DOI: 10.2298/CSIS100804016L

# Feature Diagram Formalization Based on Directed Hypergraphs

Miguel A. Laguna[1], José M. Marqués[1], and Guillermo Rodríguez-Cano[2]

[1] Department of Computer Science, University of Valladolid
Campus M. Delibes,
47011 Valladolid, Spain
{mlaguna,jmmc}@infor.uva.es
[2] wileeam@acm.org

**Abstract.** Feature models are used to represent the variability and commonality of software product lines (SPL), and to decide on the configuration of specific applications. Several variants based on tree or graph hierarchical structures have been proposed. These structures are completed with additional constraints, generally expressed in parallel with the feature diagram. This paper proposes the use of hypergraphs to integrate both concepts in a unique characterization. Therefore, the definition, validation and selection of feature configurations can be internally based on the hypergraph properties and well-known algorithms, while the concrete visual syntax remains unchanged for domain engineers. The implemented hypergraph algorithms have been tested using a complete set of feature diagrams. Finally a feature meta-model can be derived directly from the formal definitions, providing the foundations for building feature modeling tools.

**Keywords:** feature diagram, hypergraph, feature diagram configuration.

## 1. Introduction

Software product lines (SPL) constitute a successful reuse paradigm in industrial environments despite their complexity [3]. Feature diagrams (FD) represent the variability and commonality of software product lines and permit the configuration of each specific application to be selected.

Feature diagrams were first introduced in the Feature Oriented Domain Analysis (FODA) method [11], and Fig. 1 shows an example of a partial feature diagram of an eCommerce SPL inspired by Lau [15]. This proposal defined features as the nodes of an and/or tree related by various types of edges. As depicted in Fig. 1 a feature diagram is a structure diagram showing the hierarchical (parent/child) structure between the features. A feature diagram is composed of the tree root or concept (*eCommerce*), and its subfeatures showing mandatory, e.g. (*Payment*), optional, e.g. (*Registration*), and alternative features (*CreditCard, DebitCard, ElectronicCheque*). The relations (edges) between features (nodes) can be: AND, XOR and OPTIONAL relationships. To improve their expressiveness, several extensions have been proposed, incorporating the OR

decomposition [12], changing the visual syntax, or using directed acyclic graphs (DAG) instead of simple trees and UML-like multiplicity constraints to annotate multiplicities for sets of features [18]. Schobbens et al. [20] have surveyed and evaluated the diverse FD variants, 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, XOR, OR, multiplicity) are expressed. They propose a generic formalization of the syntax and semantics of FD and a new non-redundant variant FD or VFD. One of the conclusions is that these FD description languages have a different concrete syntax but share many aspects of an abstract syntax and semantics based on graphs.

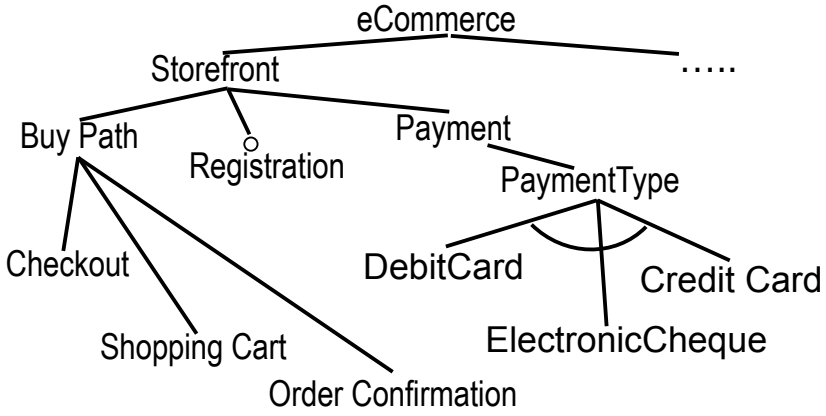


**Fig. 1.** Example of a FODA feature diagram

However, the use of standard graphs as the underlying structure of FDs forces the use of *ad hoc* internal representations to differentiate the diverse feature relationships. For example, the internal representation of the grouped alternative decomposition of the feature *PaymentType* of Fig. 1 (One-To-Many relationship), and the internal representation of the decomposition of the feature *Buy Path* (three binary relationships) must be different.

Moreover, an FD alone cannot capture all domain restrictions. Constraints between features (a feature requires another feature or two features are mutually exclusive) have been added, in textual or graphical formats, to complete the semantics of the models. If textual, separate constraint documentation must be handled to have the global picture of the model. Modifying a feature diagram can be difficult, since changing the hierarchy can cause modifications in constraints.

This leads us to consider the formalization of feature diagrams using directed hypergraphs [9]. A directed hypergraph is a generalization of the concept of directed graphs, an example is illustrated in Fig. 2 . In directed hypergraphs, One-To-Many, e.g. $E_4$ hyperarc, Many-To-One, e.g. $E_2$ hyperarc, and Many-To-Many,e.g. $E_1$ hyperarc, relations are naturally represented, including, as particular cases, labeled graphs, And-Or graphs, DAGs, and simple trees. Thus, conversion from a concrete FD to a hypergraph is simple and straightforward: features are the nodes of the hypergraph; decompositions and constraints are represented by hyperarcs.

The formalization of feature models using the structure of a hypergraph provides several benefits. Firstly, the use of hypergraphs as the underlying structure clarifies and simplifies the feature meta-model (only two basic types of elements are defined: features and relationships). Consequently, the definition and construction of automated tools is easier than in the case of graph based structures, as fewer elements are needed to express the FD semantics. More interestingly, analysis and configuration problems can be treated algorithmically, taking advantage of the progress made in hypergraph theory. From the practitioners' point of view, this will not affect them directly, because the FD language (i.e., the concrete syntax) is not modified.

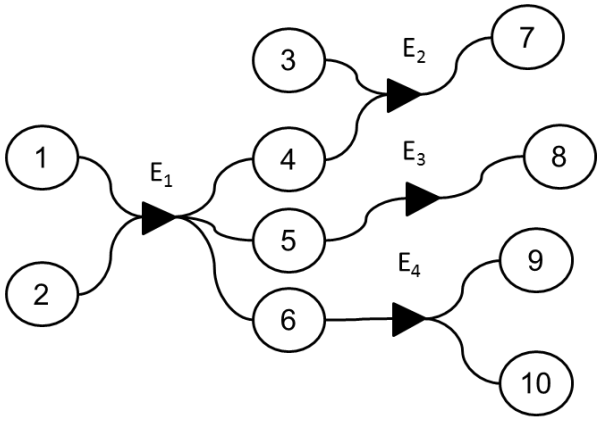


**Fig. 2.** An example of hypergraph

The rest of the article 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. Experimental results are also provided. Section 4 uses the formal definition to build a feature meta-model. Finally, Section 5 presents related work and Section 6 concludes the paper and considers future work.

## 2. Hypergraphs and Feature Diagrams

This section describes the conceptual framework and the formal structure, based on hypergraphs, used to model features and relationships among features in an FD. First, we define the feature modeling concepts to be used in the rest of the paper. Next, basic definitions of hypergraphs are given. Finally, the formulation of FDs is presented.

