Feature Diagrams: a Formalization and extensible Meta-model Proposals

(GIRO Technical Report 2009/02-v2.0 2009-07-05)

Miguel A. Laguna, José M. Marqués

Department of Computer Science, University of Valladolid,
Campus M. Delibes, 47011 Valladolid, Spain
{mlaguna, jmmc}@infor.uva.es

Abstract: Feature models are used to represent the variability and commonality of software product lines, and permit the configuration of specific applications. However, universally accepted definitions of feature and feature diagrams are missing. This paper proposes the use of hypergraphs to integrate the different versions of these concepts in an extensible characterization. The definition, validation and selection of feature configurations are based on hypergraph properties and existing algorithms. Once the formalism is stated, the definition of a feature meta-model is straightforward and a set of modeling tools, compatible with the different flavors of feature diagrams, can be readily built. Finally, configuration and transformation of feature diagrams into UML models are defined as algebraic and QVT transformations.

Keywords: Feature model, hypergraph, feature meta-model, CASE tool

1 Introduction

Software product lines constitute a successful reuse paradigm in industrial environments in spite of their complexity [3]. Feature models represent the variability and commonality of software product lines and permit the configuration of each specific application to be selected. However, a universally accepted definition of feature and feature diagram (FD) is missing and many of the variants more frequently used cannot solve some problems. The original proposal, included in the Feature Oriented Domain Analysis (FODA) method [12], defines features as the nodes of a tree, related by various types of edges. The tree root or concept is decomposed by AND, X-OR and OPTIONAL relationships (see the example of Figure 1a), but FODA does not cover disjunction (OR).

Several extensions have been proposed: incorporating the OR decomposition (Figure 1b) [11], changing the visual syntax, or using directed acyclic graphs (DAG) instead of simple trees. However, in spite of using graphs, the situation of Figure 1c is not possible when the multiplicity is a property of the feature. Riebisch et al. [17], for example, proposes moving the multiplicity constraint to arcs instead of nodes. Constraints between features (a feature requires another feature or two features are mutually exclusive) can be added in textual or graphical formats. Schobbens *et al.*

[18] have evaluated the diverse variant of FDs, clarifying the differences and establishing a generic semantics. The study classifies the existing proposals using several characteristics: the FD is a tree or a DAG, the constraints are textually or graphically shown, and the way the decomposition relationships (AND, X-OR, OR, multiplicity) are expressed. They propose a new non-redundant variant FD or VFD.

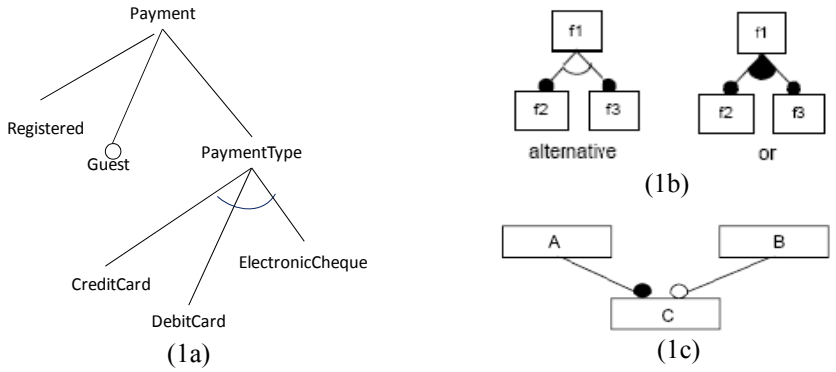

Fig. 1 An original FODA feature diagram, alternative and OR decomposition extensions and a problem that feature multiplicity cannot solve

Considering these antecedents, we propose the use of directed hypergraphs as a formal structure to define an FD instead of trees or simple graphs. The main reason is to substitute the several types of decomposition (Mandatory/Optional and Alternative/Or groups) and features (Solitary and Grouped Features) with only two elements: features (nodes) and generic decomposition (labeled hyperarcs). An additional advantage is that *requires* or *mutex* constraints can be reformulated using hyperarcs as well, contrary to the current proposals. Most authors (see [7] for example) deal with the structural constraints implicit in the features tree (or graph) independently from the additional *mutex*/*requires* constraints. The configuration, therefore, has to solve the problems in two stages. Using hypergraphs, the product configuration problem can be reduced to detecting connections and/or hyperconnections in the hypergraph.

The rest of the report is as follows: the next Section introduces hypergraphs and formally defines the structure underlying an FD. Section 3 analyzes the configuration problem and sketches the hypergraph algorithms used for deriving the configuration, starting from a set of features selected by the user. Section 4 uses the formal definition to build an extensible meta-model. In Section 5 a previously defined transformation into UML models is adapted to the proposed meta-model, generalizing tree structures to acyclic graphs. Finally, Section 6 presents related work and Section 7 concludes the paper and considers future work.

2 Hypergraphs and Feature Diagrams

A hypergraph is a generalization of a graph wherein edges can connect more than two vertices and are called hyperedges. Directed hypergraphs extend directed graphs, and

have been used as a modeling and algorithmic tool in many areas: formal languages, relational databases, manufacturing systems, public transportation systems, etc [9]. A technical, as well as historical, introduction to directed hypergraphs has been given by Gallo et al. [9]. The main reason for introducing this type of graphs is to represent Many-to-One relations, for which simple DAG or trees are not well equipped.

A *directed* hypergrah or simply hypergraph is a pair H = (V, E), where

- V = $\{v_1, v_2,..., v_n\}$ is the set of *nodes*
- E = $\{ e_1, e_2,..., e_m \}$, with $e_i \subseteq$V for i=1,…, m, is the set of *hyperarcs*

Where a *hyperarc* is an ordered pair, e = (t(e),h(e)), with t(e) ∩ h(e) = ∅. t(e) is the *tail* of e, while h(e) is its *head*. A Forward hyperarc, or simply *F-arc*, is a hyperarc e = (t(e),h(e)) with |t(e)|=1. An *F-graph* (or F-hypergraph) is a hypergraph whose hyperarcs are F-arcs, that is, all the hyperarcs have only a node as their tail.

2.1 Feature Diagrams as Hypergraphs

A Feature Diagram can be modeled as a directed hypergraph, where the features are the set of the nodes and there is a hyperarc for each decomposition relationship between features. Each hyperarc is assigned a label which corresponds to the multiplicity of the decomposition. The obtained hypergraph is an acyclic labeled F-graph. Each feature decomposition is mapped to an F-arc in the following way:

- mandatory features to hyperarcs where |h(e)|=1, and label 1..1
- optional features to hyperarcs where |h(e)|=1 and label 0..1
- pure alternative (X-OR) features to hyperarcs where |h(e)|=q, with q>1 and label 1..1
- OR features to hyperarcs where |h(e)|=q, with q>1 and label 1..q

The first two situations are the original mandatory and optional relationships. The third is a pure alternative situation and the last one is the generic OR. Therefore, all the possible decomposition variants [18] are covered. To facilitate the validation and later configuration of the FD, constraint relationships are considered formally as additional hyperarcs. The semantics of *requires* is that the *A requires B* constraint establishes a compulsory relationship between features *A* and *B*, i.e. a hyperarc between *A* and *B* with label 1..1. In fact, we could generalize to *A requires S, S* being a set of features. The multiplicity minimum and maximum of the hyperarc would be equal, in both cases, to the cardinality of *S*. We would like to point out that introducing this kind of arc (hyperarc) cycles may arise in the hypergraph. This is an undesired and nonsensical situation; therefore, once all the *requires* hyperarcs have been defined, the acyclicity of the hypergraph could be tested with the F-Acyclic procedure described in [8].

