

Criteria for Comparing Requirements Variability Modeling Notations for Product Lines

Olfa Djebbi

*CRI, University Paris 1 - Sorbonne
90, rue de Tolbiac, 75013 Paris, France
olfa.djebbi@malix.univ-paris1.fr*

Camille Salinesi

*CRI, University Paris 1 - Sorbonne
90, rue de Tolbiac, 75013 Paris, France
camille.salinesi@univ-paris1.fr*

Abstract

Software product families have proven to be an effective approach to reuse in software development.

For planning requirements reuse, several variability approaches are developed.

This study is made in an industrial company producing blood analysis automatons. It aims at finding the most suitable notation to model requirements variability for the product line developed by the company.

The paper provides a comparative survey on feature-based notations for requirements variability modeling. It introduces an evaluation framework based on criteria that are derived by studying the main engineers' expectations for such a notation. The evaluation is fulfilled by making out notations' metamodels. The use of these languages is systematically illustrated with the same example, adapted from the company context, in order to refine the notation selection approach. Finally, recommendations are done, and issues on making the approach systematic are discussed.

1. Introduction

This paper reports our experience in an industrial company developing blood analysis automatons. These machines are embedded and real-time systems. They communicate with external systems like hospitals, laboratories and remote control hosts. Several stakeholders (biological technicians, biological researchers, mechanical engineers, electronic engineers, software engineers, norm experts, industrialization specialists and marketers) collaborate to their development.

When the company started, it was a small laboratory. The first developed automaton was constructed by a group of friends. They did not apply any development process, and did not elaborate any

specification document. It was rather a pragmatic development following the 'codex fix' kind of paradigm. Faced to a diverse and ever-increasing demand, the group of developers was enlarged. The communication and collaboration between the different co-workers became more and more difficult. Due to the lack of a formal process, the company continued to produce automatons in an 'ad-hoc' manner. In a highly competitive environment, and in order to guarantee a better productivity, the company felt the need for a method to coordinate specification management, and tools that enable development of automatons by extension/adaptation of former ones.

The automatons produced by the company fit into a product line: all of them share the same core part with the main blood analysis functionalities. Each automaton has also its own characteristics and differs from the others. These variable parts can be as simple as color, weight or user interface of the machine; or more complex such as programming technology, capacity in term of number of plasma tubes handling and mechanical and electronic technologies.

This problem is already addressed in the field of software product lines (PL) engineering. Several methods have been published to handle with supporting variability during software product line development. However, while variability affects the whole product line process, from requirements to code, the variability issues are mainly addressed in design and implementation phases. In our case, variability must be already managed at the requirements engineering phase to enable efficient communication between the designers and the stakeholders about requirements and thus variability.

Some methods are proposed to model requirements variability explicitly. However, they are diverging in terminology and processes, and no standard notation neither tool has been adopted yet. Therefore, the first challenge is to find out the suitable method to apply to the particular domain of haemostasis diagnosis.

This step belongs to a wider project that is to develop a derivation process for the analysis automata from the product line specifications. These specifications include requirements variability, hardware and software reference architecture variability, as well as the relationship between them.

At this moment, the paper deals with a part of our research project. It tries to meet up the first challenge by finding an appropriate notation to model requirements variability for blood analysis automata.

This paper aims at providing a comparative survey on the different techniques to model requirements variability. It introduces an evaluation framework based on criteria that are derived by studying the main requirements engineers can expect from a notation designed to model variability. The criteria are first presented independent of any approach. They are then applied to a specimen of approaches to illustrate the main characteristics of the notations, and to show some of the notations' pros and cons.

Several families of approaches were identified according to the formalism they use: those based on features, on use cases, on UML classes, on goals and on aspects. This study focuses on feature based approaches. This family was chosen because features represent the first mean conceived to represent variability. They are now widely used. Besides, much research on variability is elaborated through features, notably researches on variability dependencies. Moreover, in order to get a better insight in the notations, the paper illustrates them with a case study that we have adapted from our company context.

First, this paper introduces in section 2 a comparison framework for evaluating requirements variability modeling notations. Then, in section 3, the selected notations are briefly presented and illustrated with the case study. In section 4, notations are compared against the framework. Then, recommendations to choose the most suitable one is presented in section 5. Section 6 discusses the validity of the approach, its limitations, and the further work. Section 7 closes the paper. Finally, the appendix in section 9 regroups all the paper figures.

2. The evaluation framework

Today, there is a host of requirements variability modelling techniques, but little has been done to evaluate them [1] [2]. As result, there are too few decision-making criteria for deciding whether or not to use a given method.

This paper seeks to perform a comparative survey on relevant feature based notations, with the aim of finding an explanation for this diversity and studying

which method is the most appropriate to the company situation. The question is, however, how to evaluate those notations? In other words, what characteristics should have such notations to satisfy all system stakeholders' needs?

A brainstorming session gathering stakeholders was organized. Discussions, aided by small examples, have led to elaborate an evaluation framework. This framework is an attempt to provide a common conceptual ground for comparing and contrasting different ways to undertake requirements variability.

The proposed list of basic requirements, and their measurement parameters, is:

1. The notation should graphically visualize, in a readable form, common and variable parts of a product line. Readability expresses the facility to visually apprehend a model. Its measurement is based on two sub-criteria: clearness and minimality.

Clearness is a purely aesthetic criterion founded on the graphic arrangement of the elements that make the diagram. If the model is complex, the representation contains crossing arcs that penalize its readability.

A diagram is minimal if each concept is represented once and only one. A bad choice of the concepts of the model can involve a duplication of the relations between these concepts, hindering its minimality.

2. The notation has to be simple and expressive. A diagram is expressive if it represents obviously the user's needs and can be easily understood without additional explanation. A diagram is simple if it contains a minimal number of objects. The complexity of a model is related to the number of relations between these objects. Consequently, a diagram is so simple as the number of entities is higher than the number of its relations.

3. The notation should distinguish between types of variability: the notation must be able to explicitly express all different types of variability (options, alternatives, etc.)

4. The notation should allow specifying properties of variation points: it should provide means to explicitly model variation points and their characteristics aiding to make a decision about their selection, i.e. specification of binding times, justification of the variability, etc.

5. The notation has to represent dependencies between variable parts of the product line: the description of dependencies is necessary since variabilities are not independent of each other (one can restrict, require ... the other).

6. The notation should support the evolution of the product line: as models will evolve over time, it should be easy to add new requirements, to integrate change and update already modeled requirements. The notation should also capitalize company's experience

and manage the traceability of the models' updates over time.

7. The notation has to be adaptable: it should be flexible in order to fit each company specific needs.

8. The notation should be scalable: it has to allow modeling large-scale systems.

9. The notation has to be supported by a CASE tool and be integrated into existing tool kits: serious software development without appropriate CASE tool support is quite impossible. A tool facilitates automating handling of models and hence his large adoption by developers' community.

10. The notation should be unified into the whole product line development cycle: it should provide means to ensure traceability with the remainder development phases.

