PLiMoS, a DSML to Reify Semantics Relationships: An Application to Model-Based Product Lines

Stephen Creff, Joël Champeau

PLiMoS, a DSML to Reify Semantics Relationships: An Application to Model-Based Product Lines

Stephen Creff and Joël Champeau

Lab-STICC, ENSTA Bretagne, UEB - 2, Rue F. Verny 29806 Brest Cedex 9, France
{stephen.creff,joel.champeau}@ensta-bretagne.fr

Abstract. In the Model-Based Product Line Engineering (MBPLE) context, modularization and separation of concerns have been introduced to master the inherent complexity of current developments. With the aim to exploit efficiently the variabilities and commonalities in MBPLs, the challenge of management of dependencies becomes essential (e.g. hierarchical and variability decomposition, inter-dependencies between models). However, one may observe that, in existing approaches, relational information (i) is mixed with other concerns, and (ii) lacks semantics and abstraction level identification. To tackle this issue, we make explicit the relationships and their semantics, and separate the relational concern into a Domain Specific Modeling Language (DSML) called PLiMoS. Relationships are treated as first-class entities and qualified by operational semantics properties, organized into viewpoints to address distinct objectives, e.g. product derivation, variability consistency management, architectural organization. This paper provides a description of the PLiMoS relationships definition and its implementation in a model-based product line process using two variability languages: Feature Model and OVM. The independence with variability and core assets modeling languages provides benefits to cope with the product line maintenance.

Keywords: Model-Based Engineering, Product Lines, Dependencies, Semantics Relationships, DSML.

1 Introduction

Product Line Engineering (PLE) [1] aims at increasing product quality, enhancing time-to-market, and reducing production costs. Product Lines (PLs) promote intra-organizational reuse through the explicitly planned exploitation of similarities (*commonality*) among related products. A PL is a family of products built in a prescribed manner from a common set of core development assets [2]. The explicit modeling and management of the *variability* and *commonality* is a key point in PLE [2]. Several techniques, and thus Domain Specific Languages (DSLs), have been defined to explicitly model the variability. Some of them are defined as orthogonal approaches, dealing with the variability as a separated

viewpoint connected to core assets to automate the product derivation. The most popular, and historical one introduced by Kang *et al.* [3], is the Feature Model (FM), another one, also used later on in this paper, is the Orthogonal Variability Model (OVM) language [2].

In the Model-Based Product Line Engineering (MBPLE) context, clear modularization and separation of concerns have been introduced to master the inherent complexity of current developments. With the aim to exploit efficiently the variabilities and commonalities in model-based PLs, the challenge of management of dependencies becomes essential (e.g. hierarchical and variability decomposition, inter-dependencies between models). Efforts have been provided in existing variability modeling techniques to separate concerns, either by defining categories of features and providing classifications (e.g. [2, 4, 5]), or by defining intra-variability model relations (e.g. [4, 6]) and inter-model ones (e.g. [7, 5]). A survey on of the variability modeling approaches in the literature that handle explicit relationships between variable elements [8] shows that many authors faced the need to define and use specific relationships combined to variability languages. Relationships are defined for various purposes such as: Hierarchical refinement, Configuration, Specific Usage, and Inter-concerns management. Nevertheless, the relationships are often buried in the variability model and can lack semantics: their description can be quite vague and imprecise, especially when describing the specific usages. Up to now, existing approaches do not consider relational information as a global preoccupation in the MBPL. However, this information represent the core concepts necessary to improve the management of the MBPL and its evolution. For example to integrate extensions in the variability models, or to take into account evolution of the application models and modeled core assets of the PL. Therefore, efforts have to be provided to (i) clarify all the relationships, (ii) their semantics and (iii) their corresponding usages, in order to federate all the models of the MBPL modeling space.

To tackle this issue, we make explicit the relationships and their semantics, and separate the relational concern into a Domain Specific Modeling Language (DSML) called PLiMoS. Relationships are treated as first-class entities and qualified by operational semantics properties, organized into viewpoints to address distinct objectives, e.g. product derivation, variability consistency management, architectural organization. We consider PLiMos as an independent relational language to federate models of the MBPL, supporting dependency processings and semantic consistency checkings. The language is amalgamated with existing variability modeling languages to introduce model management capabilities, allowing interrelating heterogeneous variability languages. This paper provides a description of the PLiMoS language and an implementation in a model-based product line process using two variability languages (Feature Model, and OVM), and existing modeling core assets (models at different levels of abstraction). The independence with variability and core assets modeling languages provides benefits to cope with the product line maintenance and evolution.

The remainder of this paper is organized as follows. In the second part, as the root of our work, we give a brief overview of the background on relation-

ships in PLE, describing the lack of related approaches and our motivations. In the third section, we present and formalize the PLiMoS DSML. In the fourth section, we describe an application of the language to two variability modeling languages (FM and OVM) for illustration purpose. The fifth section outlines the implementation, providing details on the concrete syntax and tooling. The last section draws our conclusion and introduces future works.