The semantics of *mutex* is that we cannot select simultaneously more than one of the two or more features involved. Then, independently of the implicit constraints imposed by the graph structure, a new relationship is imposed. This can be reinterpreted as a hyperarc from a common node (the root) to the involved nodes with 0..1 multiplicity: at most one of the features involved can be selected.

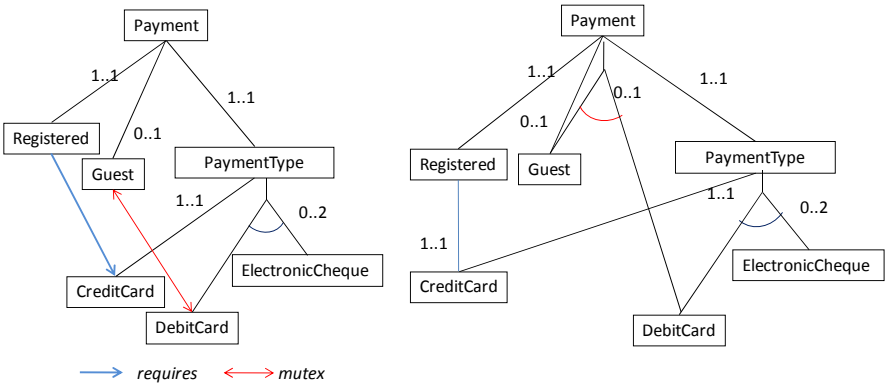

Fig. 2 Graphical constraints reinterpreted as 1..1(requires) and 0..1 (mutex) hyperarcs

Figure 2 is an attempt to graphically express these reinterpretations. While the left-hand side of Figure 2 is visually more expressive and tools can continue using that representation, the (hidden) hypergraph model (the right-hand side of Figure 2) is more easily handled by the validation/configuration algorithms.

2.2 Formal definition

A multiplicity value mv is a pair of integers mv=(min,max) with min ≥0, max>0 and min≤max. We denote by M the set of all possible multiplicity values, $M \subset \mathcal{N} \times \mathcal{N}^*$.

A Feature Diagram is an *acyclic F-hypergraph* F= (N, E, r, δ) where

- N is its set of nodes (or features)
- E={e_1, e_2,..., e_m}, with $e_i \subseteq N$ for i =1,…, m, is the set of *decomposition F-arcs*; q is the cardinality of the head of e_i q=|h(e_i)|.
- r ∈N is the root of the diagram (it is the only node not contained in the head of any hyperarc of the hypergraph, i.e it is the only node whose *Backward Star* is ∅ [8]): R is the root set of the hypergraph, $R \subseteq N \wedge R=\{r\}$. BS(r)=∅ ∧ BS(n) ≠ ∅ ∀ n ∈ N \ R
- δ: E→M assigns each F-arc *e* with a multiplicity (min, max), max ≤ q=|h(*e*)|

A particular type of Feature Diagram is the Feature Tree. If each node has no more than one parent, then the generic graph structure is a hypertree:

- A *Feature Tree* FT is a Feature Diagram, such that each node has at most one entering hyperarc (root r has none): |BS(n)| = 1 ∀ n ∈N. n ≠ r

Two extensions are possible. We introduce separately typed features and constraints, but the integration of the two definitions is straightforward. Constraints are introduced as additional hyperarcs with multiplicity 1..n (requires) or 0..1 (mutex).

Given a Feature Diagram F=(N,E,r,δ), a *Constrained Feature Diagram* is a Feature Diagram $C_{F=}$ (N, E', r, δ') where

- $E' = E \cup E_r \cup E_m$ and $\delta' = \delta \cup \delta_r \cup \delta_m$

- $E_r = \{r_1, r_2, ..., r_k\}$, with $r_i \subseteq N$ for $i = 1, \ldots, k$, is the set of *requires constraints F-arcs*; in general $|h(r_i)| \geq 1$
- $\delta_r: E_r \rightarrow M$ assigns each F-arc r with a fixed multiplicity min = max = $|h(r)|$. In particular, if the requires constraint involves two nodes, multiplicity is 1..1.
- $E_m = \{m_1, m_2, ..., m_l\}$, with $m_i \subseteq N$ for $i = 1, \ldots, l$, is the set of *mutex constraints F-arcs*; in general $|h(m_i)| \geq 2$ and $t(m_i) = r$ is the root.
- $\delta_m: E_m \rightarrow M$ assigns each F-arc m with a fixed multiplicity 0..1.

Finally, typed features allow a type (default type is NONE, others are classical predefined types such as INTEGER, BOOLEAN, STRING, etc.) to be assigned to each leaf (any feature not contained in the tail of any hyperarc, i.e. whose *Forward Star* is $\varnothing$ [8]). As there is no consensus about this concept in the literature, we treat it as an optional extension.

Given a Feature Diagram $F=(N,E,r,\delta)$, a *Typed Feature Diagram* is a Feature Diagram $T_{F=}(N, E, P, r, \delta, \tau)$ where:

- P is a set of types P={INTEGER, REAL, BOOLEAN, STRING, NONE}
- $\tau: L \rightarrow P$ assigns each leaf with a type value, $L \subset N \wedge |FS(l)| = 0 \; \forall \; l \in L$.

2.3 Discussion

To show the equivalence with previous FD definitions [18], a simple transformation of our Feature Diagram into a DAG can be considered: a) each hyperarc with h(e)=1 is transformed into an AND or OPTIONAL decomposition (the 1..1 or 0..1 multiplicity is assigned to the child feature); and b) each hyperarc with h(e) > 1 is transformed into a Grouped decomposition, connecting the parent and child nodes (the original multiplicity is assigned to the decomposition). The result is a constraintless DAG with multiplicity expressions or VFD, using Schobbens terminology [18]. As in [18], VFD is proved to be powerful enough to represent the rest of the FD variants, and our definition can equally generate any FD variant. As pointed out by Schobbens, expressiveness, succinctness and non-redundancy are key points. Most variants of Feature Diagram have two ways of expressing multiplicity: group multiplicity and feature multiplicity. For instance, feature multiplicity (an optional 0..1 or mandatory 1..1 feature) could be combined with an OR group, making it clear that some of the grouped features are always selected and the others are purely optional (see CreditCard in Figure 3a). This has a meaning: the group semantic indicates that the (in fact) mandatory feature is closely related to the rest of the optional features. The problem is that this possibility opens the door to unnecessary redundancies, or inconsistencies, allowing situations like *Registered* (optional as feature, mandatory as decomposition) in Figure 3a. The normalization of the diagrams, using only the decomposition based multiplicity is preferred, as inconsistency and redundancy are impossible, maintaining sufficient expressivity.