In general terms, a feature model defines features and their dependencies, covering the commonality and variability of software products in a Software Product Line. An FD is a structure defined by features, decomposition relationships, and constraint relationships. To model an FD by means of a supporting hypergraph we are only interested in its underlying structure formed by the following elements:

**Features** The collection of identified characteristics of a system which determines the scope of the Product Line under study. A feature is "a system property that is relevant to some stakeholder and is used to capture commonalities or discriminate between systems" [5].

**Root Feature** Also known as the concept, this is a distinguishable feature used to denote the feature diagram. It is always present in the feature model.

**Feature relationships** These are used to model directed associations between features. They are used to model the decomposition of a feature (parent feature) into one or more detailed features (children features). Each feature relationship will be labeled with a UML-like multiplicity value specifying the range of allowable features to be selected.

**Constraints** To further restrain the valid configurations of an FD, additional constraints can be formulated as a (dependency) relationship between features. The constraints considered are *requires* and *mutex*. Requires establishes a compulsory relationship between a source feature and a subset of target or required features. The *requires* relationship defines an implication where, if the source feature is selected, then the implied set of features must be selected as well. Unlike requires, the *mutex* relationship defines an exclusion between features. Thus, if two or more features are associated by a *mutex* constraint, one and only one of the features can be selected in a product configuration.

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. 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 graph is to represent Many-to-Many relations, for which simple DAG or trees are not well equipped. Definitions of *hypergraph* and the *Backward Star* function, useful for our purposes, are given:

**Definition 1.** *[8] A directed hypergraph, or simply hypergraph, is a pair* $H = (V, E)$*, where* $V$ *is a non empty set of vertices (or nodes) and* $E$ *is a set of*

*directed hyperedges; a directed hyperedge or hyperarc $e \in E$ is an ordered pair, $e = (T(e), H(e))$, where $T(e) \subseteq V$ is the tail of $e$, while $H(e) \subseteq V \backslash T(e)$ is its head.*

**Definition 2.** *[8] Given a hypergraph $H = (V, E)$ and a node $v \in V$, Backward Star of node $v$, $BS(v) \subseteq E$, denotes the set of hyperarcs entering $v$, $BS(v) = \{e \in E \mid v \in H(e)\}$*

The hypergraphs we use in the rest of the article are F-hypergraphs, a particular case of acyclic directed hypergraphs, characterized by the fact that the tail cardinality of the hyperarcs is always one (Forward hyperarcs, or simply F-hyperarcs). So the next definition states:

**Definition 3.** *An F-hypergraph is a pair $H = (V, E)$, where $V$ is a non empty set of nodes and $E$ is a set of F-hyperarcs; an F-hyperarc $e \in E$ is an ordered pair, $e = (t(e), H(e))$, where $t(e) \in V$ is the tail of $e$, while $H(e) \subseteq V - \{t(e)\}$ is its head.*

### 2.1. Feature Diagrams as Hypergraphs

A feature model, as the result of a Domain Analysis, defines features and their relationships, the feature diagram being its key element. Therefore, the starting point for the formulation of an FD in terms of hypergraphs is to associate one node with each feature and one hyperarc with each feature relationship. The feature diagram can thus be described by an F-hypergraph. Each hyperarc is assigned a label which corresponds to the multiplicity of the relationship.

More formally, for a given FD, let $F$ denote the set of all its features, $F = \{f_1, f_2, \ldots, f_k\} \cup \{f_c\}$, where $f_c$ is the concept or root feature of the FD; $f_c$ is the only node not contained in the head of any hyperarc of the hypergraph, i.e. it is the only node in the FD whose Backward Star, $BS(f_c)$, is the empty set.

A feature relationship takes the general form $e = (f, S, [m \ldots n])$, where:

- $f$ is a feature of $F$
- $S$ is a non empty subset of features of $F - \{f, f_c\}$, $|S| \geq 1$, and
- $[m \ldots n]$ is the relationship multiplicity with $m, n$ integers, $m \geq 0; 0 < n \leq |S|; m \leq n$

The semantics of the relationship multiplicity is that if feature $f$ is selected, at least $m$ and no more than $n$ features of subset $S$ must be selected. From the point of view of feature modeling, $f$ is the parent feature and $S$ the set of its children features or subfeatures. We associate a hyperarc with each feature relationship $e$, having $f$ as its tail and $S$ as its head. Fig. 3 shows how feature decomposition relationships of an FD language are formulated as hyperacs.

The constraint relationships will be formulated as feature relationships. The *requires* relationship establishes, in general, a compulsory relationship between a feature $f$ and a subset of features $R$ such that, if $f$ is selected, all features of subset $R$ must be selected. In other words, a *requires* relationship can be

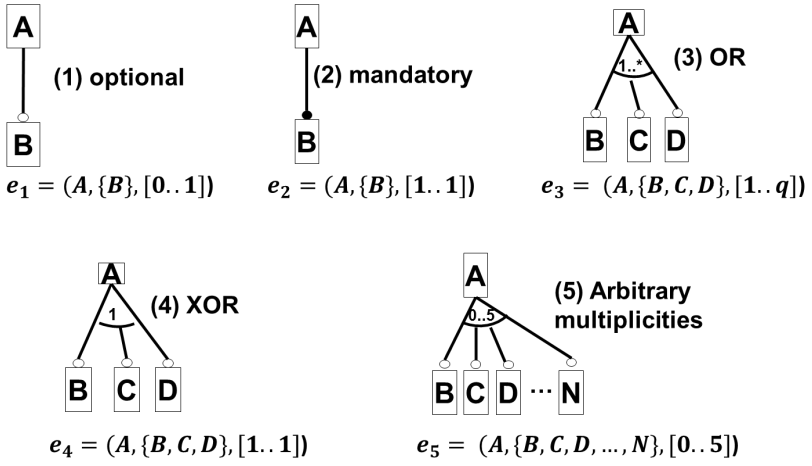


**Fig. 3.** Example of feature relationships [18] modeled as hyperarcs

written in terms of a feature relationship as $e = (f, R, [q \ldots q])$ with $|R| = q$. We would like to point out that introducing this kind of hyperarc, cycles could 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 transformation of a *mutex* relationship into a feature relationship is slightly more elaborate. It must be reinterpreted as a feature relationship from a common and always present feature – the root feature fc – to the involved features in a *mutex* constraint. In brief, if a *mutex* constraint exists among a subset of features of FD, $M \subset F - \{fc\}$, then the constraint can be modeled as: $e = (f_c, M, [0..1])$

Some authors propose enhancing *requires* and *mutex* binary constraints with arbitrary n-ary logical expressions between features. The semantics of these complex expressions can be difficult to understand but, in any case, they can be added as hyperarcs to the hypergraph [17].

We now have a one-to-one correspondence between the concepts of a feature model and the structure of a hypergraph. It should be pointed out that the above formalization of features and feature relationships belongs to the abstract syntax level being applicable, without loss of information, to the concrete syntax variants of most representative FD languages [20].

