GENERATION OF SCA DOMAIN PROFILE DESCRIPTORS FROM UML 2.0 MODELS

John Hogg (Zeligsoft, Gatineau, QC, Canada; hogg@zeligsoft.com)
Francis Bordeleau (Zeligsoft, Gatineau, QC, Canada; francis@zeligsoft.com)

ABSTRACT

This paper describes how SCA descriptors can be automatically generated from sets of UML diagrams. It explains the mappings from UML models and their graphical representation to XML. The relationship of this work to other aspects of SCA component implementation is explained, and also the relationship to other MDA approaches and aspects. Finally, initial results in this area using a commercial tool are briefly described.
The main finding of this paper is that SCA Domain Profile descriptor defects can be reduced, descriptor creation time can be greatly reduced and the need for scarce technical resources can be minimized by modeling graphically and applying MDA techniques.

1. INTRODUCTION

Software-defined radio (SDR) uses component-based architectures to deliver flexible radios that maximize reusability and minimize effort and time-to-market for families of products. This paper describes how the Unified Modeling Language (UML) and the Model Driven Architecture (MDA) approach, two of the main technologies developed by the Object Management Group (OMG), can be used to improve the development of SDR products.

UML constitutes the de facto standard in the industry for software development. UML 2.0 9, which was adopted in June 2003 and which is now in its finalization stage, was specifically intended to support the MDA approach 9. The essence of MDA is working at the model level at all stages of system development. The part of MDA that has received most attention is the (usually automated) transformation of models to application implementations, possibly in several steps. Other applications of MDA can also be extremely valuable 9.

The Software Communication Architecture (SCA) specification 9, 9, which constitutes the de facto standard for SDR system development in the defence industry, defines the core components and interfaces that are required to built SCA-compliant applications and platforms. It also defines the set of XML descriptor files, called the Domain Profile, which must be provided by application and platform developers to make their products SCA-compliant. The descriptor files contain information concerning the set of components that compose applications and platforms, the configuration of the components, the location of component implementation files (containing the executable code), the interconnection between the components, and the deployment requirements of the components. The set of Domain Profile descriptor files are required to enable the automated deployment and configuration of applications on SCA-compliant platform. Unfortunately, the required Domain Profile descriptors are hard to read, tedious to write and fertile ground for mechanical errors.

The SCA specification is defined using the Unified Modeling Language (UML). It makes heavy use of the port/connector concepts that are part of the 2.0 standard. Furthermore, the information contained in descriptors can be captured graphically in UML 2.0 component models. For these reasons, SCA component (resources and devices), application and platform development can greatly benefit from the type of automated transformation defined by the MDA approach.

1.1. MDA

In the context of this paper, "MDA" or model-driven architecture primarily involves graphically modelling an SCA component system, validating the correctness of the model then generating a complete set of SCA-compliant profiles (XML descriptor file sets) from the model. The generated descriptor sets are no longer the primary IP; the reusable architectural IP is captured in the model. The XML files are in fact throwaway artifacts that can be regenerated at any time. They are analogous to object code in component implementations—the precious, version-controlled artifacts are the source files.

2. BENEFITS

There are several anticipated benefits to applying an MDA approach to SCA component development. Among them are comprehensibility, speed, quality, reduced need for scarce resources and early architectural validation.

1.2. Comprehensibility

Humans learn and understand in many different ways, but very few of them find raw XML easier to comprehend than graphical diagrams. For confirmation, consider how developers use whiteboards: boxes and lines figure prominently when systems are explained. This is especially true with respect to a standard like the SCA where two boxes and a line may represent multiple hexadecimal unique IDs.

1.3. Scare Resources

SCA is a comparatively new standard. This makes SCA experts hard to find. It is also a powerful but complex standard, which makes SCA experts hard to train. When the authors of this paper asked SDR project leaders to identify their single greatest problem in developing a software radio, "finding resources" topped the list.

Resources are obviously needed to create descriptor files initially. Additionally, descriptor files must be maintained as an application evolves. Since domain profiles have many cross-references between files, a local change may have unanticipated consequences elsewhere. Anecdotally, suboptimal architectures have been retained because the risk of modifying them was considered to be too great.

Expertise is also needed simply to review descriptors in environments where every human-generated artifact must be reviewed. This is time-consuming, error-prone work and the XML notation puts a lot of syntax in the way of the semantic content that should be the focus of an intelligent, expensive human. For these reasons, any technique that reduces the need for SCA expertise will help SDR projects.

1.4. Development Speed

"A picture is worth a thousand words" and this is especially true in MDA. A few graphical elements created with mouse clicks and drags can represent many lines of XML descriptor files.