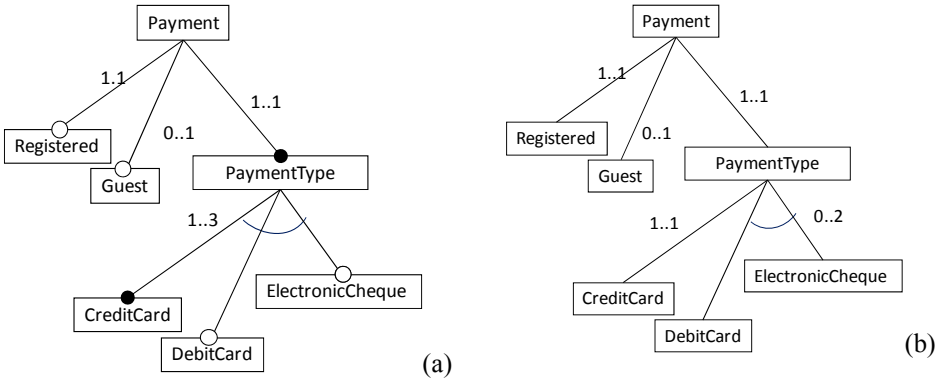

Fig. 3 Redundant and normalized versions of the same situation in an e-commerce product line

3 Configuration of a Feature Diagram

A (partial) configuration of a Feature Diagram is a sub-set of the original Feature Diagram where the variability is (partially) removed. In general, a manual process of node selection is carried out, obeying the constraints expressed in the Diagram. Some of these constraints are implicitly imposed by the diagram structure. Defining *mandatory (non-mandatory,* respectively*) decompositions* as decompositions where the minimum multiplicity is equal (less, respectively) than the number of its children, the following rules apply:

Rule 1. The root feature and all the features connected with the root feature through mandatory decompositions are intrinsically present in any configuration.

Rule 2. A feature connected with a selected feature through mandatory decompositions must be selected.

Rule 3. A non-mandatory feature can be selected only if at least one of its parents is selected.

Rule 4. A non-mandatory feature is selected if any of its descendants has been explicitly selected and it has not been selected through an alternative path. For example, the descendant has been directly selected by the user.

Rule 5. If a feature is present, the final number k of features selected as children of its decomposition must be between the minimum and maximum of the original hyperarc multiplicity: min $\leq$ k $\leq$ max. (Clearly, the number of children of the decomposition must be identical to the minimum and maximum of the hyperarc multiplicity for all the hyperarcs present in the configuration, i. e., no multiplicity is needed for the configuration hyperarcs.)

Other groups of constraints, when used (in a Constrained Feature Diagram), are imposed by the *requires* and *mutex* relationships:

- *Requires* constraints mean that, for each feature in the configuration, all the elements required by it must also be present. In the hypergraph representation, this is equivalent to a mandatory decomposition (rules 1 and 2 apply).

- *Mutex* constraints over a set of features mean that, if an involved feature is present in the configuration, the others must be absent. In the hypergraph representation, this is equivalent to a non-mandatory decomposition with maximum multiplicity equal to 1 (rules 3 and 4 apply).

Consequently, the configuration procedure can be applied uniformly to the constrained hypergraph, instead of dividing it into two phases or transforming the feature tree (or graph) into a set of propositional formulas, as proposed in the literature [15].

3.1 Formal Definition of Valid Configuration

A *Valid Configuration* $G = (N_G, E_G, r)$ is a sub-hypergraph of a (Constrained) Feature Diagram $F= (N, E, r, \delta)$ where

- N_G is a subset of nodes of N: $N_G \subseteq N$
- E_G is a set of hyperarcs: $E_G = \{e_G \mid \exists e \in E \wedge t(e_G) = t(e) \wedge h(e_G) \subseteq h(e)\}$
- The root is present: $r \in N_G$
- All the features connected with the root feature through mandatory decompositions are intrinsically present in any configuration. These are the *core* features of the product line and are always present. Any feature connected with a selected feature through mandatory decompositions must be selected (forward paths):

  $$\forall e \in E\ .\ (t(e) \subset N_G \wedge |h(e)|=\min) \Rightarrow h(e) \subset N_G$$

- All the paths P_{rn} in h from r to each feature in N_G are included in the configuration c (backward paths). This implies that a parent feature must always be present if a non-mandatory feature is selected:

  $$\forall n \in N_G\ e_j \in E\ \forall e_j \in P_{rn} \Rightarrow e_j \in E_G$$

  $$\forall n \in N_G\ n_j \in N\ \forall n_j \in P_{rn} \Rightarrow n_j \in N_G$$

  (Alternatively $\forall n \in N_G\ P_{rn} \subset c$)

- Denoting $h'(e)$ as the head of any hyperarc $e \in E_G$ in the configuration G, the head of e in G is a subset of the original head of e in F:

  $$\forall e \in E_G\ |h'(e)| > 0 \wedge h'(e) \subseteq h(e)$$

- If the parent feature of a non-mandatory decomposition is present, the number of children selected must be equal to or greater than the original decomposition minimum (forward paths) and less than or equal to the original maximum:

  $$\forall e \in E\ .(t(e) \subset N_G \wedge |h(e)|>\min) \Rightarrow \max \geq |h'(e)| \geq \min$$

3.2 Configuration Procedure of a Feature Diagram

The definition of Configuration guides the characterization of the Configure procedure. An obvious pre-condition is that the Feature Diagram is correct and has no inconsistencies. Many works (see for example, [2]) are devoted to solving that question, and therefore, we do not consider it here. Once the application engineer has expressed his/her preferences by selecting a set of non-mandatory features, we must recognize a usual problem: once the initial set of non-mandatory features has been selected, and assuming they are compatible, it is possible that some non-mandatory feature groups (hyperarcs with |h(e)|=m and multiplicity 1..n < m) remain undefined. There are at least two ways in which the configuration process can be dealt with: a) finding the (probably ordered) set of all valid configurations that fulfill the defined selection; and b) guiding the engineer until a unique valid configuration is found. The first option is a complete but computationally costly solution. The second is more realistic, but it remains largely a manual process, accomplished with FD tools. Staged configuration [7] is a classical approach for solving this problem in several steps. We find using a topological order in the set of features included in the head of each hyperarc useful for facilitating the process. For F-graphs, such node preordering can be accomplished by the F-Acyclic procedure [8]. This option implies that the domain engineer has assigned a preference order to each group of features. An example can clarify the idea: in an e-commerce product line credit card payment is more frequent than check or phone based payments and, in consequence, if the application engineer does not explicitly decide to change the payment method, credit payment will be selected by default.