Figure 4 depicts a feature diagram of a simple *WebPortal* system based on the feature model described by Mendonça et al. [17]. As pointed out above, there are some semantics which can not be expressed in a feature diagram using features and decomposition relationships, therefore additional constraints are required, the two best known being *requires* and *mutex*. Feature constraints

to be considered for the *WebPortal* system displayed in figure Fig. 4 , expressed in textual form, are:

| *WebPortal* constraints |
|---|
| $Auth.$ **REQUIRES** $UserLogin$ |
| $Transfer$ **REQUIRES** $https$ |
| $Ms$ **MUTEX** $https$ |

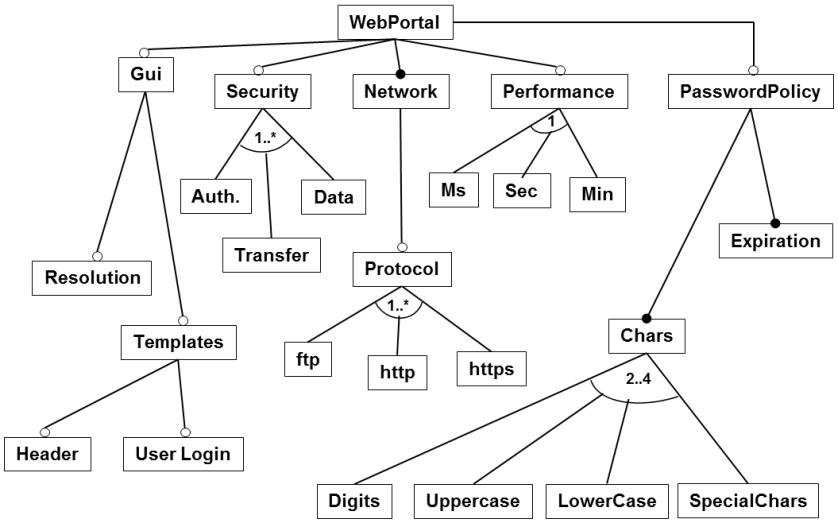


**Fig. 4.** *WebPortal* system feature diagram inspired by [17] using Riebish et al. [18] notation

To describe the *WebPortal* feature model as an f-hypergraph model, the aforementioned mapping rules will be applied. Let $WP = (F, E)$ denote the derived hypergraph, where $F$ is the set of features, the nouns shown in Fig. 4, $F = \{Gui, Security, ...\} \cup \{WebPortal\}$ and $E$ are, repectively, the set of hyperarcs representing the decomposition and the constraint relationships between features. These transformation are shown in table 1, where $n_1 \rightarrow n_2$ means $n_1$ is a parent of $n_2$; $n_2$ is a subfeature of $n_1$; and $n_1 \rightarrow [n_2, n_3, \ldots, n_k]$ means $n_1$ is a parent of the set of features $n_2, n_3, \ldots, n_k$, and each $n_i \; i = 2 \ldots k$ is a subfeature of $n_1$

Figure 5 illustrates the full F-Hypergraph transformation of the *WebPortal* system feature diagram depicted in Fig. 4 according to the proposed hyperarc transformation processes shown in Table 1. Note that the hyperarcs representing *require* and *mutex* constraints between features have been identified using an empty arrowhead instead of the solid arrowhead used for hyperarcs derived from the FD decomposition relationships. The existence of two kinds of adorn-

**Table 1.** F-arcs of F-hypergraph $WP = (F, E)$

| FEATURE MODEL RELATIONSHIP | F-ARC |
|---|---|
| WebPortal → Gui | $(WebPortal, \{Gui\}, [0 \dots 1])$ |
| WebPortal → Security | $(WebPortal, \{Security\}, [0 \dots 1])$ |
| WebPortal → Network | $(WebPortal, \{Network\}, [1 \dots 1])$ |
| WebPortal → Performance | $(WebPortal, \{Performance\}, [0 \dots 1])$ |
| WebPortal → PasswordPolicy | $(WebPortal, \{PasswordPolicy\}, [0 \dots 1])$ |
| Gui → Resolution | $(Gui, \{Resolution\}, [0 \dots 1])$ |
| Gui → Templates | $(Gui, \{Templates\}, [0 \dots 1])$ |
| Security → [Auth., Transfer, Data] | $(Security, \{Auth, Transfer, Data\}, [1 \dots 3])$ |
| Network → Protocol | $(Network, \{Protocol\}, [0 \dots 1])$ |
| Performance → [Ms, Sec, Min] | $(Performance, \{Ms, Sec, Min\}, [1 \dots 1])$ |
| PasswordPolicy → Expiration | $(PasswordPolicy, \{Expiration\}, [1 \dots 1])$ |
| PasswordPolicy → Chars | $(PasswordPolicy, \{Chars\}, [1 \dots 1])$ |
| Templates → Header | $(Templates, \{Header\}, [0 \dots 1])$ |
| Templates → User Login | $(Templates, \{UserLogin\}, [0 \dots 1])$ |
| Protocol → [ftp — http — https] | $(Protocol, \{ftp, http, https\}, [1 \dots 3])$ |
| Chars → [Digits, Uppercase, LowerCase, SpecialChars] | $(Chars, \{Digits, Uppercase, LowerCase, SpecialChars\}, [2 \dots 4])$ |
| $Auth.$ **REQUIRES** $UserLogin$ | $(Auth, \{UserLogin\}, [1 \dots 1])$ |
| $Transfer$ **REQUIRES** $https$ | $(Transfer, \{https\}, [1 \dots 1])$ |
| $Ms$ **MUTEX** $https$ | $(WebPortal, \{Ms, https\}, [0 \dots 1])$ |

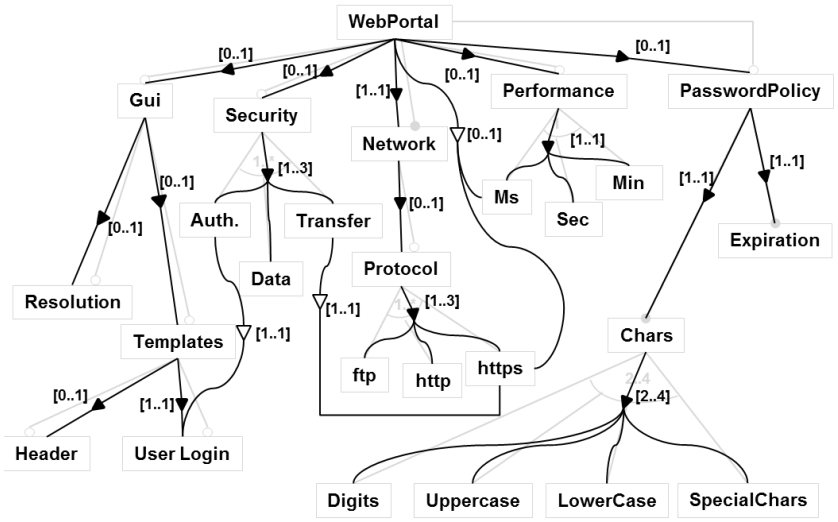


**Fig. 5.** *WebPortal* system F-hypergraph transformation

ments for hyperarcs has no special meaning, being used only for graphical purposes.

