Tool Interview – FeatureIDE

1 INITIAL CONCEPTUAL MODEL

The initial conceptual model describes essential concepts of modeling variability of a software system in space and time. Additionally, the model unifies those concepts to represent revisions of variable system parts. In Table 1 we provide a definition of the involved concepts.

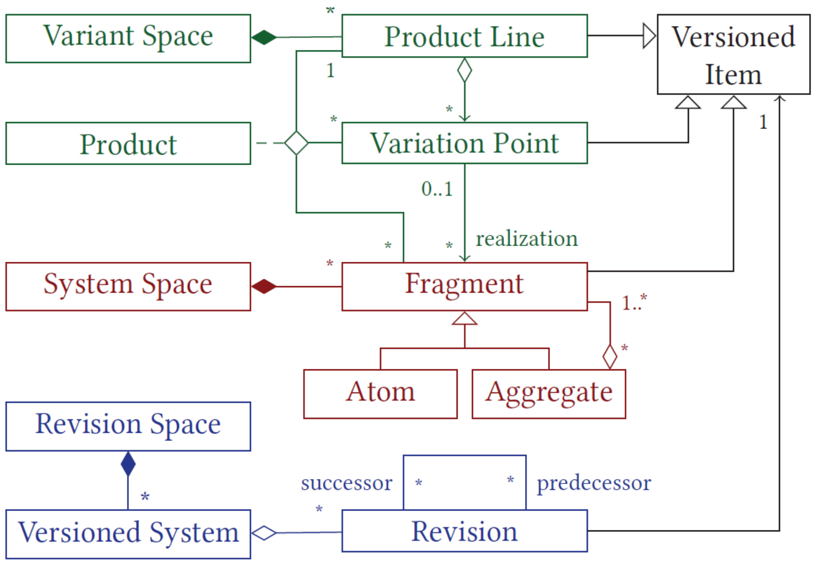

Figure 1: The Initial Conceptual Model with essential and combining Concepts for Variability in Space and Time.

Table 1: Definition of concepts in the Conceptual Model.

Concept	**Direct relation to other Concepts**	**Definition**
Fragment	*Variation Point, Product*	*Fragments* are the essential concept to describe a system. A *Fragment* can either be an atom or an aggregate, e.g. a single file, character or the node of an AST. A hierarchical structure of containments is not enforced but instead *Fragments* can be composed to various combinations.
Product Line	*Variation Point, Versioned Item*	A *Product Line* represents the configurable space regarding spatial variability and is composed of a system's *Variation Points*.
Variation Point	*Product Line, Fragment, Product, Versioned Item*	A *Variation Point* expresses the variability of a system by representing an option set for variation of the *Product Line*.

Product	*Product Line, Variation point, Fragment*	A *Product* is fully specified if all existing *Variation Points* in the *Product Line* are bound to *Fragments*. A partial *Product* does not require the binding of every *Variation Point*.
Revision	*Versioned Item*	A *Revision* of the *Fragment* evolves along the time dimension and is intended to supersede its predecessor, e.g., due to a bug fix or refactoring.
Versioned System	*Revision*	A *Versioned System* represents the configurable space regarding temporal variability. It is composed of a system‘s revisions.
Versioned Item	*Revision*	The *Versioned Item* represents versioning of the introduced concepts for *Fragment*, *Variation Point* and *Product Line* by putting them under revision control.

Table 2: Particular Relations of the Conceptual Model.

Relation	Direct relation to Concepts	Definition
Realization	*Variation Point, Fragment*	Each *Variation Point* has a set of possible options for variation whereby each option is realized by *Fragments*.
Configuration	*Product Line, Variation Point, Fragment*	A *Configuration* defines one particular *Product* of a *Product Line* by resolving the variability of a *Product Line*, i.e., binding all relevant *Variation Points* of a *Product Line* to *Fragments*.
Branching / Merging	*Revision*	To represent *branching* (which is considered a temporary divergence for concurrent development) along with *merging*, multiple (direct) successors and predecessors relate to a revision. This relation gives rise to a revision graph, which is a directed acyclic graph where each node represents a unique revision.

2 INTERVIEWS

Please inspect

1. If
2. and if yes, how

concepts of the conceptual model are represented by constructs used in your tool. Therefore, the representation of each concept in the tool and their (direct) relation to other constructs is considered separately.

Table 3: Concept Mapping between Conceptual Model and Tool.

Concept	**Representation of Concept in Tool**	**Relation to other Constructs**
Fragment	Artifact type depends on composer (Antenna -> Source Code Lines in Java Files; Munch -> XML files; AHEAD -> Jak files), but different artifacts types can't be mixed which is a design decision in FeatureIDE.	Feature, Mapping, (Attributes) (Attributes represent either a property of a feature or an input for a configuration)
Product Line	Feature Model + Cross-Tree Constraints (propositional logic); Multi-Product-Lines incorporated by the tool *Velvet* which composes multiple feature models to one monolithic model with the same root feature.	Feature Model Feature, Constraint, Configuration
Variation Point	Features and Attributes.	Artifact, Constraint, Feature Model, Configuration
Product	A product is the result of a configuration and is represented by artifacts of one particular artifact type. The derivation of partial products is not supported.	Configuration, Artifact
Revision	No	
Versioned System	No	
Versioned Item	No	