2 Relationships in PLE

Modularization and Separation of Concerns (SoC) has been used as an effective solution to tackle the growing complexity of products and to enhance the understandability of software development. The variability models describe abstractions from core modeling artifacts related through variability mechanisms, and the relational information between these core artifacts is sometimes represented into the variability modeling viewpoint. Moreover, efforts have been made to refine variability and separate concerns, even though sometimes lacking some formalization in relating manipulated artifacts, e.g. FORM [4]. Besides, the modularization issue has been pointed out by many authors, e.g. [5, 9, 10]. A survey of the relationships in the variability models literature reveals some categories [8]:

- *Logical group relationships:* are found in any variability language and describe the implication/exclusion of variants from a given logical group (e.g. Or, And, Card).
- *Hierarchical refinement relationships:* where initially called *Consist-of* [3]. These co-implication relationships are symmetric and transitive, and are specialized, depending on the approaches, into *generalization*, *aggregation*, *decomposition*, and *classification*, e.g. [4, 6].
- *Configuration dependencies:* are mainly based on the two following constraints: *Implication* (dissymmetric and transitive), and *Exclusion* (symmetric but not transitive), e.g. [2–4, 6, 11, 12].
- *Usage specific relationships:* correspond to various viewpoints (e.g. *runtime* dependencies [6]), and delimits one concern.
- *Inter-concerns dependencies:* can be defined in a single variability model (e.g. [4, 6]), between variability models (e.g. [2, 5, 7, 9, 10]), or between variability models and core artifacts (e.g. [2, 13]).

The relationships lack semantics: their description can be quite vague and imprecise, especially when describing the specific usages, e.g. [4, 9]. Some approaches provide formalizations through metamodels (like in [12, 14]), or with mathematical formalisms (e.g. [5, 15]), but do not cover the entire range of relationships. For example, a relationship like *recommends* (e.g. [16]), which is a weak form of *requires* has no precise semantics and associated operations. Moreover, existing approaches introduce relationships buried in the variability model, every one extending variability languages with some relationships. Our approach differs in that we do not focus on variability but only on relationships between the variation point and variants, providing a generic language to describe specific

relationships adapted to a given PL. The relationships described are not limited to Boolean equivalent relations like in [5] and associated to consistency checking, but can be established for various purposes. Moreover, unlike other approaches, relations considered cover the modeling core assets space.

Besides, related to traceability approaches, a recent work [17] identifies four orthogonal traceability dimensions in PLE (refinement, similarity, time and variability) and provides an implementation of traceability mechanisms in a generic framework, which does not enforce the semantics. Some applications of the traceability domain have been made to PLE [18, 19], with the aim to enforce the relationships semantics. In comparison, our approach is more pragmatic and targets domain concerns first (variability modeling and the PLE process) to extract semantics relationships that can be a basis for classical traceability activities. As a result, our approach goes further in describing the relationships granularity and usage.

3 The PLiMoS DSML

In this section, to tackle the exposed issue, we make explicit the relationships and their semantics, and separate the relational concern into a Domain Specific Modeling Language called PLiMoS to model the relational information of the Product Line Modeling Space. The section introduces (i) the approach, and provides details on the PLiMoS (*specification*) generic metamodel, a base to generate the PLiMoS (*implementation*) specific metamodel.

3.1 Overview of the Approach

We define the semantics of the PLiMoS language on the basis of the abstract syntax. The DSML is implemented with Ecore (*eclipse modeling framework*) concepts and gathers various concerns. Figure 1 depicts the process associated to each concern; in the process, a PLiMoS$_{specification}$ metamodel provides facilities to build a specific part of the DSML (PLiMoS$_{implementation}$), dedicated to a particular modeling space, by generating it from a PLiMoS$_{specification}$ model. This paper provides a description of the PLiMoS language in this section and its application and implementation to a modeling space, using Feature Model and the OVM language in Section 4: due to space limitation, the model transformation (specification to implementation) stays out of the scope of the paper.

3.2 The PLiMoS Specification Language

This section introduces the PLiMoS$_{specification}$ metamodel. A separation of concerns is applied to reveal the basic relationships and the associated operational semantics. So, conceptually, the language is based on a set of *Basic Relationships* (BR), some *Semantics Interpretations* (SI), and functions to bind SI to BR.

$$PLiMoS_{specification} \triangleq \langle BR, SI, \Phi(SI) \rightarrow BR \rangle$$

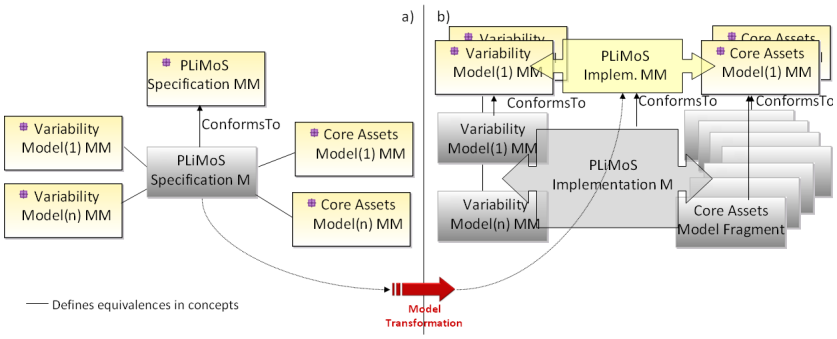

Fig. 1. Overview of the PLiMoS language

In traceability approaches, relations are very often deeply qualified by their causality (if not unidirectional), i.e. relationships usually have a source (the cause) and a target (the effect). With the aim to add genericity, we consider a *BasicRelationship* to be a mapping (symmetrical link) established between two sets of artifacts, i.e. with a Left Hand Side (LHS) and Right Hand Side (RHS). Figure 2 presents an excerpt of the metamodel. One may note that the *BasicRelationship* associate EObjects as mapping ends; this implementation details allows to associate the links in PLiMoS$_{specification}$ models to variability and *core assets* metamodel elements. Moreover, the *BasicRelationships* have by default multi-cardinality on both sides and could be constrained later to a specific range. These relationships can be defined to realize a structural infrastructure (serializable) or considered to be derivable (and therefore inferred on-demand for various activities, e.g. consistency checking, viewpoint picturing), volatile and transient (*derived* property in Fig. 2).

