Introduction to a CVL to Clafer transformation project

Tom Wijsman

Model Driven Engineering 2014-2015
University of Antwerp

Abstract

Clafer and CVL are currently developing modeling languages that combine modeling languages with variability. They tackle historical problems when introducing variability in existing models. Clafer is textual and concise; CVL is visual and an expansion. In order to make comparisons, interactions and migrations between these two modeling languages more possible; a project is introduced that will attempt to transform CVL to Clafer, after first studying Clafer and CVL and their tools. For completeness, note that the transformation in the opposite direction already exists in the Clafer Compiler.

Keywords: Choices, Clafer, Class, Constraint, CVL, Feature, Generation, Feature Model, Language, Reference, Representation, Textual, Transformation, Unification, Variability, Variation, Visual

1. Introduction

The context and history in which this project takes place is revealed, then the problem is highlighted and an overview of this paper is given.

1.1. Context

For the course Model Driven Engineering at the University of Antwerp a project is studied and implemented. This project incorporates most of the knowledge exchanged at the course lectures and practicums, which covers (but is not limited to) theory and practice about modeling languages, syntax, semantics and transformations.

Email address: `Tom.Wijsman@student.uantwerpen.be` (Tom Wijsman)

1.2. History

There have been multiple attempts in the past to combine a modeling language with variability, as Kacper et al. (2011) cites in their introduction. For example, let's assume an instance of a class diagram is attempted to be combined with a feature diagram. In such example, a composition of both the class diagram instance and a feature diagram is problematic as such composition does not have a binding between the class diagram and the feature diagram. Therefore there is a need for a modeling language that can efficiently bind variability to an existing modeling language.

Clafer and CVL are two such modeling languages that have been developed in this decennium. Clafer is a general-purpose lightweight modeling language, developed at the GSD Lab, University of Waterloo and MODELS group at IT University of Copenhagen. It follows a less-is-more idea, where features are a first-class citizen and therefore no different from a class. CVL is a language for variability modeling, it is developed by multiple parties (IBM, Thales, Fraunhofer FOKUS and TCS) and is a standardization attempt that has been submitted to the OMG. In contrast with Clafer, CVL does compose an existing modeling language with a variability specification tree and uses variation points to bind the existing modeling language elements with the variability specifications. Clafer and CVL are both solutions to that historical problem of needing an efficient binding.

Clafer will be further studied in section 2, CVL in section 3.

1.3. Problem

Each language has its own representation by design; Clafer is textual, whereas CVL is graphical. Someone interested in such languages will compare representations and perhaps prefer one representation over the other. In interactions between people there can be interactions where one person uses Clafer, whereas the other person uses CVL or vice versa. It is also possible that someone needs to switch between Clafer and CVL. Being able to transform one representation into the other and vice versa could help for these comparisons, interactions and migrations.

Searching for existing solutions yields that the Clafer compiler can produce a CVL representation; however, the transformation in the other direction from CVL to Clafer does not seem to exist. This project will attempt to implement this transformation from CVL to Clafer.

1.4. Overview

Clafer is studied and its tools are reviewed in section 2. CVL is studied and its tools are reviewed in section 3. The first design of the CVL to Clafer transformation is given in section 4. The implementation verification is explained in section 5. A conclusion is given in section 6. Future work of the project is planned in section 7. The papers read are listed in section 8.

2. Clafer

Clafer (**cla**ss, **fe**ature and **r**eference) is a class modeling language with first-class support for feature modeling.

2.1. Introduction

Clafer is designed with a concise natural textual notation that is a mixture of meta-models, feature models and models. It has features such as constraints and inheritance that allow the arrangement of these models into specializations and extensions. It is postulated to satisfy the following design goals:

- provide a concise notation for feature modeling;
- provide a concise notation for meta-modeling;
- allow mixing feature models and meta-models;
- use minimal number of concepts and have uniform semantics.

Its first-class support for feature modeling comes from the need for unificating the concepts; in other words, they do not want Clafer to be a hybrid language.

2.2. Details

A Clafer model is a set of concepts such as type definitions, features, and constraints. A type corresponds to both a class and a feature without distinction. Features correspond to attributes or role names of association and composition relationships. Constraints limit the variability of these types and features.