The immediate result is that graphical SCA profiles can be created much more quickly than manually written ones. Review time is also decreased since reviewers can more readily understand the meaning of the graphical model—and whether it means what it is supposed to. Maintenance time also goes down since changes are easier to understand and can be automatically propagated across sets of files.

This increased speed allows more system iterations to be delivered, improving quality.

1.5. Quality

Increased development speed is not the only contributor to increased quality in an MDA approach to SCA configuration.

The details of generated XML are "correct by construction". Syntactic errors do not occur. At a higher level, designs are validated for SCA compliance.

Correct details and SCA compliance do not guarantee a correct architecture; it will always be possible to create a fundamentally wrong-minded model. However, an architecture displayed as a graphical model is much easier to visualize. It is also much easier to explain to colleagues, peer reviewers and managers, so errors can be discovered and resolved quickly. At the same time, the assured low-level correctness allows the architect to concentrate on the large and important issues in the design: are the right elements connected to each other, and using the appropriate protocols? The errors made at this level are the really expensive ones, and this is where graphical models provide the greatest value.

Increased development speed increases quality, and increased quality increases development speed through reduced rework time. The feedback loop is positive.

1.6. Architectural Guidance

A third advantage of modelling and validating SCA architectures is the architectural guidance this provides: design conformance with the SCA can be checked from early in the software and system lifecycle. This is related to but distinct from low-level quality validation; it is "doing the right thing" as opposed to "doing the thing right".

This value comes from applying MDA early in the lifecycle, not at the end after implementation is complete. It is in the domain of architects, not implementers. MDA should be employed throughout the software lifecycle to reap full benefits.

3. SCA AND COMPONENT-BASED DEVELOPMENT

An SCA application is composed of components that communicate through ports and interfaces connected by connectors. The ports and interfaces are described in a set of XML descriptor files that accompany each set of implementations of a component, and an application has its own descriptor file that specifies how the components are connected together through their ports.

In the application delivery cycle, the building blocks are the components of the system. These may be reused from a library or created specifically for an application. At the profile level, they are sets of interfaces and components. The architect (or team of architects) assembles components into applications by wiring together their ports and interfaces.

The architect will not necessarily provide all details of a component; that's the job of the implementer (or component developer or software engineer, depending on the preferred title for the role). The architect and implementer may in fact be the same person, but the roles are different because the tasks are different. The architect is concerned with external views and how components are connected, without reference to their internals. The component developer's responsibility is to deliver that external contract by delivering an executable with the specified behaviour—and a descriptor set that includes full information about the component, including low-level details that the architect elides.

4. SOFTWARE PROFILES

How does MDA actually apply in an SCA environment? How are SCA constructs represented graphically? Most of the notation will be familiar to SCA experts because it appears in the (non-normative) *SCA Developer's Guide* 9.

1.7. Components

The basic building block of an application is the component (device or resource). The *Guide* doesn't present a graphical notation for defining the interface of a component, but that is natural and intuitive: a component is a rectangle where the standard "lollipop" symbol from UML 2.0 is used to represent an interface, and a square on the border of the rectangle represents a port. The conjugation of a port is shown by its colouring: white for a Provides port and black for a Uses port.

1.8. Assemblies: Structures

The key to representing SCA assemblies is to use UML 2.0 structure modelling, also known as role modelling. This is a new concept to most UML users, but it is vital to validating and generating XML software profiles.

Structures are architectures: they show the elements of an application and who speaks to whom. An SCA application assembly diagram is a structure diagram showing component roles connected through their ports and interfaces by connectors.

Role or structure modelling is not class modelling. A class model describes properties common to all instances of a class; for instance, an Antenna component has a connection to a Receiver component. This sounds adequate as long as every component is a singleton: i.e., as long as there is only one component of that class in the system. If the system has two Antennas and two Receivers, the class diagram cannot describe which communicates with which.

Role modelling is not object modelling. At first glance they look very much like object models, or models of fully reified instances. Object models are very useful for understanding a system during the analysis phase, but they lack reusability. If an architecture may have multiple instances of an element, the element itself cannot be represented by a single instance.

The SCA itself appears to ignore this issue: the name for a resource component element is "componentinstantiation". Because the SCA doesn't support hierarchical modelling directly today it may appear that components can be adequately represented using instance modelling. This is not the case. Leaving aside the potential representation of hierarchical structures (discussed below under "Future Work"), instances don't adequately represent devices.

1.9. Devices

An SCA software profile is a highly portable entity: it can potentially be deployed on a range of platforms, or in a number of ways on a single platform. This means that a software profile cannot include any information about a specific hardware device which may or may not be present in the target platform. The actual device is not known at configuration time when an XML file is written or a graphical model is created; it is only defined dynamically at deployment time.