11. The notation should be able to be standardizeable: sharing a unique language promotes interoperability among users and tools. Standardizing a notation is an indication of its level of maturity and stabilization. To expedite the standardization process, features must have a precise semantics and avoid conflicting interpretations.

3. Overview of the approaches

As afore mentioned, four families of notations modeling requirements variability are considered, namely: features, use cases, UML classes, goals and aspects based approaches. As agreed upon, this study will focus on features based approaches, of witch we can enumerate: FODA (Feature-Oriented Domain Analysis) [3], FORM (Feature-Oriented Reuse Method) [4], FOPLE (Feature Oriented Product Line Software Engineering) [5], FeatuRSEB (Feature/Reuse-driven Software Engineering Business) [6], GP (Generative Programming) [7], FORE (Family Oriented Requirements Engineering) [8] and Bosch' approach [9].

Among them, some representative notations have been chosen, making thereby the specimen on which the evaluation framework will be applied. For each one of them, syntax is first described. Then, it is used to model a concrete example, so that the real difficulties of the notation are deduced. In order to make remarks plain and straight, the same case of study, inspired from the industrial background, will be used.

3.1. The case study

This simplified case study proposes to model some aspects of the product line produced by the considered company. It aims to apply the notations to a real case so that their real capacities are pointed up; but also to

draw up a feasibility study as for the possibility of using the considered notation to model the requirements variability for the company's instruments.

Thereafter, the nominal case of an automaton' use is described:

In order to make analysis tests, biologist loads the tubes of patients' plasma as well as the tubes of reagents in the instrument. While loading, tubes have to be identified manually or by a light pen, or both.

The biologist must then choose an analysis methodology and launch the tests. The instrument deals with the tubes, accomplishes analyses according to selected methodologies, makes measurements, and returns the results to the biologist.

By default, plasmas and reagents loading system is continuous. Moreover, the instrument must enable the realization of at least one of the following options: the loading of an urgent tube, the loading of other reagents of the market, and the multi-positioning loading (random positioning of the tubes).

Analysis methodologies can be static or parameterizeable. As for measurements, they are of three kinds: chronometric, colorimetric and immunological. The instrument must establish exactly two kinds of measurements. The results can be provided to the biologists either in gross unit (Sec, D.O/min, Δ D.O), or in calculated unit (INR, μ g/ml, UI/ml), or both. Besides, the instrument can be equipped with a printer to print tests' results; it can also transfer them to the hospital or laboratory' host.

Note that if the instrument allows the loading of the other reagents of the market, the biologist must be able to identify these tubes manually. Also this option can not exist if the biologist cannot parameterize correspondent analysis methodologies.

3.2. On the notion of Feature

This section introduces the notion of 'feature' and 'feature diagram'.

Kyo C. Kang, one of the main references considering features and feature diagrams for software product lines, defines a feature as "a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems".

The features are different from the functions, the objects and the aspects that are abstractions mainly used to specify the internal details of a system. The features are focused rather on the external characteristics of the products to identify their communalities and their variabilities. Using those variable characteristics, functions, objects and aspects are then elaborated in a reusable form [10].

A feature diagram constitutes a tree composed of nodes and directed edges. The tree's root represents the concept that is refined using mandatory, optional, mutually exclusive (alternative/ XOR-features) and OR-features. In feature diagrams, mandatory features are features always included in every product i.e. common features. Variable features are defined as all features except common features. Variation points are features that have at least one direct variable subfeature.

In the next subsections, four notations based on features will be analyzed: FOPLE's notation, FeaturSEB's notation, GP's notation and FORE notation.

3.3. FOPLE notation

The FOPLE method [5] was elaborated in 2002 as a refinement of the FORM and FODA methods.

Feature diagrams are trees capturing the relationships among features. All feature names are depicted in boxes. Like in FODA, a feature is mandatory unless an empty circle is attached above its name indicating an optional feature. Alternative features that are children of the same parent feature and connected by an arc, define the alternative features set from which almost one feature can be selected. More dependencies between features like «requires» or «excludes» can be expressed with composition rules to avoid graphical overload.

FOPLE extends FODA feature diagrams with four layers representing different viewpoints of the different stakeholders: the capability layer, the operating environment layer, the domain technology layer and the implementation technique layer. The formed graph constitutes the so called feature model.

FOPLE defines in a more precise manner the relations between features. There are three types of relations: 'composition', 'generalization/specialization' and 'Implemented by', that can be modeled throughout the different layers of the feature model.

In the appendix, the FOPLE meta-model and its syntax are represented respectively in figure 1 and figure 5.

The figure 9 presents the feature model of the analysis automaton product line. The case study formulated above has been respected, but some other details have been brought in order to complete the different layers of the model.

3.4. FeaturSEB notation

FeaturSEB [6] is a combination between the FODA method and the RSEB method [11].

The RSEB is a use case driven systematic reuse process, where variability is captured by structuring use cases and object models with explicit variation points and variants. FeaturSEB makes feature models from RSEB models and provides explicit steps for constructing and transforming such feature models.

To create feature diagrams, FeaturSEB introduced UML-like notational constructs. Hence, a feature model is a graph of features linked together by UML associations. The explicit representation of variation points, like it has been defined in RSEB use cases, is also introduced.

There are four important constructs in a FeaturSEB feature diagram:

- The composed-of relationship: indicates that a feature is composed of sub-features and is represented by a simple line.
- The existence attribute: determines whether a feature is optional or mandatory. An optional feature is distinguished by an empty circle above the feature name.
- The alternative or XOR-relationship: defines a feature as a variation point and its sub-features as variants from which only one can be bound. This variation point feature is represented by an empty diamond under its name and linked to its variants by a simple line.
- The OR-relationship: defines a feature as a variation point and its sub-features as its variants from which one or more can be bound. This variation point feature is represented by a black diamond under its name and linked to its variants by a simple line.

The figures 2 and 6 of the appendix illustrate respectively the meta-model and the syntax of the FeaturSEB notation.

FeaturSEB is used to model of the analysis automaton product line. The resulting model is presented in the figure 10.

3.5. GP notation

The generative programming [7] is an approach that makes it possible to generate automatically programs starting from precise specifications. The generation of the software is done by assembling reusable elementary components.

The GP feature diagram slightly extends the FODA notation with or-features:

- A mandatory feature is included in the description of a concept instance if and only if its parent is included in the description of the instance. A mandatory feature is represented with a simple edge ending with a filled circle.

- An optional feature may or may not be included if the parent is included. An optional feature node is represented with a simple edge ending with an empty circle.
- An alternative feature is a feature in a set of features from which only one can be chosen if the parent of this set is included.
- An or-feature is the parent feature of a set of alternative features which are pointed to by edges connected with a filled arc. If an or-feature is included in the description of the concept instance, then any of its sub-features can be included. A feature may have one or more sets of or- sub-features. An or-feature can also be an optional feature.

The figures 3 and 7 illustrate the GP meta-model as well as its syntax.

