Goals and MDA in Product Line Requirements Engineering

Miguel A. Laguna and Bruno González-Baixauli

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

Abstract: One of the most important factors of success in the development of a product line is the elicitation, management, and representation of the variability. In this context, this article explores the possible advantages of recent proposals such as Goal Oriented Requirements Engineering and the Model Driven Architecture (MDA) initiative. Goals and feature graphs can be considered special models in the context of MDA. The global picture is therefore a sequence of models from goals to features and from both to the architecture (a UML model). The conclusion is positive in both aspects but more effort is needed to further evaluate some of the proposed ideas related to MDA automated transformations. At last, traceability is essential if we want to exploit these possible benefits.

Technical Report GIRO-2005-01

1 Introduction

Product lines (PL) have become the most successful approach in the reuse field, due to the combination of coarse-grained components, i.e. software architectures and software components, with a top-down systematic approach, where the software components are integrated in a high-level structure. However, product lines is a very complex concept that requires a great effort in both technical – architecture definition, development, usage and instantiation [2, 7] – and organizational – business view [1] – dimensions. In addition, the standard proposals of the software development process traditionally ignore reuse issues, in spite of their recognized advantages [16]. These characteristics move many organizations away from software reuse, because they cannot afford the effort or the investment needed to initiate a product line, changing from a standard process to an entirely new one. Our proposal is to introduce a reuse approach based on product lines that requires less investment and presents results earlier than more traditional product line methods [21]. This proposal incorporates the best practices in reuse approaches, mainly from the domain engineering process, into conventional disciplines of the application engineering process and is open to new techniques that can be added, replacing others that are weak or obsolete. In this paper we focus on two techniques of the product line Requirements Engineering discipline: Goal Oriented Requirements Engineering and Model Driven Architecture (MDA).

The Goal Oriented Requirements Engineering proposes an explicit modeling of the intentionality of the system (the "whys"). Intentionality has been widely recognized as an important point of the system, but it is not usually modeled. The main advantages of the goal-oriented approach are that it can be used to study alternatives in software requirements (it uses AND/OR models for the different alternatives) and that it can easily relate functional and non-functional requirements (NFR). A goal is an objective that the system under consideration should achieve [28]. There are two types of goals: (hard) goals and *soft-goals*: goal satisfaction can be established through verification techniques, but *soft-goal* satisfaction cannot be established in a clear-cut sense (it is usually used to model non-functional characteristics of the system) [28]. The dependence between goals and *soft-goals* can be established. The NFR framework defines these correlations [6].

Model Driven Architecture (MDA) was introduced by the Object Management Group (OMG) and is based on the Platform Independent Model (PIM) concept. The PIM is a specification of a system in terms of domain concepts and with independence of platforms (e.g. CORBA, .NET, or J2EE) [25]. The system can then transform the PIM into a Platform Specific Model (PSM) [25]. The model driven development paradigm that historically connects with other attempts like model compilers, fourth-generation languages, etc., has seen a resurgence over the last few years due to this support by the OMG. The MDA Initiative has been adopted enthusiastically by many companies that had been working for a long time with different approaches in code generation in concrete niches. Also, some great companies like IBM have reoriented their products to support MDA. Some registries of successes achieved in critical product development have been reported and publicized on the OMG Web site. As the main strength of MDA is the manipulation and transformation of different models and feature

and goal models are introduced in our process, it is worth exploring the relations of these models with UML conventional models in the MDA context.

The rest of the paper is as follows: The next section briefly introduces product line Requirements Engineering and its adaptation to introduce the goal/*soft-goal* paradigm. Section 3 discusses the benefits that MDA can bring to the product line approach. Sections 4 and 5 present the definition and implementation issues of transformations from Feature to UML models. Section 6 concludes the paper and proposes additional work.

2 Product Lines and Goal Oriented Requirements Engineering

Our work in the software process is founded on a coarse-grained reuse model and a related reuse library that offers the operative support to the reuse process [12]. The model defines the structural view of a coarse-grained reusable software element (or *mecano*), made up of a set of fine-grained reusable assets, classified in one of three possible abstraction levels: requirements, design and implementation. The development of a product line involves two main categories of software artifacts: the artifacts shared by the members of the product line and the product-specific artifacts [2]. This division is shown in figure 1. From a fine-grained point of view, a product line is a set of related reusable assets, where the three abstraction levels presented in our model can be clearly identified. The requirement level is represented by the product line business model, the requirements of the product line and the product line variability graph. The design level includes by the product line architecture. Finally, the implementation level includes the generic components, compliant with the constraints of the product line architecture.

