INTRODUCTION TO THE 3rd WORKSHOP ON DOMAIN-SPECIFIC MODELING

Workshop web site: http://www.cis.uab.edu/info/OOPSLA-DSM03/

Juha-Pekka Tolvanen  
MetaCase Consulting  
Ylistonmentie 31  
FIN-40500 Jyvaskyla, Finland  
jpt@metacase.com

Jeff Gray  
University of Alabama at Birmingham  
Computer & Information Sciences  
115A Campbell Hall, 1300 University Blvd.  
Birmingham, AL 35294-1170, USA  
gray@cis.uab.edu

Matti Rossi  
Helsinki School of Economics  
FIN-00100 Helsinki, Finland  
mrossi@hkkk.fi

ABSTRACT

Current modeling languages are based on the concepts taken from programming languages, leading to working on problems of the solution domain instead of the problem domain. Domain-specific modeling allows faster development of applications, based on models of the products rather than on models of the code. A domain-specific modeling language applies concepts and rules that are natural for the experts of the domain. Together with generators and components it can automate a large portion of software production. This can be especially useful for companies that use product families or product platforms. In these proceedings we report on recent advancements in this area.

1 INTRODUCTION

Domain-driven development (3D) has recently popularized the importance of model-based research. A broad range of new research topics in this space have emerged and are being explored in numerous contexts. At the past two OOPSLA DSVL workshops, an international group of researchers assembled to discuss topics related to modeling and domain-specific visual languages. The prior workshops had a wide selection of topics (including generative/transformation techniques from models to code), but this version of the OOPSLA 2003 DSM workshop will focus solely on issues at the modeling level. There are two reasons for this: first, we believe that there is now a sufficient body of work on the modeling domain, which is evident in these proceedings, and second, there are separate workshops on generators and model driven architectures.

A contributing factor to the rising interest in domain specific modeling comes from the realization of productivity gains that have been attributed to a shift in focus toward software represented at higher levels of abstraction. In the past, abstraction was improved when programming languages evolved towards higher levels of specification. Today, domain-specific modeling provides a trajectory for continuing to raise the description of software to more abstract levels. Much investigation is still needed in order to advance the acceptance and viability of model-driven techniques.

Domain-specific modeling raises the level of abstraction, while at the same time narrowing down the design space, often to a single range of products for a single company. When applying DSM, and domain specific languages, the models are made up of elements
representing things that are part of the domain world, not the code world [5]. The language follows the domain abstractions and semantics, allowing developers to perceive themselves as working directly with domain concepts. The models are simultaneously the design, implementation and documentation of the system, which can, and should, be generated directly from them [1].

This is unlike current visual modeling languages that are based on the code world using the semantically well-defined concepts of programming languages (like UML, SA/SD). Here, developers have to leap straight from requirements into implementation concepts, and map back and forth between domain concepts, UML concepts, and program code. This requires a lot of time and resources and easily leads to errors.

The final products are automatically generated from the high level models with domain-specific code generators [4, 5]. There is no longer any need to make error-prone mappings from domain concepts to design concepts and on to programming language concepts. Industrial experiences of this approach show major improvements in productivity, time-to-market responsiveness and training time [2, 6].

2 DOMAIN SPECIFIC MODELING PREREQUISITES

Three things are necessary to achieve full automatic code generation from domain modeling: firstly a modeling tool supporting a domain-specific modeling language, secondly a code generator, and lastly a domain-specific component library. The top level is made once by the organization for a given domain. This forms the start-up cost of the DSM approach.

Normally one or two experts will make the DSM metamodel and code generation, normally with a metaCASE tool [4, 5]. The metamodel is the implementation of the domain-specific modeling language, and captures the essential concepts of the domain. In a sufficiently well known domain there should be concrete implementation components available and thus large portion of the systems can be generated from high level models.

Once the modeling language has been specified by the method experts the models can be drawn by normal developers, i.e. domain experts, which are not necessarily implementation experts. Development time can often be further reduced by reusing chunks of models which are common to several products. The code generation and component instantiation require no effort by the developer. Similarly documentation is handled by the model generators. In this scenario work can be divided by the domain specializations of the modelers (for example usability, processes, functions) instead of programming capabilities.

The DSM language captures the semantics of the domain and the production rules of the instantiation environment. The code generator transforms the concept structures into physical implementations in code. In some cases the code will be fully self-contained; more often significant parts of the code will be calls to components. Since the code is generated, syntax and logic errors do not occur, given that the semantics and modeling rules of the DSM are sufficiently well captured in the metamodel of the language.
3 ABOUT THE ARTICLES IN THESE PROCEEDINGS

The papers in this compilation present ten different views to DSM research and practice. The papers are divided into three sections, each comprising one workgroup in the actual workshop. In the first section we have three papers that present different cases of practical implementations of DSM languages. This section begins Grunske’s a visual language for embedded systems that uses hypergraphs. The second Chapter presents Amaral et al’s approach to a domain specific query language for the domain of high energy physics. In the third article Deng et al proposes a model driven approach to inventory tracking.

The second section considers model management in DSM’s. The first article by Celms Kalnins & Lace considers mapping of different diagrams into a common metamodel. In chapter five Wang and Liu present a formal model for integrating different models. Oglesby et al. present a dynamic view generation approach to model-based development in Chapter six. More transformational approaches to DSM are presented by the articles “Model Migration through Visual Modeling” by Sprinkle & Karsai and “Checking Program Synthesizer Input/Output” by Grant & al.

The third section finalizes this volume by two views to tools for DSM modeling. First paper presents UDM, a tool infrastructure for implementing DSM’s, and is written by Magyari & al. The last paper in this compilation, by Bichler, discusses tool support for generating implementations of MOF-based DSM’s. Together these papers give an excellent snapshot of the current state-of-the-art in DSM research.

REFERENCES

Contents

Introduction to the 3\textsuperscript{rd} workshop on Domain-Specific Modeling

**DSM practice:**

**Approaches for implementing DSMs**

1. A Visual Architecture Description Language for Embedded Systems with Hierarchical Typed Hypergraphs
   
   *Lars Grunske*
   
   1

2. A Domain Specific Visual Query Language for High Energy physicists
   
   *Vasco Amaral, Sven Helmer, Guido Moerkotte*
   
   9

3. Model Driven Development of Inventory Tracking System
   
   *Gan Deng, Tao Lu, Emre Turkay, Andrey Nechypurenko*
   
   15

**Model management:**

**Approaches for model integration and transformation**

4. Diagram Definition Facilities Based on Metamodel Mappings
   
   *Edgars Celms, Audris Kalnins, Lelde Lace*
   
   25

5. A Formal Model Integration
   
   *Jiayang Wang, Mengchi Liu*
   
   35

6. Cross-aspect Queries and Dynamic Views for Model-based Development
   
   *David Oglesby, Kirk Schloegel, Eric Engstrom*
   
   43

7. Model Migration through Visual Modeling
   
   *Jonathan Sprinkle, Gabor Karsai*
   
   51

8. Checking Program Synthesizer Input/Output
   
   *Emanuel Grant, Jon Whittle, Rajani Chennamaneni*
   
   59

**Tools:**

**Tools for implementing and using DSMs**

9. UDM: An Infrastructure for Implementing Domain-Specific Modeling Languages
   
   *Endre Magyari, Arpad Bakay, Andras Lang, Tamas Paka, Attila Vizhanyo, Aditya Agarwal, Gabor Karsai*
   
   69

10. Tool support for generating implementations of MOF-based modeling languages
    
    *Lutz Bichler*
    
    77