# Questionnaire – SiPL

# 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.

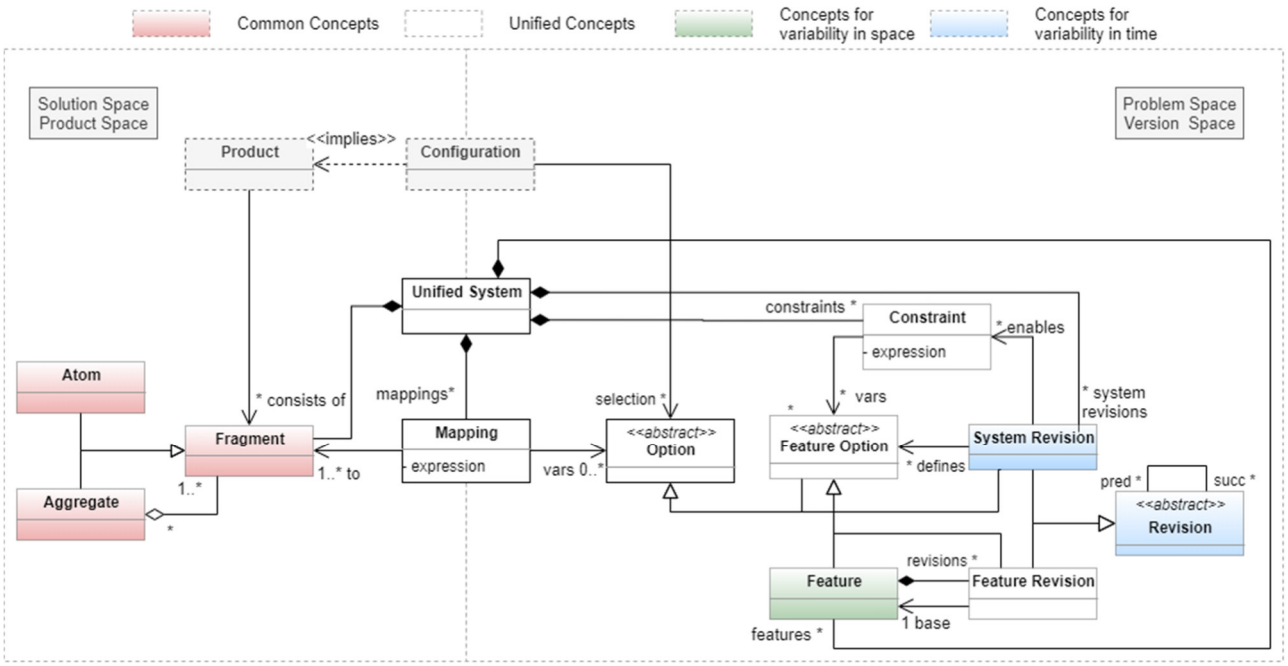


Figure 1: The Conceptual Model with common and unified Concepts for Variability in Space and Time.

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. |
| *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. |
| *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 2, please inspect whether an equivalent construct exists in your tool and complete the form according to the following scheme in **Fehler! Verweisquelle konnte nicht gefunden werden.**:

Table 2: 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 | Delta Module, Core Model<br><br>Addition based on Skype Call with all delta-based tool-experts: Fragments are represented by a Core Model and Delta Modules that modify that Core Model. | Does map | |
| Product | Product | Does map | |
| Unified System | Delta Module Set | Does map | |
| Mapping | Delta Application Condition | Does map | Each delta module comes with an application condition. A Delta Application Condition is a propositional expression over features of the feature model. A delta application condition specifies the condition when a delta module is to be applied for the sake of product generation. |

| | | | |
|---|---|---|---|
| Option *(abstract)* | | Does not exist | There is no direct mapping to any abstract construct in SiPL. Options are always features, so the abstract concept is only represented by a the concrete construct Feature in SiPL. |
| Feature Option *(abstract)* | | Does not exist | There is no direct mapping to any abstract construct in SiPL. Feature Options are always features, so the abstract concept is only represented by a concrete construct called Feature in SiPL. |
| Feature | Feature | Does map | SiPL uses Feature Models in the problem space. The Feature Model, in turn, contains the Features. |
| Revision *(abstract)* | | Does not exist | SiPL itself does not provide any specific constructs for managing variability in time but particularly considers variability in space. However, all the Fragments are versioned using version control systems such as Git or SVN. Hence, there a global System Revision can be managed using the version control system. Versioning at the feature level is no supported this way. |
| System Revision | | Partially maps (through the usage of a VCS) | see Revision |
| Feature Revision | | Does not exist | see Revision |
| Configuration | Configuration | Does map | |
| Constraint | Constraint | Does map | Constraints are attached to feature models. They are specified by a propositional formula over a set of features of the feature model. |
| Remarks | | | |

