United Business Media

April 2008

Embedded Systems Design europe

Use software to define silicon

p. 15

Improving productivity and quality with domain-specific modeling

BY JAMES L. HAMMOND

As calls for increased software development productivity continue to mount, organizations are constantly searching for effective ways to satisfy the demand. Domain-specific modeling (DSM), with its capacity for automated full code generation, is achieving increased attention as a proven approach that achieves this productivity goal, while also improving quality. This article will introduce the fundamental concepts and benefits of DSM, and discuss the implementation of a DSM language for the development of IP telephony services using the latest release of MetaEdit+.

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 in a language that uses concepts and rules from the problem domain. Raising the level of abstraction has historically been the catalyst for significant development productivity improvements (e.g. the move from assembler to third generation languages), and the abstraction rise that occurs with DSM is another jump forward.

Once these high-level models are created, code can then be automatically generated from them using customized code generators. Code can be produced in any programming language or for any development paradigm. Existing platforms, libraries, components, and legacy code can all be effectively utilized during the generation process. While it certainly is possible, DSM does not require that all code is generated from the models – but it does take the position that anything modeled by the modeler should generate complete code that does not need to be changed manually afterwards. The generated code is functional and efficient– ideally looking like the code handwritten by the lead developer that defined the generator(s).

DSM helps to hide the implementation concepts from the models –

Model of a call redirect to voicemail service

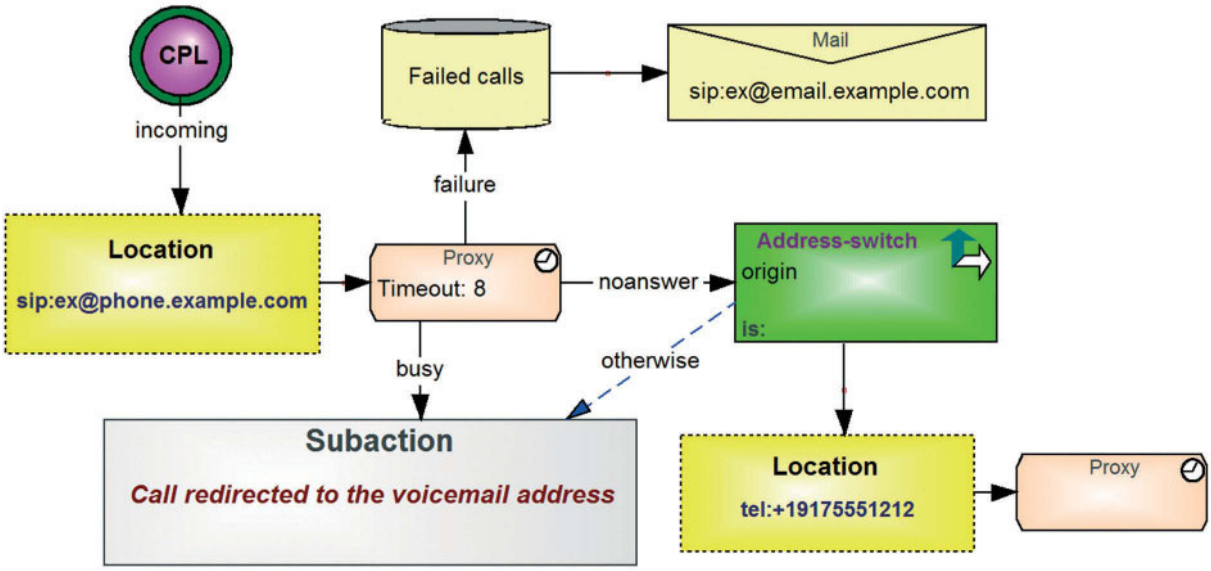

Figure 1

making it unnecessary for the modeler to be concerned with them. In addition, a single model can be used to generate code for multiple targets and functions (e.g. localization, early prototyping, build scripts) – an immensely valuable ability for the myriad of companies that expend considerable time and energy porting their products or components.

This automated code generation from models is possible because of domain specificity: the modeling language, code frameworks, and generators are designed to tightly fit the requirements of a single company.

