Modeling Turnpike: a Model-Driven Framework for Domain-Specific Software Development

Hiroshi Wada and Junichi Suzuki
Department of Computer Science
University of Massachusetts, Boston
Boston, MA 02125-3393
hiroshi_wada@otij.org and jxs@cs.umb.edu

Katsuya Oba
OGIS International, Inc.
444 High Street, Suite 200
Palo Alto, CA 94301
oba@ogis-internatinoal.com

ABSTRACT
This paper describes a new model-driven development framework, called Modeling Turnpike (or mTurnpike). It allows developers to model and program domain-specific concepts, and to gradually transform them to the final (compilable) source code. By leveraging UML metamodeling and attribute-oriented programming, mTurnpike provides an abstraction to represent domain-specific concepts at the modeling and programming layers simultaneously. This paper overviews the design, implementation and performance implications of mTurnpike.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and techniques – *Computer-aided Software Engineering*

General Terms
Design, Languages

Keywords
Model Driven Development, Domain Specific Language, UML, Attribute-Oriented Programming.

1. INTRODUCTION
Modeling technologies have matured to the point where they can offer significant leverage in all aspects of software development. Given modern modeling technologies, the focus of software development has been shifting from implementation technologies toward the domain-specific concepts (ideas and mechanisms specific to a particular business or technology domain) [1]. One of the goals of modeling technologies is to map modeling concepts directly to domain-specific concepts [1].

Domain Specific Language (DSL) is a promising solution to directly capture, represent and implement domain-specific concepts [1, 2]. DSLs are the languages targeted to particular problem domains, rather than general-purpose languages that are aimed at any software problems. Several experience reports have demonstrated that DSLs can improve the productivity in implementing domain-specific concepts (e.g. [3]).

This paper proposes a new model-driven development framework, called Modeling Turnpike (or mTurnpike) [4]. It allows developers to model and program domain-specific concepts in DSLs and to transform them to the final (compilable) source code in a seamless and piecemeal manner. Leveraging UML metamodeling and attribute-oriented programming, mTurnpike provides an abstraction to represent domain-specific concepts at the modeling and programming layers simultaneously.

OOPSLA'05, October 16–20, 2005, San Diego, California, USA.
ACM 1-59593-193-7/05/0010.

2. DESIGN AND IMPLEMENTATION
mTurnpike consists of the frontend and backend systems (Fig. 1). The frontend system is implemented as Domain Specific Code (DSC) Generator, and the backend system is implemented as DSL Transformer. Both systems are implemented with Java.

The frontend system transforms domain-specific concepts from the modeling layer to programming layer, and vise versa (Fig. 1), by providing a seamless mapping between Domain Specific Models (DSMs) and DSCs. In mTurnpike, a DSL is defined as a metamodel that extends the UML 2.0 standard (superstructure) metamodel with UML's extension mechanism. The UML extension mechanism provides a set of model elements such as stereotype and tagged-value in order to add application-specific or domain-specific modeling semantics to the UML 2.0 standard metamodel. In mTurnpike, each DSL defines a set of stereotypes and tagged-values to express domain-specific concepts.

Given a DSL, a DSM is represented as a set of UML 2.0 diagrams (class and composite structure diagrams). Each DSC consists of Java interfaces and classes decorated with the J2SE 5.0 annotations. The frontend system of mTurnpike maps the stereotypes and tagged-values in a DSM to the annotations in a DSC, and vise versa, in order to transform domain-specific concepts between the modeling and programming layers (Fig. 1).

The backend system of mTurnpike transforms a DSM and DSC into a more detailed model and program by applying a given transformation rule (Fig. 1). mTurnpike allows developers to define arbitrary transformation rules, each of which specifies how to specialize a DSM and DSC to particular implementation and deployment technologies. mTurnpike combines the specialized DSM and DSC to generate compilable source code (Fig. 1).

In mTurnpike, the frontend and backend systems are separated by design. mTurnpike clearly separates the task to model and program domain-specific models (as DSMs and DSCs) from the task to transform them into the final compilable code. This design strategy improves separation of concerns between modelers/programmers and platform engineers. Modelers and programmers do not have to know how domain-specific concepts are implemented and deployed in detail. Platform engineers do

not have to know the details of domain-specific concepts. As a result, mTurnpike can reduce the complexity in application development, and increase the productivity of developers in modeling and programming domain-specific concepts.

This design strategy also allows DSMs/DSCs and transformation rules to evolve independently. Since DSMs and DSCs do not depend on transformation rules, mTurnpike can specialize a single set of DSM and DSC to different implementation and deployment technologies by using different transformation rules. When changing a running application, modelers/programmers make the changes in the application's DSM and DSC and leave transformation rules alone. When retargeting an application to a different implementation and/or deployment technology, e.g. Java RMI to Java Messaging Service (JMS), platform engineers define (or select) a transformation rule for the new target technology and regenerate the final compilable source code. As such, mTurnpike can make domain-specific concepts (i.e. DSMs and DSCs) more reusable and extend their longevity, thereby improving productivity and maintainability in application development.