In the figure 11, the feature model of the analysis automaton product line following the GP notation is presented.

3.6. FORE notation

This notation [8] has been developed in 2003 by Reibisch et al. within the scope of the Alexandria project [12].

In this notation, feature diagrams are extended with UML multiplicities. Some multiplicities are implicitly represented in the previous notations (0..1 / 1 / 0..N / 1..N / N). But shortcomings appear as simple multiplicities, like $p/0..p/1..p/p..N/m..p$ ($1 < m < p < N$; N representing the total number of features in a set), can not be represented.

In the FORE notation, a feature diagram is a directed acyclic graph. Relations between features are expressed by edges that direction is indicated by a circle at its end. If this circle is filled, then the relation between features is said to be mandatory. If the circle is empty, the relation is optional.

Optional relations that originate from the same feature node can be combined into a set. Each relation can only be part of one set. A set has a UML-multiplicity that donates the minimum and maximum number of features to be chosen from it. Visually, a set is shown by an arc connecting all the edges that are part of the set. The multiplicity is drawn in the center of the arc.

Dependencies between features (e.g. “requires”) can be represented on the graph. For features that are not located in adjacent parts of the graph, and for clearness, such dependencies can be described separately in a textual form by using the UML Object Constraint Language (OCL) [13].

The FORE meta-model and its syntax are represented respectively in figure 4 and figure 8 of the appendix.

The figure 12 represents the analysis automaton product line modeled with the FORE notation.

4. Comparative Analysis

In the previous section, some existing feature based notations to model requirements variability are presented. These notations will be here evaluated to determine to what extent they address the requirements presented in section 2. The table below shows the result of this evaluation. A black square denotes full support for the considered requirements; while a white square denotes no support, and the two-colored square denotes support with restrictions.

Table 1: Notations comparative table

	FOPLE	FeatuRSEB	GP	FORE
Readability	□	■	◻	◻
Simplicity and expressiveness	□	◻	■	■
Type distinction	◻	◻	◻	◻
Documentation	◻	◻	■	■
Dependencies	◻	◻	◻	■
Evolution	□	◻	□	■
Adaptability	□	□	□	■
Scalability	□	◻	◻	◻
Support	□	□	■	◻
Unification	□	◻	□	■
Standardizeability	□	□	□	◻

Readability: All notations make available graphical means to visualize variability requirements (c.f. schemas of notations’ syntax). Nevertheless, no one except FeatuRSEB, allows the explicit distinction of variation points, while FORE allows distinguishing features that are used only for structuring the model (represented by a gray rectangle). Generally, we can consider that a variation point is a feature that is refined into other optional and/or alternative subfeatures.

Modeling four layers in FOPLE’ notation generates much complexity. Moreover, relationships can be modeled throughout the layers. For a great amount of features, relationships will crosscut and the model will take on more than one page, making it illegible.

As for the remaining notations, they represent systems only on one view. FeatuRSEB and GP' notations organize model elements as a tree, with maximize their clearness. In FORE notation, a feature diagram is a graph. But, since it is an acyclic graph, it had minimal effect on its clarity.

The duplication of concepts or relationships hinders the minimality of the model. Yet, due the notion of feature itself, this situation may occur. Indeed, some features are non functional features and have to be associated to other functional features. This leads to duplicate either dependency relationships or features.

Simplicity and expressiveness: The concept itself of 'feature' is quite intuitive. So, feature notations express obviously users' needs.

As shown in the notations' syntax figure, FORE and GP' diagrams are the simplest because they have less constructs. By using UML constructs, FeatuRSEB notation does not need to be further explained. Nevertheless, FOPLE' notation contains the most objects and relationships. The same feature model encompasses customer' needs, physical architecture, business data, and software/ physical implementation techniques. It represents also all dependency relationships as well as composition, generalization, specialization and implementation relationships that can relate all these features.

Type distinction: All notations allow modeling commonalities and variabilities. FOPLE models optional and alternative features. FeatuRSEB and GP add OR-features. Finally, due to the graph nature of FORE diagrams, it attributes optionality, enriched with cardinalities, to relationships rather than features.

As for the architecture variability, only FOPLE allows representing it in low level layers. For a product line of embedded systems, the physical architecture undergoes itself some variation from one product to another. So, it can be modeled in an independent feature diagram with various levels of abstraction and where various types of relations are used.

Documentation: In FOPLE and FeatuRSEB notations, variability is commented in separated documents that are the dictionary, the feature definition and the justification documents. They contain various information such as feature source, feature type, binding time, etc.

GP notation specifies more precisely binding times and modes, and adds other information such priority, availability, stakeholders, constraints, etc.

FORE integrates within its data model all needed information through glossary, references, stakeholder model, etc.

Dependencies: Features are not independent entities. Selecting, adding or removing a feature from the system has an impact on the other features. It is thus very useful to study the interactions between features early in the development cycle in order to avoid the undesirable ones and to envisage those with cause side effects [14].

In FOPLE, FeatuRSEB and GP notations, the 'requires' and 'mutex' relationships are expressed but not expressive. Indeed, as mentioned in [15] and validated in our case study, confusion exists between composition and "requires" relationships, as well as between exclusion and alternative relationships. The figure 13 illustrates this confusion.

Moreover, in practice, other dependencies relationships should be expressed. They can be more complicated to model, especially if they interfere with more than two features.

To define constraints, FORE uses FCL (Feature Constraints Language), a language similar to OCL. Thus, the feature model can be checked in an automated way for consistency and the selection of each feature set can be validated against the constraint rules.

With the great amount of features and thus dependencies that can have large-scale systems, notations have to take up a new challenge that is the management of the inconsistencies and the conflicts between the various constraints.

Evolution: Traceability is a prerequisite for successful integration of changes into the system without neglecting its consistency. Traceability over time is also required to save features history. Evolution may consist on the integration of new features, but also, on the update of the relationships and the features types themselves (mandatory, optional ...).

Neither FOPLE nor GP supports product line evolution. In FeatuRSEB, although links between features and requirements exist, traceability over time is not ensured.

However, the FORE data model is able to store and maintain information elaborated. Thus, FORE has built-in traceability and keeps data consistent.

Adaptability: Only FORE method satisfies this requirement. It is composed of three main building blocks. The requirements engineering phase for the system family, the data model holding all development assets and the requirements engineering phase for an application derived out of the family. Within the requirement engineering phase for the system family, there is a tailoring step where the document model is

set up, to reflect the company specific structure of specification, user and developer documents.

Scalability: The case study demonstrates that FOPL notation is not adequate for large scale systems. Although the case study is extremely simplified compared to what is in reality, the model took a whole page. While it is interesting for the company to model features related to the physical architecture of the automations, it would be very difficult, in real projects, to model all features from all views and with all the relations that can straggle throughout these views. That makes the model much complex for a re-use or a possible evolution.

As for the remaining notations, they can be scalable, if they are supported by CASE tools, and then handled automatically.