Furthermore, a relationship can be characterized by the abstraction level criterion (*process Level*). It can be either vertical if the relationships cross levels, or horizontal whether it does not. Another characteristic is the homogeneity/heterogeneity nature of the relationships: within the variability model, within core assets or between the two. No property is associated to such a characteristic in PLiMoS as it is inferred from *lhs* and *rhs* relations ends type.

Variability models describe abstractions of core modeling artifacts related through variability mechanisms. Any variability technique defines propositional connectors and associations among variable elements that are translated in the basic types: *implication* (noted ι, is non-symmetric but transitive), *co-implication* (noted I, is symmetric and transitive) and *exclusion* (noted η, is symmetric and non-transitive). For example, the FM language uses "consist-of" (co-implication between feature and sub-feature), and the Or operator (exclusion between sub-features), or other languages like OVM defines specific constraints equivalent to implication and exclusion (e.g. "requires$_{v_vp}$" or "excludes$_{vp_vp}$"). Some specific *rationales* and descriptions can refine the relationship to add semantics; these rationale properties are detailed in Section 3.3.

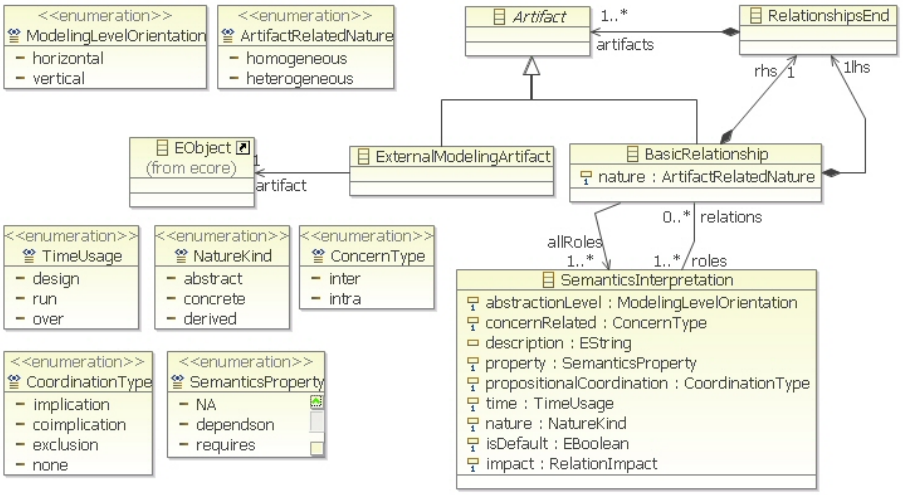

Fig. 2. Excerpt of the PLiMoS language

Every model syntactically correct and conforms to the metamodel must be interpreted as a definition for a dedicated modeling space relational language.

3.3 Defining Semantics Relationships

Relationships need semantics connected to specific usage and activities, the language contains some usual PL relationships.

Product Line Relationships Characteristics Based on the observation from the survey [8], the relationships in PL approaches can be classified by five main properties: **(1)** *Concern*: the relationships do or do not cross concerns (intra or inter concerns); **(2)** *Artifact Related*: the relationships link the same set of artifacts (homogeneous) or different sets (heterogeneous); **(3)** *Abstraction level*: the relationship is defined as vertical (crossing abstraction levels) or horizontal; **(4)** *Time*: the relationships is temporally related to (i) design time, being structural (established by logical groups or hierarchy) or dependency (basic configuration or usage specific ones), (ii) run-time (operational activities), or (iii) over-time (evolution concerned); **(5)** *Impact*: the relationships define (i) induction (implication, exclusion or activation), (ii) modification (interaction, substitution, deletion).

For example, the five main categories of relationships described in Section 2 are combinations of the previous properties:

- *Logical group relationships*: intra-concern, homogeneous, horizontal, structural design, co-implication;
- *Hierarchical refinement relationships*: intra-concern, homogeneous, vertical, structural design, co-implication;

- *Configuration dependencies*: intra-concern, homogeneous, horizontal, design dependencies, implication or inclusion impact;
- *Usage specific relationships*: intra-concern, homogeneous/heterogeneous, horizontal, dependencies, various impacts;
- *Inter-concerns dependencies*: inter-concern, homogeneous/heterogeneous, horizontal, dependencies, various impacts.

The PLiMoS language is designed to be generic, allowing the specialization of all existing relationships.

Semantic Properties in a Library PLiMoS$_{specification}$ includes predefined rationale properties to reveal the causality of the relationship, and provides semantics for understandability and process enactment. It includes the four hierarchical roles to specialize a *co-implication* links (viz. *generalization*, *aggregation*, *decomposition* and *classification*), necessary to connect PLiMoS to feature modeling languages.

