Platform-based development on a high programming abstraction level

by Martijn Iseger, MetaCase

In a similar way as compilers transform code in current programming languages to lower abstraction level machine code, MetaEdit+ allows programming on a higher abstraction level than code. This method increases productivity.

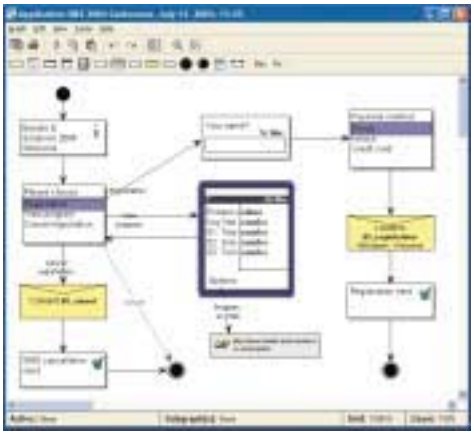

Figure 1: A domain-specific model of a mobile application that allows a user to register for the Boards and Solutions 2004 Conference

In software engineering we are constantly trying to increase developer productivity. Over the years, the most successful productivity improvements have occurred because of an upward shift in the level of abstraction in which we solve development problems. The move from assembler to third generation languages resulted in a massive 400% increase in productivity. A key success factor of this result was the automatic generation of lower-level assembler from the higher-level language. True code generation, so to speak. Not to be confused with the model-to-code translation that so many model-driven development tools nowadays offer.

Since the increase in abstraction allows major productivity growth, we need to start programming on a higher abstraction level than that of current code to attain better productivity again. A graphical representation of the code, as demonstrated by traditional modeling languages, does not constitute such a raise. The models feature the same concepts that we also find in the code, like "parameter", "class", "return value" and so on. When the models operate at this same level as the code does, it is not correct to speak about code "generation". For the same reason it is not possible to gain an order of magnitude increase in productivity from using tools that support traditional modeling languages like UML. Many of these modeling languages were designed to offer support for virtually every situation.

No matter if it is mobile phones, multimedia or navigation systems, medical, GPS or MIDI devices, you can use UML regardless of the product application you are developing. UML therefore is based on the lowest common denominator, which is the code itself. A facet of this is that it contains rules about object-oriented programming languages. However, unfortunately, it contains no rules about any specific product domain, and so it is possible to model things that make absolutely no sense or that are unwanted in specific product domains. Model-driven development tools that support these all-purpose modeling languages do not include rules about specific product domains either, nor do they offer automatic support for them. With these tools, it is possible to include rules about a specific domain in the models, but only on the same abstraction level as the models. This means that the models have to be made even more detailed manually by every developer. The developer thus needs to know and remember what is legal and stay up to date with this. The tools basically offer an environment that takes the models, efficient or not, and translates them into production code. Because models can be wrong or inefficient, the production code suffers from poor quality. A long testing and debugging phase, the most costly phase in software development, is finally required to remove all inefficiencies, inconsistencies and bugs.

What is more is that with current model-driven development tools, the association between design models and implementation (code) is proprietary to the vendor. In other words, it is the vendor that decides how code is derived from models. Despite promises from vendors that full code is "generated", developers in company or product-specific situations still end up manually editing code. The tools either form only a code skeleton or bulky, inefficient and therefore useless code. This is partly due to the modeling language used which lacks expression capabilities to make the models fit the product-specific situation, and partly to the mechanism for code production. This one-size-fits-all mechanism is simply too general for effective application in product-specific situations. Fact is that code generation is typically poor when we use a one-size-fits-all modeling language with a one-size-fits-all code generator, the rise and fall of CASE tools in the 1980s is a good example of this.

MetaEdit+ takes a different approach in that it lets companies define and apply their own product-specific modeling language and product-specific code generator. It belongs to a new breed of tools called meta-CASE tools. Domain-specific languages and generators

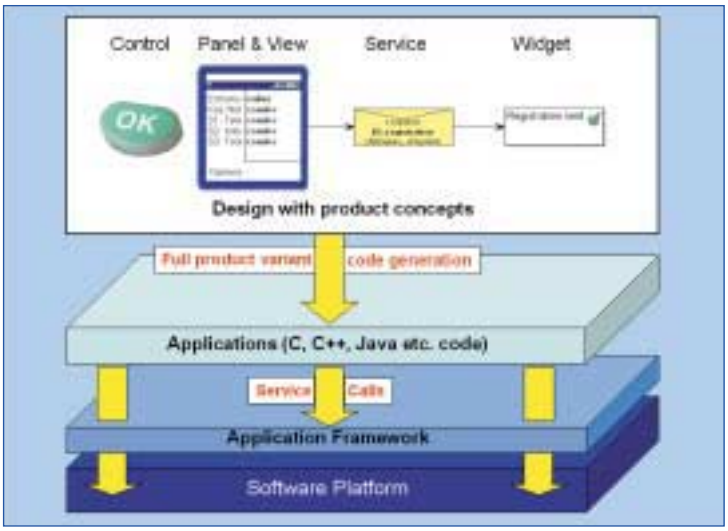

Figure 2: MetaEdit+ allows raising the abstraction level to product concepts within the confinement of a development platform

(see: http://www.dsmforum.org) offer once again the opportunity to raise the level of abstraction beyond that lowest common denominator, the code, toward product-specific concepts. From there it is possible to automatically generate lower-level code. If, for example, you are developing mobile phone software, it allows you to use concepts like "display", "soft button", "icon", platform services like "send SMS" and widgets like "menu", "list" and "notification" directly in the modeling language (see figure 1). These concepts capture mobile phone applications much better than C++ or Java classes do and all developers already know them.