Support: The GP notation is the only notation that has dedicated CASE tools. It is supported by the AmiEddi [16] and its successor Captain Feature [17] tools, developed at Waterloo University. They are GUI-based tools for interactively editing feature models that are stored into a repository in an XML format.

FORE uses a complete data meta-model that is described using UML and implemented with XML. It is intended to form an XML-based standard for storing the information elaborated in the requirements engineering phase of system family development. Thus, it can be fully integrated into current family development environments. A prototype is under development.

Other tools such as pure::variants [18] [19] and FeaturePlugin [20] are feature modeling plug-in for Eclipse. They have their own feature meta-model.

All these CASE tools are integrated into existing tool kits. Eclipse is an open source tool that has many plug-in for the different modeling stages. Moreover, the XML format allows import/export between different tools.

Unification: The expression of the product line variability should be the onset of the development life cycle. However, it is difficult to establish the link between features and the other notations of the remainder phases, in particular the design and architecture models (UML, ERM, UC models ...). Indeed, information on variability is localized in the feature models and is not easily transferable to other artifacts [21].

The FeatuRSEB notation is less elaborated compared to the FOPL notation. However, its merit is that it restores the link between features and requirements modeled by use cases.

In [22], Riebeish proposes that the feature model is expressed from the customer point of view. So, it should capture the product line variability and support the other development activities by creating traceability links with the requirements, design and implementation models. Thus, FORE uses a complete data model containing interconnected requirements, features and explaining UML-models [23].

Standardizeability: The feature notation is not standardized and there is no consensus on the definition of features and feature diagrams. To be standardized, a feature notation has to help to understand the system more efficiently and with less misunderstanding.

Nevertheless, some ambiguities are revealed in representing optional and alternative features as well as their relationships.

For instance, in the FOPL notation, the alternative feature concept does not have a clear definition. It is defined as a member of a feature set from which only one can be selected. However, its type is not précised. So, in the literature, alternative features are sometimes represented as optional ones [24] [15], other times as mandatory ones [25]. This situation generates interpretation ambiguities (see table 2).

Difficulties to distinguish between feature relationships are also encountered. As an example, a composition relationship binds an abstract feature (feature Concept) to a more concrete feature and can thus be confused with a generalization relationship. In the case study, this difficulty is noted with the features “tubes identification” and “methodologies list”.

Ambiguities are also encountered in considering views. As an example, GUI languages that are customer visible system properties, can be also considered as operational environment feature, especially when it is a requirement of FDA norms.

In the other hand, GP notation allows the coexistence of OR and XOR links with optional and mandatory features. So several ambiguities are generated additionally to those revealed in the FORM notation. Indeed, it implies, for the same feature, two choices on two different levels that can be conflicting (see figure 14).

For instance, in (2), considering the OR relation, a nonempty feature set must be selected. Given that the resulting set contains optional features, we can choose once again. So, we can obtain an empty set, which leads to an inconsistent situation.

Other ambiguities are generated by similarities of interpretation for different relation combinations between features. They are identified in the table 3.

FeatuRSEB presents also some ambiguities in modeling feature relationships. Indeed, it is difficult,

for a feature, to decide whether it is optional or related with an OR-relation. Moreover, the possibility of optionality and OR/XOR relations coexistence at variants as well as at variation points is not indicated.

We believe that it would be judicious to enrich notations by usage rules, albeit obvious, in order to formalize representations and avoid ambiguities. An example of such rules would be:

- A “requires” dependency link must point on an optional feature or on the value of a mandatory feature, but never on a mandatory feature.
- A variable feature must have at least one mandatory subfeature or a set of alternative subfeatures.
- Alternative features are conventionally represented as mandatory features.
- An exclusive dependency link cannot point on a mandatory feature.

Some other ambiguities are revealed in representing the features to be selected within a set. FORM, GP and FeatuRSEB notations allow some but not all possibilities (see table 4).

The FORE notation takes out those ambiguities. In FORE notation, all variable features of a feature set are optional. The number of features to be chosen is specified by cardinalities.

However, in all notations there are difficulties to model features witch nature does not depend on other features but on environmental constraints.

Indeed, non-functional features that are associated to functional features can be modeled through dependency relationships such “requires” relationship. But, when the non-functional character of the feature is not absolute and depends on the configuration of the system, the usual dependencies relationships do no longer solve the problem. An example relating to the case study is the feature “sequencing arms algorithm” into the figure 1. There are four types of algorithms according to the number of arms. Each one of them can or not be parameterized depending on whether the system has or not static methodologies. Thus, representing this association with “requires” or “mutex” relationships is not a suitable mean. In the model, an additional feature (“parameterized algorithm”) is added to encounter this problem. A more efficient means would be to extend the current dependencies relationships format by conditions (e.g. feature-A [condition1][...] feature-B). It adds complexity to the feature model since it requires the management of the conditions in addition to the dependencies. Besides, this form of relations and then some can be ensured by the Object Constraint Language (OCL) [13], as FORE notation already does.

5. Recommendation

Managing its business knowledge according to a product line approach is very beneficial for the company. By selecting and thus reusing existing components, the application engineers can develop new applications within a short time period and with less resources compared to conventional software development.

For the company, the two most important criteria for selecting a modeling variability notation are modeling requirements and architectural variabilities as well as their dependencies. Besides, notation should be expressive, unambiguous and easy to understand.

Indeed, modeling optional and alternative requirements as well as their related variable architectural features can be considered during the value analysis in the feasibility phase of projects with cost objectives. Moreover, dependencies between requirements variability and technical and architectural variability is an important tool of negotiation between the company and its customers. Besides, variability sets can be defined, evaluated, classified and incorporated in the Requirements Specification Document. Decisions about their selection will be made in later development stages.

The following extract of the table presented above shows the main company’ requirements and how different notations address them.

	FOPLE	FeatuRSEB	GP	FORE
Type distinction	☐	☐	☐	☐
Dependencies	☐	☐	☐	■
Standardizeability	☐	☐	☐	☐
Simplicity and expressiveness	☐	☐	■	■
Readability	☐	■	☐	☐

As we can notice, FORE is the most suitable notation as it supports the majority of the presented criteria.

Unfortunately none of the notations is adequate for modelling variability in embedded product lines, though recent studies aim at adapting FORE to hold this kind of systems [26]. Since product families are often embedded systems, supporting the physical dimension in the product lines engineering becomes a genuine challenge.

6. Validity issues

In this paper, a comparative framework for evaluating requirements variability modeling notations is presented. Using a real case study, a model is constructed for each notation. Then system stakeholders have evaluated them basing on the framework criteria.

Stakeholders are mainly architects and analysis engineers who have a big picture on the whole system.

However, they have judged the quality of the different models in a subjective manner. No measurements were realized to formalize these judgments.

Further research will focus on the refinement of the criteria and their measurements. Questionnaires are being elaborated including measurements techniques (like statistics, curves, etc.) to evaluate the ease of modeling, the effort expended, etc.

Other stakeholders having other responsibilities within the system, as well as academic researchers, will be also invited to collaborate in this work.