Considering a given intention, the dependency relationships (ι, I, and η) can express specific causality, i.e. depending on the usage, as a base for activities like product derivation or consistency checking. The rational property for constraints may offer benefits to various usages, and PLiMoS includes the properties listed in the following. For *implication* and *co-implication* relations: NA (Not Applicable by default, whether the causality is not informed), depends-on, requires, resource, includes (implication only), impacts, satisfies, uses, allocate-to, induces, design, core, implemented-by. For *exclusion*: incompatible, mutex-with, and conflict. The three relationships rational property can be set to "other" and can be filled with the "Description" property. As built upon basic relationships of implication and exclusion, a translation of the implementation language to the propositional logic technological space is eased.

Some of the properties are illustrated and described with more details in Section 4. One may note that the PLiMoS language does not consider, up to now, runtime activation dependencies, like those defined in [6].

Relationships are treated as first-class entities and qualified by operational semantics properties. The relationships typed and qualified by their rationales are specified for a given concern; the latter define viewpoints (cf. graphical filters in the Section 5). Typed relationships can be organized into viewpoints to address distinct objectives, e.g. product derivation, variability consistency management, architectural organization. For example, many constraints have to be checked and verified for product derivation: typed "resource" relationships, allocation and implementation, and so forth; on the other side, only operational relationships have to be used to perform a run-time simulation.

The semantics relationships layers provide semantic enrichment of variability modeling languages and consequently of the entire PL.

4 Application to Two Variability Modeling Languages

This section illustrates the adaptation (generative realization of the PLiMoS$_{implementation}$ part) of our DSML to a given modeling process introduced by Metzger *et al.* [5]. Briefly, the variability space is modeled with two kinds of variability models, for i) the PL variability, ii) the software design solution. The core assets space, which is composed of Model-Based Engineering (MBE) artifacts, is represented at two levels of abstraction: problem specification (e.g. with Use-Cases), solution design (e.g. Component, class diagrams), as depicted in Fig. 3. In the following, for representation convenience, the proposed formalisation does not reflect the separation between the relational structure and the semantics interpretations (i.e. the main interpretation is used to define the relation).

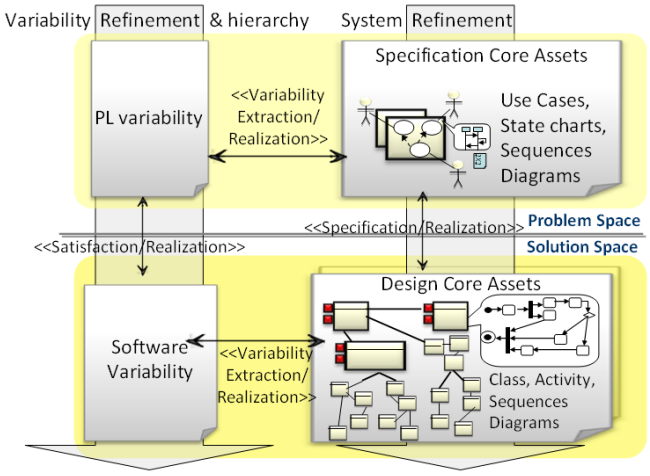

Fig. 3. Applicative Modeling Space Overview

4.1 Introducing the Variability Languages

The variability is modeled with two types: FM for PL variability, and OVM for the Software one. This section describes the two languages used. Every model that is syntactically correct and conforms to the metamodel must be interpreted as a valid variability model (FM or OVM).

Feature Model Starting from the work performed by Schobbens *et al.* [20] and the FFD pivot, we derive a metamodel for feature models in Ecore with OCL constraints. A Feature Model $fm \in \mathcal{L}_{fm}$ is represented in the article as a 8-tuple $\langle F, P, C, E, r, \lambda, CE, \Phi \rangle$ where:

- F is non-empty and finite set of Features;

- $P \subseteq F$ is a set of Primitive features (i.e. leaves);
- $C \subseteq F$ is a set of Concrete features (i.e. non abstract, connected to other artifacts, to be consistent $P \subseteq C$);
- $E \subseteq F \times F$ is a finite set of Edges. As FMs are directed, features $f_1, f_2 \in F, (f_1, f_2) \in E$ will be noted $f_1 \rightarrow f_2$ where f_1 is the *parent* and f_2 the *child*. E is acyclic: $\nexists f_1, \ldots, f_k \in F. f_1 \rightarrow .. \rightarrow f_k \rightarrow f_1$. E is a tree: $\nexists f_1, f_2, f_3 \in F. f_1 \rightarrow f_2 \wedge f_3 \rightarrow f_2$;
- $r \in F$ is the root of the FM, r is unique and has no parent $(\forall f \in F.(\nexists f' \in F. f' \rightarrow f) \iff f = r)$;
- the function $\lambda : F \rightarrow O$ labels each node with an operator - O - from $(And \cup Or \cup Xor \cup \{Opt_1\})$;
- $CE \subseteq F \times GCT \times F$ is the set of intra-model constraint edges, GCT - Graphical Constraint Type - is {implication, co-implication, exclusion}, relations described hereinafter;
- A set $\Phi \in \mathbb{B}(F)$ of textual constraints in propositional formulae;

