Reusing Conceptual Models

A Conceptual Modeling Language and Extensible Compiler

Quenio Cesar Machado dos Santos & Raul Sidnei Wazlawick
Computer Sciences, Universidade Federal de Santa Catarina, Brazil
`quenio.santos@grad.ufsc.br & raul@inf.ufsc.br`

Abstract
Presents a textual programming language for conceptual modeling (based on UML classes/associations and OCL constraints) and its compiler that can generate code in any target language or technology via extensible textual templates, both currently under initial stage of development. The language and compiler should allow the specification of information managed by ever-changing, increasingly distributed software systems. From a single source, automated code generation should keep implementations consistent with the specification across the different platforms and technologies. As the technology landscape evolves, the target templates may be extended to embrace new technologies. Unlike other approaches, such as MDA and MPS, the textual nature of this modeling language and its extensible templates is expected to facilitate the integration of model-driven software development into the workflow of software developers.

Introduction

This conceptual modeling language and its extensible compiler are an alternative to the Metaprogramming System (MPS), as presented by Voelter [10]. While MPS is an integrated development environment based on domain-specific languages (DSLs), this proposal (henceforth called CML) is a compiler that:

- accepts as *input* source files coded in its own conceptual modeling language, which has an abstract syntax similar to (but less comprehensive than) a combination of UML [7] and OCL [6].
- generates as *output* target files in any programming language, which is accomplished by text-based extensible templates, provided by the base library bundled with the compiler, by third-parties, or still by the developers themselves.

Figure 1 shows an overview of the CML compiler's arquitecture.