Moreover, the repeatability of the comparison process will be studied. The purpose is to define a systematic process for utilizing the comparative table in different companies.

We propose that the user defines his needs according to his specific situation. Viable approaches are then identified and criteria are balanced and attributed to the required situation. Columns set up the approaches by indicating how they satisfy the various elements of the considered situation. Thereby, a recommendation can be established by applying a multi-criteria decision-making technique.

7. Conclusion

This study is made in an industrial context. Its purpose is to find the most suitable notation to model requirements variability for the product line developed by the company.

The paper has compared four feature-based notations for requirements variability modeling: FOPLE, FeaturSEB, GP and FORE' notations, according to a specially developed evaluation framework.

The list of criteria is not exhaustive but it gathers the principal elements collected during the investigations of users' satisfaction.

The evaluation is fulfilled by making out notations' metamodels. The use of these languages is systematically illustrated with the same example (that we have adapted from the company context) in order to refine the notation selection approach.

The approach viability depends mainly on the quality of the defined criteria and their measurements. The approach was validated firstly by debriefing industrial stakeholders about their expectations for a modeling variability notation. Then, drawn recommendations were confronted against users' needs and provided their satisfaction.

Further collaborations aim at validating over again this approach by variability and quality models experts like Patrick Heymans and John Krogstie.

8. References

- [1] John Krogstie, "A Semiotic Approach to Quality in Requirements Engineering", IFIP WG 8.1 *Conference on Organizational Semiotics*, Montreal, 2001.
- [2] P-Y. Schobbens, P. Heymans, J-C. Trigaux, Y. Bontemps, "Feature Diagrams: A Survey and A Formal Semantics", to appear in the proceedings of the International *RE Conference*, Minneapolis, USA, September 2006.
- [3] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study", Technical Report CMU/SEI-90-TR-21, *Software Engineering Institute*, Carnegie Mellon University, November 1990.
- [4] K. Kang, S. Kim, J. Lee, G. Kim and E. Shin, "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures", *Annals of Software Engineering 5*, 1998, pp. 143-168.
- [5] K. Kang, K. Lee, J. Lee, "FOPLE - Feature Oriented Product Line Software Engineering: Principles and Guidelines", Pohang University of Science and Technology, 2002.
- [6] M. Griss, J. Favaro, M. Allesandro, "Integrating Feature Modeling with RSEB", *Proceedings of the Fifth International Conference on Software Reuse*, Vancouver, BC, Canada, 1998.
- [7] K. Czarnecki, U.W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*, Addison Wesley, 2000.
- [8] D. Streitferdt, *Family-Oriented Requirements Engineering*, PhD Thesis, Technical University Ilmenau, 2004.
- [9] J. v. Gorp, J. Bosch, and M. Svahnberg, "On the Notion of Variability in Software Product Lines", in *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA)*, 2001.
- [10] K. Kang, K. Lee, J. Lee, "Concepts and Guidelines of feature Modeling for Product Line Software Engineering",

Proceedings of the 7th International Conference on Software Reuse: Methods, Techniques, and Tools, 2002, pp. 62 - 77.

[11] I. Jacobson, M.L. Griss, P. Jonsson, *Software Reuse: Architecture, Process, and Organization for Business Success*, Addison-Wesley, May 1997.

[12] <http://www.theinf.tuilmenu.de/~riebisch/pld/index.html>

[13] OCL 2.0 Specification Version 2.0, June 2005.
<http://www.omg.org/docs/ptc/05-06-06.pdf>

[14] J. P. Gibson, "Feature Requirements Models: Understanding Interactions", in *Feature Interactions, in Telecommunications IV*, Canada, IOS Press, June 1997.

[15] M. Riebisch, D. Streitferdt, I. Pashov, "Modeling Variability for Object-Oriented Product Lines", *ECOOP Workshop*. Springer, LNCS, 2003.

[16] <http://www.generative-programming.org/>

[17] <https://sourceforge.net/projects/captainfeature/>

[18] <http://www.software-acumen.com/purevariants/featuremodels/>

[19] O. Spinczyk, D. Beuche, "Modeling and Building Software Product Lines with Eclipse", Proceedings of the *International Conference on Object-Oriented Programming Systems, Languages and Applications*, 2004.

[20] M. Antkiewicz, K. Czarnecki, "FeaturePlugin: feature modeling plug-in for Eclipse", Proceedings of the *OOPSLA workshop on eclipse technology eXchange*, 2004.

[21] S. Bühne, K. Lauenroth, K. Pohl, "Why is it not sufficient to model requirements variability with features models?", Proceedings of the *Workshop: Automotive Requirements Engineering (AURE04)*, IEEE Computer Society Press, Los Alamitos, 2004.

[22] M. Riebisch, D. Streitferdt, I. Pashov, "Modeling Variability for Object-Oriented Product Lines", the *ECOOP Workshop*. Springer, LNCS, 2003.

[23] Detlef Streitferdt, "Integration of Current Models Towards Family Oriented Requirements Engineering", in IESE-Report of the Third *International Workshop on Software Product Lines: Economics, Architectures, and Implications*, 24th International Conference on Software Engineering, Orlando Florida, USA, 2002.

[24] M. Riebisch, "Towards a More Precise Definition of Feature Models", Technical University Ilmenau, Germany, 2003.

[25] S. Robak, B. Franczyk, K. Politowicz, "Extending the UML for modelling variability for system families",

International Conference on Algorithmic Mathematics and Computer Science, Vol.12, No.2, 285–298, 2002.

[26] Detlef Streitferdt, Periklis Sochos, Christian Heller, Ilka Philippow, "Configuring Embedded System Families Using Feature Models", in the *Object Oriented Software Design for Real Time and Embedded Computer Systems Workshop at Net.ObjectDays Conference*, Erfurt, September 2005.