Orthogonal Variability Model From the description by Pohl *et al.* [2], we formalize the OVM DSML, considering that an OVM model Ω is a 7-tuple $\langle VP, V, \lambda_{VP}, DE_{VP}, Opt, Parent, CD \rangle$, such that:

- $VP (\neq \emptyset)$ is the non-empty set of Variation Points (VP);
- $V (\neq \emptyset)$ is the non-empty set of Variants; $VP \cap V = \emptyset$;
- the function $\lambda_{VP} : VP \rightarrow \mathbb{N} \times \mathbb{N}$ labels each variation point with the possible variant cardinality;
- $DE_{VP} \subseteq VP \times V$ is the finite set of decomposition relations between a VP and its Variants.
- $Opt : V \rightarrow \mathbb{B}$ provides insight on a variant optionality;
- $Parent : V \cup VG \rightarrow VP$ is the function that returns the parent Variation Point of a given Variant;
- $CD \subseteq (VP \times CT_{OVM} \times VP) \cup (VP \times CT_{OVM} \times V) \cup (V \times CT_{OVM} \times V)$, with CT_{OVM} ={implication, exclusion} represents the set of configuration constraints;

Some additional constraints are defined to ensure that: each VP is a parent of at least one V, a V has only one parent, cardinalities are correct.

4.2 The PLiMoS Implementation Metamodel

PLiMoS$_{implementation}$ makes an overlap with variability languages and core assets ones (cf. Fig. 1). PLiMoS$_{implementation}$ is generated from a specification given by a PLiMoS$_{specification}$ model. We consider in the paper, the PLiMoS$_{implementation}$ specific language to be composed of a set of *FM relationships* (FMR), a set of OVM ones ($OVMR$), and a set of inter-model relationships (IMR): $PLiMoS_{implementation} \triangleq \langle FMR, OVMR, IMR \rangle$.

Relationships Specialized for the VMs Feature Model relationships (FMR) are defined as a tuple $\langle E, IDC, CIDC, EDC\rangle$:

- *Consist-of* (χ): are co-implication and 1-to-1 relationships, homogeneous, non-derived, and vertical. This basic hierarchical relation is refined giving an intention, e.g. part-of, generalization ($E \subseteq F \times \{\text{generalization, decomposition}\} \times F$);
- *Implication* (ι): the "requires" is a non-symmetric but transitive configuration constraint ($IDC \subseteq F \times F$, $IDC \subseteq CE$); The *CoImplication* (I) is a symmetric ι ($CIDC \subseteq F \times F$, $CIDC \subseteq CE$); these relationships are 1-to-1, homogenous, non-derived and horizontal;
- *Exclusion* (η): is a symmetric and non-transitive "mutex" dependency constraint ($EDC \subseteq F \times F$, $EDC \subseteq CE$); this relationship is 1-to-1, homogenous, non derived and horizontal;

And obviously, $E \cap IDC \cap CIDC \cap EDC = \emptyset$. Considering a given intention, the dependency relationships (ι, I, η) may express specific causality, i.e. depending on the usage, as a base for activities like product derivation or consistency checking. In the running example we do not define any specific usage.

In order to make the connection with the OVM language, some relationships need to be specified in the PLiMoS$_{specification}$ model, those ones are:
$OVMR \triangleq \langle DE_{VP}, AD, CD\rangle$.

- *variability dependencies* (DE_{VP} as define before) between VP and V (mandatory and optional), are co-implication and 1-to-1 relationships, homogenous, non-derived, and vertical;
- *artefact dependencies* (AD) and *VP artefact dependencies*, as implication and 1-to-1 relationships, heterogeneous, non-derived, and horizontal (part of the inter-model relationships);
- *constraints dependencies*: (CD, as defined before) "requires_V_V", "requires_V_VP", "requires_VP_VP", 'excludes_V_V", "excludes_V_VP", "excludes_VP_VP". Consecutively, implication and exclusion relationships, 1-to-1, homogenous, non-derived and moreover, both horizontal and vertical ones are needed (implied by the modeling space - cf. previous section)

Relationships Specialized to Describe Inter-model dependencies The PLiMoS$_{specification}$ model is also concerned with specific modeling space considerations (depicted in Fig. 3), to build operational relationships. In this case, the inter-model Relationship Model rm is a tuple $\langle SR, IR, RR\rangle$ where:

- $SR \subseteq \mathcal{P}(C) \times \mathcal{P}(VP)$ is a finite set of *Satisfaction* (ζ) relationships. The marketing Features from the PLVM are satisfied by technical VPs from the SVM: the design solution variability models meet the expectations and needs of the PL;
- $IR \subseteq (VP \times VP) \cup (V \times V)$ is a finite set of *Identity* (Ξ) relationships. The identity mapping between two features means the similarity wrt. variability and concepts; therefore, associated core assets, whether different, must be totally disjoint; *Identity* is used to separate concerns at the same level of abstraction (horizontal relationships);

– $RR \subseteq (C \times CA) \cup (V \times CA) \cup (VP \times CA)$ is the set of *Realization* (P) relationships. A variant can be realized by a "Core Assets" (CA) entity reified in the metamodel (CA are specialized for UML or any other DSLs involved). *Refinement* (ρ) is the relationship that describes technological refinement between abstraction levels in a model-based process. A given variant can be mapped and realized by multiple core assets; however, no n-ary relationships are defined, the CA concept represents a set of assets (Γ relationships make the links between the set entity and the assets);

Additional relationships could be introduced as derived, volatile and transient. In the MDPL context, variability models relate elements with variability operators; in a complex modeling space, these relations can be made explicit to help enforce the semantic cohesiveness of the sets of related models.

