Tool Interview – DeltaEcore

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.

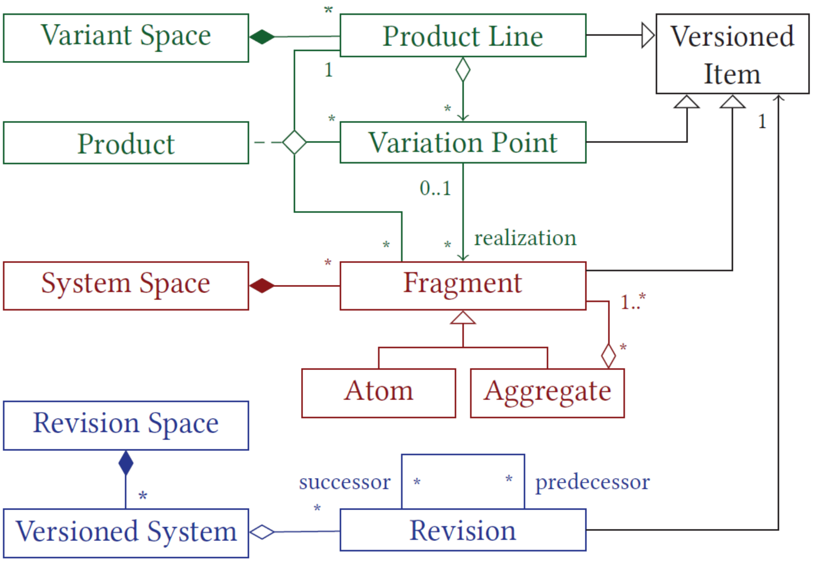

Figure 1: The Initial Conceptual Model with essential and combining Concepts for Variability in Space and Time.

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, Product, Versioned Item*	A *Variation Point* expresses the variability of a system by representing an option set for variation of the *Product Line*. 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 Conceptual Model and Tool.

Concept	Representation of Concept in Tool	Relation to other Constructs
Fragment	A fragment can be any type of realization artifact and may span code, models, documentation etc. Due to technical reasons, a prerequisite is that these fragments have a representation based on EMF Ecore, i.e., a meta model that is suitable to represent concrete fragments as models of that meta model. There are no further requirements on, e.g., the structure of the meta model or regarding marking of variation points. Fragments can be arranged to various combinations representing a graph structure.	A fragment may reference another fragment, e.g., an import of a class.
Product Line	A product line is comprised of a set of feature arranged in a tree-structured feature model with additional cross-tree constraints along with delta modules that invasively modify fragments via transformation and a mapping between feature combinations and lists of delta modules. Each feature can have multiple versions, which are related in a tree-structure as well, i.e., branching of feature versions is supported but merging is not (due to technical reasons; conceptually possible). Cross-tree constraints and mapping obey versions as well. As prerequisite for using a product line in practice, each language used to specify fragments and should be subjected to variability needs an adequate delta language (which can be specified/generated with the tool), e.g., for	Variation Point

	State Machines, one needs a State Machines Delta Language. The problem space comprises a feature model + cross-tree constraint (both can be version-aware), while the solution space encompasses configuration delta modules (features) and evolution delta modules (versions of features).	
Variation Point	Implicit due to invasive nature of delta modeling, i.e., one can reference any fragment (via user-/language-customizable links) and modify it so that there is no explicit notion of a variation point in the fragment being modified.	Fragment (there is no relation to feature model in the first place)
Product	Represented on two levels: A product on the conceptual level is represented by a configuration, i.e., a selection of features in specific versions. A product on the realization level is referenced as a product, i.e., the fragments in the variation representing the selection of the configuration (in other words, the software system resulting from the configuration). A product is represented by fragments of any type of realization, e.g., state machine + java code. A product can either be created from scratch (pure delta modeling) or based on another existing product.	
Revision	Represented as a version of a feature, which denotes an implementation of a feature at a particular point in time. Versions are perceived as being incremental, i.e., version 1.1 is a change to the functionality of version 1.0. This is important in the manifestation of revisions where, starting from a selected version, first, the branch back to the original version is determined and, then, all versions along this branch are applied as sequential transformations. Transformations may generally add, remove or modify artifacts so that previous changes may effectively be reverted. However, changes of previous versions do not have to be replicated.	Feature, Version

	Evolution delta modules are mutable but are not expected to be changed (“at the users own risk”).	
Versioned Item	Not in sense of the conceptual model. Fragments are versioned but they are unaware of it. Features may declare versions but are not versioned themselves.	
Versioned System	The sum of all fragments and evolution delta modules represents the versioned system. When also considering the variation space, configuration delta modules have to be considered as well.	Fragment, evolution delta modules, configuration delta modules
Realization	A mapping model represents the realization link between features and its versions to evolution and configuration delta modules.	Delta Modules, Mapping Model
Configuration	A configuration on conceptual level is a selection of features with exactly one version elected for each feature.	Hyper-Feature Model
Branching / Merging	Branching is supported, merging is not.	Version
Remarks	Ideas for refinement of conceptual model: • Include concept of a Feature? Multiplicity between feature and variation point is usually 1:n (one feature can be served by different variation points, each variation point representing a variation of a feature, e.g., if it is an alternative feature) • Remove spaces to keep model lean? What are the advantages of spaces? Interpret spaces as views on the system? • Apply versioning on a configuration and product? • The conceptual model represents an ontology (open-world assumption) → adapt syntax of model regarding an ontological description? • Include implicit and explicit variation points in the model (or address it at least in the definition of a variation point) • Include DarwinSPL and SiPL in evaluation (possibly BitLocker; differentiation to work of Conradi and Westfechtel; Change-Oriented Programming (Assmann))	

	Core concepts of Delta Ecore: • (Hyper) Feature Model (HFM) • Version-aware constraint language • Delta Modules (evolution, configuration, customization) • Mapping model • Configuration • Delta Dialects (differentiation possible of dialects shall be used only for evolution, e.g., ID changes) • Source Language (included by tool, not visible for user • Delta Language • Application order constraints *Would it be possible to treat variability in space and time separately or is it inseparable?* → It is possible to separately address both dimensions of variability, but the integrated management of variability in space and time and the primary use case of DeltaEcore

3 USE CASES

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

- DeltaEcore is an integrated approach for handling variability in space and time in software families.
- Delta language creation infrastructure for different source languages
- Setting and unsetting the value of single-valued references, adding and removing values of many-valued references, modifying the value of attributes and detaching an element from its container, insert a value into a many-valued reference
- Decentralized development of individual variable assets of a software family by different vendors with individual and potentially unsynchronized release cycles

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: • Analyses regarding the validity of a configuration, e.g., the root feature is part in all configurations, all propositional formulas of the cross-tree constraints have to be satisfied by the configuration • *automatic version selection procedure*

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 Mechanism

A mapping model associates version-aware logical expressions over features and feature versions with sets of delta modules.

A configuration of features and versions from an HFM is resolved to **a set of required delta modules by means of a mapping model** + a large part of the delta modules' ***application order*** *is derived* from the structure of the HFM.

Variability realization mechanism:

As variability realization mechanism, delta modeling as transformational mechanism is applied + a delta language creation infrastructure for different source languages.

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.