Domain-specific modeling significantly reduces development time

By James L. Hammond, MetaCase

This article introduces the fundamental concepts and benefits of DSM and discusses the implementation of a DSM language for a home control system with the latest release of MetaEdit+.

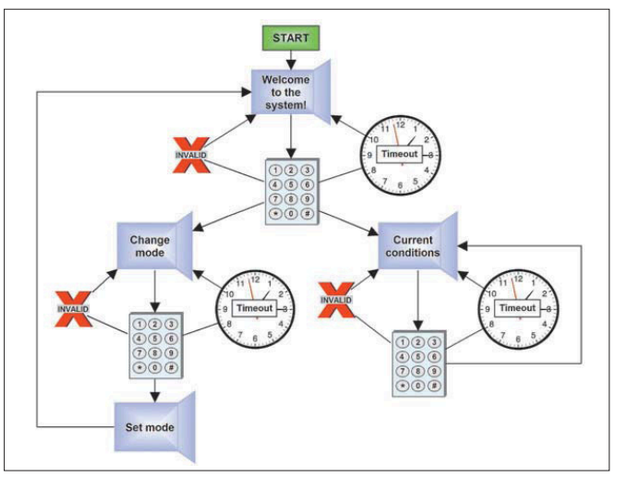

Figure 1. High-level model based on MenuOption language

The need for greater productivity - in the face of mounting complexity - is a ubiquitous challenge that the entire software engineering community is dealing with. Nowhere is this challenge more obvious than in the embedded arena; manual coding practices are simply not able to cope with ever-growing demands, and the embedded industry is now turning towards more modern development approaches to achieve its current and future goals. Domain-specific modeling (DSM), with its capacity for automated full code generation, is rapidly becoming the preferred approach for many embedded developers.

Domain-specific modeling is a model-based software development approach that focuses on the use of visual models as primary artifacts in the development process. DSM raises the level of abstraction beyond normal programming languages by directly specifying the solution using domain concepts. The desired code can then be automatically generated from these high-level models using customized code generators. In addition, a single model can be used to generate code for multiple targets (e.g. localization, early prototyping, build scripts in addition to code). This automated code generation from models is possible because of domain specificity: the language and generators are designed to tightly fit the requirements of a single company. The result of this specificity and automation is a dramatic increase in productivity. Industry experiences from a wide range of companies and application domains (Alcatel-Lucent, Panasonic, Nokia, EADS) have consistently proven that DSM is 5 to 10 times faster than current, manual development practices. The reason for this is that software engineers are able to focus on the functionalities they are looking to develop - not on their laborious, routine implementation.

Quality improvement is another key benefit of DSM. Quality remains a significant issue for all embedded developers, but particularly for those involved in safety-critical systems. Generated code avoids the risks of careless mistakes, syntax problems, and logic errors. It is common in manual coding practices for errors to be introduced during updates, an understandable occurrence when multiple developers need to make multiple changes to multiple sections of the code. With DSM, a single change in the generator is often enough to correct all the occurrences simultaneously. A compelling study from the United States Air Force comparing DSM with component-based development found that domain-specific languages and generators were not only three times faster than the code components, but led to 50% fewer errors - a significant finding in an industry focused on mission-critical systems.

In the following section we will look at the creation of a DSM language for a component of a remote home automation system using the latest release of MetaEdit+. The component of the system we will discuss is a remote control module that enables bi-direction communication between the control system and a mobile phone. Two modeling languages are used in this case: a high-level interaction layer (OptionMenu language) and a low-level audio output layer (AudioOutput language). Figure 1 shows an example of a model created with the high-level OptionMenu language. Modern modeling tools, such as MetaEdit+ in this study, enable the sharing and reuse of objects and items across different projects, languages, or models, thereby making it simple to integrate these two languages into a common solution. For these high-level models it is possible to generate the code into the 8-bit microcontroller.