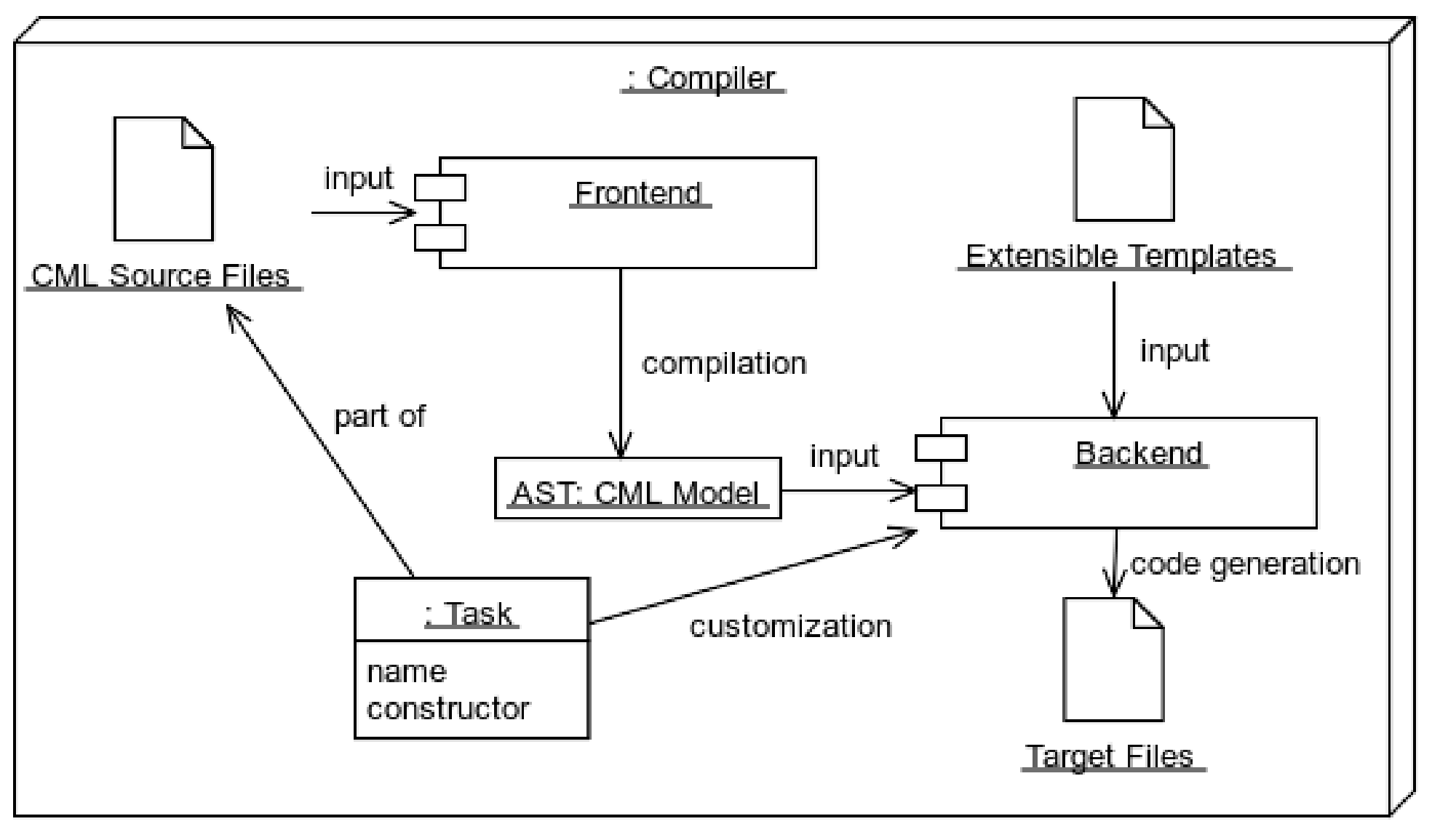

Figure 1: Architectural Overview of The CML Compiler

Main Objectives

1. Following the principle established by the *Conceptual-Model Programming (CMP) manifesto* [3], CML intends to enable modeling-as-programming, allowing software developers to incorporate conceptual modeling into the same workflow they are used to doing software development.
2. In order to validate the use of the CML language and compiler for modeling and implementing its own metamodel, this initial version of CML provides generalization/specialization, associations with cardinality zero-or-one (optionals) and zero-or-more (sequences), and the ability to define derived atributes/associations using expressions similar to OCL [6].
3. CML allows developers to customize, annotate and extend the generated code via extensible templates in order to be able to leverage conceptual models across different programming languages and technologies
4. While other initiatives, such as the *Model-Driven Architecture* [5], require the use of languages and tools from different vendors, CML combines into a single, open-source language/-compiler package, the ability to model and to customize code generation, allowing models to last a longer lifespan than it is normally viable with specific technologies.

The Language

Generalization/Specialization

Figure 2 presents some examples of generalization/specialization relationships declared in CML.

```
@concept Shape
{
    color: String;
    area: Double;
}

@concept Rectangle: Shape
{
    width: Double;
    height: Double;

    /area = width * height;
}

@concept Rhombus: Shape
{
    p: Double;
    q: Double;

    /area = (p * q) / 2.0d;
}

@concept Square: Rectangle, Rhombus
{
    side_length: Double;

    /width = side_length;
    /height = side_length;

    /p = side_length * 1.41421356237d;
    /q = p;

    /area = side_length ^ 2.0d;
}
```

Figure 2: Generalization/Specialization in CML

Associations

Figure 3 presents some examples of *associations* declared in CML.

```
@concept Vehicle
{
    plate: String;
    driver: Employee?;
    owner: Organization;
}

@concept Employee
{
    name: String;
    employer: Organization;
}

@concept Organization
{
    name: String;
    employees: Employee*;
    fleet: Vehicle*;
    /drivers = fleet.driver;
}

@association Employment
{
    Employee.employer;
    Organization.employees;
}

@association VehicleOwnership
{
    Vehicle.owner: Organization;
    Organization.fleet: Vehicle*;
}
```

Figure 3: Associations in CML

Extensible Templates

Figure 3 presents some examples of *extensible templates* declared in CML.

```
model_files(task, model) ::= <<
pom_file|pom.xml
>>

concept_files(task, concept) ::= <<
concept_file|<task.packagePath>/<concept.name; format="pascal-case">.java
>>

...

import "/design/poj.stg"

concept_file(task, concept) ::= <<
package <task.packageName>;

import java.util.*;

public <class(concept)>
>>
```

Figure 4: Extensible Templates in CML

Related Work

When compared to CML, the text-based languages are the most relevant. MPS [10] is a development environment for DSLs. Strictly speaking, its DSLs are not textual, since their AST is directly edited on projectional editors. However, the editors allow textual representations. Unlike MPS, the DSLs created with the M language [2] are truly textual. It was part of the discontinued Oslo project from Microsoft, which incorporated into Visual Studio similar capabilities to what is available on MPS. Xtext/Xtend [1] allows the definition of textual DSLs to generate code from conceptual models edited on Eclipse. It is similar to the Oslo project from Microsoft, and based on EMF [8]. MM-DSL [9], on the other hand, allows the definition of metamodels (abstract syntax; not the actual DSLs), which serve as input to generate domain-specific modeling tools. ThingML [4] is also a language and code generation framework for the development of software in embedded devices.

References

[1] Lorenzo Bettini. *Implementing domain-specific languages with Xtext and Xtend*. Packt Publishing Ltd, 2016.

[2] Hugo Brunelière, Jordi Cabot, Cauê Clasen, Frédéric Jouault, and Jean Bézivin. Towards Model Driven Tool Interoperability. In Thomas Kühne, editor, *ECMFA 6th. Proceedings*, pages 32–47. Springer, 2010.

[3] David W. Embley, Stephen W. Liddle, and Oscar Pastor. Conceptual-Model Programming: A Manifesto. In David W. Embley and Bernhard Thalheim, editors, *Handbook of Conceptual Modeling: Theory, Practice, and Research Challenges*, pages 3–16. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[4] Nicolas Harrand, Franck Fleurey, Brice Morin, and Knut Eilif Husa. Thingml: A language and code generation framework for heterogeneous targets. In *Proceedings of the ACM/IEEE 19th MODELS*, pages 125–135, 2016.

[5] OMG. Model Driven Architecture (MDA) MDA Guide rev. 2.0, 2014.

[6] OMG. Object Constraint Language (OCL), Version 2.4, 2014.

[7] OMG. Unified Modeling Language (UML), Superstructure, Version 2.5, 2015.

[8] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.

[9] Niksa Visic and Dimitris Karagiannis. Developing Conceptual Modeling Tools Using a DSL. In Robert Buchmann, editor, *KSEM: 7th International Conference. Proceedings*, pages 162–173. Springer, 2014.

[10] Markus Voelter. *Generic Tools, Specific Languages*. PhD thesis, Delft University of Technology, June 2014.