Validation of the Conceptual Model for Unifying Variability in Space and Time

1 UNIFIED CONCEPTUAL MODEL

The unified conceptual model (Figure 1) describes essential concepts for modeling variability of a software system in space (variants) and time (revisions). It follows an open-world assumption (descriptive) instead of a closed-world assumption (prescriptive).

In Table 1, we provide a definition of the involved concepts.

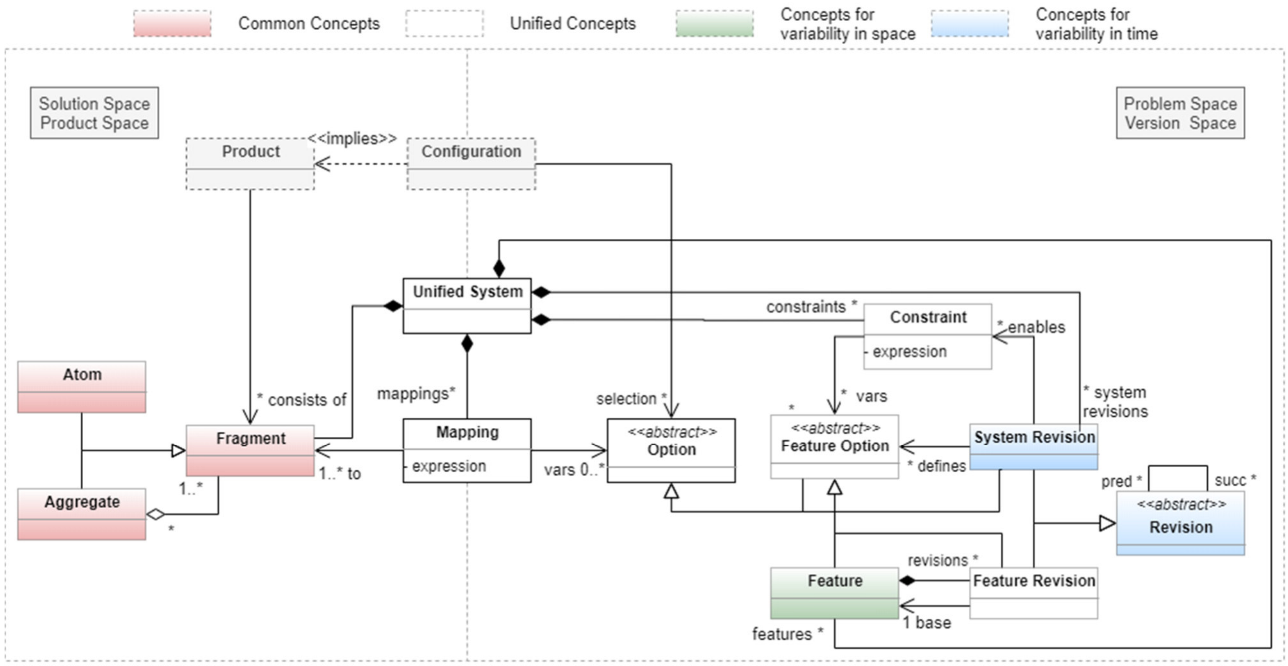

Figure 1: The Reference Model with common and unified Concepts for Variability in Space and Time.

Table 1: Definition of concepts in the Conceptual Model.

Concept	**Relation to other Concepts**	**Definition**
Fragment	*Product, Unified System, Mapping*	*Fragments* are the essential concept to describe a system on implementation level. A *Fragment* can either be an atom or an aggregate, e.g. a single file, character or the node of an AST. We explicitly do not specify the level of granularity for an atom or aggregate to remain as generic as possible. A hierarchical structure of containments is not enforced. Instead, *Fragments* can be composed to various combinations.

Product	*Configuration, (consists of *) Fragment*	A *Product* is implied by a configuration. A *Product* is not part of the system's state but can be computed from it based on the configuration. + Implementation level
Unified System	*(Contains *) Fragment, Mapping, Configuration, Constraint, Feature, System Revision*	The *Unified System* represents the unified configurable space regarding spatial and temporal variability. It subsumes concepts from both solution and problem space.
Mapping	*Unified System, (has *) Option variables, (references 1..*) Fragment*	A *Mapping* is an arbitrary expression (e.g., Boolean formula) that consists of *Option* variables that are mapped to fragments. Therefore, the Mapping connects concepts from the solution space (fragments) to concepts in the problem space (options).
Option	*Configuration, Mapping, Feature Option, System Revision*	An *Option* expresses the variability of a system. This can either manifest as variability in space (i.e., *Feature*) or variability in time (i.e. *System Revision or Feature Revision*).
Feature Option	*(Extends) Option, Constraint, System Revision, Feature, Feature Revision*	A *Feature Option* represents the configurable space on feature level.
Feature	*(Contains *) Feature Revision*	" A prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems [1]"
Revision	*(Has *) predecessor and successor Revision*	A *Revision* evolves along the time dimension and is intended to supersede its predecessor by an increment, e.g., due to a bug fix or refactoring. This relation forms a revision graph, which is a directed acyclic graph (DAG) with each node representing a unique revision.
System Revision	*(Extends) Revision, (defines *) Feature Option, (enables *) Constraint*	A *System Revision* extends the *Revision* and represents the evolutionary state of the entire system at one point in time. This state involves the definition of Features and Feature Revisions (e.g., System Revision 2 involves feature A in revision 1 and Feature B in revision 2) along with Constraints that are valid for the respective System Revision.
Feature Revision	*(has 1 base) Feature, (extends) Feature Option, (extends) Revision*	A *Feature Revision* extends the *Revision* and represents an evolutionary state of one particular *Feature* at one point in time.