5 Implementation

In this section we highlight tooling concerned with Domain Engineering: Feature Model, OVM and relationships representation, giving a concrete syntax to our infrastructure. One common end user request is to have an integrated, end-to-end tool support, instead of having different tools for closely related problems. Existing tools are based on Eclipse and Obeo Designer[1] (commercial Model-Based Tool which provides graphical multi-viewpoint support within an Eclipse integrated environment). A viewpoint is defined for variability modeling (for FM and OVM), another for relationship modeling, both based on the previously introduced metamodels. The choice of a graphical syntax was made to be consistent with the selected variability models (FM and OVM are commonly manipulated through a graphical syntax). This section introduces a running example and provides insights on the tooling.

5.1 Introducing the Running Example

The running example is simple and abstracts, in some points, real systems at Thales. Anti-aircraft systems communicate with sensors and actuators e.g., radars and consoles (human machine interfaces). In the following we will use two radars: Radar1 which sends speed and position, Radar2 which sends two positions (*t*, *t+1*). Treatments are position processing and optionally speed processing (whether needed). Resulting data are sent to the user via consoles: Console1 gives polar coordinate, Console2 cartesian ones. Variability comes from a great extent from external actors like in our applications.

Describing the PL variability The Product Line Variability Model (PLVM) (cf. Fig. 3, and Fig. 4-a) can be seen as an interface, as it contains the user-visible variability, in the problem space. The model encompasses variabilities seen from a marketing perspective. This VM is specific to the missions of the family. This

[1] http://obeo.fr/pages/obeo-designer/en

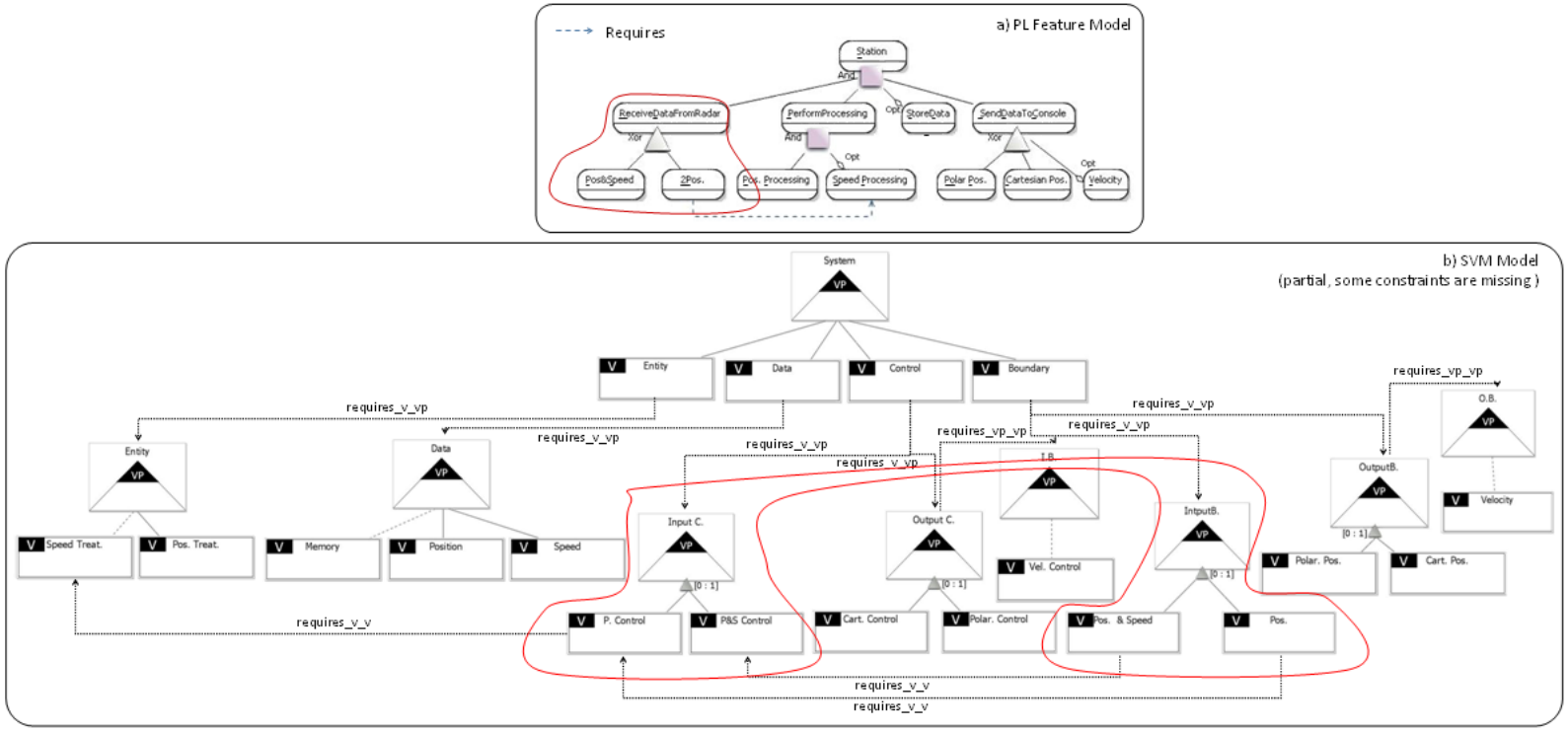

Fig. 4. Variability Models: air defense case study

model containing the PL variability does not, per se in its hierarchy, reveal design constraints of the solution - technical and architectural.

