# Clafer: a concept modeling language

Athanasios Koutoulas

19 Dec 2013

## abstract

The purpose of this paper is to provide a description of the Clafer modeling language and its properties in the context of my project for the course Model Driven Engineering. This paper refers to Clafer a textual language for concept modeling and is based on the analysis of its properties. After a brief introduction about Clafer, the ability of this language to combine Feature and Class Modeling is discussed. Moreover this paper contains the description of the Clafer to Alloy translator, the tool that gives precise semantics to Clafer. Finally I would like to mention that the biggest part of this paper focuses on the syntax of this language and the analysis of Clafer models by means of some examples.

## introduction

The word Clafer is an acronym coming from the words class feature and reference. This fact implies that Clafer language supports class modeling, feature modeling and reference modeling. Moreover Clafer is a lightweight modeling language which is designed to have a small footprint, it is an easy language to implement and it has minimalistic syntax and features. Clafer is a textual language which is used in concept modeling and specification of software product lines [2] and it has the ability to provide a uniform syntax and semantics to class and feature models. Its goal is to make lightweight modeling more accessible to a wider range of users and domains, including enterprise systems. Clafer has several applications and it can be used for Domain and structural modeling. Clafer models can encode feature, class, and meta-models which contain complex constraints. Moreover the language provides model verification and validation. It has the ability of testing the consistency of models; check if given examples correspond to correct instances of models and derive multiple examples from models. Another important application of Clafer is the Model completion. The tool contains the ClaferIG which helps to automatically configure models and specify attribute vales to derive fully-specified model instances. For the configuration of a model, the engineer can specify only some properties while the rest of them will be automatically completed by the reasoner. Finally Clafer offers Debugging. When a model is inconsistent, ClaferIG highlights the contradicting constraints and it can also show inconsistent models.

## Class, Feature Modeling and Mapping

In general, a Clafer model is a set of type definitions, features, and constraints. A type can be understood as a class or feature type. A type definition can contain one or more features. A feature definition creates a feature and, implicitly, a new concrete type, both located in the same name space. Features are slots that can contain one or more instances or references to instances. The parent-children hierarchy is indicated by simply indenting sub-features under parent feature. The Features have feature cardinalities, which constrain the number of instances or references that a given feature can contain. Moreover features and types have group cardinalities, which constrain the number of child instances, i.e., the instances contained by sub-features. For the Class modeling a model is again a set of type definitions, features and constraints. The abstract modifier indicates that no instance of the type will be created, unless extended by a concrete type. Now features correspond to attributes or role names of association or composition relationships in UML. The mapping between class and feature modeling is provided by adding a constraint to the original feature model resulting in a constrained feature model.

## Clafer model

In this section I will describe how a Clafer model looks like and what does it contain. First of all Clafer models consist of Clafers and constraints. Clafers express the domain concepts and the variability among them with the help of nested constraints. There are two kinds of Clafer abstract Clafers and concrete Clafers. The declaration of abstract Clafers defines a new type (the set of abstract Clafers represent the Metamodel of the domain of the system under test).The Concrete Clafers represent a possible set of instances of an abstract one: there are several features associated with a Clafer model.

- Hierarchy / nesting: this is supported by means of indentation.
- Cardinality: defines how many instances of a given Clafer can appear as children of the parent Clafer.
- Group cardinality: defines how many children or a given Clafer can be instantiated.
- Inheritance: where both concrete and abstract clafers inherit everything from their superclafers.

- Reference: which are used for defining relations between Clafers.

We can understand these features of Clafer by means of the following example which is illustrated in the following figure.

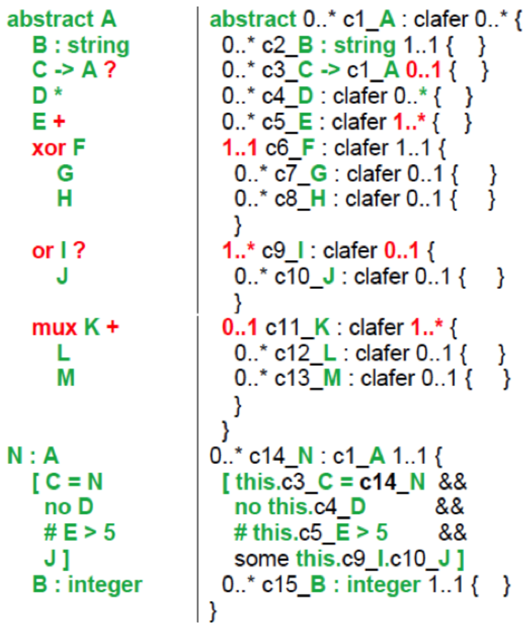
```
abstract Vehicle
    serialNo : int

Automobile : Vehicle
    xor Engine
        Gasoline
        Electric
    or Radio ?
        CDPlayer
        Tape

[Electric  ⟹  Radio]

numOfAutomobiles : int
[numOfAutomobiles = #Vehicle]
```