Nonetheless, it is necessary to specify that software components have connections to devices. Devices must therefore appear in graphical models as *roles*, not objects. A device in a software assembly has no defined class; it is known only by the connections it has with components and the interfaces it provides to them. At deployment time this role is played by an actual logical device.

These roles are the essence of structure modelling. They not only allow portability, they also support highly dynamic architectures. When an element in an assembly is seen as a role instead of an instance, it can be dynamically replaced with any object that can play the role (i.e., has a matching set of interfaces and semantics.) This applies not only to devices, but also to dynamic components created by a ResourceFactory component.

1.10. Non-Normative Graphical Representation

The SCA contains the structural concepts of UML 2.0 (structures, ports and connectors) but the only notation specified in the standard itself is the XML representation.

The graphical representation of 9 is non-normative. There are probably two reasons for this. First, the SCA preceded the release of UML 2.0, and the interpretation of ports and connectors is more closely related to the OMG's CORBA Component Model (CCM) 9. Second, the tools available at the time the SCA was released couldn't depict structures. The Whorfian hypothesis may apply here.

5. PLATFORM PROFILES

Platform-independent SCA software profiles are deployed on environments specified by platform profiles: descriptors of the logical hardware. Models of platform profiles are comparatively simple in the SCA today.

An SCA node is composed of devices, services and managers. Informally, it can be thought of as representing a box in a complex system. It can be graphically represented using the node representation of UML deployment diagrams. Each node is composed of devices and services, and these are represented in separate areas of the node showing the underlying Core Framework elements (Managers and Services) and the devices that make up the node.

A platform as such does not appear as an element in the SCA, but it is a set of nodes. It is convenient (but not essential) to present the collected nodes in a platform as a UML deployment diagram.

The mapping from the resource components of the software profile and the devices of the hardware profile is done at deployment time, well after the configuration modelling described in this paper. That task is currently entirely within the core framework, although there are opportunities to assist this task using model-level specifications outside the current SCA. Today, the most significant specifications in this area are hostcollocation constraints.

6. EXPERIENCE

The theoretical advantages of using an MDA tool to model and generate SCA descriptors are appealing. However, the true value can only be determined by applying an industrial tool in an industrial setting.

Results have been accumulating from multiple SCA development projects across various companies delivering SCA applications, including some with hundreds of ports and connections. They are clear, consistent and very positive. While the impact of improved quality is still being quantified, compelling feedback is available in two areas: productivity and architectural guidance.

1.11. Productivity

Initial demand for descriptor creation automation is usually driven by the simple return on investment of reduced developer effort. Productivity is fairly easy to quantify: simply track the resource requirements to write a representative software profile by hand in XML, then model the profile (or one of equivalent complexity) and generate the XML.

The numbers from user (as opposed to tool vendor) evaluations have been reassuringly consistent, with automated productivity in the range of an order of magnitude higher than manual productivity. In the larger applications, manual creation and maintenance of descriptors would be a truly daunting task; the sheer potential for mechanical error might be even more significant than the person-years saved.

1.12. Architectural Guidance

One striking lesson from user experience was the payback in using an MDA tool at the beginning of the development cycle to validate basic architectures. Quantifying this kind of return on investment is difficult, but anecdotal results suggest that it can be even greater than the productivity ROI.

Users have discovered that early tool-based architectural verification can prevent the need for rework. The result is decreased development effort and increased schedule predictability.

1.13. Simplifying the Task

Any approach to software development can look good from the proverbial 50,000' level. When real descriptors must be delivered, the details can be messy and difficult.

In practice, the most confusing commonly used pattern in the SCA lexicon seems to be deviceusedbythiscomponentref. An SCA software profile specifies the software components of an application and their connections, but it is (mostly) independent of the hardware platform or platforms on which it will be deployed. The connection between the two involves an allocation property on a device, a "uses dependency" between a component and a property (independent of where that property is located) and multiple cross-references to entities as either human-readable strings or hexadecimal unique identifiers. Simply determining the range of values used can be difficult.

This is the kind of place where model-based architecture can greatly simplify the developer's task. Simply presenting the user with a menu of valid choices eases the task. Auto-filling the most likely values (where they can be determined) can further decrease the required domain knowledge.

7. FUTURE WORK

Is the application of MDA to SCA development a completely solved problem? Most definitely not. Several areas invite further work.