Fig. 4-a represents our running example: a simple control system family; variability visible for the market is limited to: type of data to treat (delivered by the external radars), type of processing consequently needed, presence/absence of internal memory and type of data processed and delivered. The variability comes here, for a part, from external actors, e.g. connecting a new radar with different properties; a characteristic of real applications.

Describing the Software variability The Software Variability Model (SVM), expressed in the OVM language, contains the internal variability of the architecture and design. The OVM model is organized upon architectural concerns and related to MBE solution artifacts (Component, class diagrams, etc.).

Note, in the simple example Fig. 4-b, that the architectural decomposition appears, e.g. Fig. 4-b pictures, from left to right, *Entities* (treatment rules entities and their relationships), *Data* (data management, including persistence), *Controls* (interface data consistency and translation into core data) and *Boundaries* (interface rules) features which come from a logical pattern application. Given a domain thesaurus, features and variants can have different names between the PLVM and the SVM, the tree hierarchy can be modified and variation points can be duplicated (e.g. red shape in Fig. 4 part b that corresponds to the one in part a); co-implication constraints may appear to keep the VM consistent. All the requires constraints are not represented in the figure for readability reason.

Overview of the inter-model relationships Product management for a PL comprises the positioning of the PL, seen in light of the interplay between the technical view of a product line (common technical platform, SVM) and the marketing view of a PL (products targeting a similar market, not necessarily technically related, PLVM). Figure 5 illustrates some Satisfaction and Realization relationships: The "HMIData" feature is satisfied by the "O.B." variation point, the "O.B." Variation Point is identical to "OutputB.", and the "Vel" variant is realized by a core asset.

5.2 Variability Models Designer Viewpoints

Over the "classical" FM representation, specific Obeo Designer filters are built for relationship representations: i) features can show/hide satisfaction link presence (Example Fig. 6 a), the "2 Pos." satisfied feature is colored differently) and represent technical constraints into an optional layer; ii) features can show/hide realization and link presence. It provides stakeholders an initial understanding of the relations among features and assets.

The same kind of filters are defined for the OVM viewpoint to show/hide realization relationships.

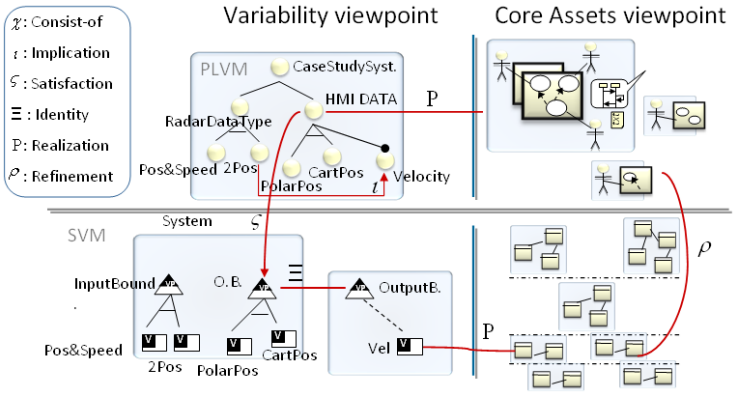

Fig. 5. Some inter-model relationships: air defense case study

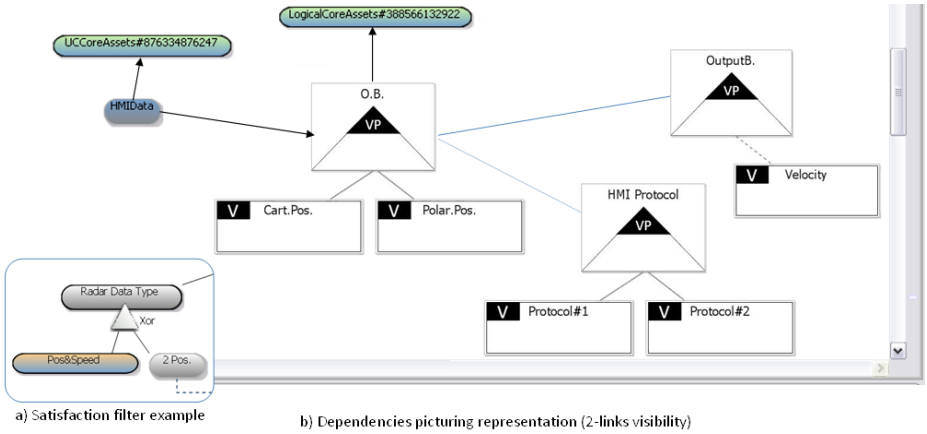

Fig. 6. Variability modeling and dependencies

5.3 Dependency Designer Viewpoint

The dark side of features is feature interaction, which is implicit in feature composition and therefore difficult to understand. We define the dependency viewpoint to get the big picture of the overall dependencies. One could choose an epicenter element; dependencies will be extracted, or computed whether necessary, and shown (we define the operator: *extract_relationships(element,range)*). For example, in Fig. 6 b) the focus is made on the element "O.B.", with a limit of two links visibility. "HMIData" is part of the PLVM; "O.B.", "OutputB." and "HMI Protocol" are elements of the SVM. The set of required features (of a particular feature) - *required_features(feature)* operator - is given by a transitive closure operator.

