A High Level Constraint Language for Product Line Enginering

Ángela Villota Gómez

March 21, 2018

1 HLCL

HLCL is a constraint-based conceptual modeling language for specifying product line models. A Product Line Model (PLM) defines all the legal combinations of variable artifacts in a product line by means of relationships among them. Variable artifacts refer to reusable elements that are part of some, but not all, products that can be build from the product line [6].

As a conceptual modeling language, HLCL comprises a set of constructs and rules to create PLMs. The set of constructs and rules is defined by the HLCL grammar. A collection of statements derived from the HLCL grammar is called a *script*, this script is an specific product line model in HLCL [3]. Scripts in HLCL are the result of the product line design process and can be obtained either by manual specification or by automated transformation from existing models. Thus, HLCL can be used as (i) a language to directly specify PLMs, (ii) an intermediate language between product line models and the execution language (iii) a common language to represent a PLM spread in several views, and (iv) a pivot language to represent a product line designed with PLMs in different specification languages.

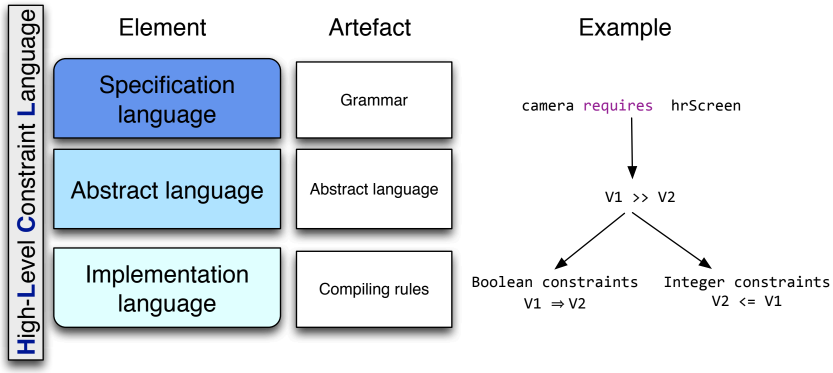

Figure 1: HLCL Overview

2 Specification Language

This section describes the grammatical rules to derive product line models in HLCL... In this section we introduce the grammatical rules for the operations of the HLCL, Assuming a set of names $\mathcal{N}$ and

3 Abstract Language

Product line models are defined by a set of variables representing its elements, a set of constraints between these elements and a set of product line predicates. Thus, a product line model M is defined by the triplet $M = \langle \mathcal{V}, \mathcal{C}, \mathcal{P} \rangle$ where:

$\mathcal{V}$ is the set of variables representing elements in a product line (i.e., requirements, artifacts, parts, etc).

$\mathcal{C}$ is the set of constraints. There are two constraints *requires* and *excludes* denoted by $V_i \gg V_j$ and $V_i \ll V_j$, respectively.

$\mathcal{P}$ is the set of product line predicates. There are four product line predicates *attribute*, *clone_at_least*, *clone_at_most*, *clone_exactly*.

Definitions

- A *product* is a set containing elements in $\mathcal{V}$.
- The *domain space* with respect to M, denoted as S_M, is a set containing all the products that can be built using the elements in $\mathcal{V}$. Thus the solution space is equivalent to the power set of $\mathcal{V}$ without the empty set $\emptyset$ (It is not possible to have a product without elements).

