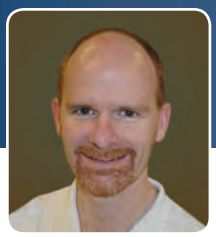

By Curt Schwaderer

CompactPCI & AdvancedTCA Systems

Pioneering model driven development

Since software and systems engineering has been around, so too has systems and software modeling. Modeling delivers benefits that include proving out the system prior to detailed design and implementation so that errors can be detected and eliminated earlier in the development cycle.

Witnessing the operational aspects of system requirements is also beneficial. Modeling provides a proof-of-concept that saves time and money during the development cycle by facilitating the collective agreement of all stakeholders pertaining to the requirements and functional operation of the system.

However, modeling has historically been disjoint from the development cycle. Often the model is discarded once the concept has been proven, as was the situation with CASE tools in the 1990s, and development begins from scratch. This chasm between modeling and the development process leads to design flaws and an implementation that does not behave according to the original model. For many years developers have been striving to make the model a more integral and iterative part of the process. This approach would enable the model to serve as the design and implementation starting point for the development. What's more, the model could be evolved as the development proceeds. In this column, we will explore a development approach called Model Driven Development or MDD. The evolution of MDD has been going on for some time, but recent modeling and development tool innovations are making MDD a viable approach to software and systems engineering.

Proliferating MDD throughout the embedded space

PrismTech, a company whose core competency has historically been software tools and middleware, believes now is the time to make MDD a reality. The company embraces the proliferation of MDD throughout the embedded systems space. PrismTech sees MDD as useful for enterprise, net-centric, and mission-critical applications. They have successfully used the approach to develop real-world embedded software products. (At www.prismtech.com a comprehensive white paper is available that covers the details of the MDD approach.)

What is model driven development?

MDD is a development practice where high-level, agile, and iterative software models (often domain-specific) are created and evolved as software design and implementation takes place. The key defining characteristic of MDD is that the model literally becomes part of the development process. Contrast this with an approach such as the waterfall development process where modeling appears as a separate step in the process and tends to get left behind once the development proceeds to the next phase.

Work in defining model driven development best practice is ongoing by various industry groups, including the Object Management Group (OMG), an organization of end users and software vendors for developing industry standards for the software life cycle. The idea behind MDD is to model an application and use a single repository where the high-level model of that application (and its systems environment) is maintained. This enables non-software-engineering stakeholders (for example systems designers and engineers) to maintain control (well past the initial modeling stage) of requirements and functionality for the system during development. Thus, MDD allows business and technical personnel to help define and maintain the model in a very real and meaningful way, resulting in systems and applications that more accurately satisfy requirements on initial deployment.

MDD's component architecture together with automatic code generation (latter-day interpreters between the model and the source and unit test code) and high-performance, low-overhead middleware meet the needs of networked enterprise and embedded systems where application functionality is distributed among network nodes.

As a result, MDD is highly compatible with publish/subscriber middleware, such as Data Distribution Service (DDS) and distributed object middleware such as CORBA.

MDD hurdles

Software and system modeling has been around for many years. Typically the modeling occurs at the proof-of-concept stage using third generation languages such as C/C++ or Java. Universal Modeling Language (UML) tools have been refined over time. Use of UML has spanned anywhere from modeling to being used to develop designs. However, until this point, UML has historically lacked semantics to describe the system at the business or embedded application level. Work must be done to define the application within its domain, with a separate step required to translate that description to UML. Software frameworks, libraries, and software components available on the Web in open source form have also been used for modeling.

The biggest obstacle to achieving MDD with these traditional tools and components is *domain specificity.* Application stakeholders need to describe the system requirements and functionality in the domain of that system, including:

- Information that the system keeps
- Behavior of the components in the system
- Availability of the information to those components

The semantics of UML or third generation languages such as XML, C/C++, and Java, should not limit these descriptions. Nothing is worse than to have an experienced stakeholder in the domain unable to express to the engineering team important aspects of the system because the programming language puts up obstacles.