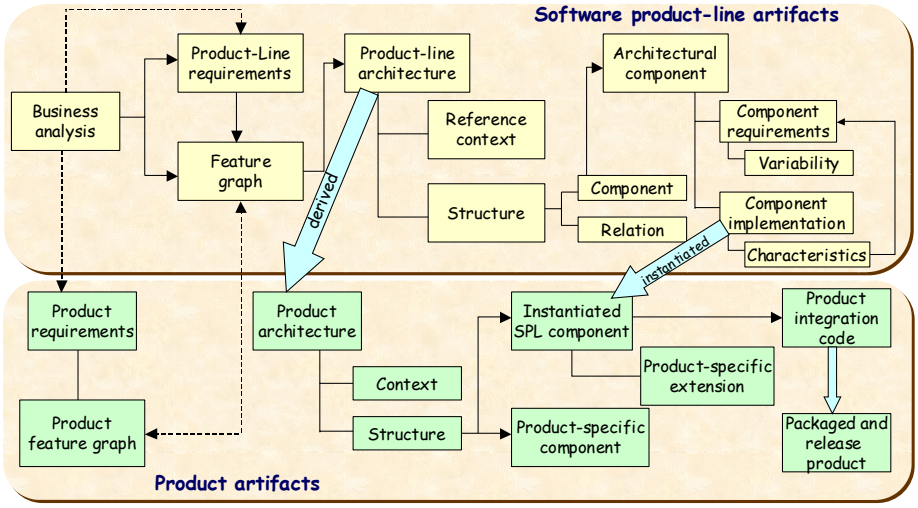


Fig. 1 Product-line artifacts, as defined by Bosch [2]

We have defined two processes separately [21]: a specialized one for domain engineering in the spirit of FORM [22] or Bosch [2] and a process adaptation, based on a conventional software process, where some changes are introduced. Product line engineering and asset

management are continuous processes without external observable output. Product process is iterative but has a final release as relevant difference. This approach is very similar to the RUP profile for asset management that has recently been incorporated to IBM/Rational tools, based on the Reusable Asset Specification (RAS), adopted as OMG standard in 2004 [26]. The RUP Asset-based Development plug-in describes workflows for identifying, producing, managing, and consuming assets.

This coincidence has encouraged us to continue the improvement of the process and, incorporating the best practices appropriate to each key activity in the process. To this end, we have initiated a systematic evaluation of recent technologies with the intention of incorporating them (in an adapted manner if needed) to the domain and application processes.

The first point we focus on, as it is one of the most critical, is the elicitation and analysis of variability in the requirements. The idea that we initially proposed in the original process was based on the work of the SEI on use cases and features [5]. In addition to the information that expresses the requirements themselves, it is important to know the variability of the requirements, and the dependencies between them. To represent this kind of information, the requirements are usually structured in definition hierarchies [18, 17]. Thus, each user requirement is an identifiable functional abstraction, or feature. The features are organized by a graphical AND/OR hierarchy diagram, i.e. the feature graph or feature diagram, which captures the logical structural relationships between requirements.

Requirement elicitation can also be based on use case analysis [13] (use cases is usually a more familiar technique). The question of which analysis must guide the other depends on the PL requirements analyst and his knowledge of the domain or the domain expert's availability. If the analyst has experience and domain experts are available, the best strategy is a feature-driven one; otherwise, a use-case-driven strategy is better [5].

Although its effectiveness has been proven in many projects, we think that these strategies have an intrinsic weakness: they are oriented to the solution more than to requirements. We therefore believe that specific requirements engineering techniques can help. Not only the functionality but also non-functional aspects must be taken into account. This has led us to consider other possibilities to take advantage of the recent advances in Requirements Engineering. In this sense, we are working in the field of Goal Oriented Requirements Engineering [24] as a way of introducing intentionality in the elicitation and analysis of this requirement, and consequently, these goals will allow a rationale to be used in the selection of variants in the application development process. The Figure 2 shows the dependencies between Product Line and Application development models.