Table 1: HLCL Grammar

Product line model		
$\langle \mathit{model} \rangle$	::=	$\langle \mathit{identifier} \rangle$ $\langle \mathit{variables} \rangle$ $\langle \mathit{splConstraints} \rangle$
Variables and variants		
$\langle \mathit{variables} \rangle$	::=	‘`variables:`’ $\langle \mathit{varDeclaration} \rangle^{+}$
$\langle \mathit{varDeclaration} \rangle$	::=	$\langle \mathit{varType} \rangle$ $\langle \mathit{identifier} \rangle$ ‘`variants:`’ $\langle \mathit{variantDeclaration} \rangle$
$\langle \mathit{variantDeclaration} \rangle$	::=	$\langle \mathit{variantsInterval} \rangle \mid \langle \mathit{variantsEnumeration} \rangle$
$\langle \mathit{variantsInterval} \rangle$	::=	$\langle \mathit{value} \rangle$ ‘`::`’ $\langle \mathit{value} \rangle$
$\langle \mathit{variantsEnumeration} \rangle$	::=	‘`[`’ $\langle \mathit{valuesList} \rangle$ ‘`]`’ $\mid$ ‘`[`’ $\langle \mathit{idsList} \rangle$ ‘`]`’
Constraints		
$\langle \mathit{splConstraints} \rangle$	::=	‘`constraints:`’ $\langle \mathit{consDefinition} \rangle^{+}$
$\langle \mathit{consDefinition} \rangle$	::=	$\langle \mathit{idConstraint} \rangle$ ‘`:`’ $\langle \mathit{consExpression} \rangle$
$\langle \mathit{consExpression} \rangle$	::=	$\langle \mathit{idConstraint} \rangle \mid \langle \mathit{refinement} \rangle \mid \langle \mathit{rule} \rangle \mid \langle \mathit{foda} \rangle$
$\langle \mathit{refinement} \rangle$	::=	$\langle \mathit{assignment} \rangle \mid \langle \mathit{varRefinement} \rangle \mid \langle \mathit{setRefinement} \rangle$
$\langle \mathit{assignment} \rangle$	::=	$\langle \mathit{identifier} \rangle$ ‘`is`’ $\langle \mathit{value} \rangle$
$\langle \mathit{varRefinement} \rangle$	::=	$\langle \mathit{identifier} \rangle$ ‘`in`’ $\langle \mathit{variantsInterval} \rangle \mid \langle \mathit{variantsEnumeration} \rangle$
$\langle \mathit{setRefinement} \rangle$	::=	’`(`’ $\langle \mathit{idsList} \rangle$ ’`)`’ ‘`variants`’ ‘`[`’‘`(`’ $\langle \mathit{valuesList} \rangle$ ‘`)`’ { ‘`,`’‘`(`’ $\langle \mathit{valuesList} \rangle$ ’`)`’ $\}^{+}$ ‘`]`’
$\langle \mathit{rule} \rangle$	::=	$\langle \mathit{consExpression} \rangle$ ‘`-->`’ $\langle \mathit{consExpression} \rangle$
$\langle \mathit{foda} \rangle$	::=	$\langle \mathit{identifier} \rangle$ $\langle \mathit{fodaOP} \rangle$ $\langle \mathit{identifier} \rangle$
Syntactic categories		
$\langle \mathit{value} \rangle$	::=	$\langle \mathit{integer} \rangle$ $\mid$ ‘`selected`’ $\mid$ ‘`unselected`’
$\langle \mathit{varType} \rangle$	::=	‘`boolean`’ $\mid$ ‘`numeric`’
$\langle \mathit{valuesList} \rangle$	::=	$\langle \mathit{value} \rangle$ (‘`,`’ $\langle \mathit{value} \rangle)^{*}$
$\langle \mathit{idsList} \rangle$	::=	$\langle \mathit{identifier} \rangle$ (‘`,`’ $\langle \mathit{identifier} \rangle)^{*}$
$\langle \mathit{fodaOp} \rangle$	::=	‘`optional`’ $\mid$ ‘`mandatory`’ $\mid$ ‘`requires`’ $\mid$ ‘`mandatory`’
$\langle \mathit{idConstraint} \rangle$	::=	‘`C`’ $(\langle \mathit{letter} \rangle^{*} \langle \mathit{number} \rangle^{*})^{+}$

- A *product line* is a set of products specified by a product line model M. All the products in PL are consistent with the constraints and the predicates in M. A product line is a proper subset of the *domain space*, $PL \subseteq S_M$. *Note: this may change if we modify the predicates.*

Table 2 presents an example of a product line model with four variables, two constraints and zero predicates. The example shows that the product line represented by the product line model contains less products than the domain space. The following section presents the semantics of the constraints and the predicates.

3.1 Variables

Variables represent elements in a product line. In this representation all the variables represent reusable elements that are part of some, but not all, products in a product line. In this sense, variables may or may not be included in a

Table 2: Example

M	$=$	$\langle \mathcal{V}, \mathcal{C}, \mathcal{P} \rangle$
$\mathcal{V}$	$=$	$\{V_1, V_2, V_3, V_4\}$
$\mathcal{C}$	$=$	$\{V_2 \gg V_4, V_1 \ll V_3\}$
$\mathcal{P}$	$=$	$\{\}$
S_M	$=$	$\{\{V_1\}, \{V_2\}, \{V_3\}, \{V_4\}, \{V_1, V_2\}, \{V_1, V_3\}, \{V_1, V_4\}, \{V_2, V_3\}, \{V_2, V_4\},$ $\{V_3, V_4\}, \{V_1, V_2, V_3\}, \{V_1, V_2, V_4\}, \{V_2, V_3, V_4\}, \{V_1, V_3, V_4\},$ $\{V_1, V_2, V_3, V_4\}\}$
PL	$=$	$\{\{V_1\}, \{V_3\}, \{V_4\}, \{V_1, V_4\}, \{V_2, V_4\}, \{V_3, V_4\}, \{V_1, V_2, V_4\}, \{V_2, V_3, V_4\}\}$

product. For instance, consider products $\{V_1, V_3\}$ and $\{V_2, V_3\}$ in table 2, these products does not contain the variable V_4.