Given a (Constrained) Feature Diagram F=(N, E, r, δ), and U an identified (selected manually) subset of compatible nodes of N: $U \subset N$), a valid configuration $G=(N_G, E_G, r)$ is obtained based on hypergraphs algorithms. Observe that user selected features are of special relevance to determine which features are or not included in the configuration, see Rule 4. Thus, a selected feature may require that some optional features, min=0, become a non-optional ones, min=1. Therefore, we must to check and replace these multiplicities before the configuration procedure is applied. Obviously, all nodes belonging to U are set to multiplicity min=1.

Taking into account the above considerations and the defined configuration rules, a valid configuration is obtained as follows:

1. For each $u_k \in U$ a procedure, adapted from $A\_frontier(F, F', r, u_k)$ procedure [16] is applied. Procedure assigns multiplicity 1..1 (1..n) to each arc $e \in F'$ with multiplicity 0..1(0..n). This step is performed for all nodes of subset U. Procedure transform Feature Diagram F=(N, E, r, δ) in Feature Diagram F'=(N, E, r, δ').
2. Procedure Visit [8] is applied to root node r and Feature Diagram F', $sit(r, F')$. This procedure finds all nodes connected to r and returns a set of paths connecting them to r. Procedure Visit has to be adapted in order to limit the number of nodes of h(e) to be examined. Note that for each feature group only min features has to be selected, being (min,max) their associated multiplicity value. The procedure selects nodes and defines hyperarcs to be considered in the final configuration hypergraph $G=(N_G, E_G, r)$.

4 Feature Meta-model

One of the advantages of the definition of Feature Diagrams as F-hypergraphs is that we have only two types of elements, features and decompositions, instead of introducing an additional element (grouped features) to complete the semantics. In consequence, the definition and implementation (as CASE tools) of the meta-model is easier. The proposal is modular, allowing several versions, from the simplest Tree based meta-model to the complete F-hypergraph/constrained/typed meta-model. The definition style uses the package merge mechanism and is the same that the UML2 meta-model uses extensively in the OMG documentation. This approach allows all the variants of the feature diagrams mentioned in Sections 1 and 5 to be covered (figure 4).

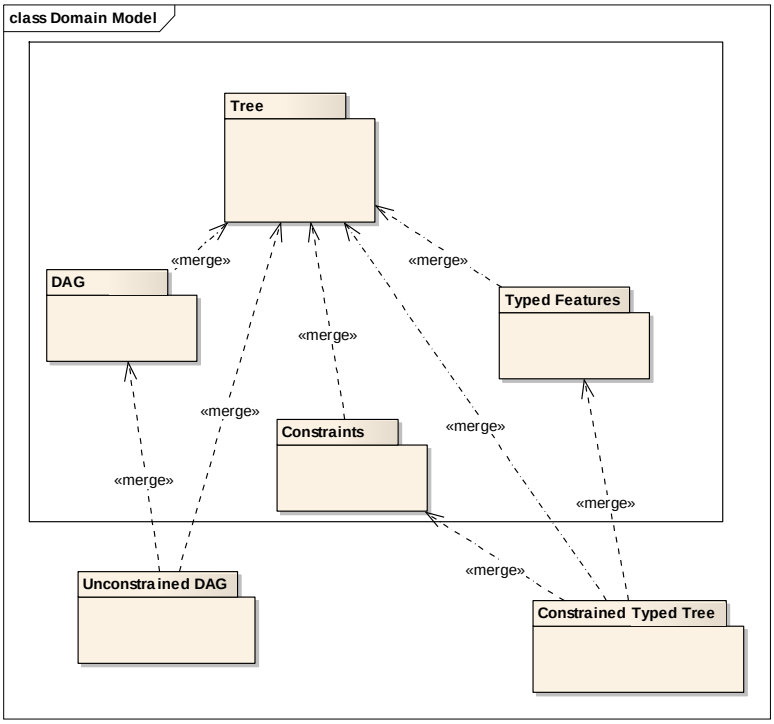

Fig. 4 Extensible Feature Meta-model

As an example, the details of the base package are shown in Figure 5. A *FeatureDiagram* has a *Root* (feature), a set of zero or more (non-root) *Features*, and a set of *HyperArcs* (in this case *Decompositions)*. Each Decomposition connects a parent *Node* (Root or Feature) with one or more child Features. As multiplicity of children meta-association indicates, a Feature can only be child of one Decompostion (and indirectly of a parent feature). This is a constraint that makes the structure into a tree with a root that has no parents. Decomposition has an associated *MultiplicityElement* that must conform to the associated OCL constraint: maximum

value (upper) must be less than or equal to the number of children of the Decomposition. To convert a tree based meta-model into the general F-hypergraph version, we need to merge the package DAG. Other possibilities are the Typed or Constraints packages.

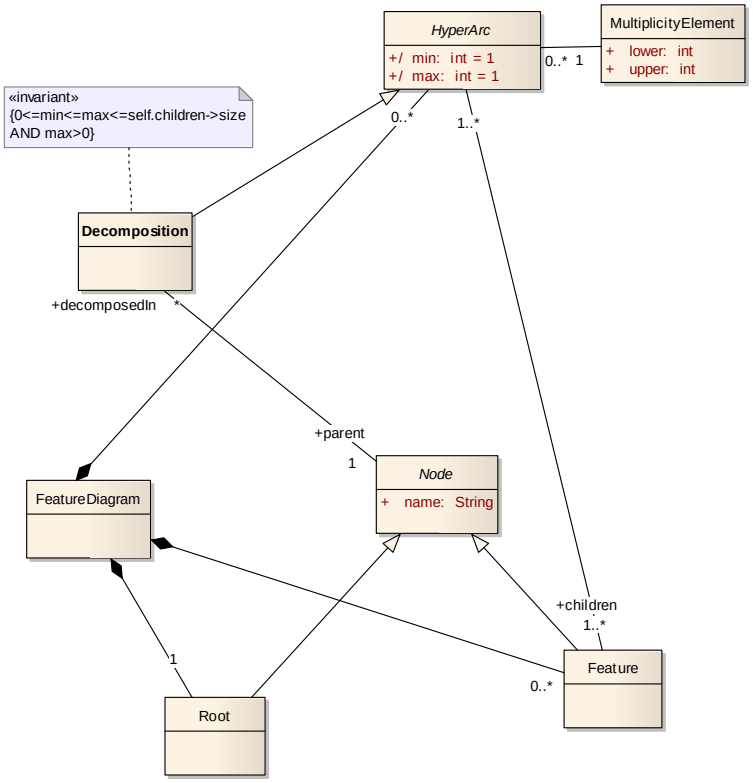

Fig. 5 Detail of the basic package of the proposed extensible Feature Meta-model

The basic meta-model of the Feature Configuration (rarely used) is the same of Figure 10 but with the invariant {0<min=max=self.children-> size}, taken into account that the possibility of election among children features must be null.

