Tool Interview – SuperMod

1 INITIAL CONCEPTUAL MODEL

The initial conceptual model describes essential concepts (or a superset of it) for modeling variability of a software system in space and time and shall subsume functionality related it. Additionally, the model unifies those concepts to represent revisions of variable system parts. The conceptual model follows an open-world assumption (descriptive) instead of a closed-world assumption (prescriptive) as metamodels commonly do. In Table 1 we provide a definition of the involved concepts.

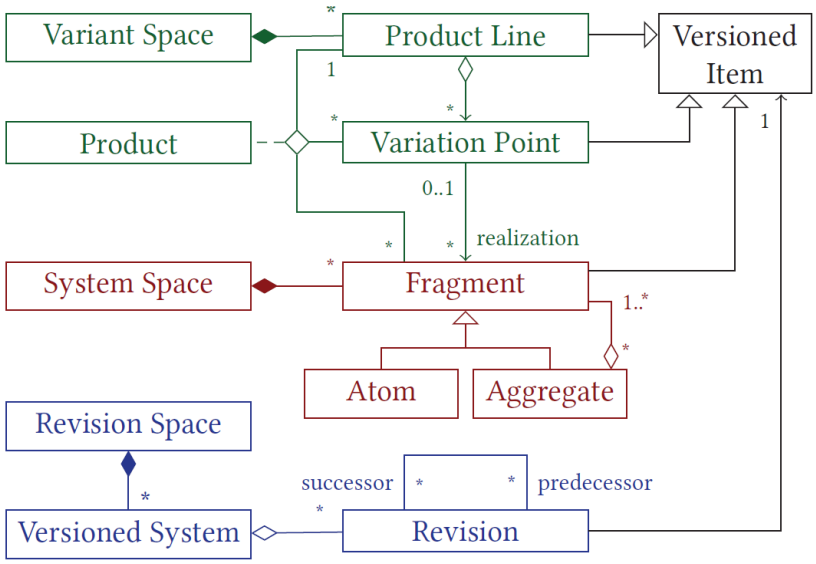

Figure 1: The Initial Conceptual Model with essential and combining Concepts for Variability in Space and Time.

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 on realization level. 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,*	A *Variation Point* expresses the variability of a system by representing an option set for variation of the *Product Line*.

	Product, Versioned Item	A *Variation Point* can either be explicit (e.g., if-defs or a plug-in system with a compositional variability realization mechanism) or implicit (a reference between a feature module and fragment represents the implicit variation points, therefore the fragment is not aware of its variation e.g., FOP, AOP, delta modeling).
Product	*Product Line, Variation point, Fragment*	A *Product* is fully specified if all existing *Variation Points* in the *Product Line* are bound to *Fragments* or *Variation Points* are not bound explicitly, e.g., if a feature is optional and not selected for product (hence, all to a configuration relevant *Variation Points* 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 by an increment, 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 the Conceptual Model and Tool.

Concept	**Representation of Concept in Tool**	**Relation to other Constructs**
Fragment	Equivalent notion: *artifact* Artifacts of a software project under Version Control are composed in a file hierarchy within the repository containing arbitrary models (EMF models) and non-model resources (e.g., plain text and XML files).	Variation Point, Feature Model, Revision Graph
Product Line	A feature model provides a logical variability model for the product line and is additionally subject to evolution. The feature model is contained within both the version and the product dimension. Not covered by Conceptual Model: *Version rules* (constrain the set of available choices and ambitions realized as logical expressions over the option set. Version rules are used, e.g., in order to implement constraints such as mutual exclusion within feature models, or to designate subsequent revisions → transparently mapped to feature model constraints) *Visibilities* (logical expressions over the option set, which are attached to elements of the feature or domain model. In order to test an element's presence in a specific version, the bindings specified by the respective choice are applied. Visibilities are modified automatically during the commit	Variation Point