In part **a** of figure 1 an example of Clafer in concise notation is shown. In part **b** of this figure, the same Clafer example in completely desugared code

```
abstract options
  xor size
    small
    large

  cache?
    size -> int
      fixed?

  [ small && cache =>
    fixed ]
```

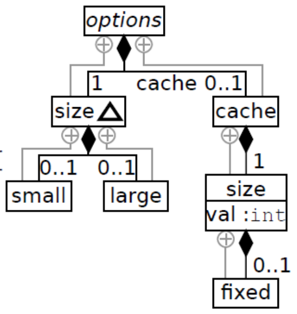
a) Concise notation

```
abstract <0-*> options {
  <1-1> size 1..1 {
    <0-*> small 0..1 {}
    <0-*> large 0..1 {}
  }

  <0-*> cache 0..1 {
    <0-*> size -> int 1..1 {
      <0-*> fixed 0..1 {}
    }
  }

  [ some this.size.small &&
    some this.cache =>
    some this.cache.size.fixed ]
}
```

b) Full notation



c) UML class model

Figure 1: Example of Clafer notations

(including resolved names in constraints) is shown. In part **c** of this figure, the UML class diagram for this Clafer example is shown.

Indentation is used to denote that concepts belong to the concept of one indentation level above. For example; in figure 1, the abstract options definition consists of a mandatory size and an optional cache and it has a constraint (between brackets). For the size options, the two indented options reveal that one can pick between small and large. If a cache is chosen; one needs to pick a size, as well as denote whether or not that cache size should be fixed. The constraint further limits the choices that can be made.

This example shows us that there is no difference between a class and a feature; all the types can be interpreted as both, therefore Clafer has no need for an explicit binding between classes and features.

For more details and features of Clafer and the source of this information and pictures, see Kacper et al. (2011).

At the time of writing, a new up-to-date version of the paper is being published, see Kacper et al. (2014).

2.3. Tools

Clafer has an integrated set of tools called Clafer Tools; since this set is extensive and can do basic requirements such as validation and instance generation, it suffices for this project. In figure 2 the architecture and capabilities are summed up. The most notable tools are:

1. **Clafer Compiler** provides a reference language implementation that can export models in Clafer to other formats (e.g., Alloy, DOT, HTML, JS, Python, XML) to allow for processing and reasoning with existing tools (Alloy Analyzer, Choco3, and Z3 SMT solver);
2. **Clafer Wiki** allows for embedding Clafer model fragments in wiki pages and provides model authoring support including code highlighting, parse and semantic error reporting, hyperlinking from identifier use to its definition, and graphical view rendering;
3. **Clafer IG** (Clafer Instance Generator) is an interactive tool that generates instances and counter examples in a Clafer model. If there are no contradicting constraints, the generator produces valid instance data. Otherwise, the generator produces an unsatisfiable core which included all contradicting constraints and generates a counter example by removing one constraint from the core.
4. **Clafer Configurator** is an interactive web-based configurator for attributed feature models with inheritance that provides a novel approach to feature configuration; whereby the configurer works with multiple correct configurations at the same time instead of working with a single configuration, making configuration steps and resolving configuration conflicts.
5. **Clafer Moo Visualizer** visualizes a set of non-dominated optimal variants (Pareto Front) and allows for exploration and trade-off analysis.

Most of these tools are based on a web front end and make use of Alloy, Choco3 and Z3 instance generators and multi-objective optimizers; which gives this set of tools an accessible, scalable and interactive user environment.

For more details about the Clafer Tools and the source of this information and pictures, see Micha et al. (2013).

3. CVL

CVL (Common Variability Language) is a domain-independent language for specifying and resolving variability. It facilitates the specification and resolution of variability over any instance of any language defined using a MOF-based meta-model.