3.2 Constraints

Constraints are relations between variables used to characterize a product line. In this sense, the set of products represented by the PLM (e.g., product line PL) is reduced by each constraint. To explain how each constraint reduces the set of products in a product line, we first define the projection of a constraint over the domain space. Then, we present the definition of product line in terms of the projections of the constraints. Finally, a definition of the semantics of the constraints and predicates is presented.

Let $M = \langle \mathcal{V}, \mathcal{C}, \mathcal{P} \rangle$ be a product line model with domain space $S_M = \mathcal{P}(\mathcal{V}) - \emptyset$.

Projection of a constraint. The projection of a constraint C_i over the domain space denoted as Π_{C_i} is the set of products $P_i \in S_M$ satisfying the constraint C_i.

Product line. A product line PL is defined as the intersection of all the projections of constraints in the product line model. Let $M = \langle \mathcal{V}, \mathcal{C}, \mathcal{P} \rangle$ be a product line model with constraints $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$. The set of products represented by M is the product line:

$$PL := \Pi_{C_1} \cap \Pi_{C_2} \cap \cdots \cap \Pi_{C_n}$$

3.2.1 Requires

A requires constraint is a binary relation denoted as $V_i \gg V_j$. If a product line model contains the constraint $V_i \gg V_j$ then, the inclusion of V_i in a product implies also the inclusion of V_j. The requieres constraint is directional, therefore $V_i \gg V_j$ is different than $V_j \gg V_i$. The projection of $V_i \gg V_j$ is the set of products $\Pi_{V_i \gg V_j}$ containing all the products in the domain space that: (i) the product contains V_i, V_j at the same time, (ii) the product does not contain V_i, V_j, and (iii) the product contains V_j But does not contain V_i. Let $M = \langle \mathcal{V}, \mathcal{C}, \mathcal{P} \rangle$

be a product line model, with variables $\mathcal{V} = \{V_i, \ldots, V_n\}$ and constraints $\mathcal{C} = \{V_i \gg V_j\}$, the projection $\Pi_{V_i \gg V_j}$ is the set

$$\Pi_{V_i \gg V_j} = \{P \mid V_i, V_j \notin P\} \cup \{P \mid V_i, V_j \in P\} \cup \{P \mid V_i \notin P \wedge V_j \in P\}$$

Table 3 presents an example showing how to compute the projection of a requires constraint.

Table 3: Example of the requieres semantics

M	$=$	$\langle \mathcal{V}, \mathcal{C}, \mathcal{P} \rangle$	product line model
$\mathcal{V}$	$=$	$\{V_1, V_2, V_3, V_4\}$	variables
$\mathcal{C}$	$=$	$\{V_2 \gg V_4\}$	constraints
$\Pi_{V_2 \gg V_4}$	$=$	$\{\{V_2, V_4\}, \{V_1, V_2, V_4\}, \{V_2, V_3, V_4\},$ $\{V_1, V_2, V_3, V_4\}\} \cup \{\{V_1\}, \{V_3\}, \{V_1, V_3\}, \} \cup$ $\{\{V_4\}, \{V_1, V_4\}, \{V_3, V_4\}, \{V_1, V_3, V_4\}\} \cup$	projection

3.2.2 Excludes

A excludes constraint is a binary relation denoted as $V_i \ll V_j$. When the constraint $V_i \ll V_j$ is included in a product line model, the products derived from the product line model cannot contain variables V_i and V_j at the same time. This constraint is bi-directional, therefore, $V_i \ll V_j$ is equivalent to $V_j \ll V_i$. The projection of $V_i \ll V_j$ is the set of products $\Pi_{V_i \ll V_j}$ containing only the products that does not contain V_i, V_j. Let $M = \langle \mathcal{V}, \mathcal{C}, \mathcal{P} \rangle$ be a product line model, with variables $\mathcal{V} = \{V_i, \ldots, V_n\}$ and constraints $\mathcal{C} = \{V_i \ll V_j\}$, the projection $\Pi_{V_i \ll V_j}$ is the set

$$\Pi_{V_i \ll V_j} = \{P \mid V_i, V_j \notin P\}$$

Table 4 presents an example showing how to compute the projection of a excludes constraint.