	operation. A visibility may be an option reference or a composed expression (e.g., and, or, not)).	
Variation Point	Encapsulated within the feature model.	Feature Model, Artifact
Product	A product represents the resolution of all variation points (which is defined during check-out). Partial resolution of variation points is not supported to avoid consistency issues which may occur otherwise (e.g., different names for same class in workspace). A product exists outside of the SuperMod scope and within the workspace of a user.	Feature Model, Variation Point, Artifact
Revision	Realization artifacts are subject to variability in space (features) and time (revisions). The product line itself is subject to only variability in time. For historical versioning, the framework utilizes a sequence of revisions, such that branches are disallowed. Each revision corresponds to one historical state of the product line; the selection of multiple or no revision is forbidden. In this way, extensional versioning is realized. Extensional versioning is on top of intensional versioning which is chosen as base mechanism for version definition (a visibility is mapped to each versioned Element). Rather than being available for arbitrary modification, the revision graph may be extended only indirectly by committing one new revision per edit session -> an artifact in a particular revision is immutable.	Artifact, Feature Model
Versioned System	Equivalent notion: *repository* A SuperMod repository consists of three layers: 1) The revision graph (DAG) (version space) 2) The multi-version feature model (product space) 3) The multi-version domain model (both) The repository is built upon the uniform version model UVM adding higher-level representations for both the version space (by feature models and revision graphs) and the product space (by the use of EMF models).	Feature Model, Revision, Artifact

	→ The complete system (i.e., product line) is put under revision control, not single features.	
Versioned Item	Similar concept: Versioned Element A Versioned Element represents a super class for all elements to be versioned, e.g., the feature model and the domain model. ➔ Difference to CM: Versioned Items exist on a rather fine-grained level, i.e., variation points and fragments.	Feature Model, Artifact
Realization	Relations between artifacts and corresponding conceptual units (i.e., features) are represented through annotations (not by explicit relations).	Artifact, Feature Model (?)
Configuration	As the feature model defines features as configuration options, a unique selection in the feature model corresponds to a feature configuration. The user is requested to assign a unique selection state (selected or deselected) to each feature. Moreover, the selected configuration must conform to version selection rules defined in the feature model, including parent-child relationships, groups, and requires/excludes relationships ➔ A feature configuration in SuperMod specifies choices and ambitions. Choices and ambitions are represented by Option-Binding, which maps options to selections.	
Branching / Merging	Deliberately no support for branching, but for a (temporary) three-way merge.	Revision
Remarks	• Constraints are not covered by the Conceptual Model • Changes performed on the feature model are propagated back so that feature model in repository is updated SuperMod comprises three metamodels: • The Hybrid Version Model • The Extensible Extrinsic Product Model • The Consistency-Preserving Dynamic Editing Model	

3 USE CASES

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

- Targets the incremental development of model-driven software product lines
- Integrates temporal and logical versioning (variability in time and space)
- Supports Single- and Multi-User operation
- Relies on a single version workspace and enables checkout, modify and commit operations

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.

Analyses mechanisms

Variability model:

- The variability model shall be valid with respect to constraints induced by the model, no feature defect shall be present.

Configuration:

- Consistency of feature selection is covered by the configuration mechanism which is based on the feature model. The feature model induces constraints by its tree hierarchy and cross-tree constraints between features (i.e., so-called version rules in SuperMod).

Fragment:

The fragment shall be valid with respect to its notation (syntactical validity is ensued by the tool, but not its semantical validity which is left to the user (this is a conceptual model; java code does not need necessarily to realize the pragmatics of a feature for which it is intended))

Mapping model + variant derivation mechanism

The variant derivation mechanism in SuperMod is applied during check-out after a feature configuration (*Choice*) has been defined. The mechanism evaluates all values of versioned elements (which can be either true or false) and assembles all elements whose values are evaluated to true (which, as a whole, represent the final product within the user workspace).

Variability realization mechanism:

The variability realization mechanism is realized by symmetric deltas, i.e., negative or annotative variability.

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.