### 2.2. Formal Definitions

This subsection summarizes the described concepts in a compact form:

**Definition 4.** *A multiplicity value denoted as* $[m \dots n]$ *is a pair of integers* $(m, n)$ *with* $m \geq 0$, $n > 0$ *and* $m \geq n$.

**Definition 5.** *An Unconstrained Feature Diagram is an (arc labeled) acyclic F-hypergraph* $UD = (F, E, f_c)$ *where:*

- $F$ *is its set of nodes (or features)*
- $E = \{e_1, e_2, \dots, e_p\}$ *is the set of decomposition F-arcs, with* $e_i = (f_i, R_i, [m \dots n])$ , $R_i \subseteq F - \{f_i\}$ *and* $n = |head(e_i)|$ *for* $i = 1, \dots, p$
- $BS(f_c) = \emptyset \ \wedge \ BS(f) \neq \emptyset \ \ \forall f \in F - \{fc\}$

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

**Definition 6.** *A Feature Tree FT is an Unconstrained Feature Diagram, such that each feature has at most one entering hyperarc (root* $f_c$ *has none):* $|BS(f)| = 1 \ \forall f \in F - \{fc\}$

Constraints are introduced as extensions, adding additional hyperarcs with multiplicity $1 \dots n$ (*requires*) or $0 \dots 1$ (*mutex*)

**Definition 7.** *Given an Unconstrained Feature Diagram* $UD = (F, E, f_c)$*, a Constrained Feature Diagram, or simply a Feature Diagram, is an acyclic F-hypergraph* $FD = (F, E', f_c)$*, where*

- $E' = E \cup E_r \cup E_t$
- $E \cap E_r = E \cap E_t = E_r \cap E_t = \emptyset$
- $E_r = \{r_1, r_2, \dots, r_k\}$ *is the set of requires constraints F-arcs, with* $r_i = (f_i, R_i, [q \dots q])$ *where* $f_i \in F$, $R_i \subset F - \{f_i\}$ *for* $i = 1, \dots, k$ *such that* $1 \leq |head(r_i)| = q$
- $E_t = \{t_1, t_2, \dots, t_l\}$ *is the set of mutex constraints F-arcs, with* $t_i = (f_c, R_i, [0 \dots 1])$*, where* $R_i \subset F - \{f_c\}$ *for* $i = 1, \dots, l$ *and* $|head(t_i)| \geq 2$
- *For each* $e \in E'$*, the function isConstraint is defined:*
  - *isConstraint(e) = true, if* $e \in E_r \cup E_t$
  - *isConstraint(e) = false, if* $e \in E$

The last Boolean function is defined so as to be used in the configuration process, as explained further in Section 3.

As it is the most complete type of FD, from here on, we refer to the Constrained variant simply as the Feature Diagram or FD and denote it by default as $FD = (F, E, f_c)$. Section 3 shows how the well-known hypergraph traversal algorithm can contribute to the FD validation and configuration procedures.

## 3. Configuration of Feature Diagrams

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 feature 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*) as decompositions where the minimum multiplicity is equal to (less than) the number of children, the following rules apply:

**Rule 1** The root feature must be present in any configuration.
**Rule 2** A feature can be selected only if at least one of its parents is selected.
**Rule 3** If a feature is present, the features connected to it, through mandatory decompositions, must be selected.
**Rule 4** If a feature is present, the number $k$ of non mandatory features selected as children of its decompositions must be between the minimum and maximum of the original hyperarc multiplicity: $m \leq k \leq n$.

Two more rules are imposed by the *requires* and *mutex* constraints:

**Rule 5** *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 (Rule 3).
**Rule 6** *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 multiplicity equal to [0..1] (Rule 4).

Consequently, the configuration procedure can be applied uniformly to the constrained hypergraph, instead of dividing it into two problems or transforming the feature tree (or graph) into a set of propositional formulas, as proposed in the literature [17]. In fact, generalizing the several variants, all the above rules can be reformulated in a comprehensive way: In a valid configuration, defined as a subset of features of one FD, the present features must satisfy two simple properties:

**Property 1** For each feature other than the root, at least one of its *structural* parents (the tail of a hyperarc not representing a mutex/requires constraint) must also be present.
**Property 2** For each leaving hyperarc of the considered feature, with $[m \dots n]$ multiplicity, at least $m$ and at most $n$ children features of the head of the hyperarc must also be present in the configuration

These properties are enough to accomplish the rules 1 to 6. Considering the possible combinations of children given by the hyperarc multiplicity, the remaining rules can be trivially deduced. The second property implicitly defines a type of reduction [8] of the original FD hyperarcs into each one of the possible configuration hyperarcs.

**Definition 8.** *Given a Feature Diagram $FD = (F, E, f_c)$, and a hyperarc $e \in E$ where $e = (tail(e), head(e), [m \dots n])$, we define a C-reduction of $e$ as a hyperarc $e_G = (tail(e_G), head(e_G))$ such that $tail(e_G) = tail(e)$, $head(e_G) \subseteq head(e)$, $m \leq |head(e_G)| \leq n$.*

**Definition 9.** *A Valid Configuration $G = (F_G, E_G)$ is a hypergraph obtained from a (Constrained) Feature Diagram $FD = (F, E, f_c)$ by replacing each selected hyperarc of $E$ by one of its C-reductions:*

- *$F_G$ is a subset of features of F: $F_G \subseteq F$*
- *$E_G$ is a set of hyperarcs $E_G = \{e_G\}$ where $e_G$ is a C-reduction of one $e \in E$.*
- *The root is present: $f_c \in F_G$*
- *For each feature in the configuration, at least one of its parents in FD is present, giving as a result an F-connected hypergraph:*
  *$\forall f \in F_G \; f \neq f_c \; \exists e \in E \wedge isConstraint(e) = false \wedge f \in head(e) \wedge tail(e) \in F_G$*
- *For each feature $f$ in $F_G$, for each leaving hyperarc $e$ of $f$ in $E$, $e = (tail(e), head(e), [m \dots n])$, with $m > 0$, one C-reduction $e_G$ of $e$, must be present in the configuration:*
  *$\forall f \in F_G \wedge e \in E \wedge tail(e) = f \wedge m > 0 \Rightarrow \exists e_G \in E_G$ such that $e_G$ is a C-reduction of $e$*

Considering that the semantics of feature modeling is expressed by the concept of FD configuration [20], we can say that an FD is valid if at least one valid configuration can be derived from it and if each feature of the FD is present in at least one configuration (no *dead* features). The trivial cases are one FD with only one feature (the root itself) or with only mandatory features (no variability at all, only one valid configuration). For the useful cases, to validate one FD, it is enough to prove that each feature is present in at least one valid configuration. The next Subsection gives a procedure to find a valid configuration of one FD, given a set of selected features. The repeated application of that procedure to each individual feature of the FD will serve to trivially prove the validity of the FD itself.