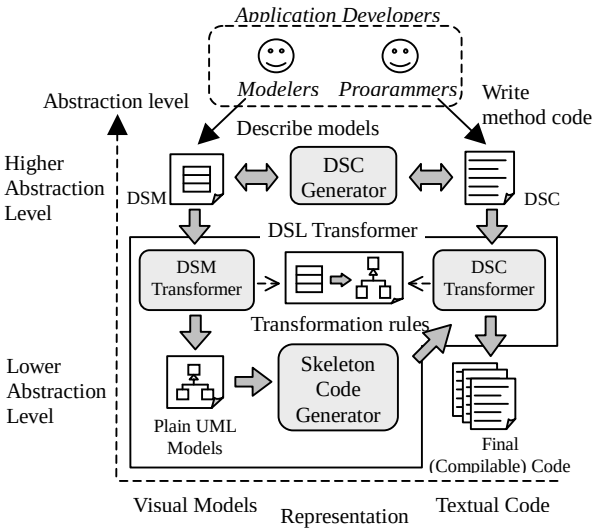

Fig. 1 mTurnpike Architecture and its key components

3. APPLICATIONS

In order to demonstrate how to exploit mTurnpike in application development, it has been used to develop distributed systems with a DSL for Service-Oriented Architecture (SOA) [4]. SOA is a distributed systems architecture that models a distributed system as a set of services and connections between them in a platform independent manner. The SOA DSL abstracts distributed systems using two major domain-specific concepts, *service interface* and *connections between services*, and hides the details of implementation and deployment technologies (e.g. programming languages and remoting systems). mTurnpike currently specializes DSMs to Java applications running with Java RMI and JMS.

In addition to the proposed SOA DSL (a DSL for a horizontal domain), mTurnpike will support DSLs for vertical domains as well. mTurnpike will support at least three[1] horizontal/vertical DSLs, and it will be tested to generate compilable code through combining DSMs and DSCs written in those multiple DSLs.

[1] [5] suggests investigating at least three applications on a framework in order to examine generality and reusability of the framework.

4. EMPIRICAL EVALUATION

Empirical measurement results of the mTurnpike frontend system shows that its transformation overhead is small enough (below 5 seconds) and consumes no more than 15MB memory in small-scale to mid-scale application development. mTurnpike does not interrupt developers' modeling and programming work severely, and it is not necessary for them to upgrade their development environments (e.g. memory modules in their PCs).

5. CONTRIBUTIONS

This section summarizes the contributions of this work.

Higher abstraction for programming domain-specific concepts. mTurnpike offers a new approach to represent domain-specific concepts at the programming layer, through the notion of attribute-oriented programming. This approach provides a higher abstraction for developers to program domain-specific concepts, thereby improving their programming productivity. Attribute-oriented programming makes programs simpler and more readable than traditional programming paradigms.

Seamless mapping of domain-specific concepts between the modeling and programming layers. mTurnpike maps domain-specific concepts between the modeling and programming layers in a seamless manner. This mapping allows modelers and programmers to handle the same set of domain-specific concepts in different representations (i.e. UML models and annotated code), yet at the same level of abstraction. Thus, modelers do not have to involve programming details, and programmers do not have to possess detailed domain knowledge and UML modeling expertise. This separation of concerns can reduce the complexity in application development, and increase the productivity in modeling and programming domain-specific concepts.

Modeling layer support for attribute-oriented programs. Using the bi-directional mapping between UML models and annotated code, mTurnpike visualizes annotated code in UML. This work is the first attempt to bridge a gap between UML modeling and attribute-oriented programming.

6. REFERENCES

[1] G. Booch, A Brown, S Iyengar, J. Rumbaugh and B. Selic, "An MDA Manifesto," In *The MDA Journal: Model Driven Architecture Straight from the Masters*, Chap. 11, Meghan Kiffer, Dec. 2004.

[2] S. Cook, "Domain-Specific Modeling and Model-driven Architecture," In *The MDA Journal: Model Driven Architecture Straight from the Masters*, Chap. 3, Meghan Kiffer, December 2004.

[3] S. Kelly and J. Tolvanen, "Visual Domain-specific Modeling: Benefits and Experiences of using metaCASE Tools," In *Proc. of Int'l workshop on Model Engineering, ECOOP*, 2000.

[4] H. Wada and J. Suzuki, "Modeling Turnpike Frontend System: a Model-Driven Development Framework Leveraging UML Metamodeling and Attribute-Oriented Programming," In *Proc. of the 8th ACM/IEEE MoDELS*, October 2005. to appear.

[5] D. Roberts and R. Johnson, "Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks," In *Pattern Languages of Program Design 3*, Chap. 26, Addison Wesley, 1997.