figure1: Clafer Model

The model consists of an abstract Clafer with the name vehicle that contains the integer attribute serial number. The Clafer vehicle is a super-Clafer of the Automobile Clafer and on the other way around Automobile is a sub-clafer of the vehicle Clafer. Automobile is a concrete Clafer as long as it can give instances to the abstract Clafer vehicle. The Automobile Clafer also contains the attribute of the serial numbers as long as its superclafer contains it. As we can see the model is based on indentation. Automobile has two children engine and radio. The XOR in front of the engine declares group cardinality and it stands for exactly one of (1..1 cardinality). Similarly the OR has the same role and it stands for at least one (1..* cardinality). The attribute Gasoline and Electric are children of the attribute Engine and descendants of Automobile. The same happens with the attributes CD player and Tape. As we can see the attribute Radio contains the symbol “?” which declares cardinality and stands for 0. . . 1 cardinality. In other words the attribute Radio is optional attribute and it defines that the Radio can appear as an instance of its parent Clafer Automobile. Moreover we can see that we can apply constraints with Clafer based on indentation. We can add the integer attribute of numOfAutomobiles that stores the number of all vehicles in the model. The constraint in the last line specifies that the number of Automobiles is equal to the number of vehicles. Finally the arrow (=¿) declares implication. If the engine of the automobile is Electric then the Automobile must contain Radio. At this point I would like to mention that if we don’t specify the cardinality of some elements Clafer assumes that by default Clafers are mandatory and their cardinality is equal to 1..1.

## Clafer to Alloy Translator

Clafer is designed simultaneously with the clafer to alloy translator. The translator/compiler takes a Clafer model and transforms it to corresponding Alloy model. This happens in order to give precise semantics to the language by preforming semantic analysis and establishing a mapping to Alloy. The translator comprises of lexer and parser of layout resolver which makes a Clafer model shorter and easier to read by reconstructing the code (for example the relationship parent-child can be simply expressed by identifying sub features further). The translator also contains the desugarer. Clafer is composed of two languages the core language and the full language [3]. The desugarer takes out the large amount of syntactic sugar from the full language turns it into a minimal language with well-defined translational semantics. The separation of the two languages simplifies the semantics analysis. The translator also contains the semantic analyzer. The semantic analyzer translates the same parts that can cause conflicts because of the different nature of Alloy and Clafer. Finally the code generator transforms the core language into Alloy. In the following figure we can see a contrived example of the desugarer.



```
abstract A               | abstract 0..* c1_A : clafer 0..* {
   B : string            |  0..* c2_B : string 1..1 {   }
   C -> A ?              |  0..* c3_C -> c1_A 0..1 {   }
   D *                   |  0..* c4_D : clafer 0..* {   }
   E +                   |  0..* c5_E : clafer 1..* {   }
   xor F                 |  1..1 c6_F : clafer 1..1 {
      G                  |    0..* c7_G : clafer 0..1 {   }
      H                  |    0..* c8_H : clafer 0..1 {   }
                         |    }
   or I ?                |  1..* c9_I : clafer 0..1 {
      J                  |    0..* c10_J : clafer 0..1 {   }
                         |    }
   mux K +               |  0..1 c11_K : clafer 1..* {
      L                  |    0..* c12_L : clafer 0..1 {   }
      M                  |    0..* c13_M : clafer 0..1 {   }
                         |    }
                         |  }
N : A                    | 0..* c14_N : c1_A 1..1 {
   [ C = N               |  [ this.c3_C = c14_N  &&
    no D                 |    no this.c4_D       &&
    # E > 5              |    # this.c5_E > 5    &&
    J ]                  |    some this.c9_I.c10_J ]
   B : integer           |  0..* c15_B : integer 1..1 {   }
                         | }
```

figure2: Desugared Model

Fragments in green and bold are the same in both models. Fragments in red and bold on the left are expanded into corresponding fragments on the right. Fragments in black on the right are results of applying defaults, resolving indentation, and making the identifiers unique [4]. As we can see in the desugared model the following changes are made: - Names were mangled to be unique (note that a new Clafer B is declared as a child of N and its unique name is different from the Clafer B declared as a child of A)

- Indentation was resolved into explicit nesting using curly brackets .
- Clafer and group cardinalities were specified explicitly .
- The default super Clafer “clafer” was added .
- The multi-line constraint is written now into an explicit conjunction .
- The default qualifier “some” was added in the last constraint .

The distinction between the concise and desugared syntaxes is not strict. It depends on the modelers to choose the level of detail that fits best to their needs. For example, explicit nesting using may be freely mixed with indentation. Also, modelers may choose different styles of specifying cardinality [4].

## Clafer tools description

In this section the tools that Clafer is using are briefly discussed. Clafer contains a Clafer compiler which is used in order to translate Clafer models to Alloy, XML, and desugared Clafer. It also contains Clafer Instance Generator which is used for generating instances of Clafer models; to show counter examples, and conflicting constraints. Moreover Clafer contains the Clafer Configurator , an interactive, web-based, configurator which is used for attributed feature models with inheritance subset of Clafer. ClaferMOO is another part and is used for performing multi-objective optimization over Clafer models limited to the attributed feature models with inheritance subset. The ClaferMOO Visualizer visualizes optimal instances of Clafer models and allows the performance analysis of the Pareto Front. Finally the Clafer Wiki integrates informal documentation in natural language with more formal Clafer models [1].

## Refrences

1. http://www.clafer.org

2. Attributed Feature Models in Clafer,Kacper Bak, Generative Software Development Lab, University of Waterloo, Canada.

3. Clafer: a Unified Language for Class and Feature Modeling, Kacper Bak Generative Software Development Lab ,University of Waterloo, Canada

4. Domain Concept Modeling Using Clafer, A Tutorial By Michał Antkiewicz Version 9.2, Mar 20, 2012

5. Scotiabank Mortgages in Clafer, Draft v. 3 April 24, 2011 By Michał Antkiewicz