Configuration	*(Has a selection of *) Options, implies Product*	A *Configuration* implies one particular *Product* of the *Unified System* and consists of a selection of *Option* variables. It is not part of the system's state. + Conceptual level
Constraint	*Unified System, System Revision, (has *) Feature Option*	The *Constraint* is an arbitrary expression (e.g., Boolean formula) that constrains *Feature Options* that can be combined in a *Configuration*.

2 MAPPING

To assess the mapping between concepts and relations of the unified conceptual model regarding the selected tool, each concept and relation is considered separately. For the sake of simplicity, we omit inheritance relationships.

2.1 CONCEPTS

For each concept of the conceptual model listed in Table 3, please inspect whether an equivalent construct exists in your tool and complete the form according to the following scheme in Table 2:

Table 2: Exemplary Mapping of ECCO (incomplete).

Concept in Model	**Maps to Construct (Name)**	**Does not map / Does not exist**	**Please comment, if concept is only partially reflected**
Fragment	Artifact	-	-
Product	-	✓	Because it is not part of the state of the system but exists as output in the form of files in the file system.
System Revision	-	✓	ECCO considers Feature Revisions only.

Table 3: Concept Mapping between Conceptual Model and Tool.

Concept in Model	**Maps to Construct (Name)**	**Does not map / does not exist**	**Please comment, if concept is only partially reflected**
Fragment	Configuration Delta Modules, Evolution Delta Modules Addition based on Skype Call with all delta-based tool-experts: Fragments are represented by a Core Model and		There are also dedicated fragments, i.e., implementation artifacts represented as models. These are not defined as part of DeltaEcore but they are modified in the course of SPL management.

	Delta Modules that modify that Core Model.		
Product	Product		
Unified System	Product Line		
Mapping	Mapping (Mapping Model)		
Option *(abstract)*	Configuration Selection		In DeltaEcore, a configuration consists of a combination of feature selections and feature version selections. These “configuration selections” can be considered an option.
Feature Option *(abstract)*	Feature Selection		s.o.
Feature	Feature		
Revision *(abstract)*			No counter part on abstract level. Feature Versions and Baselines (see below) are considered fundamentally different.
System Revision	-	✓	DeltaEcore considers Feature Revisions only. There is no direct counterpart to this construct, however, it can be emulated by the use of a “baseline”: a collection of feature versions that should be used together to form a system revision.
Feature Revision	Feature Version		
Configuration	Configuration		

Constraint	(Version-aware Cross-tree) Constraint		
Remarks	Might need a phone call to clarify some of the more obscure features of DeltaEcore. ☺		
Unmapped constructs in tool			

2.2 RELATIONS

For each relation of the conceptual model listed in Table 5, please inspect whether an equivalent relation exists in your tool and complete the form according to the following scheme in Table 4:

Table 4: Exemplary Mapping of ECCO (incomplete).

Name of Relation in Conceptual Model	**Maps to Relation**	**Does not map / Does not exist**	**If relation is only partially mapped, please name divergence (*source, target, multiplicity, direction* and *kind)***
Graph-based Fragment structure	*Tree-based* Fragment structure with cross-tree references	-	Uses strong containment instead of weak containment for children of fragments. To mitigate this limitation, ECCO uses cross-tree references.
Mapping *has 1..** Fragments	*equivalent*	-	
System Revision *defines ** Feature Options	-	✓	ECCO considers Feature Revisions only.

Table 5: Relation Mapping between Conceptual Model and Tool.

Name of Relation in Conceptual Model	**Maps to Relation**	**Does not map / Does not exist**	**If relation is only partially mapped, please name divergence (*source, target, multiplicity, direction* and *kind)***
Graph-based Fragment structure	*equivalent*		Dictated by EMF Ecore, which has a containment tree but a reference graph, so equivalent.
Product *consists of ** Fragments	*equivalent*		True but, practically speaking, a product consists of 1..* fragments.
Mapping *has 1..** Fragments	*equivalent*		True

Configuration *implies* Product	*equivalent*		True, via the use of delta modules to manifest variability (both in space and time).
Configuration has a *selection of* * Options	*equivalent*		True but, practically speaking, a configuration consists of 1..* options.
Unified System *has* * fragments	*equivalent*		1..*
Unified System *has* * Mappings	*equivalent*		True
Unified System *has* * Constraints	*equivalent*		True
Unified System *has* * Features			Not exactly. It has 1..* features (needs at least one when feature model is used but can be operated without feature model altogether as well, i.e., manual selection of delta modules at will).
Unified System *has* * System Revisions		✓	DeltaEcore considers Feature Revisions only. See comment above on baselines.
Mapping *has* * Option variables	Addendum after discussion: equivalent		Do not understand this, sorry.
Feature *has* * Feature Revisions	*equivalent*		True
Constraint *has* * Feature Option variables	Addendum after discussion: equivalent		Do not understand this, sorry.
System Revision *defines* * Feature Options		✓	DeltaEcore considers Feature Revisions only. See comment above on baselines.
System Revision *enables* * Constraints		✓	DeltaEcore considers Feature Revisions only. See comment above on baselines.
Revision *has* * successor	*equivalent* Sequence of		True but has * successors (not sure if this was in the original comment).

(Branching/Forking) and predecessor (Merging) Revisions	revisions (subsumed by DAG)		
Remarks			
All unmapped relations in tool			

A. REFERENCES

[1] K. Kang, J. Hess W. Novak, and A. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study.," Carnegie Mellon University, 1990.

[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.

[3] 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.