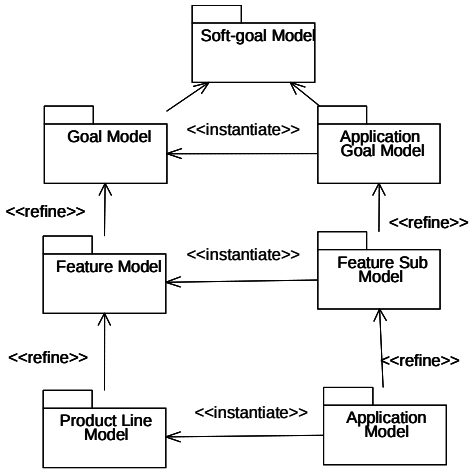


Fig. 2 Model dependencies in Product Line development with Goal support

Figure 3 shows the agents, resources, functionality (goals), and non-functional requirements (*soft-goal*s) as points of variability, starting from a typical i* strategic diagram [31] in the domain of communication for handicapped people. This model represents a kind of context diagram in the goal paradigm. From our point of view, all elements in this diagram are potentially variable but not independent. The scheme of and-or trees in figure 4 serves to investigate the variability and to establish a traceability between the requirements, their correlations and the architectural decisions taken at the time of deriving an architecture for a specific application

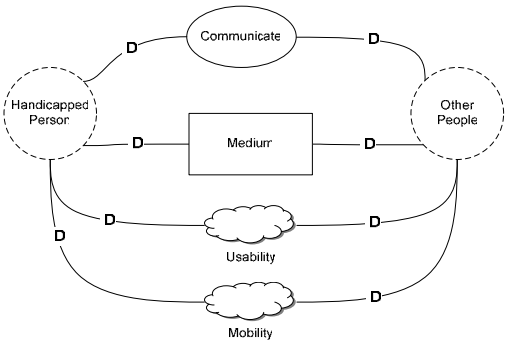


Fig. 3 i* schema of agents, resources and goals/*soft-goal*s for a communicator

It is worth noting that goals and *soft-goal*s are related by correlation forces (contributions): a “poor movement precision” but “mental deficiency” combination of *soft-goal*s influences the decision “input message by words” in the goal model. These contributions provide a rationale for the decisions that an analyst must take when a product is derived from the domain model (of course a carefully defined set of traceability links between *soft-*

goal/goals/features must be previously established). Certainly, a tool that can evaluate these contributions automatically and present the results visually to the analyst is of invaluable help. A prototype has been built to support these calculations [14]. One of the advantages of the separate treatment of goals, *soft-goal*s and features is the early separation of some aspects as proposed by the Aspect Oriented Software Development (ASOD) paradigm [19]: different non-functional aspects are elicited as *soft-goal*s and their relation with functional requirements explicitly expressed (contributions relate *soft-goal*s with goals and indirectly with features by means of traceability links).

This goal approach to variable requirements elicitation has been treated in detail elsewhere [13, 14]. In this paper we focus on another technique that can be of help for our approach: Model Driven Architecture. The next section discusses its interest in requirements elicitation and specification in a product line development scenario.

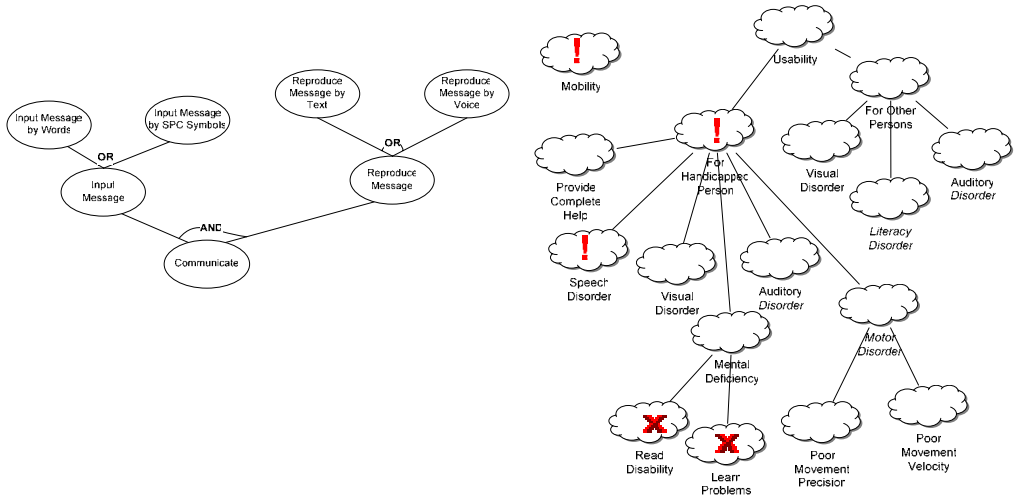


Fig. 4 Variability in goals and *soft-goal*s. **!** and **X** symbols indicate some decisions taken by the analyst