To convert a tree based meta-model into the general F-hypergrah version, we need to merge the package DAG. Other possibilities are the Typed or Constraints packages. As an example the combination of the four packages, respecting the UML merge rules results in the meta-model of Figure 6.

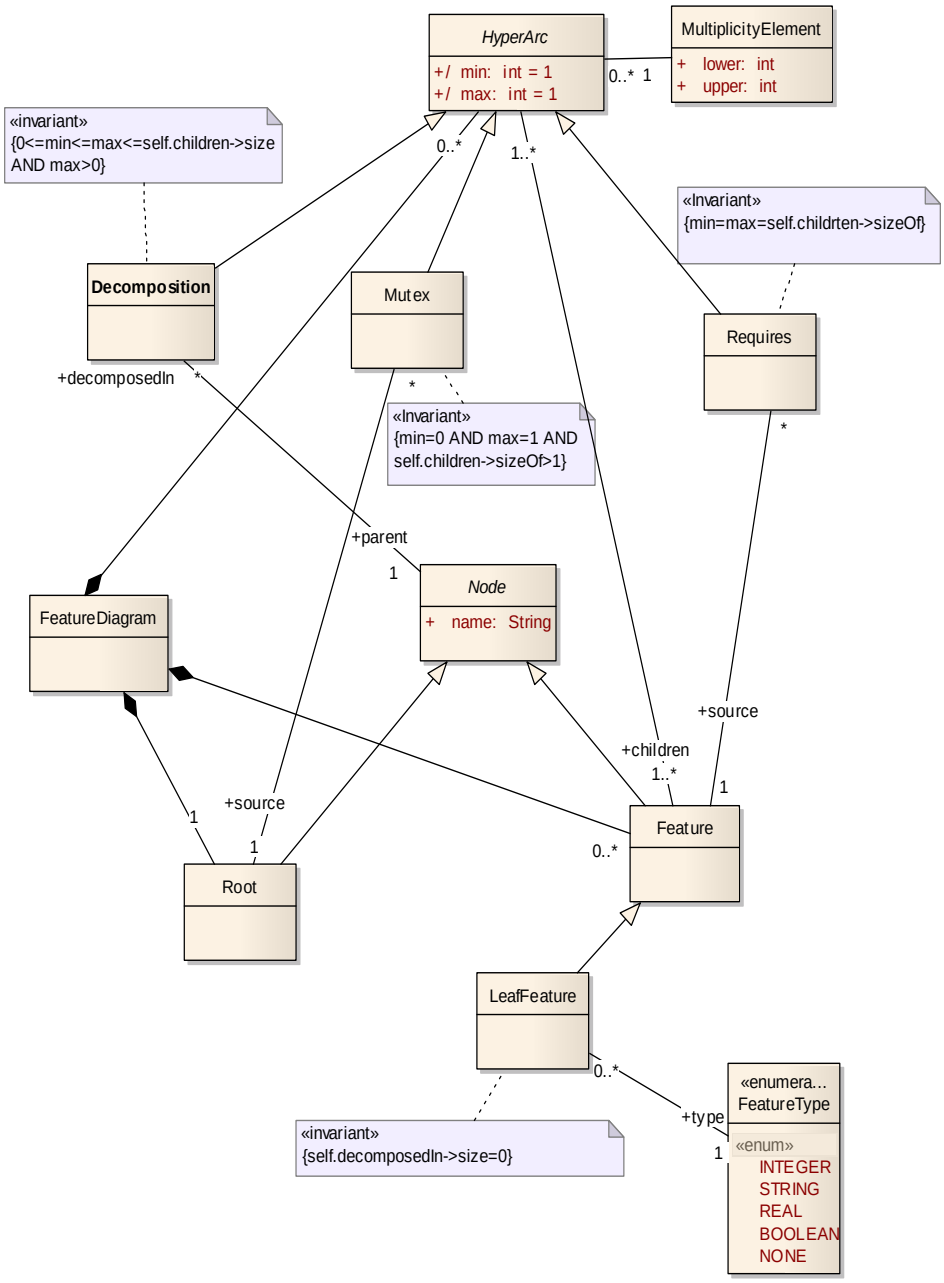

Fig. 2 Final Constrained Typed DAG version of Feature Meta-model

To implement the meta-model, we have used GMF[1]. The Eclipse Graphical Modeling Framework (GMF) provides a generative component and runtime infrastructure for developing graphical editors based on EMF and GEF. A plain implementation of the meta-model has been defined. The visual syntax of the Features and Root are rectangles, while the Decomposition graphical representation is a circle with multiplicity details. The Tree/DAG variants require a different multiplicity value but visually are similar. The type is easy to add as an attribute of Feature. The graphical constraints are simple arrows.

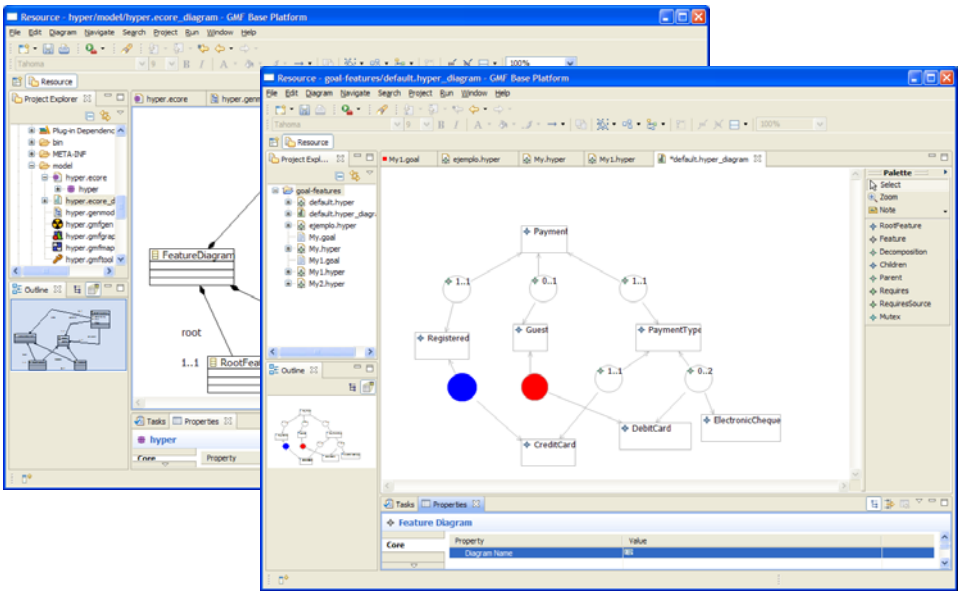

Fig. 7 Using GMF to implement the proposed meta-model

5 Transformation of Feature Models

Feature Diagrams have associated design models (generally expressed by UML models, including class, use case, and interaction diagrams). We use the UML package merge mechanism to preserve the traceability form feature to UML models, as explained in [13]. The proposed meta-model (and the consequent tool availability) opens the door to the definition of enhanced versions of the feature model to UML transformation presented in [13]. We propose to refine the original version (based on feature trees) into a more general version. We face the problem in two phases:

- The simplest situation occurs when the hypergrah is a hypertree (and then the transformation is trivial)
- The general case when a F-hypergrah is considered

[1] http://www.eclipse.org/modeling/gmf/

Feature Tree transformation

If the Feature Diagram is a Feature Tree, the strategy consist of that each variability point detected in the feature model must originate a package that will be combined, or not, in product development time, according to the selected configuration [13].

