Are your 8- and 16-bit designs running out of steam?

Tools from TTE Systems support the rapid creation of embedded software. The resulting systems have very limited resource requirements.

Our latest tool - RapidiTTy Preview - offers a complete development environment for single-processor time-triggered designs targeting 32-bit, ARM-based microcontrollers.

RapidiTTy Preview is currently available for £495 / seat.

A 30-day free trial edition can be downloaded from our WWW site.

www.tte-systems.com
info@tte-systems.com

Copyright © 2007 TTE Systems Ltd. All rights reserved. All trademarks acknowledged.
Boosting productivity with domain-specific modelling

JAMES HAMMOND explains how domain-specific modelling can improve embedded software development productivity

Domain-specific modelling (DSM) has the ability to redefine embedded software development productivity by raising the level of abstraction developers work with, providing full code generation directly from models.

Model-driven development (MDD) is continuing to gain a reputation in embedded markets as a productivity enhancing approach to software development. DSM, a particular form of MDD that makes use of domain-specific languages (DSLs), is at the forefront of this productivity rise. DSM raises the level of abstraction beyond programming by specifying directly using domain concepts. The final products are then generated from these high-level specifications. This automation is possible because both the language and generators only need to fit the requirements of a single company and domain.

The most significant increases in software development productivity over the years have involved a fundamental rise in abstraction. The first major rise came with the move from machine code to assembly language. Years later there was a progression to third generation languages (3GLs) such as Fortran and C. Decades have now passed, but the industry as a whole has not yet embraced a new, abstraction raising approach to software development. The introduction of Java, UML and the plethora of new languages that have emerged on the scene, have only served to improve productivity by 10 to 20% at best, far less than triple digit percentage increase we experienced with the first 3GLs.

MDD has long been on the industry watch list as the next productivity boosting technology, but many adopters have achieved results far below their expectations (for example, there is surprisingly little, if any, productivity increase reported from using UML or Sys ML). These early failures have understandably made organisations wary of adopting other model-based techniques, thus impeding their widespread use. The reason these earlier MDD approaches failed to deliver is because they failed to raise the level of
A US Air Force study showed DSM as three times faster than code components and halved the errors in the resulting code.

abstraction and therefore did not reduce domain complexity. These unsuccessful approaches used the same programming concepts found in the programming languages they were intended to support. Most notably they have not been able to automate development by generating production code that can be executed; they tend to climax with the production of skeleton code.

DSM
DSM accepts that each domain – and each company – is unique and therefore does not subscribe to this one size fits all attitude. DSM focuses on a single problem domain and uses modelling concepts that directly mirror real world concepts, allowing for the capture of all the necessary information needed to develop software applications or systems fully and completely. This domain-specificity is at the heart of DSM and is the source of its dramatic productivity improving abilities. In DSM, a company’s expert developer – or a small group of expert developers – defines the DSL containing the domain concepts, rules and associated code generators. Expert developers are well familiar with the concepts of their domain, and are a preferable choice for defining these concepts than a tool vendor. It should be the tool vendor’s job to provide the proper tools for creating and using the DSL, not the language itself.

Language creation
Creating a DSM language begins with defining the domain modelling concepts. It is a good idea to begin with a thoroughly understood portion of the domain, and then to extend the language to cover additional components. Domain concepts can come from various sources including product architecture, output requirements and physical structure. The process of defining the modelling concepts is highly iterative and may need to be repeated several times. Next, rules regarding how concepts are identified and connected to one another are defined. Rules can be defined for within a model as well as between models. These rules define what types of connections exist and how many are allowed. Once the concepts and rules are defined, the expert would move on to creating the symbols associated with them. DSM tools typically provide a symbol-editing tool for this purpose and many support the importation of external graphics (such as SVG). Whenever possible, it is useful to implement symbols that closely approximate the real-word concept the object represents. This results in a very expressive design language that can specify applications in the problem domain accurately, elegantly and efficiently.

