Recurring Return on Modeling Investment: A Conceptual Modeling Language and Extensible Compiler

Quenio Cesar Machado dos Santos[1] and Raul Sidnei Wazlawick[2]

[1] Computer Sciences,
UFSC - Universidade Federal de Santa Catarina, Brazil,
queniodossantos@gmail.com
[2] Associate Professor of Computer Sciences Department,
UFSC - Universidade Federal de Santa Catarina, Brazil,
raul@inf.ufsc.br

Abstract. Proposes a textual programming language that enables conceptual modeling (similarly to UML classes/associations and OCL constraints) and a compiler that allows code generation (via extensible textual templates) to any target language or technology. Together, the language and the compiler make it feasible to specify (in a single high-level language) the information of ever-changing, increasingly distributed software systems. From this single source, the automated code generation keeps the implementations (across the different platforms and technologies) consistent with the specification. Also, as the technology landscape evolves, these textual models allow the recurring use of the investment made on their specification. Unlike other approaches, such as MDA and MPS, the built-in tooling support, along with the textual nature of this programming language and its extensible templates, facilitates the integration to the workflow of software developers, which is expected to promote its adoption. A cost-benefit analysis model is also provided, which should assist the stakeholders in measuring the return of their investment in modeling.

Keywords: conceptual modeling, omg, mda, uml, ocl, mof, mps, mde, mdsd, er, entity-relationship model, programming language, compiler, code generation, model-driven software development, model-driven engineering, modeling investment, classes, associations, constraints, specification, software tools, metaprogramming, generative programming

1 Introduction

In order to address the challenges of the ever-changing, increasingly distributed technologies used on software systems, the Model-Driven Architecture (MDA [1]) initiative by the Object Management Group (OMG) has been promoting model-driven software development. In particular, MDA has guided the use of high-level models (created with OMG standards, such as UML [3], OCL [2]

and MOF [4]) to derive software artifacts and implementations via automated transformations. As one of its value propositions, the MDA guide [1] advocates:

> "Automation reduces the time and cost of realizing a design, reduces the time and cost for changes and maintenance and produces results that ensure consistency across all of the derived artifacts. For example, manually producing all of the web service artifacts required to implement a set of processes and services for an organization is difficult and error-prone. Producing execution artifacts from a model is more reliable and faster."

MDA provides guidance and standards in order to realize this vision, but it leaves to software vendors the task of providing the tools that automate the process of generating the implementations from the models.

The key role played by tools has been demonstrated by Voelter [5] in his *Generic Tools, Specific Languages* approach for model-driven software development. Voelter [5] has used domain-specific languages (DSL's) with the Metaprogramming System (MPS) in order to generate the software artifacts. Unlike MDA, which is based on UML/MOF models, MPS allows the specification of models using domain-specific editors. MPS itself is a generic tool, but it enables the definition of the abstract syntax, the editors and the code generators for DSL's.

The conceptual modeling language and extensible compiler presented here are an alternative approach to MPS. While the latter is a fully integrated development environment based on domain-specific languages and their projectional editors [3], the former (hereby called CML) is a compiler that has:

- as *input*, source files defined using its own conceptual language, which provides an abstract syntax similar to (but less comprehensive than) a combination of UML [3] and OCL [2];
- and, as *output*, any target languages based on extensible templates, which may be provided by the compiler's base libraries, by third-party libraries, or even by the developers themselves.

The next two sections present the conceptual model language (section 2) and the compiler with its extensible templates (section 3). The section 4 discusses how CML can be integrated into the development workflow. The Section 5 provides a cost-benefit analysis model that can be used by organizations in order to determine whether integrating CML into its development workflow would provide recurring return on their modeling investment. We conclude in section 6, reiterating the objectives achieved by CML and exploring future possibilities to expand CML's reach.

[3] As explained by Voelter [5], projectional editors in MPS do not rely on parsers. Instead, the abstract syntax tree (AST) of the DSL is modified directly. The projection engine renders the visual representation of the AST based on the DSL editor definition.

2 The Language

This section presents an overview of the conceptual modeling language. The concrete syntax will be presented using examples. The appendix A provides a formal description of the concrete syntax, along with its mapping to the abstract syntax. The abstract syntax will be presented in subsections; each one focusing on a key concept of the language's metamodel.

Before exploring the abstract syntax, a concrete example is displayed and commented below:

```
concept BookStore
{
    books: Book+;
    customers: Customer*;
    orders: Order*;
    /goldCustomers = customers | select totalSales > 1000;
    /orderedBooks = orders.items.book;
}

concept Book
{
    title: String;
    price: Decimal;
    quantity: Integer = 0;
}

concept Customer
{
    orders: Order*;
    /totalSales = orders | collect result += total;
}

concept Order
{
    items: Item+;
    customer: Customer;
    /total: Number = items | collect result += item.amount;
}

association CustomerOrder
{
    Order.customer: Customer;
    Customer.orders: Order*;
}
```

Fig. 1. This example is adapted to CML from the fictional Livir bookstore, which is presented as a case study in Wazlawick [6].

The model specified by the example above will be parsed and instantiated by the CML compiler into the following abstract syntax tree:

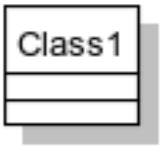

Fig. 2. This is the caption of the figure displaying a white eagle and a white horse on a snow field

3 The Extensible Compiler

4 The Development Workflow

5 The Cost-Benefit Analysis Model

6 Conclusion

A The Concrete/Abstract Syntax

This appendix provides a formal description of the concrete syntax, along with its mapping to the abstract syntax.

References

1. OMG: Model Driven Architecture (MDA) MDA Guide rev. 2.0 (2014). URL: http://www.omg.org/cgi-bin/doc?ormsc/14-06-01
2. OMG: Object Constraint Language (OCL), Version 2.4 (2014). URL: http://www.omg.org/spec/OCL/2.4
3. OMG: Unified Modeling Language (UML), Superstructure, Version 2.5 (2015). URL: http://www.omg.org/spec/UML/2.5
4. OMG: Meta Object Facility (MOF) Core Specification, Version 2.5.1 (2016). URL: http://www.omg.org/spec/MOF/2.5.1
5. Voelter, M.: Generic Tools, Specific Languages. Ph.D. thesis, Delft University of Technology (2014). URL: http://repository.tudelft.nl/view/ir/uuid%3A53c8e1e0-7a4c-43ed-9426-934c0a5a6522
6. Wazlawick, R.S.: Object-Oriented Analysis and Design for Information Systems: Modeling with UML, OCL, and IFML. Morgan Kaufmann (2014). URL: http://www.sciencedirect.com/science/book/9780124186736