1.14. Further Simplification of Modelling

As tool experience on real-world problems grows, further opportunities to guide users through model complexities arise. For example, even with the simplifications described previously, deviceusedbythiscomponentref still a complex pattern. Empirically, allocation properties and dependencies are sometimes created simply to support it with no deeper model significance. Given this, a fully automated creation of all elements (properties, dependencies and references) could ease the profile creation task.

1.15. Hierarchical Structures

The SCA today supports only a single level of composition: Components are combined together into Software Assemblies, but Assemblies are not reusable. Hierarchical decomposition is perhaps the most fundamental tool that humans use to solve any large problem, and SCA applications will only grow over time. Without MDA-based approaches, the SCA will hit scalability limits, and soon. This is recognized in similar efforts such as the OMG's Deployment and Configuration specification 9, which does support hierarchical composition.

The beauty of MDA is that restrictions at one level of modelling can be abstracted away at a higher level. The SCA is a reified platform for defining component architectures. These architectures can be abstracted using the UML 2.0-based non-normative graphical notation of 9, and the individual component instances in an assembly can themselves be reusable structures 9.

Hierarchical composition of components is only possible if they are modelled as structures. Object models cannot be composed because the objects are concrete instances that cannot be reused in multiple places. However, structure modelling is mature and well-understood, at least by a core community of modellers.

1.16. Adaptability

The current validation checks and generation mappings from models to XML are largely fixed, although some customization is possible. Consortia, companies or projects may have special needs or standards. Therefore, user-definable mappings are an appealing concept.

1.17. Further Domains

Taking adaptability one stage further, not all software-defined radio systems will be based on the SCA standard, and not all SCA applications are SDR. Furthermore, not all component-based development is either. The technology described in this paper will be needed in wider domains.

The simplest form of standards independence is to generate profiles for different versions of a single standard from a single, unchanged model. Today, SCA versions 2.2, 2.2.1 and 3.0 require no customization. However, future versions may need different generators.

The next level of standards independence is generating descriptor sets for different but related standards—say, either SCA or the OMG Deployment and Configuration specification. Again, a single model can be the source for either target.

Ultimately, it is inevitable that we will see mode support for modelling, validation and generation of descriptors and other artifacts for component-based development based on unrelated standards. This technology is too valuable to only be used in SCA applications.

8. CONCLUSION

The power of MDA has previously been proven in implementation of systems not based on components. It is equally applicable to component-based development of SCA applications. The anticipated benefits of development speed, improved quality, reduced resource needs and improved architectural guidance have been proven in practice.

The future of MDA in software-defined radio looks very bright.

9. REFERENCES

[1] OMG Specification: *UML 2.0 Superstructure Final Adopted Specification*, http://www.omg.org/cgi-bin/doc?ptc/2003-08-02, 2003.

[2] J. Miller and J. Mukerji (eds.), *MDA Guide*, http://www.omg.org/docs/omg/03-06-01.pdf, 2003.

[3] J. Hogg, "Model-Driven *: Beyond Code Generation", *OMG MDA Implementers' Workshop*, http://www.omg.org/news/meetings/workshops/MDA_2004_Manual/5-1_Hogg.pdf, Orlando, FL, USA, 2004.

[4] Joint Tactical Radio System (JTRS) Joint Program Office, *Software Communications Architecture Specification Version 2.2*, http://jtrs.army.mil/documents/sca_documents/V2.2/SCA_v2_2.zip, 2001.

[5] Joint Tactical Radio System (JTRS) Joint Program Office, *Software Communications Architecture Specification Version 3.0*, http://jtrs.army.mil/documents/sca_documents/V3.0/SCA-V3.0.zip, 2004.

[6] A. Gonzales and R. Hess, *SCA Developer's Guide*, http://jtrs.army.mil/sections/technicalinformation/developersguide/pdfs/pdfs.zip, 2002.

[7] OMG Specification: "UML Profile for CCM: Final Adopted Specification", http://www.omg.org/cgi-bin/doc?ptc/2004-03-04, 2004.

[8] J. Hogg, "Applying UML 2.0 to Model-Driven Architecture", *OMG MDA Implementers' Workshop*, http://www.omg.org/news/meetings/workshops/MDA_2003-2_Manual/5-1_Hogg.pdf, Burlingame, CA, USA, 2003.

[9] OMG Specification: “Deployment and Configuration Final Adopted Specification”, http://www.omg.org/cgi-bin/doc?ptc/2003-07-08, 2003.

[10] J. Hogg, “The Problems and Promise of UML 2.0 Structures for SCA”, *OMG Software-Based Communications Workshop*, http://www.omg.org/news/meetings/workshops/SBC_2004_Manual/05-2_Hogg.pdf, Arlington, VA, USA, 2004.