The models can be used for a variety of purposes, and the final step in the process is the creation of the generators. DSM environments usually include tools specifically designed for creating model-based code generators. Alternatively, various third party tools for creating generators are also available. Vendor-propriety or fixed code generators are not advisable for DSM, particular when as both the problem and solution domains are certain to evolve over time. In addition to the limitless code generation possibilities, reports can also be readily generated from these models for various functions: to check the consistency of the models, produce build scripts, create metrics, documentation, configuration information and so on.

Productivity increase
As with most development methodologies and techniques, DSM’s productivity increase is highly visible and measurable, including:

- **Productivity improvement**
  - The models can be used for a variety of purposes, and the final step in the process is the creation of the generators. DSM environments usually include tools specifically designed for creating model-based code generators. Alternatively, various third party tools for creating generators are also available. Vendor-propriety or fixed code generators are not advisable for DSM, particular when as both the problem and solution domains are certain to evolve over time. In addition to the limitless code generation possibilities, reports can also be readily generated from these models for various functions: to check the consistency of the models, produce build scripts, create metrics, documentation, configuration information and so on.

- **Language creation**
  - Creating a DSM language begins with defining the domain modelling concepts. It is a good idea to begin with a thoroughly understood portion of the domain, and then to extend the language to cover additional components. Domain concepts can come from various sources including product architecture, output requirements and physical structure. The process of defining the modelling concepts is highly iterative and may need to be repeated several times. Next, rules regarding how concepts are identified and connected to one another are defined. Rules can be defined for within a model as well as between models. These rules define what types of connections exist and how many are allowed. Once the concepts and rules are defined, the expert would move on to creating the symbols associated with them. DSM tools typically provide a symbol-editing tool for this purpose and many support the importation of external graphics (such as SVG). Whenever possible, it is useful to implement symbols that closely approximate the real-word concept the object represents. This results in a very expressive design language that can specify applications in the problem domain accurately, elegantly and efficiently.

- **Productivity increase**
  - As with most development methodologies and techniques, DSM’s productivity increase is highly visible and measurable, including:
the ultimate aim of DSM is to raise productivity. Whereas the vast majority of approaches only offer minimal productivity gains – if any at all – DSM has consistently demonstrated dramatic improvements in real world activities. However, due to the narrow nature of DSM languages, these significant accomplishments are seldom related outside the organisation. Many companies even go so far as to conceal actively their DSM activities to protect what they see as a competitive advantage for their organisation.

Despite the lack of a high volume of publicly available success stories, there exists some very compelling business-use information on the subject. Major embedded industrial heavyweights such as Nokia, Matsushita (Panasonic), Lucent and Eads have publicly stated that DSM significantly improved their software development productivity. Lucent (now Alcatel-Lucent) has stated that its use of DSLs has improved productivity by three to ten times, depending on the particular product being developed. Mobile phone giant Nokia claims that by using a DSM tool it was able to increase mobile phone application development by a factor of ten. This claim certainly shines some light on Nokia’s remarkable ability to introduce a large number of new device models year after year.

In addition to being able to raise productivity, DSM has been shown to improve marked- ly software code quality. Because the code generators are specified by the domain experts, the code they produce is identical to the high-quality code which that expert is capable of writing.

Automatic code generation also removes the possibility for accidental errors resulting from mistakes in typing or referencing that can easily occur during manual entry.

A study from the US Air Force comparing DSM with component-based development showed that DSM was three times faster than code components and led to 50% fewer errors in the resulting code. The study further went on to report that DSM gave “superior flexibility in handling a greater range of specifications” than did component-based approaches.

**Conclusion**

Embedded applications and systems continue to gain in size and complexity, resulting in increased pressure on already taxed software development teams. DSM seems poised to improve fundamentally development practices by raising the level of abstraction developers work with. Industry experience has shown that DSM radically improves developer productivity and code quality, resulting in fewer defects and faster time-to-market – items the industry is certainly on the lookout for.

James Hammond is sales and marketing manager for Metacase

---

**Fig. 1: Number of new product features implemented in a given time (productivity proportional to Assembler)**