9. Appendix

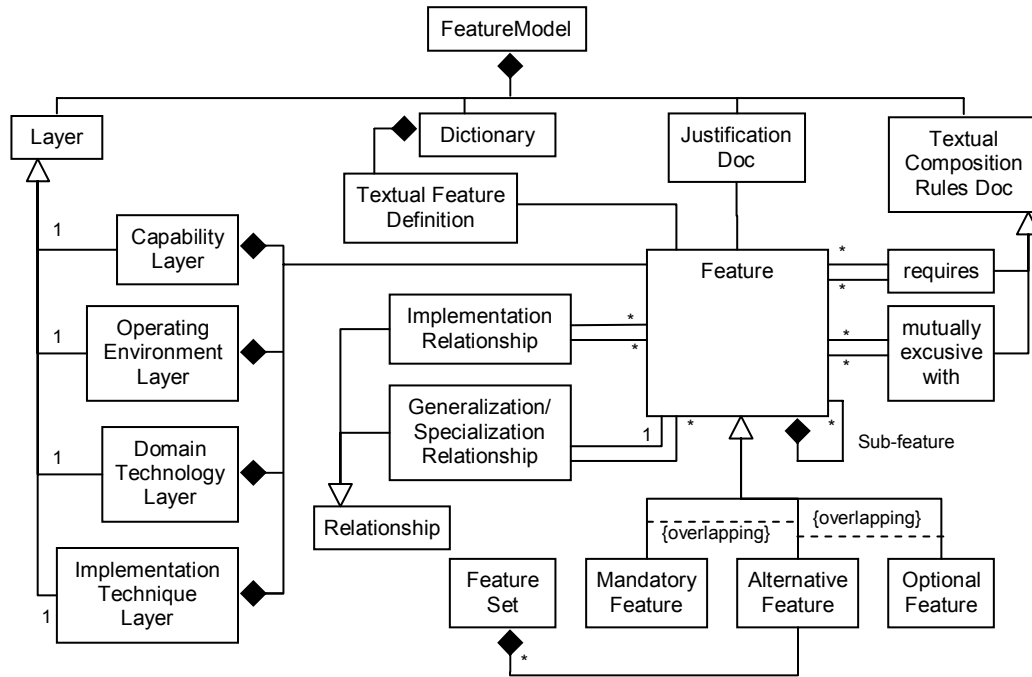


Fig 1: FOPLE' notation meta-model

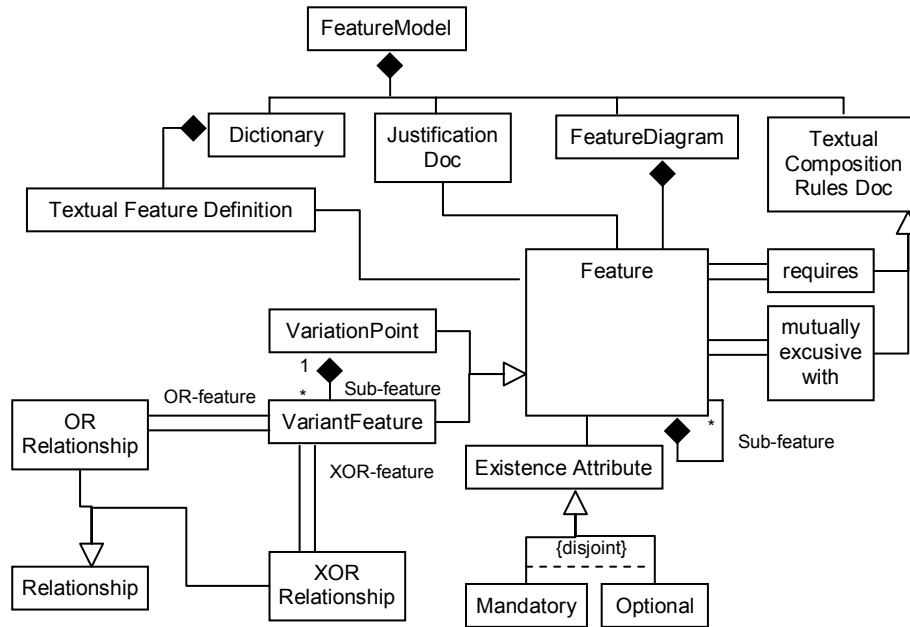


Fig 2: FeaturSEB' notation meta-model

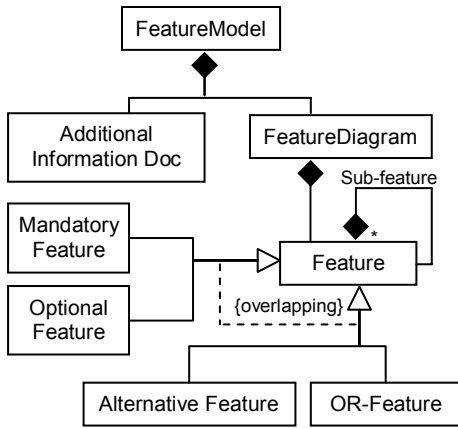


Fig 3: GP' notation meta-model

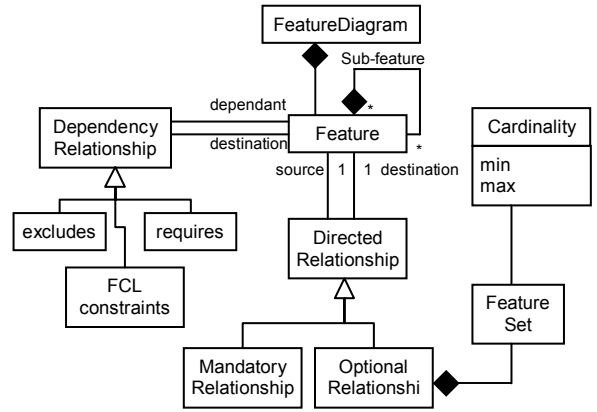


Fig 4: FORE' notation meta-model

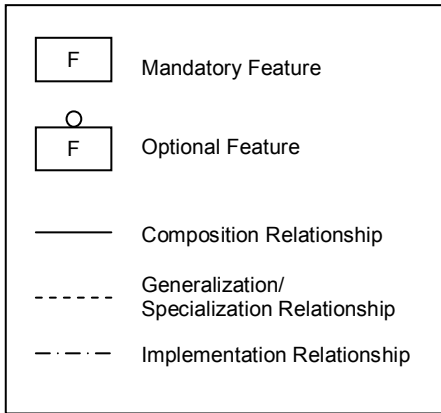


Fig 5: FOPLE' syntax

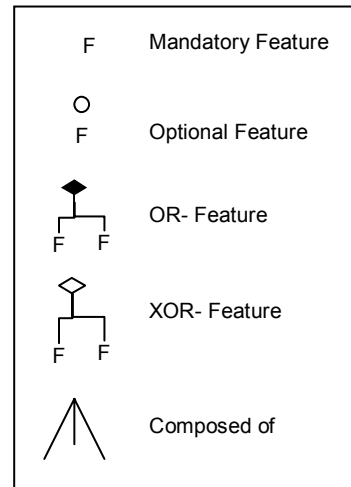


Fig 6: FeatuRSEB' syntax

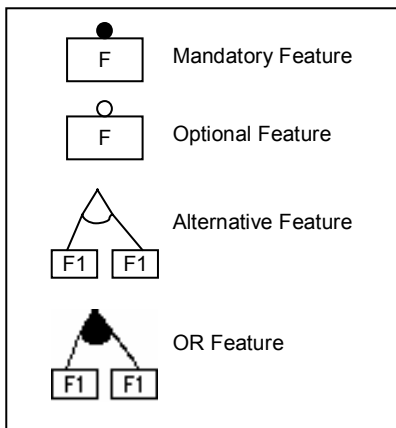


Fig 7: GP' syntax

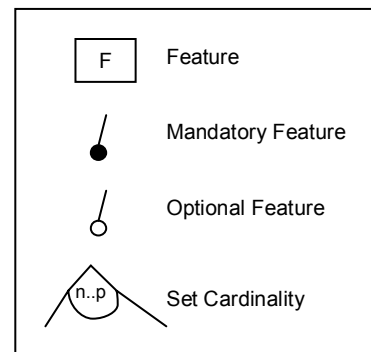
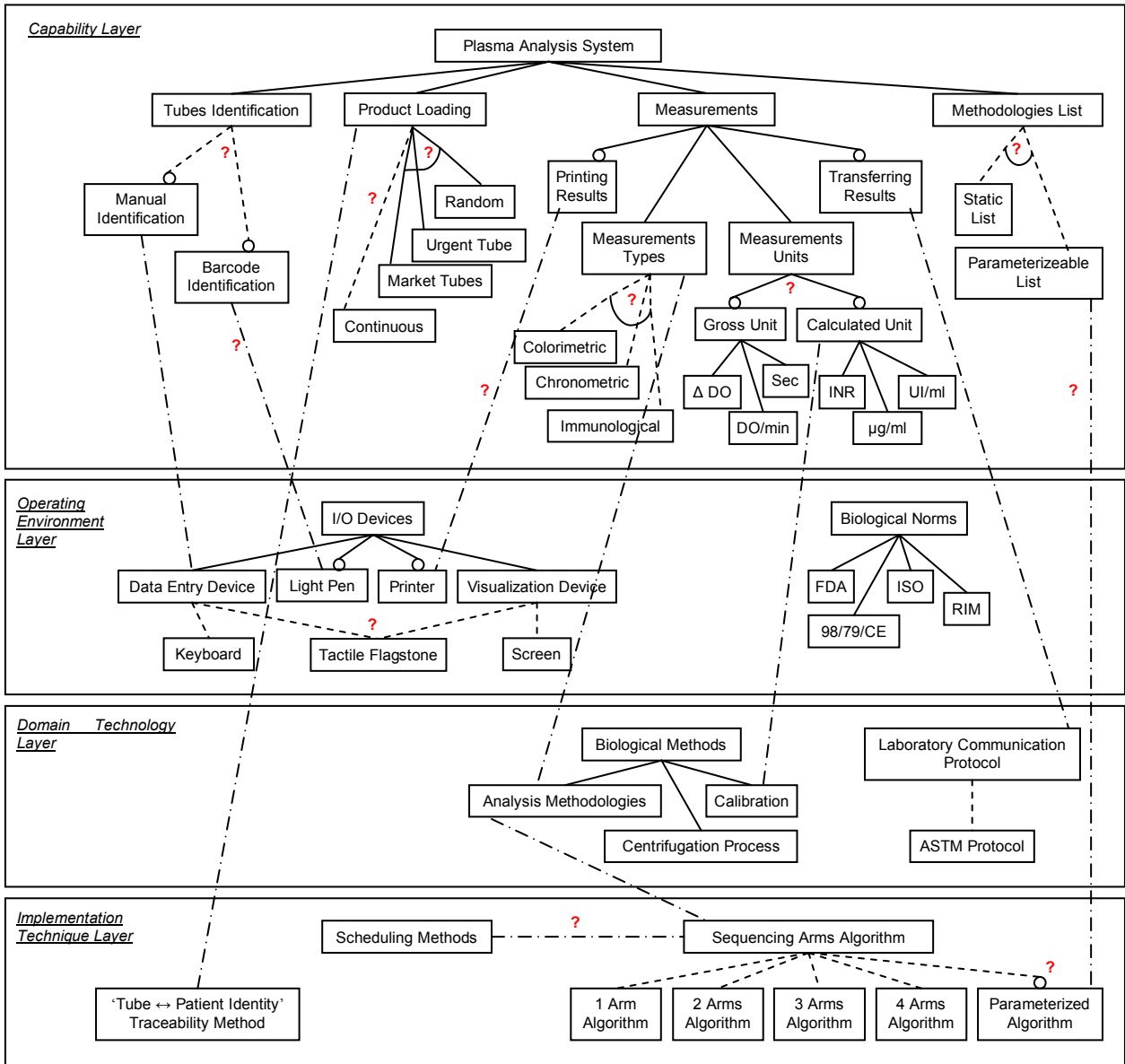