Graphical filters on relationship types are provided to focus on one specific concern, e.g., away from our example, to only show *resource* relationships (expressing resource dependencies).

6 Conclusions and Future Works

This article describes a Domain Specific Modeling Language called PLiMoS. This language does not focus on variability operators but only on relational knowledge. We propose to consider relationships between elements as an independent language to be merged with any variability modeling language to introduce model management capabilities. Relationships are treated as first-class entities and qualified by operational semantics properties, organized into viewpoints to address distinct objectives, e.g. product derivation, variability consistency management, architectural organization. We aim at providing a solution that allows the engineers to focus on their domain and reason about the relationships between the elements of the different spaces, abstraction levels and concerns.

The approach enforces the semantics of the relationships and provides benefits to cope with heterogeneous variability languages focusing on relations between concepts (the paper describes the amalgamation with two languages, a FM and OVM). The independence with variability and core assets modeling languages provides benefits to cope with the product line maintenance and evolution.

Future work will focus on adding automation to the tooling design process in relation with the PLiMoS relationships (design model generation), within a model-based process. Works will also investigate the connection to other technological spaces to perform various activities, but mainly for behavioral composition, and simulation for operational consistency checking.

References

1. Clements, P.C., Northrop, L.: Software Product Lines: Practices and Patterns. SEI Series in Software Engineering. Addison-Wesley (August 2001)
2. Pohl, K., Böckle, G., Linden, F.J.v.d.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer NY, Inc., Secaucus, NJ, USA (2005)

3. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (foda) feasibility study. Technical report (1990)
4. Kang, K.C., Kim, S., Lee, J., Kim, K.: Form: A feature-oriented reuse method with domain-specific reference architectures. Ann. Softw. Eng. **5** (1998)
5. Metzger, A., Pohl, K., Heymans, P., Schobbens, P.Y., Saval, G.: Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis. Requirements Engineering Conference **0** (2007) 243–253
6. Lee, K., Kang, K.C.: Feature dependency analysis for product line component design. In: Lecture Notes in Computer Science, Springer (2004) 69–85
7. Sinnema, M., Deelstra, S., Nijhuis, J., Bosch, J.: Covamof: A framework for modeling variability in software product families. In Nord, R.L., ed.: SPLC. Volume 3154 of LNCS., Springer (2004) 197–213
8. Creff, S., Champeau, J.: Relationships in variability modeling approaches: A survey and classification. In: Proceedings of Journée Lignes de Produits '12, Lille, France (2012)
9. Reiser, M.O., Weber, M.: Multi-level feature trees: A pragmatic approach to managing highly complex product families. Requir. Eng. **12**(2) (May 2007) 57–75
10. Hartmann, H., Trew, T., Matsinger, A.: Supplier independent feature modelling. SPLC '09, Pittsburgh, PA, USA, Carnegie Mellon University (2009) 191–200
11. Ye, H., Liu, H.: Approach to modelling feature variability and dependencies in software product lines. Software, IEE Proceedings - (2005)
12. Mei, H., Zhang, W., Zhao, H.: A metamodel for modeling system features and their refinement, constraint and interaction relationships. Software and Systems Modeling **5** (2006) 172–186
13. Heidenreich, F., Kopcsek, J., Wende, C.: FeatureMapper: Mapping Features to Models. In: Companion Proceedings of ICSE'08, ACM (2008)
14. Fey, D., Fajta, R., Boros, A.: Feature modeling: A meta-model to enhance usability and usefulness. In: Proceedings of the Second International Conference on Software Product Lines. SPLC 2, London, UK, UK, Springer-Verlag (2002) 198–216
15. Zhu, C., Lee, Y., Zhao, W., Zhang, J.: A feature oriented approach to mapping from domain requirements to product line architecture. In Arabnia, H.R., Reza, H., eds.: Software Engineering Research and Practice, CSREA Press (2006) 219–225
16. Riebisch, M., Streitferdt, D., Pashov, I.: Modeling variability for object-oriented product lines. In: ECOOP 2003 Workshop Reader. Volume 3013 of LNCS. Springer Heidelberg (2004) 165–178
17. Anquetil, N., Kulesza, U., Mitschke, R., Moreira, A., Royer, J.C., Rummler, A., Sousa, A.: A model-driven traceability framework for software product lines. SoSyM **9** (2010) 427–451
18. Jirapanthong, W., Zisman, A.: Xtraque: traceability for product line systems. Software and Systems Modeling **8** (2009) 117–144 10.1007/s10270-007-0066-8.
19. Lamb, L.C., Jirapanthong, W., Zisman, A.: Formalizing traceability relations for product lines. In: TEFSE '11 Proceedings, New York, NY, USA, ACM (2011) 42–45
20. Schobbens, P.Y., Heymans, P., Trigaux, J.C., Bontemps, Y.: Generic semantics of feature diagrams. Comput. Netw. **51**(2) (2007) 456–479