### 3.1. Configuration Procedure of a Feature Diagram

The given definition will guide the configuration process. Once the application engineer has expressed his/her preferences, by selecting a set of features, and has checked their compatibility (i.e., there are no mutex or multiplicity conflicts between them), we can find a usual problem: it is possible that feature decompositions with no children selected remain undefined (hyperarcs with minimum multiplicity m, $0 < m < |head(e)|$ ). There are at least two ways in which the configuration procedure 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 a largely manual process, accomplished with the help of FD modeling

tools. Staged configuration [7] is a classical approach for solving this problem in several steps.

To facilitate the configuration process in these undefined hyperarcs, we think that it is useful to predetermine a topological order in the set of features included in the head of each hyperarc. This option implies that the domain engineer has assigned a preference order to each group of features (alternatively, the "weight" of the feature plus its mandatory descendants could be automatically calculated and assigned to the features [8]). The aim is to have a (set of) default feature(s) when there is no explicit decision. An example can clarify the idea: in an e-commerce product line, credit card payment is more frequent than check or transfer 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.

In any case, the possible configurations can be generated in two steps that try to accomplish properties 1 and 2 respectively:

**Step 1** The partial configurations that include the selected features are found. This step is deterministic in the sense that one path from each selected feature to root must be included, and (recursively) mandatory descendants of each feature in the configuration must be added. If valid (no multiplicity limits are violated), the resulting partial configurations can be communicated to the engineer or used in step 2.

**Step 2** Each resulting partial configuration is completed using a second procedure that finds the most "economical" configuration of the FD. For each undefined hyperarc (the number of selected features of its head is less than the minimum multiplicity and the hyperarc tail is present) the default feature descendant(s) are added to reach the minimum multiplicity. Again, mandatory descendants of each new feature are added and constraints are checked.

The first step is a variation of the hypergraph $BVisist(r, H)$ algorithm. The second one can be designed as a refinement of the $FVisit(r, H)$ algorithm, both described and analyzed in [8], and running in $\mathcal{O}$(size(H)) time, $H$ being the traversed hypergraph. Thus, given a (Constrained) Feature Diagram $FD = (F, E, f_c)$, and $U$ being an identified (selected manually) subset of features of $F$: $U \subset F$, a set of valid configurations $G_i = (F_{G_i}, E_{G_i})$ is obtained in two steps.

### 3.2. Step 1 Implementation

We say that one hyperarc $e \in F$ is a *parent* hyperarc of the feature $u$ if the head of the hyperarc contains $u$, $u \in head(e)$, and it is *structural* (i.e., not a constraint type hyperarc). The value *used(e)* gives the number of features of *head(e)* included in the configuration so far. For each present feature, at least one parent hyperarc must be included in the configuration. We always start from an initial configuration of $FD$, $G_0 = (F_0, E_0)$ where $F_0$ includes only the root feature of $FD$ and $E_0$ has zero selected hyperarcs.

For $U$ and $G$, where $G$ is a (partial) configuration of $FD$, a procedure *configure(U,G)*, adapted from $BVisist(r, H)$ [8] is applied (See Procedure 1 basic scheme). For each feature $u \in U$, $u$ is selected and removed. For each hyperarc $e$, *parent* of $u$, (i.e., *requires* or *mutex* arcs are not selected), and if it is a *valid parent hyperarc* (the hyperarc $used(e)$ value is less than the hyperarc maximum multiplicity), the configuration process continues: the tail of $e$ is added to the list of selected features and a new recursive execution of procedure 1 is launched, starting from the current configuration. If there are no *valid parent hyperarcs* for a feature, the procedure execution aborts (though other branches can continue).

Clearly the original complexity of the procedure increases with the number of features with more than one structural parent, being multiplied by each number of structural entering arcs $a_i$ of each feature $f_i$ except the root, i. e. $\mathcal{O}(size(H).\Pi a_i)$ (though all or most of the values are one in the FDs published in the literature). In the implemented version of the procedure, each time one feature is added, two operations are used for efficiency reasons: (1) its mandatory descendants are also added to the selected features, using an auxiliary procedure; (2) the existence of conflicts in the resulting partial configuration is tested using an auxiliary $compatible(G)$ function (returns false if $used(e) > n$ in any hyperarc of the configuration) to discard illegal configurations as soon as possible. Procedure 1 transforms the initial $U$ and $G_0 = (F_0, E_0)$ into one set of partial configurations $G'_i = (F_{G_i}, E_{G_i})$, all compatible with the expressed requirements.

For each resulting modified $G'_i$, property 1 holds at this point, that is, each feature has at least one structural parent (and no conflicts are present). Property 2 must be accomplished in a second step.

### 3.3. Step 2 Implementation

The procedure $complete(G'_i)$ inspired by $FVisist(r, H)$ [8], is applied to each modified partial configuration $G'_i$. The original $FVisist(r, H)$ finds all nodes connected to root $r$ and returns a set of paths connecting them to $r$. This must be adapted in order to limit the number of features of $head(e)$ to be examined. Note that, for each undefined hyperarc, only $m$ features must be selected, $[m \dots n]$ being its multiplicity value. The algorithm can be improved if we consider the default features first. Thus, if valid, the algorithm will reach the configuration where all the choices are the default features(s). Only when that configuration is illegal, is backtracking applied and the algorithm continues to search for a valid configuration $G_i = (F_{G_i}, E_{G_i})$.

As in procedure $configure(U, G)$, in the optimized implemented version, the mandatory descendants of each new feature selected are also added to the selected features, using an auxiliary procedure (not shown in the basic scheme of procedure $complete(G_i)$). If new feature(s) with no parents are added to the configuration, a new call to the $configure(U, G)$ procedure is necessary to find their parents before completion ($U$includes the new features without parents).

**Procedure** configure(U,G)

```
Generation of the staged (partial) configurations;
while U ≠ ∅ do
    // Select and remove node u ∈ U
    u ← first(U) ;
    U ← U − {u} ;
    // Mark node u ∈ G as selected
    mark(u,G) ;
    // Valid number of parents
    parents ← 0 ;
    // Try entering non-constrained arcs
    foreach e ∈ BS(u) and isConstraint = false do
        // Select hyper-arc when possible
        if used(e) < max(e) then
            parents ← parents + 1 ;
            used(e) ← used(e) + 1 ;
            if not getMark(tail(e), G) then
                mark(tail(e), G) ;
            end
            U ← U ∪ tail(e) ;
            configure(U,G) ;
        end
    end
end
// Finished path
G ← void ;
// Check for non-valid configurations
if parents = 0 then
    return ;
end
```

Again the original complexity of the algorithm is increased by the number of possibilities that can be chosen in each hyperarc with $0 < m < |head(e)|$ (2 in [1..1] and $|head(e)| = 2$, 3 in [1..1] and $|head(e)| = 3$, 3 in [2..2] $|head(e)| = 3$). This combinatorial explosion is alleviated because, in practice, we only select the first *m* features of the ordered set of children features and we only intend to find the first valid configuration.