3 MDA and Product Line Requirements Engineering

The introduction referred to some successful experiences with MDA. Yet, with respect to the application of MDA to product line development, the pertinent question is: What degree of real freedom exists at the time of creating a PIM? As a typical example, in the book by Kleppe et al. [20], a translation of a PIM is a set of three PSM which are predefined concrete solutions: a PSM based on Web technology, another supported by java beans technology and the third based on relational databases. Another approach, the *executable UML* paradigm [23] is specific to a certain kind of system and requires a precise definition of the classes and operations (using an action semantic language very close to a conventional code). Nevertheless, when a product line requirements model is specified, its creation is

accompanied by other requirements of quality, security, etc. These non-functional requirements influence the type of architectural solution and technologies that must be applied. The assumption is that, in the previous examples, there is a hidden set of RNF that are not specified in the PIM. In spite of these inconveniences, it is worth analyzing the possibilities that the MDA ideas can bring to this field. Essentially we are searching for an (ideally automated) derivation of an optimal specific product in a product line, while taking into consideration functional and non-functional requirements and using the goals/*soft-goal*s and feature models and their correlations as the starting point. A set of transformations between these models can actually be carried out.

The product line Requirements Engineering discipline includes several activities. The main activity involves the specification of the domain model, which consists of the domain features. The design of a solution for these requirements constitutes the architectural asset base of the product line (typically implemented as an OO Framework).

Later, in the application engineering process, an application model must be derived by selecting alternative domain features from the domain model. In this process alternative concepts are selected based on customer functional and non-functional requirements. This activity is essentially a transformation process where a set of decisions taken by the application engineer generates the initial product model and, consequently, via traceability links, the initial architecture of the product. The variation points are selected on the conceptual level on the basis of a rationale provided by functional and non-functional requirements.

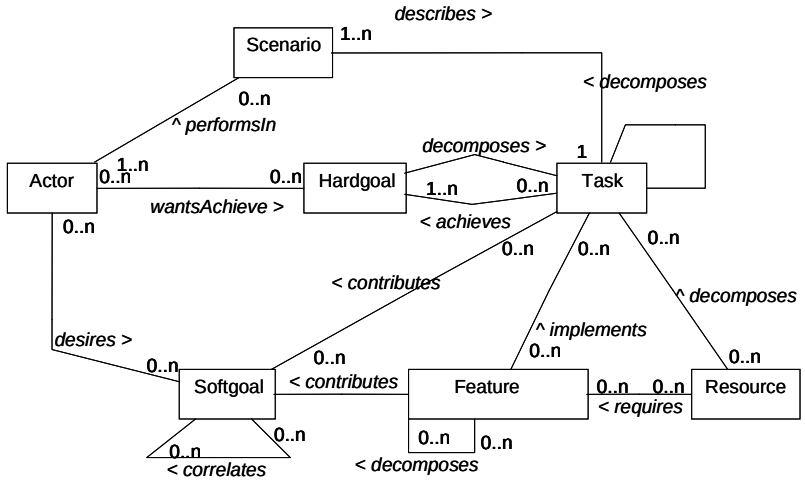


Fig. 5

Goal and soft-goal meta-model. Feature represents the connection to other models, in particular the feature meta-model

The novelty with MDA is the possibility of the automation of the transformations that specify how instances of the domain feature model are converted into a working application. A pre-requisite of the applicability of MDA is to have a meta-model of each technique. In figure 5 a meta-model of the goal/*soft-goal* paradigm is presented in a simplified way (the relationship appears implicitly) and later in this section the feature meta-model is also discussed. The transformation definition can be seen in its maturest state as a compiler for a

domain specific language. The feature/goal models combination would be compiled into a working application using the transformation definition, the asset base and the customer requirements.

In [7] MDA is presented as an approach to derive products in a specific type of product lines, configurable families. The authors identify two main benefits of applying MDA to configurable product families: a) delaying binding time and the selection mechanism for application engineering, and b) independent evolution of domain concepts, product line assets and the transformation technique. The main idea is that a software system that is specified according to the MDA approach is a particular case of product line where the most characteristic variation point consists of products that implement the same functionality on different platforms. The choice for the alternative platforms is a variation point in such a product line. This variation point can be separated from the specification models and managed in the transformation definition itself. The main benefit of MDA compared to traditional development, is that the management of the platform variation point is handled automatically by the transformation step and is not a concern for the product engineer.

However the final platform for a product is not the only variation point that needs to be managed in a product line. The various product line members differ in both their functional and non-functional requirements.