Table 4: Example of the excludes semantics

M	$=$	$\langle \mathcal{V}, \mathcal{C}, \mathcal{P} \rangle$	product line model
$\mathcal{V}$	$=$	$\{V_1, V_2, V_3, V_4\}$	variables
$\mathcal{C}$	$=$	$\{V_1 \ll V_3\}$	constraints
$\Pi_{V_1 \ll V_3}$	$=$	$\{\{V_1\}, \{V_2\}, \{V_3\}, \{V_4\}, \{V_1, V_2\}, \{V_1, V_4\},$ $\{V_2, V_3\}, \{V_2, V_4\}, \{V_3, V_4\}, \{V_1, V_2, V_4\}, \{V_2, V_3, V_4\}$	projection

3.3 Predicates

Predicates are relations between variables used to represent particular characteristics of such variables. We included in the language a set of predicates to represent relations such as: composition, and cardinalities.

Composition.

1. $attribute(a, V)$ to estate that the variable a represents an attribute of V.
2. $parent(V_1, V_2)$ to describe a hierarchical relation between V_1, V_2.

Cardinalities.

1. $clone\_at\_least(V, a)$ to state that the element represented by a variable V must be cloned at least a times.
2. $clone\_at\_most(V, a)$ to state that the element represented by a variable V must be cloned at most a times.
3. $clone\_exactly(V, a)$ to state that the element represented by a variable V must be cloned exactly a times.
4. ... other for cardinality n..m

3.4 Other variability constraints

This is a collection of variability constraints and its mapping to HLCL

Table 5: Variability constraints mapped to HLCL

Constraint (name in literature)	Description	HLCL representation
Excludes [4]	if V_1 is present in a product then V_2 must not be present	$V_1 \ll V_2$
Requires [4]	If V_1 is present in a product, then V_2 must be present too	$V_1 \gg V_2$
Mandatory [4]	If V_1 is present in a product, then V_2 must be present as well and vice-versa.	$V_1 \gg V_2 \wedge V_2 \gg V_1$
Optional [4]	If V_1 is present in a product V_2 may be present, but if V_2 is present, then V_1 is present too	$V_2 \gg V_1$
Cardinality [2] Cloning [7]	Let a be an integer. There are a copies of V_1 in the product	$clone\_exactly(V_1, a)$
	There are at least a copies of V_1 in the product	$clone\_at\_least(V_1, a)$
	There are at most a copies of V_1 in the product	$clone\_at\_most(V_1, a)$
Attributes [2, 1], Non-functional features [5]	Let $a_1, \ldots, a_n$ be variables representing attributes of V. If V is present in a product, then all its attributes are present in the product.	$(attribute(a_1, V) \wedge \cdots \wedge attribute(a_n, V)) \wedge (V \gg a_1 \wedge \cdots \wedge V \gg a_n) \wedge (a_1..a_n \gg V)$

References

[1] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés. Automated reasoning on feature models. In *Proceedings of the 17th International Conference on Advanced Information Systems Engineering*, CAiSE'05, pages 491–503, Berlin, Heidelberg, 2005. Springer-Verlag.

[2] Krzysztof Czarnecki and Chang Hwan Peter Kim. Cardinality-based feature modeling and constraints: A progress report. In *International Workshop on Software Factories*, pages 16–20, 2005.

[3] Andrew Gemino and Yair Wand. A framework for empirical evaluation of conceptual modeling techniques. *Requirements Engineering*, 9(4):248–260, Nov 2004.

[4] Kyo C Kang, Sholom G Cohen, James a Hess, William E Novak, and a Spencer Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report November, DTIC Document, 1990.

[5] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh. Form: A feature-oriented reuse method with domain-specific reference architectures. *Ann. Softw. Eng.*, 5(1):143–168, January 1998.

[6] Raul Mazo. Product line models a generic approach for automated verification of product line models, 2012.

[7] Raphael Michel, Andreas Classen, Arnaud Hubaux, and Quentin Boucher. A formal semantics for feature cardinalities in feature diagrams. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '11, pages 82–89, New York, NY, USA, 2011. ACM.