The minimal set of features present in a configuration if no features are selected (built applying both procedures to the root feature and selecting the first *m* children in each hyperarc) constitutes the *default* feature configuration. The minimal set of features present in all the possible configurations (built applying only configure to the root feature and selecting only *m* mandatory/ requires features where $m = |head(e)|$ in each considered hyperarc) constitutes the *core* features. The base package of the SPL architecture [13] is a design solution to these core features.

### 3.4. Empirical Evaluation

An implementation in Java has been coded to test the procedures and to estimate the time needed to reach the partial and final configurations. The FDs used for experimentation were downloaded from the SPLOT[3] project [16]. The configurations were created in a quasi-random way (if an alternative feature is included, the rest of the features of the group are discarded to trivially avoid inconsistent configurations). The number of selected features over the total (RCS in the Tables) range between 1% and 35%. The real FDs of the SPLOT Web (all of which are originally trees with constraints) were divided for their study into several groups according to size.

The results corresponding to the biggest real models are shown in Table 1. The columns contain the FD Size ($FD_S$), Extra constraints representativeness (ECR), Relative configuration initial size (RCS) , Zero (mandatory descendants of root), First, and Second Algorithm Average time in ms ($S_0, S_1, S_2$), Configuration time ($S_{1-2}$), First or second staged-configuration size (average ratio of the number of valid configurations obtained in first, C1, or second, C2, algorithm execution). The dashes mean that no valid random configuration is generated, as the maximum number of features to be chosen was exceeded. It can be seen that, in general, when the number of initially selected features (RCS column) is increased, the time of algorithm 1 grows while the time of algorithm 2 decreases. The total sum remains acceptable (less than 0.4 seconds). The computer used was a Mac (OS X), equipped with an Intel processor (Core 2 Duo 2.16 Gigahertz), 2 Gb of memory.

A second group of studied FDs corresponds to *benchmark* models, with a considerably bigger size: up to 10,000 features and 1,000 constraints. Some constraints (in particular ternary CNF expressions) had to be previously transformed into hypergraph constructions and/or binary constraints to be dealt with,

[3] http://splot-research.org/

**Procedure** complete($G_i$)

*Generation of a complete configuration from a partial configuration*;

```
// Collect hyperarcs to be completed
foreach e ∈ E∧ getMark(tail(e),G_i) do
    W ← W ∪ {e} ;
end
while W ≠ ∅ do
    // Select and remove node w ∈ W
    w ← first(W ) ;
    W ← W − {w} ;
    // Try hyper-arc's head non-selected features
    foreach f ∈ head(w) do
        // Select feature f and check for no conflicts
        if used(w ) < min(w) and not getMark(f,G_i) then
            mark(f,G_i) ;
            used(w ) ← used(w ) + 1 ;
            if compatible(G_i) then
                // Find partial configurations for require
                   constraints
                if count(getRequire(G_i)) > 0 then
                    G_iv ← configure(getRequire(G_i),G_i) ;
                else
                    G_iv ← G_i;
                end
                // Complete derived partial configurations or
                   current
                foreach g ∈ G_iv do
                    G_i' ← complete(g) ;
                    if compatible(G_i') then
                        G_i ← G_i ∪{ G_i' } ;
                    else
                        // Configuration no valid
                        G_i ← void ;
                    end
                end
            else
                // Configuration no valid
                G_i ← void ;
            end
        end
    end
end
if W = ∅ then
    // Configuration is completed
end
```

as described in [19]. The elapsed time in these cases is longer but acceptable with a peak of 18.5 seconds (Table 2).

**Table 2.** Time (ms) of execution of the configuration algorithms applied to real FDs

| | | | | Configuration | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Feature model | $FD_s$ | ECR(%) | RCS | $S_0$ | $S_1$ | $S_2$ | $S_{[1-2]*}$ | $C_1$ | $C_2$ |
| Home | | | 5 % | 4.36 | 1.98 | 12.66 | 15.03 | 1.0 | 1.0 |
| Integration | 67 | 11.9 | 15 % | 8.72 | 4.19 | 7.78 | 12.54 | 1.0 | 1.0 |
| System | | | 25 % | 14.59 | 1.37 | 3.17 | 5.29 | 1. | 1.0 |
| | | | 35 % | - | - | - | - | - | - |
| Ecological | | | 5 % | 9.04 | 3.52 | 25.59 | 29.65 | 0.72 | 0.72 |
| car | 94 | 4.3 | 15 % | 19.71 | 14.13 | 0.0 | 14.2 | 0.0 | 0.0 |
| | | | 25 % | 28.13 | 12.41 | 0.0 | 12.5 | 0.0 | 0.0 |
| | | | 35 % | - | - | - | - | - | - |
| Electronic | | | 5 % | 39.79 | 28.58 | 287.98 | 317.93 | 1.0 | 0.84 |
| shopping | 287 | 11.8 | 15 % | 77.1 | 55.29 | 188.74 | 245.71 | 1.0 | 0.8 |
| | | | 25 % | 124.2 | 79.46 | 147.8 | 229.73 | 1.0 | 0.8 |
| | | | 35 % | 199.17 | 110.97 | 106.54 | 219.71 | 1.0 | 0.6 |

As all these models are based on trees, we use a modified version of the "Electronic shopping" FD (the last row of table 1) to test the influence of features with more than one parent in the performance of the first and second step configuration algorithms (the first algorithm creates several alternative partial configurations, completed by the second algorithm). Table 3 shows the results. Although the time increases notably, the relation time/number of found configurations is of the same order of magnitude. Thus, we can conclude that the use of hypergraphs as a practical tool with regular FDs is viable. More details of the conducted tests can be found in [19].

## 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 relationships, 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 constrained F-hypergraph meta-model. More complete details of the meta-model definition and tool implementation with GMF can be found in [14]. The definition style uses the package merge mechanism and is the same that the UML 2 meta-model uses extensively in the OMG documentation. This approach allows all the variants of the

**Table 3.** Time (ms) of execution of the configuration algorithms applied to benchmarks FDs [16]

| | | | | Configuration | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Feature model | $FD_s$ | ECR(%) | RCS | $S_0$ | $S_1$ | $S_2$ | $S_{[1-2]*}$ | $C_1$ | $C_2$ |
| SPLOT- | | | 1 % | 69.68 | 7.72 | 3544.49 | 3553.85 | 0.98 | 0.08 |
| FM-50- | 517 | 9.6 | 2 % | 72.61 | 15.92 | 3643.55 | 3661.15 | 0.96 | 0.01 |
| SAT-1 | | | 5 % | 90.3 | 47.91 | 3218.32 | 3268.04 | 0.81 | 0.0 |
| SPLOT- | | | 1 % | 71.16 | 5.71 | 3115.95 | 3123.34 | 0.99 | 0.34 |
| FM-50- | 511 | 9.4 | 2 % | 71.48 | 12.78 | 3004.28 | 3018.68 | 0.95 | 0.25 |
| SAT-2 | | | 5 % | 88.83 | 41.88 | 2759.61 | 2803.5 | 0.82 | 0.16 |
| SPLOT- | | | 1 % | 255.7 | 21.15 | 18456.9 | 18481.16 | 0.78 | 0.26 |
| FM-100- | 1034 | 9.6 | 2 % | 274.52 | 60.65 | 9239.41 | 9301.71 | 0.36 | 0.5 |
| SAT-1 | | | 5 % | 329.89 | 224.79 | 245.76 | 470.83 | 0.01 | 0.0 |
| SPLOT- | | | 1 % | 217.36 | 23.83 | 15786.21 | 15812.61 | 0.74 | 0.38 |
| FM-100- | 1036 | 9.4 | 2 % | 227.58 | 56.24 | 5785.63 | 5842.91 | 0.25 | 0.06 |
| SAT-2 | | | 5 % | 283.67 | 184.34 | 0.0 | 184.56 | 0.0 | 0.0 |