Fig 8: FORE' syntax



Composition Rules:

- Market Tubes *requires* Manual Identification Product
- Market Tubes *mutex* Static List
- Loading *requires* Visualization Device
- Measurements *requires* Visualization Device
- Plasma Analysis System *requires* Visualization Device
- Plasma Analysis System *requires* Biological Norms
- Parameterizeable List *requires* Visualization Device

Fig 9: Analysis automatons product line model according to the FOPLE notation

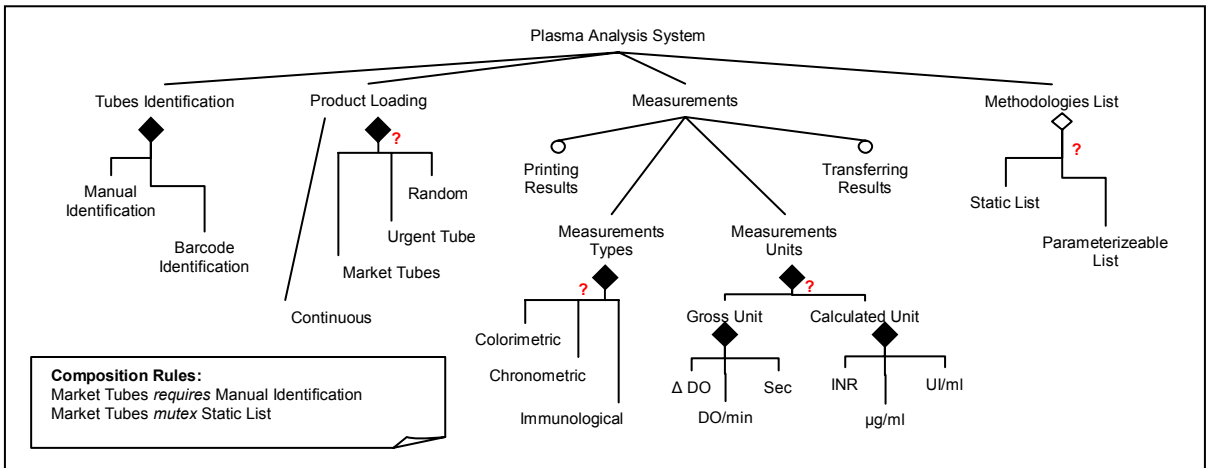


Fig 10: Analysis automaton's product line model according to the FeatuRSEB notation