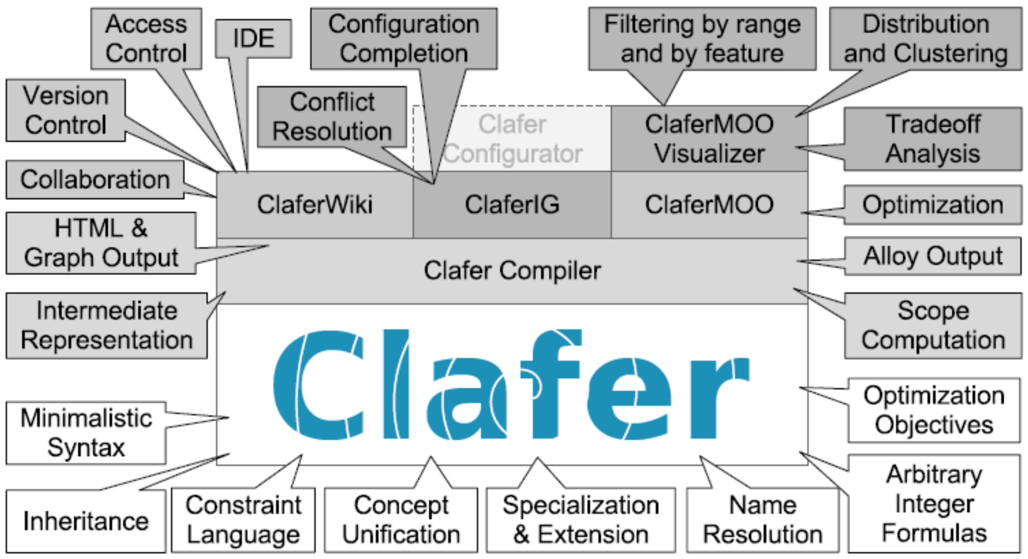


Figure 2: Architecture and Capabilities of Clafer and Tools

3.1. Introduction

Figure 3 gives an overview of how CVL works. A variability model is added to a base model to introduce variability. In order to execute CVL, resolution models that are solutions of the variability model need to be provided; execution then yields materialized models.

Variability models consist of a tree of variability specifications (which correspond to a feature diagram), this tree of variability specifications is connected with variation points to the objects in the base model. This can be seen in figure 4. The variability specifications consist of choices and variables, which the resolution model can fill in. The variation points decide what the choices and variables imply in the resolution of the base model; such variations are the existence of one or more object(s), the assignment of a value to a parameter slot and so on. These variation points allow CVL to transform the base model according to the variability specification; in other words, it allows us to approach the base model through CVL in a variable way.

For more details and features of CVL and the source of this information and pictures, see IBM et al. (2012).

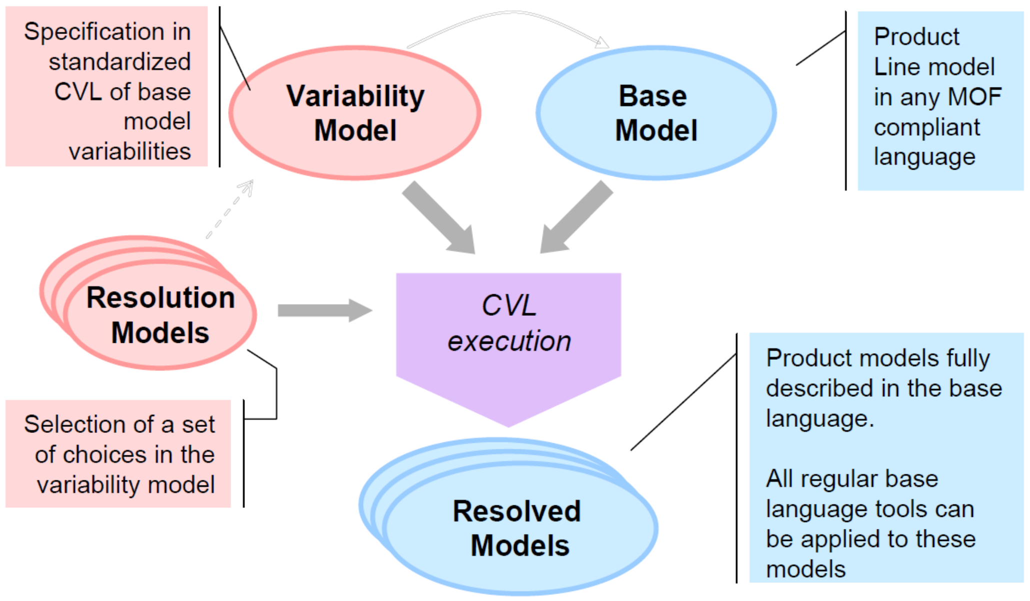

Specification in standardized CVL of base model variabilities
Variability Model
Base Model
Product Line model in any MOF compliant language
Resolution Models
CVL execution
Selection of a set of choices in the variability model
Resolved Models
Product models fully described in the base language.
All regular base language tools can be applied to these models

Figure 3: Common Variability Language as specified in the Request for Proposals

3.2. Tools

Given that CVL has been submitted to OMG last year; it is relatively new and subject to change. The amount of tools that exist for CVL is therefore limited. The ones found (for example, CVL Tool from SINTEF) are visual, the lack of a standardized textual format for CVL does not allow for direct processing. For the purpose of this project, a concrete visual syntax for CVL is created in the research framework AToMPM, which stands for ”A Tool for Multi-Paradigm Modeling”. Those whom are interested in implementing this project can look into exporting formats to the AToMPM format, or reimplementing our work for a future standardized textual format for CVL.