**Table 4.** Time (ms) of execution of the configuration algorithms applied to customized FDs with multiple parent features

| | | | | Configuration | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Cust. | $FD_s$ | ECR(%) | RCS | $S_0$ | $S_1$ | $S_2$ | $S_{[1-2]*}$ | $C_1$ | $C_2$ |
| $F_2 = 2$ | | | 5 % | 44.33 | 57.26 | 346.2 | 405.2 | 1.36 | 1.26 |
| $\wedge$ | 287 | 11.8 | 15 % | 85.6 | 147.34 | 358.66 | 509.93 | 1.83 | 1.43 |
| $F_3 = 0$ | | | 25 % | 135.71 | 189.38 | 281.41 | 474.49 | 1.95 | 1.0 |
| | | | 35 % | 209.7 | 205.67 | 216.59 | 426.25 | 1.9 | 0.9 |
| $F_2 = 2$ | | | 5 % | 47.45 | 59.96 | 384.5 | 447.08 | 1.36 | 1.3 |
| $\wedge$ | 287 | 11.8 | 15 % | 85.27 | 266.1 | 735.23 | 1006.87 | 3.57 | 2.83 |
| $F_3 = 1$ | | | 25 % | 140.35 | 470.29 | 881.43 | 1363.0 | 6.0 | 3.0 |
| | | | 35 % | 229.68 | 559.04 | 829.98 | 1404.13 | 6.6 | 4.6 |
| $F_2 = 2$ | | | 5 % | 47.32 | 218.84 | 1654.22 | 1881.47 | 4.58 | 6.5 |
| $\wedge$ | 287 | 11.8 | 15 % | 95.34 | 1303.77 | 4469.7 | 5810.7 | 20.77 | 12.8 |
| $F_3 = 3$ | | | 25 % | 152.26 | 3240.44 | 8502.7 | 11836.78 | 45.95 | 26.45 |
| | | | 35 % | 254.95 | 3762.53 | 10133.82 | 14053.74 | 57.6 | 43.2 |
| $F_2 = 3$ | | | 5 % | 42.65 | 216.88 | 1680.09 | 1903.33 | 4.56 | 5.7 |
| $\wedge$ | 287 | 11.8 | 15 % | 89.61 | 1473.03 | 5867.12 | 7380.1 | 22.73 | 18.8 |
| $F_3 = 3$ | | | 25 % | 151.83 | 3737.02 | 10166.15 | 14012.18 | 56.65 | 31.2 |
| | | | 35 % | 251.19 | 8595.86 | 30783.49 | 39636.66 | 115.2 | 97.2 |

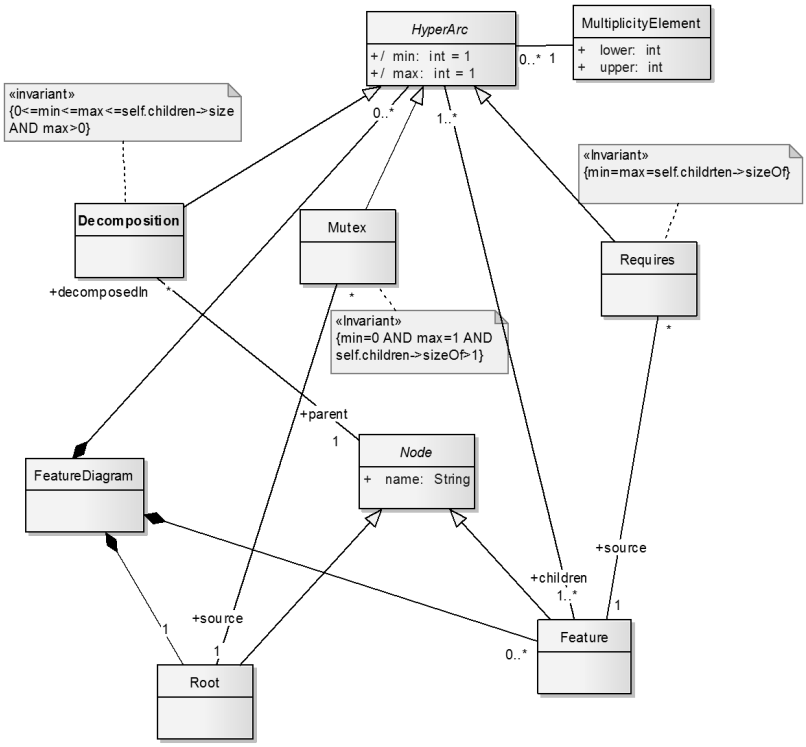


**Fig. 6.** Detail of the proposed Feature Meta-model [14]

feature diagrams mentioned in Section 2 to be covered. The details of the hypergraph based constrained meta-model are shown in Fig. 6. *HyperArc* and *Node* are abstract meta-classes. A *FeatureDiagram* has a *Root* (feature), a set of zero or more (non-root) *Features*, and a set of *Decompositions*. Each Decomposition connects a parent Node (Root or Feature) with one or more children Features. As multiplicity of children meta-association indicates, a Feature can be the child of more than one Decomposition and, indirectly, of a parent Feature. (If we change the multiplicity to 1..1, we convert 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: the maximum value (upper) must be less than or equal to the number of children of the Decomposition. Finally *Mutex* and *Requires* meta-classes are specializations of HyperArc with the adequate invariant to express the semantics of these constraints, as explained in Section 2.2: the fixed multiplicity of Mutex is 0..1, the multiplicity of requires is n..n (if n is the number of involved children, 1..1 being the typical situation). A basic implementation was presented in [14]. As part of our

industrial oriented work, we have previously implemented a Feature Modeling Tool (FMT)[4] as a *plug-in* for Microsoft Visual Studio IDE. The meta-model we used was based on constrained trees, validation was external, and the configuration used a staged approach. Work in progress includes the change of the internal meta-model and the incorporation of the validation and configuration algorithms.

## 5. Related Work

Starting with the original FODA proposal [11], several variants of feature diagrams have been proposed: FORM [12] 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. [18] 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 [20]. 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.

