Attributed Feature Models in Clafer

Kacper Bąk[1]

Generative Software Development Lab, University of Waterloo, Canada,
kbak@gsd.uwaterloo.ca

Abstract. This tutorial explains how to read and encode attributed feature models in Clafer. It also presents Clafer constructions that go beyond feature modeling, but are useful for encoding variability models. The tutorial limited the scope to those constructions that were used in Clafer encodings of real-world variability models, such as the Linux kernel and eCos.

1 What Is Clafer?

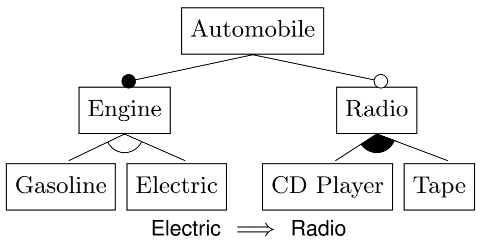
Clafer [2] is a textual language for concept modeling and specification of software product lines. In this document we are concerned with the latter application. In particular we show how to encode feature models with attributes. Those models are expressive enough to represent real-world variability models, such as the Linux kernel and eCos.

Clafer aims at providing a common infrastructure for analyses of feature and meta-models. The need for analyses was recognized, for example, in the operating systems domain by the Linux Kernel and eCos developers [1]. They both provide variability models of system kernels that are supported by configuration tools. The tools guide the configuration process, so that end-users are less likely to build incorrect kernels. Many non-trivial analyses of variability models are reducible to the NP-hard problem of finding model instances by combinatorial solvers. At present, Clafer translates input models to Alloy [3]. Alloy uses SAT solvers to do model checking, or to find model instances.

2 Feature Models in Clafer

Feature models [4] are tree-like structures that specify commonalities and variabilities within a software family. Figure 1a shows a feature model of variabilities found in an automobile product line.

Each box represents a *feature*, that is a user-relevant product characteristic, such as Engine or Radio. Mandatory features are marked with filled circles; optional features with hollow circles. In our example, every car must have an Engine, but not every car must have a Radio. Feature models also have *alternative groups* (marked with empty arcs) and *or-groups* (marked with filled arcs). An *alternative group* allows to select only one subfeature, e.g., a car must have either Gasoline or Electric engine. An *or-group* requires at least one subfeature to



a) FODA notation

```
Automobile
    xor Engine
        Gasoline
        Electric
    or Radio ?
        CDPlayer
        Tape

[Electric ⟹ Radio]
```

b) Clafer notation

Fig. 1. Feature model of automobile product line

be selected, e.g., a valid car configuration has either CD Player, or Tape player, or both of them. Finally, some constraints cannot be encoded in the tree structure. In that case we use cross-tree constraints specified as propositional formulas to restrict the feature model. For instance, the constraint below the tree says that cars with Electric engine must have some Radio player on board.

Figure 1b shows a Clafer model that corresponds to the automobile feature model from Fig. 1a. The hierarchy of model elements is established by code indentation. Feature group (*alternative* and *or*) specification precedes the name (e.g. xor before Engine). In contrast with FODA feature models, each feature in Clafer may have specified group cardinality that restricts the number of subfeatures that can be selected. For instance, for Engine exactly one of its subfeatures can be selected; for Radio at least one. Optionality of elements is marked by the question mark following the name (e.g. Radio).

FODA feature models make use of propositional constraints. Cross tree constraints are specified in square brackets in Fig. 1b. Those constraints involve feature names and a standard set of logical operators, i.e., equivalence ($\iff$), exclusive-or (xor), implication ($\implies$), conjunction (&&), disjunction (||), and negation (∼). Clafer also allows *if-then-else* expressions (*condition* => *trueExp* else *falseExp*) as logical expressions.

3 Feature Models with Attributes

Several extensions to feature models have been proposed; among them feature *attributes*. An attribute allows feature to store a value of primitive type (such as integer, string, enumeration). Let us extend the automobile example with two attributed features: MaxSpeed and Model (see Fig. 2). The former specifies car's maximum speed in km/h, the latter car model name.