The first step in the process is to define the modeling language concepts. There are various sources for these concepts, with typical examples being: product components, architecture, characteristics, required output, etc. Specifically

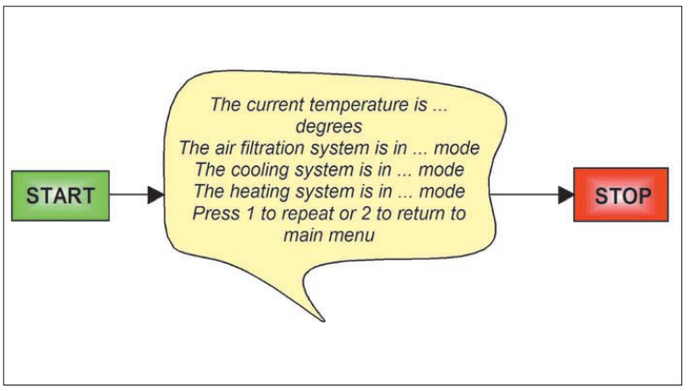

Figure 2. Low-level model based on AudioOutput language

designed tools in MetaEdit+ for describing modeling languages make it easy to transform these concepts into elements of the modeling language. As the language creation process is agile, test models can be built with the language and modified at any time during the creation process. For the OptionMenu language the main concepts would be AudioOutput (audio output from system to caller) and RemoteInput (system waits to receive input from caller). The process would go as such: from the start position the flow would move to the AudioOutput (audio menu of possible choices and responses) to the RemoteInput where the system would wait to receive a command (key press) from the mobile user. The type of input expected is specified as a property of the RemoteInput object. For each possible input there exists a relationship from one AudioOutput to another.

The AudioOutput language would then be created to control the audio responses and menus sent from the system to the mobile device. The possible audio statements would be modeled in their own AudioOutput sub-diagrams, with many audio fragments (or whole items) being reused in multiple models. Figure 2 illustrates a simple model created with this AudioOutput language.

Once the modeling concepts are defined, the modeling rules need to be established. These rules cover how the concepts can be used and connected with each other. MetaEdit+ provides a variety of predefined rule templates to choose from to address a wide range of possible scenarios - however custom rule definition is also supported. The establishment of these policies at the language level assures that all developers adhere to the same domain rules. In the home automation system, the rules would specify how the objects can be connected with various flows. For example, if there is an invalid key press there would be an IvalidInput flow back to the previous item. In the event that no input is received there would be a Timeout flow which would also return back to the previous menu. This language also allows an AudioOutput to be followed immediately by a second AudioOutput, enabling reuse of the concept.

A very useful aspect of DSM - also a great looking one - is the ability to utilize custom graphics within the models and modeling language. These symbols can either be drawn using the integrated drawing tools of MetaEdit+, or imported in vector or bitmap format. Using symbols that closely approximate the real world items they represent makes development far easier for the modeler, and also produces models that can be understood by any stakeholder familiar with the domain concepts. The symbols also support dynamic graphical behavior by using conditional property values that can display values calculated by the generators.

The final step in building a DSM solution is the creation of the generators. As was the case with the modeling language, domain-specificity is also important here. These generators can be created to for a wide variety of purposes: reports to check the consistency of the models, produce metrics, analyze model linkages, create data dictionaries, produce documentation, generate code or configuration information, and to export models to other programs (e.g. simulators, version control systems, etc). The integrated generator editor included in MetaEdit+ helps in the specification of template-based generators that crawl through models and output model values and fixed or conditional text. This domain-specific code generation assures that the code generated from every developer always meets the exact requirements established by the organization expert, because they are built directly into the code generator.

Once the language and generators are complete, the organization is ready to start creating models and achieving the significant productivity and quality benefits previously mentioned. The time it takes to fully create the language and generators is dependent on the domain, but experiences with MetaEdit+ over the years have shown that it usually ranges from a few man-days to a few man-weeks. For example, the languages needed for this remote home climate control system would take approximately 4 to 6 man-days to complete. This small initial investment in time is quickly recouped thanks to the increased productivity delivered. Development in the domain on which this case study is based became 5x faster, meaning that one week's work could now be accomplished in one day. An important point to keep in mind is that the language will almost certainly need to change and grow over time, and it is essential that the chosen tool support is able to handle this evolution. MetaEdit+ fully supports the idea of language evolution; modifications can be easily made to the modeling language at any times and these changes can be automatically and non-destructively propagated to all models defined by that modeling language. ■