Most authors (see [7] for example) deal with the structural constraints implicit in the features tree (or graph) independently of the additional mutex/requires constraints. The definition of the complete feature models, therefore, requires working with graphs (the structure) and logical expressions (the constraints). Some recent works are devoted to the global validation of feature models, mainly based on propositional formulas [1] or constraint solvers [2].

Batory [1] uses a grammar and propositional formulas to validate the PL and each PL configuration. A sound connection between FDs, grammars, and propositional formulas was established and a system (*logic truth maintenance system*) enabling the propagation of constraints as user select features, to avoid inconsistent product specifications, was proposed. Furthermore, rules for transformation between feature models and grammars and a tool named GUIDSL, which takes a grammar as input and provides a graphical user interface to create configurations, was provided. Mendonça et al. use a two stage analysis to validate the models [17]. 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 formal-

[4] http://giro.infor.uva.es/FeatureTool.html

ism. Modeling and transformation tools are consequently easier to define and implement.

## 6. Conclusions and Future Work

In this article, we have used F-hypergraphs to describe the abstract syntax of feature diagrams and their configuration. A valid feature diagram configuration is defined as a subset of its features, where the root is always present and the rest of the features satisfy two properties (at least one of its *structural* parents is present and for each leaving hyperarc at least minimum and at most maximum features of the hyperarc head are also present in the configuration). A configuration procedure has been defined and implemented.

Once the formal definition is stated, the construction of an extensible feature meta-model has been dealt with. The algebraic definition directly yields the required invariants, establishing a firm foundation for the meta-model. The advantages of simplicity and extensibility have made it possible to build modeling feature tools compatible with the different flavors of feature diagrams.

Work in progress includes the revised version of FMT, which will incorporate internally the proposed meta-model and the implemented configuration algorithms. The algorithms themselves are being optimized for their tool utilization, using FD preprocessing. Basically, the independent application of the first algorithm to each feature allows a sub-hypergraph (or an ordered set of them) to be associated with it, so that a configuration can be found faster by combining the sub-hypergraphs related to each feature (i.e., the union of features and hyperarc sets). If a valid partial configuration results, the second algorithm is applied to complete the configuration.

## References

1. Batory, D.S.: Feature models, grammars, and propositional formulas. In: Obbink, J.H., Pohl, K. (eds.) Software Product Lines, 9th International Conference, SPLC 2005, Rennes, France, September 26-29, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3714, pp. 7–20. Springer (2005)
2. Benavides, D., Martín-Arroyo, P.T., Cortés, A.R.: Automated reasoning on feature models. In: Pastor, O., e Cunha, J.F. (eds.) Advanced Information Systems Engineering, 17th International Conference, CAiSE 2005, Porto, Portugal, June 13-17, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3520, pp. 491–503. Springer (2005)
3. Bosch, J.: Design and Use of Software Architectures: Adopting and Evolving a Product Line Approach. Pearson Education (Addison-Wesley & ACM Press) (May 2000)
4. Cechticky, V., Pasetti, A., Rohlik, O., Schaufelberger, W.: Xml-based feature modelling. In: Software Reuse: Methods, Techniques and Tools: 8th International Conference, ICSR 2004, Madrid, Spain, July 5-9, 2009. Proceedings. Lecture Notes in Computer Science, vol. 3107, pp. 101–114. Springer (2004)
5. Czarnecki, K., Eisenecker, U.W.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley (2000)

6. Czarnecki, K., Helsen, S., Eisenecker, U.W.: Formalizing cardinality-based feature models and their specialization. Software Process: Improvement and Practice 10(1), 7–29 (2005)
7. Czarnecki, K., Helsen, S., Eisenecker, U.W.: Staged configuration through specialization and multilevel configuration of feature models. Software Process: Improvement and Practice 10(2), 143–169 (2005)
8. Gallo, G., Longo, G., Pallottino Sang, S.: Directed hypergraphs and applications. Discrete Applied Mathematics 42(2-3), 177–201 (1993)
9. Gallo, G., Scutella, M.: Directed hypergraphs as a modelling paradigm. Decisions in Economics and Finance 21(1), 97–123 (1998)
10. Griss, M.L., Favaro, J., d'Allessandro, M.: Integrating feature modeling with the RSEB. In: Devanbu, P., Poulin, J. (eds.) Proceedings: Fifth International Conference on Software Reuse. pp. 76–85. IEEE Computer Society Press (1998)
11. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, S.: Feature-oriented domain analysis (FODA) feasibility study. Tech. Rep. CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University (Nov 1990)
12. Kang, K.C., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: Form: A feature-oriented reuse method with domain-specific reference architectures. Annals of Software Engineering 5, 143–168 (January 1998)
13. Laguna, M.A., González-Baixauli, B., Marqués Corral, J.M.: Seamless development of software product lines: Feature models to uml traceability. In: Sixth International Conference on Generative Programming and Component Engineering (GPCE 07). Salzburg, Austria. ACM Press (oct 2007)
14. Laguna, M.A., Marqués Corral, J.M.: Feature diagrams and their transformations: an extensible meta-model. In: IEEE proceedings of the EUROMICRO SEEA (2009)
15. Lau, S.: Domain analysis of e-commerce systems using feature-based model templates. Msc thesis, ECE Department, University of Waterloo, Canada (2006)
16. Mendonça, M., Branco, M., Cowan, D.D.: S.p.l.o.t.: software product lines online tools. In: Arora, S., Leavens, G.T. (eds.) Companion to the 24th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2009, October 25-29, 2009, Orlando, Florida, USA. pp. 761–762. ACM (2009)
17. Mendonça, M., Cowan, D.D., Malyk, W., de Oliveira, T.C.: Collaborative product configuration: Formalization and efficient algorithms for dependency analysis. Journal of Software 3(2), 69–82 (2008)
18. Riebisch, M., Böllert, K., Streitferdt, D., Philippow, I.: Extending feature diagrams with UML multiplicities. In: Proceedings of the Sixth Conference on Integrated Design and Process Technology (IDPT 2002). Pasadena, CA (jun 2002)
19. Rodríguez-Cano, G.: Configuration of Hyper-Graph based Feature Diagrams. Master's thesis, Department of Information Technology (2010), `http://uu.diva-portal.org`
20. Schobbens, P.Y., Heymans, P., Trigaux, J.C., Bontemps, Y.: Generic semantics of feature diagrams. Computer Networks 51(2), 456–479 (2007)

**Miguel A. Laguna** is Associate Professor in the Department of Computer Science at the University of Valladolid (Spain). He graduated and received a PhD from the University of Valladolid. He is the leader of the GIRO research group, specialized on systematic software reuse and software refactoring.

**Jose Manuel Marques** is Associate Professor in same Department. He graduated at the Complutense University of Madrid and obtained a PhD in Computer Science from the University of Valladolid. His research interests include OO modeling, software reuse and requirements engineering

**Guillermo Rodrguez-Cano** received his degree in Computer Science in 2008 from the University of Valladolid, and obtained a Master degree in 2010 at the Uppsala University, Sweden. His research interests include software product lines and model-driven architecture.

*Received: August 4, 2010; Accepted: May 9, 2011*