A chasm

So, there is a chasm that exists between the architecture created by the domain experts and third generation languages, tools, frameworks, and other components used to implement it. In this chasm, domain specific requirements decouple from the technical design and implementation of

the system. This decoupling can lead to derailment of the project if a system does not function as intended.

MDD bridges this chasm using Domain Specific Modeling Language (DSML). Those familiar with the Eclipse Modeling environment will recognize that DSML brings high-level engineering/modeling work and low-level implementation programming together as two well-integrated parts of the same job.

The benefits of the MDD approach are numerous. There is a significant increase in cross discipline collaboration. The existence of the DSML means nontechnical and technical participants all speak the same language without getting into technical programming details that may cause miscommunication or errors in interpretation of the system requirements. Learning curves associated with this approach involve understanding of the domain specific terminology, not details behind programming languages and tools.

Model driven development and domain specific languages

So, how is model driven development done? First, a DSML must be defined for the application, forming a foundation for communication among all stakeholders. Next, a domain specific editor captures requirements and high-level operation of the system. Finally, transformation engines that take the output of the domain specific editor and generate source and/or link to functional components that can be built into the system executable complete the model driven development environment.

Using model driven development, modeling and programming activities blend into the same development process. The models become direct input into the development process, not just an aid to design activity. Domain specific models are developed to be machine processed for integration with implementation generators. Further, implementation of the model is done from the design perspective, not according to third generation language semantics.

PrismTech is using tools and middleware that implement DDS concepts and processes in its OpenSplice product family (Figure 1). OpenSplice is targeted to aid in the development of domain specific languages. It also allows for generators that transform the domain specific model into executable software for a specific enterprise or embedded platform or product family. Table 1 describes the steps involved in model driven development.

Each of the steps for model driven development is accounted for in the OpenSplice product line. Using OpenSplice, the model produces a collection of development artifacts that are used to generate the implementation.

A Software-Defined Radio using MDD

PrismTech has developed a Software-Defined Radio (SDR) product using MDD and the DDS tools. To drive home the impact of MDD, we will overview some of the key points of the PrismTech SDR product so you can have a look at MDD in action.

Like many other areas, radio technology is moving from custom hardware systems to general purpose platforms with software that implements the functionality of the radio. This enables radio systems to become easier to maintain, lengthens the deployment life cycle, and reuses or enhances existing software algorithms.

Development of SDR starts with the Software Communications Architecture (SCA). SCA is an SDR model created in 1999 by a consortium of leading military radio developers. SCA isolates abstractions and describes how they work together within the domain of an SDR.

The next step is to create a formalized grammar, or Domain Specific Language (DSL). Most SCA implementations use third generation languages applied directly from the SCA specification. The PrismTech approach raises the level of abstraction to define formalized meta-model components that are expressed in terms of a language workbench, specifically, the Eclipse Modeling Framework.