| | |
|---|---|
| All unmapped constructs in tool | SiPL uses a Feature Model for modeling variability in the problem space. The Feature Model, in turn, comprises Features and Constraints. |

## 2.2 RELATIONS

For each relation of the conceptual model listed in Table 3, please inspect whether an equivalent relation exists in your tool and complete the form according to the following scheme in **Fehler! Verweisquelle konnte nicht gefunden werden.**:

Table 3: 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 | *Tree-based* Fragment structure with cross-tree references | Does map | The tree-based structure with cross-tree references only applies to the Core Model. Please note that Delta Modules, which are the second kind of major Fragments in SiPL, also impose a Delta Relation Graph which is not part of the conceptual model. Details on the Delta Relation Graph are explained below (unmapped relations in tool). |
| Product *consists of* * Fragment | equivalent | Does map | Yes, Product consists of Fragments. However, please note that a Product does not consist of the main kinds of Fragments called Core Model and Delta Module. It rather consists of the Model Elements and Relationships contained by the Core Model or created by Delta Modules. In other words, a Product consists of the Model Elements and Relationships which are comprised by the Superimposed Model which is maintained in the background by SiPL. |
| Mapping *has 1..** Fragment | equivalent | Does map | |
| Configuration *implies* Product | equivalent | Does map | |
| Configuration has a *selection of* * Option | equivalent | Does map | Options are always Features |

| | | | |
|---|---|---|---|
| Unified System *has* * Fragment | equivalent | Does map | |
| Unified System *has* * Mapping | indirectly | Does map | The Unified System (represented by Delta Module Set in SiPL) does not directly contain the mappings. The Mappings are contained by Delta Modules (in the form of Delta Application Conditions). |
| Unified System *has* * Constraint | indirectly | Does map | Yes, but not directly. Unified System has a Feature Model which in turn comprises Constraints. |
| Unified System *has* * Feature | indirectly | Does map | Yes, but not directly. Unified System has a Feature Model which in turn comprises Features. |
| Unified System *has* * System Revision | indirectly | Provided by VCS | As explained for the concepts mappings, SiPL relies on the usage of an external VCS for managing variability in time. |
| Mapping *has* * Option | equivalent | Does map | Options are always Features |
| Feature *has* * Feature Revision | | Does not exist | Versioning (in time) on the level of Features is not supported. Only the global system revision is put under version control by an external VCS. |
| Constraint *has* * Feature Option | equivalent | Does map | Feature Options are always Features. |
| System Revision *defines* * Feature Option | indirectly | Provided by VCS. | As explained for the concepts mappings, SiPL relies on the usage of an external VCS for managing variability in time. Feature Options are always Features. |
| System Revision *enables* * Constraint | indirectly | Provided by VCS. | As explained for the concepts mappings, SiPL relies on the usage of an external VCS for managing variability in time. |
| Revision *has* * successor (Branching/Forking) and predecessor (Merging) | indirectly | Provided by VCS. | Since it is assumed that SiPL is integrated with a traditional VCS (Subversion, Git, ..), it adopts the typical branching structure of these VCSs, which corresponds to that of the conceptual model. |

| | |
|---|---|
| Unmapped relations in tool | SiPL manages relations between Delta Modules in a so called Delta Module Graph. The nodes of this graph are Delta Modules, while the edges are so-called Delta Module Relations. In SiPL, there are four kinds of Delta Module Relations, namely Conflict, Dependency, Duplicate and Transient Effect Relations. A conflict between two delta modules means that, for the sake of product generation, the delta modules cannot be applied together. A dependency between two delta modules means that both delta modules can only be applied in a certain order. A duplicate relation indicates that two delta modules lead to equivalent effects when being applied during product generation. A transient effect relation means that one delta module removes the effect of another delta module during product generation. |
| Remarks | |

# 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.