Domain-specific modeling gives these concepts first-class status and thus directly supports the knowledge already available in a company. As a result, there is no need for developers to learn a new modeling language or for a company to find developers that are proficient in such a modeling language. Due to its size and complexity (over 800 concepts), developers that are experienced with UML are hard to find, which is one of the reasons for the low adoption of it. The idea of domain-specific modeling is valid for a multitude of product development areas that conduct a repetitive development effort to produce variants, either in product or in configuration.

A present-day approach for fast, reliable and efficient product development is the use of a development platform that offers the common building blocks and services to apply and reuse during actual product development. This approach is exceptionally well suited for using domain-specific modeling because it is relatively easy to achieve a framework for automatic generation of production code. The platform provides commonalities and services that provide flexibility to focus on developing new product variants. This is crucial in reducing time to market by cutting integration and testing time that typically are very time-consuming phases of development. Nowadays, developers still need to create functional product variants by designing them first in code-based modeling languages, followed by manual coding in a programming language. In essence, they are thus solving each problem at least twice.

MetaEdit+ changes this. Instead of working with modeling and programming languages that have been designed to support development of any type of application, it allows companies to take full advantage of the confinement that is posed by the development platform (see figure 2). After all, this confinement is supposed to make things easier. MetaEdit+ allows companies to raise the level of abstraction inside this confined area and to generate the code which is then used by the platform framework.

The idea is to allow developers to define product features in domain-specific models, which offers considerable benefits. First of all, these models are at

a higher level of abstraction than the code, and developers can thus perceive themselves as working directly with the product concepts they are familiar with. The modeling language actually describes the products instead of the code and this makes the models easier to read, remember and check and allows developers to focus fully on the functionality of the application. Secondly, the created domain-specific modeling language contains the correctness rules that are specific to the product and platform. MetaEdit+ consequently supports these rules and avoids that developers design against these rules. This avoids errors already in the design phase, which in terms of development cost, is the best phase to avoid them.

A domain-specific modeling language offers much better possibilities for code generation than a general purpose modeling language because it can be tailored to support product-specific circumstances. MetaEdit+ capitalizes on this fact by providing code generators that are fully open to customization. These code generators read the model instances and generate code of a quality and in a language that the development organization, not the tool vendor, specifies. If over time the generation process needs change, then the development organization is free to change it without being dependent on the tool vendor. The combination of a modeling language and code generator, both tailored to the requirements of a single company and product line, allows for full generation of production code from high-level models. MetaEdit+ thus copies the principle that was so successful 30 years ago when we went from assembler to modern-day programming languages and implements it in today's platform-based software development processes with remarkable results. MetaEdit+ users report an astonishing increase in productivity of between 5 and 10 times compared with traditional code-based techniques.

The principle that MetaEdit+ follows is that one or a few expert developers define and maintain a modeling language that is well suited for the product and platform at hand. An expert has better knowledge about the product domain than a normal developer and also

Figure 3: Implementation of domain-specific modeling by filling out forms

is capable of writing better quality code. This gives the expert excellent qualifications for defining the mechanism of model-based code generation. A benefit of this principle of central maintenance is that when changes occur on the platform-level these changes need to be implemented in one place only. All developers automatically follow the new situation without having to remember it. Code is thus always generated consistently and of the same quality by all developers, regardless of programming skill.

The required effort of this approach, creating an own-custom modeling language and code generator, is not as hard as it might seem. The modeling language and code generator do not have to offer support for every situation imaginable, as was the case for UML. Instead, the requirement is for a modeling language to support development of one product-line only. Knowledge about the concepts and rules of the product and development platform is usually already available inside the organization. So too is knowledge about the quality and structure of the final code that needs to be produced. With MetaEdit+, the expert can define and maintain both modeling language and code generator without having to write a single line of code, simply by filling out forms (see figure 3).

The tool guides the expert and reduces the time to implement domain-specific modeling. After completion, or even partial completion, MetaEdit+'s modeling tools are configured to support the domain-specific language for the rest of the development team to use. The results in fundamentally increased productivity, faster response to requirement changes and shorter training time for new staff compare favorably to training a development staff in UML, constantly having to monitor for correct usage and manual translation of models to code and vice-versa.

The higher abstraction level, automatic code generation possibilities and automatic adherence to the rules of a specific product domain make domain-specific modeling a breakthrough method for developing product-variants. The idea is to go from domain-idea straight to implementation without unnecessary manual mapping and reverse engineering between code, models and domain. MetaEdit+ makes a significant contribution to easing the pain of implementing domain-specific modeling by offering a complete environment for definition, application and maintenance of this method. Where companies compromised with all-purpose methods and tools in the past, forced by the high costs associated with building own development tools, with MetaEdit+ they no longer have to compromise, they can simply define their own.