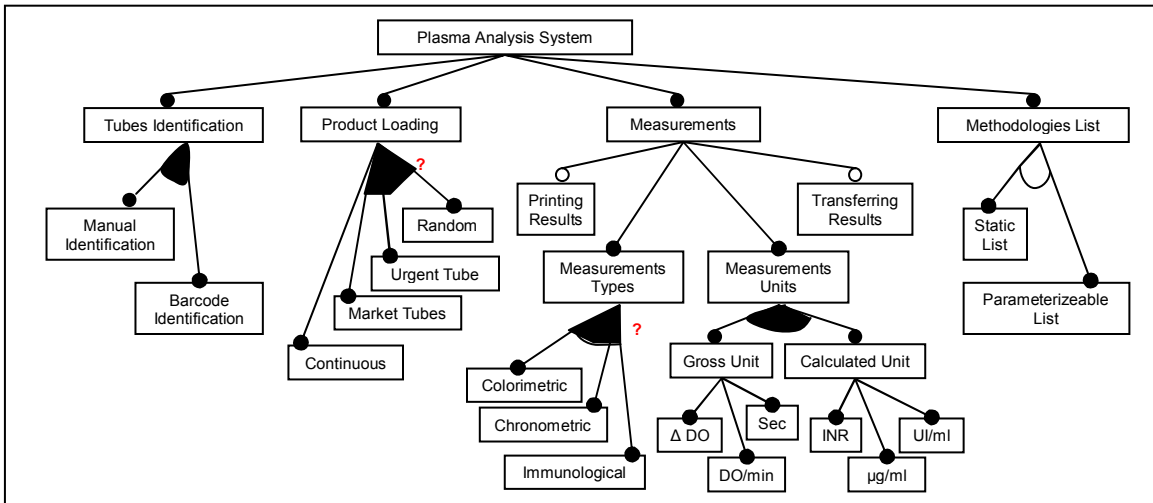


Fig 11: Analysis automaton's product line model according to the GP notation

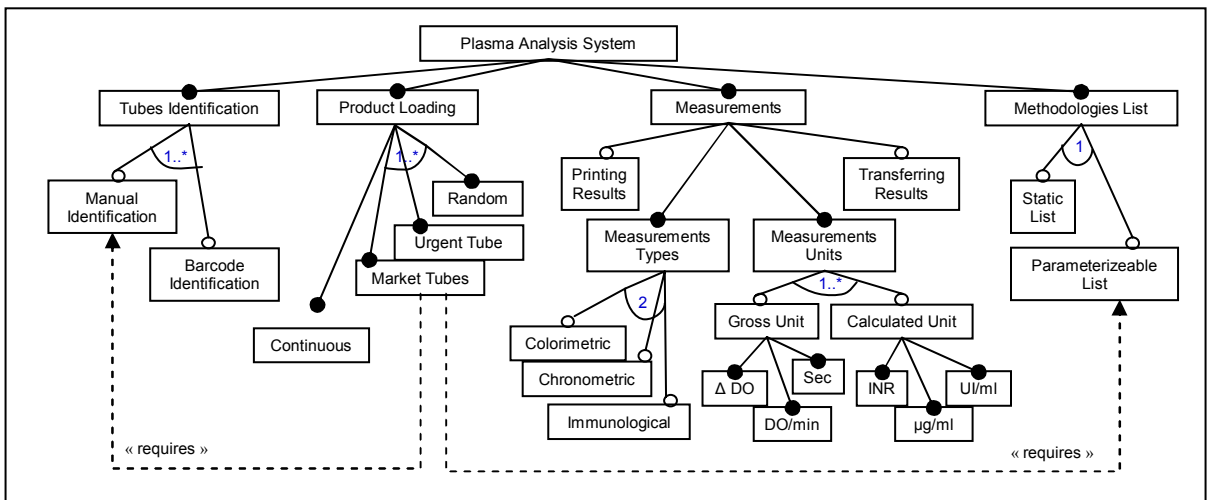


Fig 12: Analysis automaton's product line model according to the FORE notation

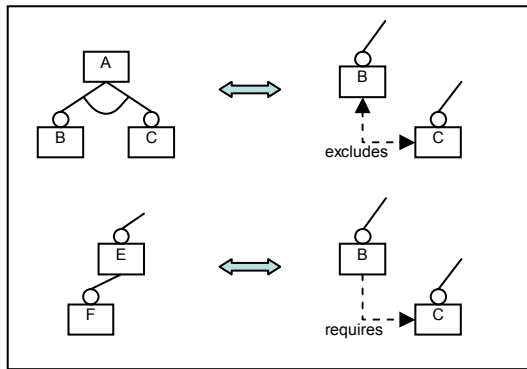


Fig 13: Similarities between hierarchical and dependency relationships

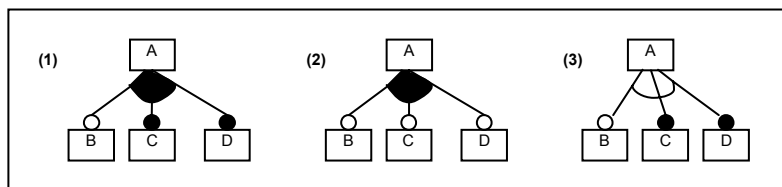


Fig 14: Two-level choice ambiguity in GP notation

Table 2: Interpretation ambiguities in FOPLE notation

Representation	Interpretation and remarks
	<p>The two representations have the same meaning.</p> <p>Since feature C is mandatory, then feature (a) or feature (b) must be selected.</p>
	<p>The two representations have also the same meaning.</p> <p>Feature C is optional, but once selected, feature (a) or feature (b) must be selected.</p>
	<p>This representation does not have a sense.</p> <p>Given that (a) et (b) are alternative features, one of them must be selected. Knowing that (a) is mandatory, it should be the selected feature. Representing (b) is then useless.</p>

Table 3: Interpretation ambiguities in GP notation

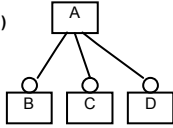
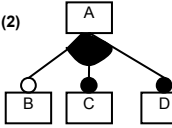
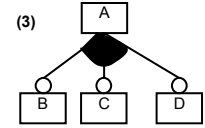
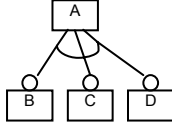
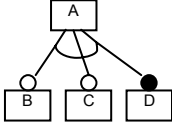

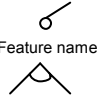

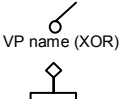


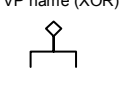

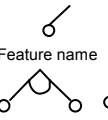
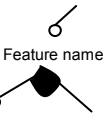
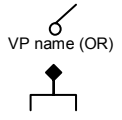
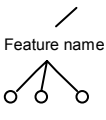
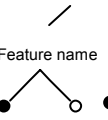
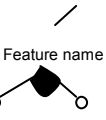
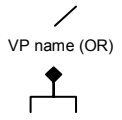
Representation	Interpretation and remarks
<p>(1) </p> <p>(2) </p> <p>(3) </p>	<p>Here, subfeatures are optional OR-features.</p> <p>In all these diagrams, we can choose none or many features.</p>
<p>(1) </p> <p>(2) </p>	<p>In this case, subfeatures are optional alternative features. In both diagrams (1) and (2), we can choose only one feature. But, because of the two-level choice problem, we can also choose none.</p>

Table 4: Cardinality expression in feature-based notations

Cardinality	FORM	GP	FeatuRSEB
0..1		 	
1			
0..N		 	
1..N		 	
N	