The result of this specificity and automation is a reduction in complexity and a dramatic increase in productivity. Experiences from a wide range of companies and application domains (e.g. Lucent[1], Panasonic[2], Nokia[3], EADS[4]) have consistently proven that DSM is 5 –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. Industry analysts are also asserting the benefits of DSM, e.g. in 2004 the Burton Group reported[5] that they saw domain-specific languages and custom metamodels to be the greatest aid to productivity.

Improved quality is another key benefit of DSM. Sustaining or improving quality standards remains a significant issue for all software developers, and maintaining quality while confronting increased complexity, can be immensely challenging. 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.

COMPELLING STUDY

A compelling study[6] 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 percent fewer errors – a significant finding in an industry focused on mission-critical systems.

An important point to keep in mind is that every 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.

Many organizations have learned this lesson the hard way, having found themselves left with useless models that are unable to evolve as needs and requirements change. MetaEdit+ fully supports the idea of language evolution; modifications can be easily made to the modeling language at any time and these changes can be automatically and non-destructively propagated to all models defined by that modeling language.

While the key benefits of domain-specific modeling – improved productivity and quality – are desirable to just about every organization, DSM as a development approach is not. When building any application or feature for the first time that has never been developed before, it is illogical to assume that one is going to be able automate the process at the beginning.

The organization must be familiar with what they are doing, before they start to automate their development practices – to the best of my knowledge this hold true for all areas of automa-

tion. Consequently, companies that work on short term projects, unaware of the general requirements of the next customer, are unlikely to achieve much success with DSM. But for those organizations that make a range of similar products, or build applications on top of a common library or framework (e.g. companies with a product line), the benefits of DSM are nothing short of astounding.

MODELING LANGUAGE CREATION

In this section we will walk through the process of creating a domain-specific modeling language with the latest release of MetaEdit+[7], an IP telephony service language as an example. This use case is particularly relevant today, as a fundamental change is now occurring in the telephone service arena. In the past, network-based services were only created by service providers due to the special knowledge required, use of proprietary tools, and lack of user-specific customization options. With the introduction of Internet based telephony, the rules have change, allowing a much wider group to define these services.

One of the first required activities when building a modeling language 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 designed tools in MetaEdit+ for describing modeling languages make it easy to transform these concepts into elements of the modeling language. The language creation process should be agile, allowing the metamodeler to define the language incrementally and to test it at any time by creating specifications.

For the IP telephony services language the modeling concepts are pure call processing concepts, e.g. signaling operations, switches, locations, and non-signaling actions. Service engineers are already familiar with all these, even if the final service creation code has been done manually by programming.

Generator editor of MetaEdit+

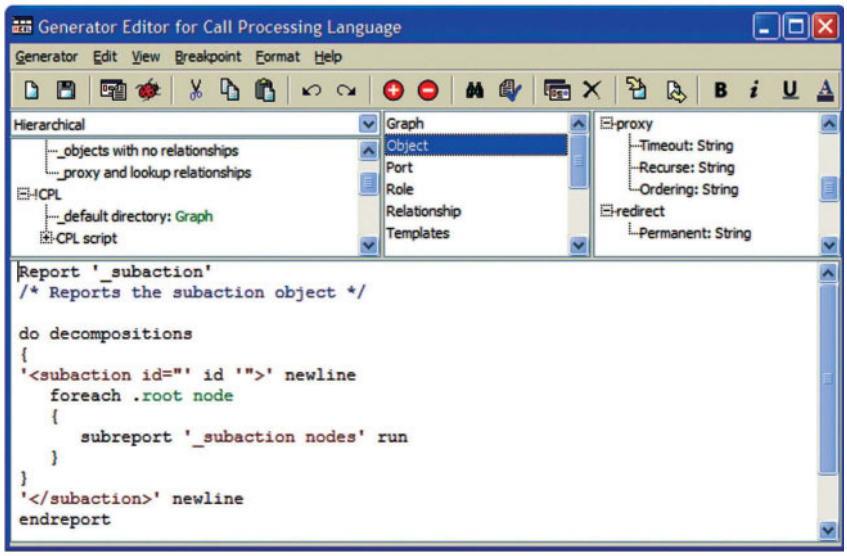

Figure 2

Now with DSM, such implementation code can be generated form these high-level service concepts.