If we define a UML Tree Package Model TPM pm = (P, M), where P is the a set of packages and M the set of ordered pairs of packages (representing merge dependencies between them, p1 requires p2): $m(p1, p2) \in M.\ p1, p2 \in P \wedge p1 \Rightarrow p2$.

Definition of Feature Tree *Transformation Operation:* $FT \rightarrow TPM$

Given a Feature Tree h = (N, E, r, δ), a UML Package Model pm = (P, M) is created applying the following rules:

- The root r generate the *Base* package $P=\{Base\} \wedge pp=Base$
- Each feature (recursively) connected by an optional hyperarc decomposition to a previous considered feature of N (including root) generates a new package and a new merge dependency from the new package to the previous.

  $\forall n \in N \quad \forall e \in E\ .\ (T(e) = \{n\} \wedge |H(e)| > min(e)) \Rightarrow$

  $p1$ is new $P = P \cup \{p1\} \wedge m1$ is new $m1(p1, pp).\ M = M \cup \{m1\} \wedge pp=p1$

In this type of transformation, where different meta-models are implied, the MDE approach is a better approach. We have previously implemented this transformation using the Czarnecki meta-model. Using the new meta-model, the transformation is easier. Figure 8 shows the QTV based definition of the transformation.

In practical terms we must consider the multiplicity of the parent decomposition. If the minimum is equal to the number of children of the decomposition (1..1, for instance, when the number of children is exactly one) the features are non optional and the related design elements must be incorporated to the existing package. If the minimum is less to the number of children (1..1, when the number of children is more than one; 0..1; 0..2; 1..2; etc.) the feature is optional and the design elements are described in a new package, merged with the existing package.

Being a tree, the transformation can be implemented by a XML style sheet and involves:

a) Transform the Feature model into a UML model.
b) Transform the RootFeature into a root Package
c) Transform each optional Feature (i.e., with multiplicity minimum less than the number of children of the decomposition from which it is part) into a package merged with the previous package.
d) Ignore the mandatory Features (i.e., with minimum multiplicity equal to the number of children), simply passing to nodes in the next level.

An example of simple application is shown in Figure 9.