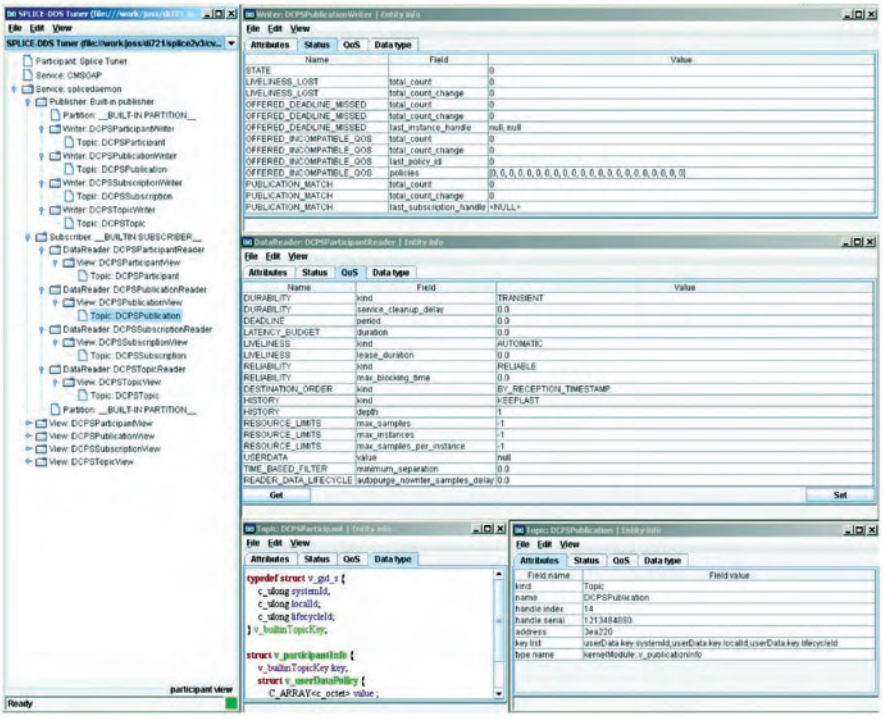

Figure 1

General MDD Methodology	PrismTech OpenSplice Process
Isolate the abstractions and how they work together	The Data Distribution Service Specification
Create a formalized grammar for these – DSL	Create a formalized DSS *meta-model*
Create a graphical representation of the grammar – DSGL	Create a DDS-specific graphical tool
Provide domain specific constraints – DSGL, DSCL	Program the constraints into the tool
Attach generators for necessary transformations	C/C++ and Java generators

Table 1

Once the DSL is completed, a meta-model for SCA is specified. The meta-model is the key ingredient that allows end users with knowledge in the domain to specify what they want to have, not how it is done. The meta-model also allows the end user to directly affect model implementation.

Next, a graphical description of the grammar is created, called Domain Specific Graphical Language (DSGL) and Domain Specific Views (DSV). The PrismTech Spectra SDR PowerTool modeling environment allows the end user to define functional components and connect them together to specify processing characteristics. All of this is done using the DSL.

The language workbench's programming facilities implement, within the domain specific graphical language, the Domain Specific Constraint Language (DSCL). So, the programming within the language workbench includes a structural framework for the rules of the domain. When end users develop their components and systems, the constraint language ensures that they are conforming to the domain requirements. The constraint language is an important part of maintaining the correctness of the model and requires domain experience.

Ultimately, the DSL is transformed into an executable format. This occurs through Domain Specific Generators (DSG). For embedded systems, PrismTech notes these generators may have multiple targets on the platform, including FPGAs, general purpose processors, or digital signal processors. As a result, the DSGL tool needs to be able to iterate over the model, interacting with multiple domain specific code generators to produce multiple types of executable code.

PrismTech's OpenSplice DGSL tool can declare a component in the SDR domain. This tool can also create software artifacts from the DSGs, keep code coverage information, and keep test case generation information for the components within the model. I was impressed by his tool chains' ability to bridge the gap between a graphical model component and its VHDL description to C source code with test case and code coverage information.

PrismTech notes a number of advantages of the OpenSplice MDD paradigm in the modeling and development of SDR versus using third generation languages. Domain experts can be far more productive in the MDD environment due to the abstraction. The domain specific generators provide a high level of correctness with prevalidated logic and the software artifacts derived directly from the model. The SCA architecture is captured in the PrismTech meta-model, lowering the cost of entry significantly for companies wishing to move to SDR. With the MDD paradigm, it's possible to determine defects in the system at modeling time rather than during system integration. This approach helps achieve architectural consistency across the model and implementation.

Conclusion

This SDR example covers specifics in the SDR domain. However, the MDD process has more far-reaching implications than one example in one domain. The OpenSplice MDD/DSS environment is capable of extending model driven development into a number of areas. Systems and software continue to become increasingly complex. Now more than ever, it's important to leverage domain expertise into the heart of the development process. Model driven development and products make this transition possible in an efficient and effective way.

For more information, contact Curt at cschwaderer@opensystems-publishing.com.