Realization	Artifacts are mapped to features (Mapping via name).	Feature, Artifact
Configuration	A configuration is a subset of all features defined in the feature model. A configuration is valid if the combination of features is allowed by the feature model (i.e., if it fulfills the semantics of groups and all cross-tree constraints). Otherwise, the configuration is called invalid. Per default, all features are set to false (abstract features need to be configured as well).	Feature Model, Constraint
Branching / Merging	No	
Remarks	Address following concepts in the Conceptual Model? • Differ fragments into fragments that compose the system and fragments that test the system. Or define that test data belongs to the system. • Configuration as concept (instead of ternary association) which could inherit from Versioned Item • Multi-Product-Lines (e.g., self-reference of product line) • Shall views be represented by the conceptual model? • Define that a versioned System is finite. • Represent mandatory features explicitly or define that they are represented by a Variation Point (with a single realization) Is it a problem if fragments are reused within the realization of different Variation Points (e.g., VP 1 -> Aggregat B (c,d,e); VP 2 -> Node c)? Since the relation between revisions is represented by a Directed Acylic Graph, it is not possible to have a successor which references a predecessor.	

3 USE CASES

Please provide an overview of use cases that your tool addresses.

- Supports all development phases of SPLs
- Product Line Modeling and Analyses (Statistics, e.g., how many products exist)
- Teaching and Research
- Testing

4 PREVIEW: SEMANTICS

The semantics of several concepts is only defined through the mechanisms that operate on them. For example, the configuration of a product from a product line, variation points and fragments is expressed in the conceptual model, but constraints that define which variation points and fragments may be selected have to be ensured by a configuration mechanism. The same applies to the generic concept of the *Versioned Item*. A mechanism that defines how the relation between revisions of product lines, variation points and fragments can be combined has to be defined. Designing such mechanisms, based on the conceptual model, is the next step towards a unifying concept for variability in space and time.
We consider semantics represented by the following mechanisms of a system that deal with variability in space and / or time:

1) *Analyses mechanisms* support the validity of:
    a. the variability model
    b. the configuration
    c. the fragment

2) The *mapping mechanism* that is used to resolve a configuration from a variability model to a set of realization artifacts

3) A *variability realization mechanism* assembles realization artifacts for a configuration in a particular manner (*annotative* variability, e.g. #ifdefs; *compositional* variability, e.g., feature-oriented programming; *transformational* variability, e.g., delta modeling).

In the following, please describe the semantics of your tool regarding the described mechanisms

Analysis mechanisms

Variability Model:

- Automated analysis and explanation of feature-model anomalies, such as dead and false-optional features, based on Sat4j
- Categorization of feature-model edits into refactoring, generalization, specialization, and arbitrary edits
- Folding and layouting that scale to large feature models with thousands of features
- Import and export of feature models to tools and languages, such as Dimacs, fmp: Feature Modeling Plug-in, GUIDSL, S.P.L.O.T., SPL Conqueror, Velvet

Configuration:

- Manual configuration mechanism + decision propagation (features that would not lead to a valid configuration cannot be selected anymore) + auto-complete (per default, all feature-values are set to false)
- T-wise sampling with IncLing, SPLCATool, CASA
- Automated generation of all valid configurations
- Automated generation of all current configurations (defined in requirements engineering)
- Automated generation of random configurations

A configuration is valid if the combination of features is allowed by the feature model (i.e., if it fulfills the semantics of groups and all cross-tree constraints (SAT Solver)). Otherwise, the configuration is called invalid.

Fragment:

- Automated product generation after changes for one selected configuration
- Propagation of compiler markers to original source code
- Conflict resolution (if a conflict in a configuration occurres due to a feature model change, the resolution is performed either manually (while providing user support) or automated, e.g., a feature is deleted)

Additional: Family-based analyses

- Detection of dead code blocks and superficial ifdefs in annotation-based product lines based on Sat4j
- Family-based type checks for FeatureHouse projects with Fuji
- Family-based parsing and type checks for CPP projects with TypeChef

Mapping mechanism

Mappingin FeatureIDE relates artifacts to features via name. Additionally, FeatureIDE provides a projective view on the mapping.

Variant realization mechanism

FeatureIDE supports several implementation techniques:

- Feature-oriented programming with AHEAD (Java 1.4), FeatureC++ (C++), and FeatureHouse (C, Java 1.5, JML, Haskell, XML, JavaCC) (classes haben feature modules, wobei jedes ein feature implementiert)
- Aspect-oriented programming with AspectJ (Java) and FeatureC++ (C++)
- Delta-oriented modeling and programming with DeltaEcore and DeltaJ (Java)
- Annotation-based implemenation with preprocessor Antenna, C preprocessor CPP by Colligens, and preprocessor Munge
- Black-box frameworks with plug-ins
- Runtime variability with runtime parameters and property files
- Statistics and code metrics for product-line implementations
- Refactoring, source-code documentation with JavaDoc, and formal specification with JML

➔ annotational, compositional and transformational variablity realization is supported

A. TABLE OF TABLES

B. REFERENCES

[1] S. Ananieva, T. Kehrer, H. Klare, A. Koziolek, H. Lönn, S. Ramesh, A. Burger, G. Taentzer and B. Westfechtel, "Towards a conceptual model for unifying variability in space and time," *Proceedings of the 2nd International Workshop on Variability and Evolution of Software-Intensive Systems,* 2019.

[2] G. Guizzardi, L. F. Pires and M. van Sinderen, "An Ontology-Based Approach for Evaluating the Domain Appropriateness and Comprehensibility Appropriateness of Modeling Languages," *Proceedings of the International Conference on Model Driven Engineering Languages and Systems,* 2005.