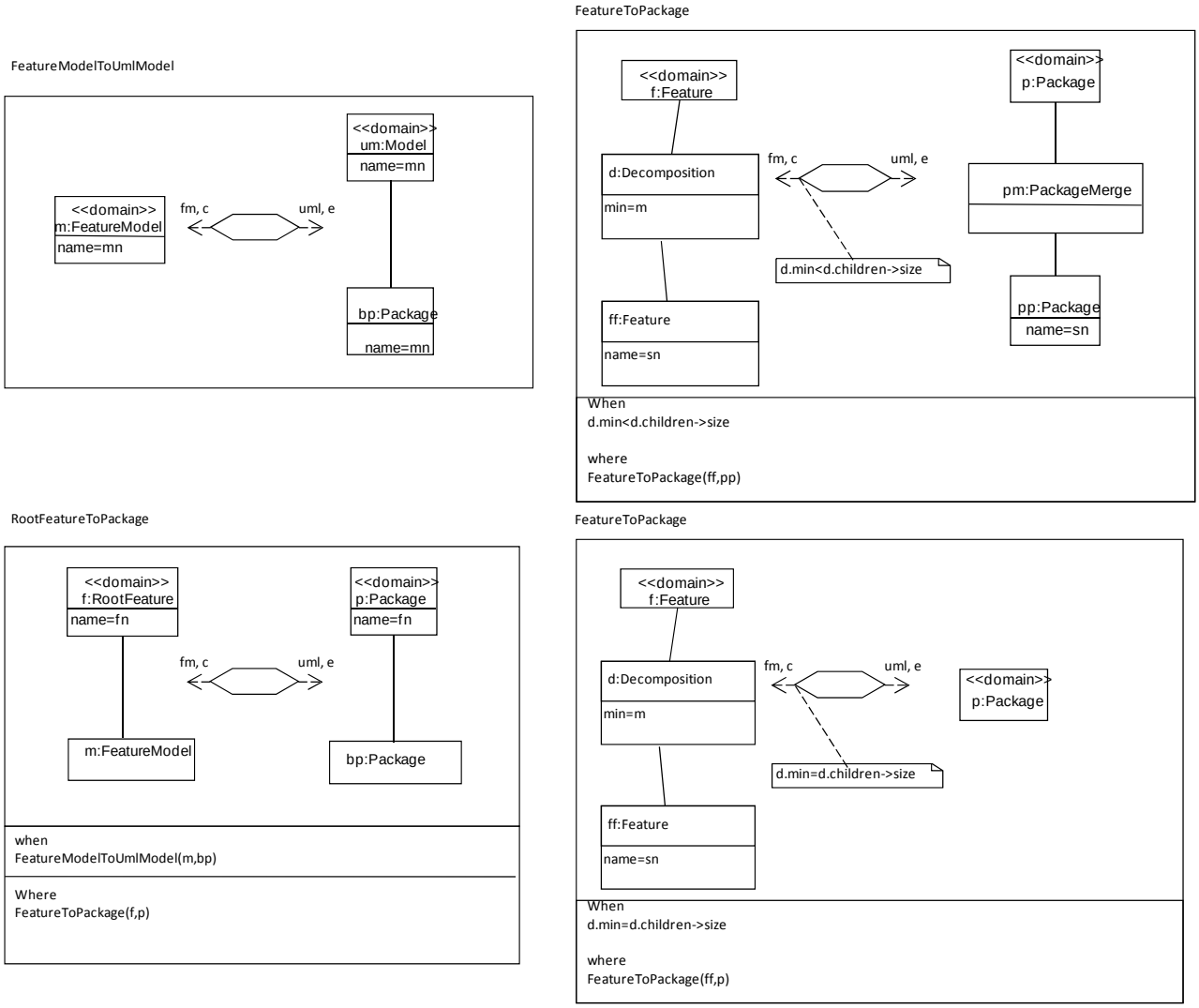

Fig. 8 Using QVT to define the proposed transformation

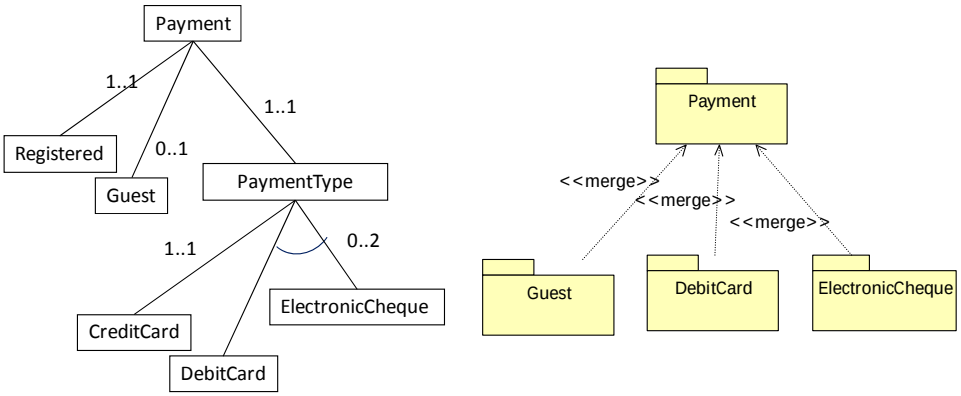

Fig. 3 Application of the proposed transformation to a simple Tree Feature Diagram

Feature Diagram transformation

If the Feature Diagram is a general hypergraph the transformation is more complex as a feature can have contradictory properties. The chosen strategy implies that each package with more than one parent in the feature model originates always a package. An optional decomposition is treated as previously and generates a package (if not created before) and a merge dependency. A non optional feature generates a package (if not created before) and one import dependency with the base package that in this case is the source of the relationship, while the new package is the target. Figure 10 shows an example of this strategy. The cost we assume is that the structure is in general not optimized. Considering the *CreditCard* package, we can argue that if is included in *Payment*, the Guest package always has access to the content of *CreditCard* and we could remove it, getting exactly the same model of Figure 9.

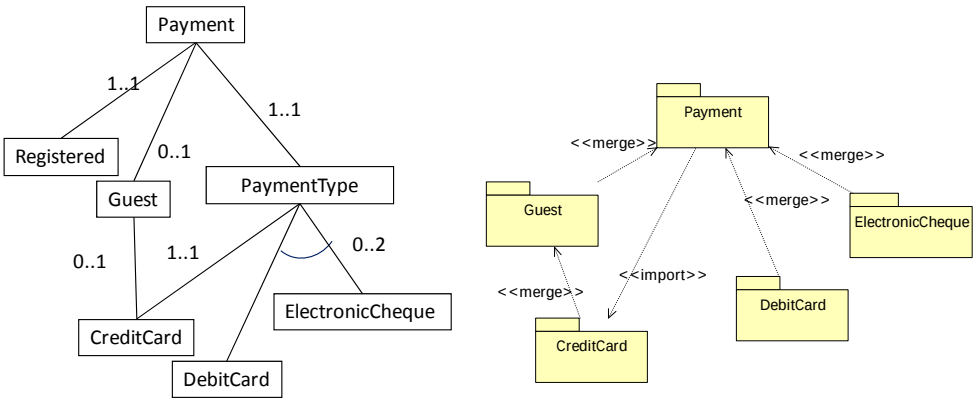

Fig. 10 Application of the hypergraph based proposed transformation to a general Feature Diagram

If we define a UML Package Model pm = (P, M, I), where P is the a set of packages, M the set of ordered pairs of packages (representing merge dependencies between them, p1 requires p2): $m(p1, p2) \in M. p1, p2 \in P \wedge p1 \Rightarrow p2$ and I the set of ordered pairs of packages (representing import dependencies between them, p1 requires p2): $i(p1, p2) \in M. p1, p2 \in P \wedge p1 \Rightarrow p2$

Definition of Feature Diagram *Transformation Operation:* FD $\rightarrow$ PM

Given a Feature Diagram FD h = (N, E, r, δ), a UML Package Model pm = (P, M,I) is created applying the following rules:

- The root r generate the *Base* package $P=\{Base\} \wedge pp=Base$
- Each feature (recursively) connected by an optional hyperarc decomposition to a previous considered feature of N (including root) generates a new package and a new merge dependency from the new package to the previous.

  $\forall n \in N \quad \forall e \in E . (T(e)=\{n\} \wedge |H(e)|>min(e)) \Rightarrow$

  $p1$ is new $P = P \cup \{p1\} \wedge$ m is new m(p1, pp). $M = M \cup \{m\} \wedge pp=p1$
- Each feature connected by a non optional hyperarc decomposition to a previous considered feature of N (including root) and with more than one parent decomposition generates a new package and a new import dependency from the previous package to the new.

  $\forall n \in N \quad \forall e \in E . (T(e)=\{n\} \wedge |H(e)|=min(e) \wedge |BS(n)| > 1) \Rightarrow$

  $p2$ is new $P = P \cup \{p2\} \wedge$ i is new i(pp, p2). $I = I \cup \{i\} \wedge pp=p2$

The general hypergraph based meta-model transformation is described with QVT in Figure 11. As in the previous transformation, we consider multiplicity but also the number of parents. If the feature is optional, the design elements are described in a new package, merged with the existing package. If the features are non optional and the number of parents is exactly one, the related design elements must be incorporated to the existing package. But if the features are non optional and the number of parents is greater than one, the design elements are described in a new package, imported by the existing package. A minor problem to manage is the fact that the new package can exist as result of a previous feature transformation. Therefore we have defined a total of eight QVT transformations.

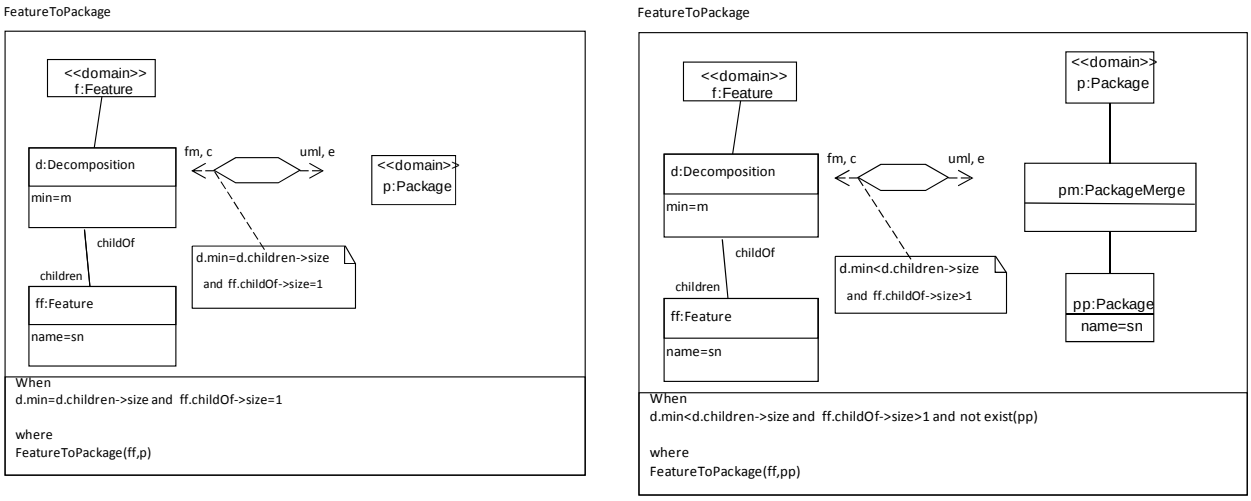

Fig. 11 Using QVT to define the proposed transformation

Being a single rooted graph, the transformation can be implemented by a XML style sheet and involves:

e) Transform the Feature model into a UML model.
f) Transform the RootFeature into a root Package
g) Transform each optional Feature (i.e., with multiplicity minimum less than the number of children of the decomposition from which it is part) into a package merged with the previous package.
h) If the number of parents of the feature are exactly one, ignore the non optional Features (i.e., with minimum multiplicity equal to the number of children), simply passing to nodes in the next level.
i) If the number of parents of the feature are greater than one, transform each non optional feature into a package imported by the previous package.

6 Related work

Starting with the original FODA proposal [12], several variants of feature diagrams have been proposed: FORM [11] is an extension where feature diagrams are single-rooted directed acyclic graphs (DAG) instead of simple trees. FeatureRSEB [10] also uses DAGs and changes the visual syntax, including a graphical representation for the constraints requires and mutex. Other authors, such as Czarnecki et al. [5,6] and Batory [1], continue to use trees as the main structure (however Czarnecki et al. add OR decomposition, graphical constraints, and distinguish between group and feature cardinalities). Riebisch et al. [17] replace AND, X-OR, and OR by multiplicities combined with mandatory and optional edges. Cechticky et al. proposed a notation

without solitary features in an attempt to reduce the number of redundant representations: a group with one grouped feature is used instead [4].

A detailed comparison of all these variants has been done by Schobbens *et al*. in [18]. The authors use a parameterized formal definition of the feature diagram, obtaining a framework useful for comparing and classifying all the variants, proving how the diverse options can be equivalent. Some recent works are devoted to the validation of feature models, mainly based on propositional formulas [1] or constraint solvers [2]. Mendoça *et al*. use a two stage analysis to validate the models [15]. The advantage of using hypergraphs is the remarkable simplification of the supporting model. Instead of transforming FDs into a set of formulas to find inconsistencies or configure the final product, the algorithms can be used directly on the constrained hypergraphs, using a unique formalism. Modeling and transformation tools are easier to define and implement as a coherent and extensible set.

6 Conclusions and future work

In this article, we have used F-hypergraphs to define the semantics of feature diagrams and their configuration. Once the formal definition is stated, the construction of an extensible feature meta-model has been dealt with. The algebraic definition directly yields the invariants of the meta-model, establishing a firm foundation. The advantages of simplicity and extensibility have made it possible to build a set of modeling tools compatible with the different flavors of FDs.

As part of our industrial oriented work, we implemented a Feature Modeling Tool (FMT) as a template integrated with the Microsoft Visual Studio IDE. The meta-model we used was based on constrained trees, validation is external, and configuration uses a staged approach. Work in progress on FMT includes the incorporation of the extensible meta-model and the implementation of the algorithms. As the tool was originally built using DSL tools and C#, the meta-model enhancement and the implementation of the algorithms are straightforward.

References

1. Batory, D.S. Feature Models, Grammars, and Propositional Formulas, in SPLC, 2005.
2. Benavides, D., Trinidad, P., and Ruiz-Cortes, A. “Automated Reasoning on Feature Models”, Conference on Advanced Information Systems Engineering (CAISE), July 2005.
3. Bosch, J. “Design & Use of Software Architectures. Adopting and Evolving a Product-Line Approach”. Addison-Wesley. 2000.
4. Cechticky, V., Pasetti, A., Rohlik, O. and Schaufelberger, W. XML-based feature modelling, in ICSR 2004, LCNS 3107, pp. 101–114. 2004.
5. Czarnecki, K., and Eisenecker, “Generative Programming: Methods, Tools, and Applications”, Addison-Wesley, 2000
6. Czarnecki, K., Helsen, S. and Eisenecker, cardinality-based feature models and their specialization, Software Process: Improvement and Practice 10 (1):7–29. 2005.

7. Czarnecki, K., Helsen, S. and Eisenecker, U. Staged Configuration Through Specialization and Multi-Level Configuration of Feature Models. Software Process Improvement and Practice, 10(2):143 – 169, 2005.
8. Gallo, G., Longo, G., Pallottino, S., Nguyen, S."Directed hypergraphs and applications," Discrete Applied Mathematics, vol. 42, Abr. 1993, pp. 177-201.
9. Gallo, G., Scutella, M.G. Directed hypergraphs as a modelling paradigm, AMASES 21: 97-123, 1999.
10. Griss, M.L., Favaro, J., d'Alessandro, M. Integrating feature modeling with the RSEB, Proceedings of the Fifth International Conference on Software Reuse, pp.76-85, 1998.
11. Kang, K. C., Kim, S., Lee, J. y Kim, K. FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. Annals of Software Engineering, 5:143-168. 1998.
12. Kang, K., Cohen, S., Hess, J., Nowak, W., and. Peterson, S. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report, CMU/SEI-90-TR-21, 1990.
13. Laguna, M.A., González-Baixauli, B., and Marqués, J.M. Seamless Development of Software Product Lines: Feature Models to UML Traceability. GPCE 07, 2007.
14. Lee, K., Kang, K. C., Chae, W., Choi, B. W. "Feature-Based Approach to Object-Oriented Engineering of Applications for Reuse". Software: Practice and Experience, 30(9):1025-1046. 2000.
15. Mendonça, M., Cowan, D., Malyk, W., Oliveira, T. Collaborative Product Configuration: Formalization and Efficient Algorithms for Dependency Analysis, Journal of Software, 3(2):69-82, 2008.
16. Nguyen, S., Pretolani, D., Markenzon, L. On some path problems on oriented hypergraphs. Informatique Théorique et Applications/Theoretical Informatics and Applications, vol. 32, no 1-3, 1–20, 1998
17. Riebisch, M., Boellert, K., Streitferdt, D., Philippow, I. Extending feature diagrams with UML multiplicities, in: Proceedings of IDPT2002, June 2002.
18. Schobbens, P., Heymans, P., Trigaux, J., and Bontemps, Y. 2007. Generic semantics of feature diagrams. Comput. Netw. 51, 2:456-479. Feb. 2007.