4. CVL to Clafer transformation

Transformation from CVL to Clafer can be done in multiple steps, which is preferable, such that existing generation and transformation languages are used. Therefore the framework metaDepth for multi-level meta-modeling will be used as an intermediate step. An overview of the steps to be performed:

1. identify a reasonable set of common features that exist in both CVL, Clafer and their constraint languages;

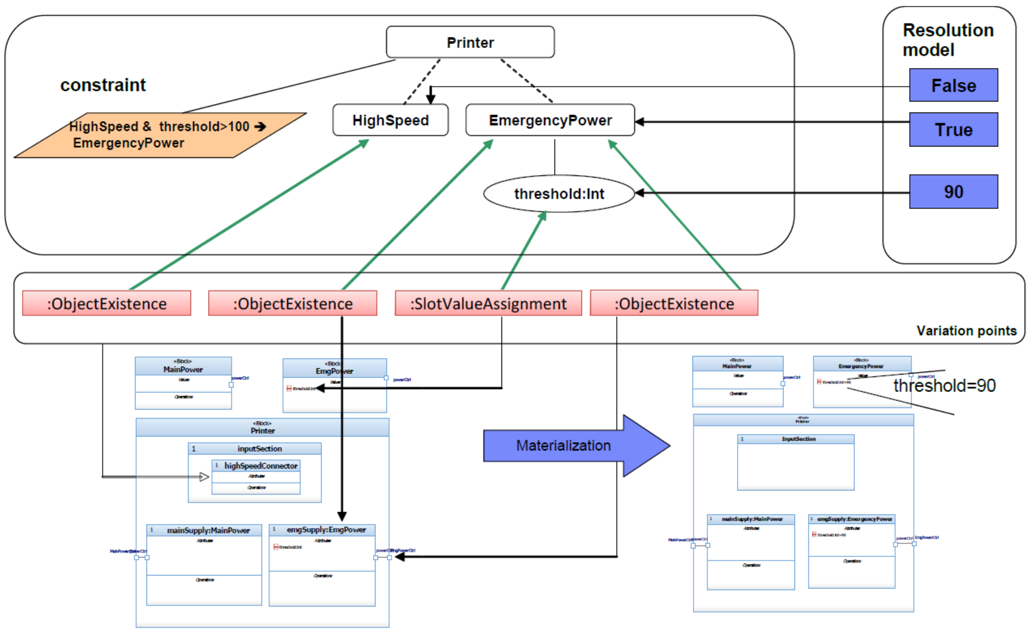


Figure 4: Variability model (top left), resolution model (top right), variation points (center), base model (bottom left) and materialization (bottom right)

2. create an abstract and concrete visual syntax of CVL in AToMPM;
3. generate a metaDepth model of CVL from AToMPM by using EGL (Epsilon Generation Language);
4. transform the metaDepth model of CVL to Clafer by using ETL (Epsilon Transformation Language);
5. generate concise syntax in Clafer from the metaDepth model of CVL by using EGL again;

5. Verification

As the Clafer Compiler uses a transformation in the other direction to visualize a Clafer model by producing a CVL model in a DOT file; the transformation can be verified by transforming the concise syntax of Clafer that our transformation produced back to CVL, then do a visual comparison of both. They should have the same structure to pass verification.

6. Conclusion

Both languages have the goal to combine a base model with variability but use an opposite approach to reach that goal; Clafer mixes everything together to create a minimal amount of concepts, CVL keeps the modeling language and variability separated and binds them together using additional logic in variation points. The core of Clafer and its tools makes one able to do similar things to the core of CVL, perhaps even the same. This makes it worth the attempt to transform from CVL to Clafer in this project. It is promising since the transformation in the other direction from Clafer to CVL already exists.

7. Future work

First the focus will be put on identifying a reasonable set of common features in CVL, Clafer and their constraint languages; which includes looking into how each feature in CVL can be transformed into a feature in Clafer. Then the various steps of the transformation (see section 4) will be carried out, after which a verification (see section 5) will be done. More features and uncommon features can be added later; if possible in the scope of this project an attempt might be done, which further progress will help decide.

8. Bibliography

IBM, FOKUS, F., Thales, Services, T. C., August 2012. Common Variability Language (CVL). See the CVL Revised Submission section online at `http://www.omgwiki.org/variability/doku.php`.

Kacper, B., Krzysztof, C., Andrzej, W., 2011. Feature and meta-models in Clafer: Mixed, specialized, and coupled. Lecture Notes in Computer Science 6563, 102–122.

Kacper, B., Zinovy, D., Micha, A., Krzysztof, C., Andrzej, W., December 2014. Clafer: unifying class and feature modeling. Software & Systems Modeling 14.

Micha, A., Kacper, B., Alexandr, M., Jimmy, L., Rafael, O., Krzysztof, C., 2013. Clafer Tools for product line engineering. In: Proceedings of the 17th International Software Product Line Conference co-located workshops. Software Product Line Conference, ACM, Tokyo, Japan, pp. 130–135.