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 – a situation that rarely occurs when relying on in-house standards manuals and documentations. In the use case language, the majority of the rules were able to be defined directly in the metamodel.

Examples of the rules established for the language are: there can only be one start element of each service, the root concept must be in only one relationship to start the call process, and the number of connections each object can have.

A very useful aspect of DSM – also a visually appealing 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 generator definition below produces the code needed to call other call processes that may be used as reusable actions within the main call process. In this way the generator becomes modular and easy to modify.

The integrated Generator Editor included in MetaEdit+ helps in the speci-

Proxy object dialogue

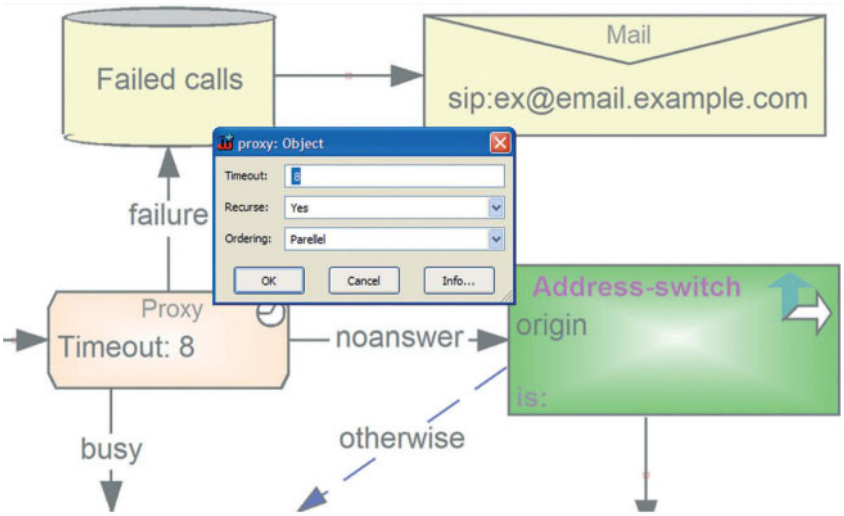

Figure 3

fication 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's expert because they are built directly into the code generator.

It should be noted that along with the rules established for the modeling language, it is also possible to include rule checking in the generators. This was the situation with this call processing case as well as all the domain rules could not be efficiently included in the metamodel since their checking would not make sense at the modeling time.

For example, each proxy needs to be accessed in the call processing flow by at least one other call element, but this kind of rule cannot be checked at the modeling time since immediately after adding a new proxy object to the model the design would be invalid. Generator based rule checking avoids this potential problem.

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 show that it usually ranges from a few man-days to a few man-weeks.

The language and generators described in this use case took approximately 5 days to complete – including testing time. This small initial investment in time is quickly recouped thanks to the radically productivity improvement.

James L.Hammond james.hammond@metacase.com) is sales and marketing director of MetaCase (Jyväskylä,Finland)

REFERENCES

1. Weiss, D, Lai, C T R, (1999) Software Product-line Engineering, Addison Wesley Longman.
2. Safa, L (2007) The Making Of User-Interface Designer: A Proprietary DSM Tool. In: Proceedings of the 7th OOPSLA Workshop on domain-specific modeling, 21—22 October. Montreal, Computer Science and Information System Reports, Technical Reports, TR-38, University of Jyväskylä.
3. MetaCase (2007) Nokia Case Study [brochure]. Jyväskylä, MetaCase.
4. MetaCase (2007) EADS Case Study [brochure]. Jyväskylä, MetaCase.
5. Haddad, C (2004) Model-Driven Development: Rethinking the Development Process. Burton Group.
6. Kieburtz, R (1996) A Software Engineering Experiment in Software Component Generation. In: Proceedings of the 18th International Conference on Software Engineering, 25 - 30 March. Berlin, IEEE Computer Society.
7. Kelly, S and Tolvanen, J-P (2008) domain-specific modeling: Enabling Full Code Generation. Wiley-IEEE Computer Society Press.

A free 31-day evaluation version of MetaEdit+, including the use case described above, is available for download at www.metacase.com.