In Clafer, feature attribute follows feature name, i.e., it is followed by colon, and then by type of the attribute. MaxSpeed is an integer number; Model a textual string. Furthermore, both features are constrained, so that values of attributes are well-defined. We set car's maximum speed to 240 in the last line.

```
Automobile
    xor Engine
        Gasoline
        Electric
    or Radio ?
        CDPlayer
        Tape
    MaxSpeed : integer
    Model : string
        [this = "Supercar"]

[Electric ⟹ Radio]
[MaxSpeed = 240]
```

Fig. 2. Attributed feature model of automobile product line

Clafer constraints are either global or specified within context of a feature. The context is determined by code indentation (similarly to feature hierarchy). Putting constraint in the context allows for easier referencing feature names. We specified car's model name as a constraint in the context of the Model feature. The constraint refers to this that resolves to Model and sets its value to "Supercar".

Integer and string attributes may participate in relational expressions that compare values: equality (=), inequality (/=), less than (<), less than or equal (<=), greater than (>), greater than or equal (>=). Integers also participate in arithmetic expressions: addition (+), substraction (-), and multiplication (∗). We use the concatenation operator (++) to concatenate strings.

4 Other Clafer Constructions

4.1 Abstract Clafers

So far the Clafer models included only concrete features. It means that the presence of features is reflected in configuration semantics. Clafer models may also incorporate *abstract* features. Abstract features declare a new type, but do not exist on their own in configuration semantics. Their subfeatures cannot be selected or deselected unless the abstract feature is extended by a concrete one. Abstract features are a way of reusing parts of models. An abstract element has subfeatures that become subfeatures of an element that extends the abstract feature.

Figure 3a presents a model similar to the one from Fig. 1. In introduces, however, the abstract Vehicle element. Each vehicle has a serial number specified as integer attribute. Automobile is a vehicle, that is, serialNo is one of its subfeatures. It is important to note that Automobile extends Vehicle. The mechanism is very similar to inheritance in object-oriented programming. Concrete features that extend an abstract feature share the same type.

```
abstract Vehicle
    serialNo : int

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

[Electric ⟹ Radio]
```

a) Abstract feature

```
abstract Vehicle
    serialNo : int

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

[Electric ⟹ Radio]

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

b) Set constraint

Fig. 3. Abstract features and set constraints in Clafer

4.2 Constraints over Sets

A single abstract feature may be extended by multiple concrete features (similarly to classes in OOP). When we want to specify constraints over abstract features, we no longer think about features of cardinality 0 or 1. We need to allow arbitrary cardinalities, because there can be any number of concrete features that extend given abstract feature. Thus, the constraints must be able to deal with sets, not only propositional variables.

Clafer constraints cover several set operations. Here we explain only the set-cardinality operator (#). Figure 3b shows a modified example from Fig. 3a. We added numOfAutomobiles that stores the number of all Vehicles in the model. Constraint in the last line specifies that numOfAutomobiles is equal to the number of Vehicles. The total number of vehicles takes into account concrete features that extend the abstract Vehicle feature. In our example numOfAutomobiles is equal to 1.

4.3 Quotation

Quotation is a syntactic sugar for reusing abstract features. To extend an abstract feature we write *fName* : *aName*, where *fName* is a new feature name, and *aName* is a name of extended abstract feature. For example, in Fig. 3a Automobile extends abstract Vehicle. At times we want the new feature (*fName*) to have the same name as the abstract feature (*aName*). For instance, in Fig. 4a there is an abstract Engine. *Automobile* has the Engine feature that extends the abstract Engine.

```
abstract xor Engine
  Gasoline
  Electric

Automobile
  Engine : Engine
  or Radio ?
    CDPlayer
    Tape

[Electric ⟹ Radio]
```

a) Explicit inheritance

```
abstract xor Engine
  Gasoline
  Electric

Automobile
  ‘Engine
  or Radio ?
    CDPlayer
    Tape

[Electric ⟹ Radio]
```

b) Implicit inheritance: quotation

Fig. 4. Quotation in Clafer

Clafer provides convenient syntax for such a construction: quotation. Figure 4b shows the same model as in Fig. 4a, but we use quotation instead of explicit inheritance. Syntactically it is a name of abstract feature (Engine) preceded by the backquote (‘) symbol.

References

1. Berger, T., She, S., Lotufo, R., Wąsowski, A., Czarnecki, K.: Variability modeling in the real: a perspective from the operating systems domain. In: ASE'10 (2010)
2. Bąk, K., Czarnecki, K., Wąsowski, A.: Feature and meta-models in Clafer: mixed, specialized, and coupled. SLE'10 (2010)
3. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. The MIT Press (2006)
4. Kang, K., Cohen, S., Hess, J., Nowak, W., Peterson, S.: Feature-oriented domain analysis (FODA) feasibility study. Tech. Rep. CMU/SEI-90-TR-21, CMU (1990)