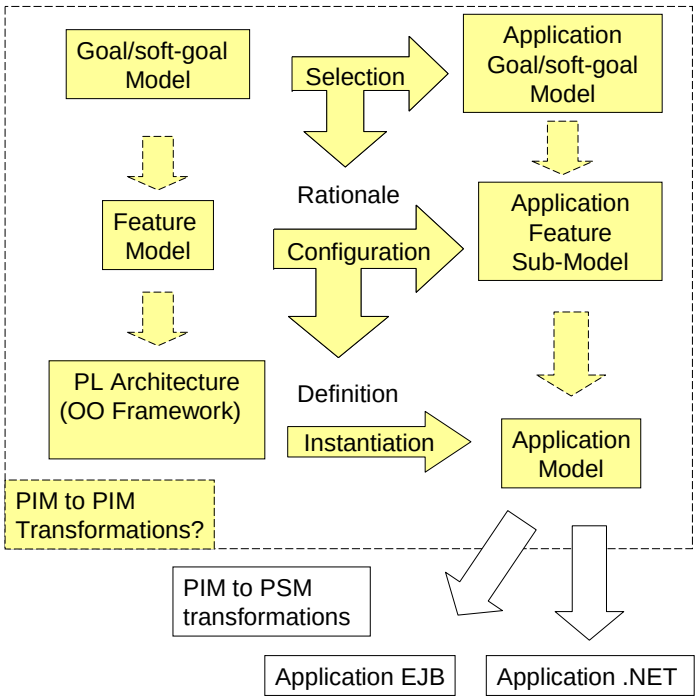


Fig. 6 Product line engineering and MDA and the scope of this study. Left and right parts of the figure refer to Product Line Application process respectively.

The central question is if MDA can easily accommodate these variable requirements by adding the information that specifies places where alternative concepts can be selected. Selection of different concepts from this domain model then results in different PIMs, specific to an application, which, provided that the adequate transformation definitions are implemented, can be automatically transformed into PSMs using the MDA approach. The general schema is presented in figure 4.

From an application model to specific platforms (.NET or EJB), a conventional PIM to PSM transformation is used and only this is a typical MDA transformation, where the initial application model obtained from the feature model and the asset base (and manually completed) is transformed to a specific PSM or set of PSMs. The rest of the models in figure 4 can be considered as PIMs.

These PIMs can be related via automated or non-automated transformations. The possibilities MDA offers will now be examined in detail. There are basically two kinds of transformation:

- *Horizontal:* selection of goals/soft-goals combination, feature model configuration, and framework instantiation
- *Vertical*: PL Goal model to PL Feature model transformation, PL Feature model to PL Architecture (Domain framework) transformation, and the parallel Application equivalents.

Some feature models can incorporate platform or context information (such as variability points) but as we use the *soft-goal* models to express non-functional variable requirements, the feature model is basically functionally oriented. From the inspection of the Figure 4 and from the *horizontal* point of view, we can extract some conclusions, independently of automation degree of the transformations.

The conventional configuration step of a feature model consists of choosing a set of restrictions which originates a sub-graph of features, possibly with some variants deferred to execution time, as two alternative types of payments (Czarnecki differentiates configuration from specialization mechanisms of derivation of feature sub models [10] but we only contemplate configuration as a *horizontal* transformation for the sake of simplicity). There are several kinds of tools to select the variants, such as wizards or graph-like languages and their use guides the instantiation of the particular Application model. The difficulty is that the combination of features must be decided by a domain expert based in his experience and not in objective data.

The interest of using our complementary goal/*soft-goal* model is that it allows the application engineer to decide (if the traceability links are carefully established) what features are needed to reach the selected goals (or functional requirements) and which is the optimal set of goals/features in the context of a set of *soft-goal*s (or NFR) of a determined priority that provides the rationale of the selection. In practice, this supposes a rise in the abstraction level of the variants selection process, making the selection in the requirements level instead of in the feature level. In conclusion, these *horizontal* transformations can be automated, but not in the MDA sense. This line of work can be supported by the tool described in [14] and, despite its scalability problem, the obtained results are promising. In the rest of this section we will

focus on the *vertical* possibilities of figure 4, basically the steps from the feature model to the architectural asset base and the application model.

The relation between goal and feature models cannot easily be considered as an MDA-type transformation because of the different objectives and building methods. Goal models determine the variability of the different ways to achieve these goals (expressed as a tree of sub-goals and tasks that *operationalize* these goals), while feature models separate the common from the variable part of the systems. The tasks have a strong relation with the atomic features in the feature model but the relation is nor one-to-one in general. This can be appreciated in figure 5, where a Feature *implements* (in a many-to-many relationship) several Tasks. This characteristic implies that, until this moment, the two models (three, if we consider the *soft-goal* model) must be built manually, but not independently, by the domain engineer. As a working hypothesis, a constraint imposing that a Task must be implemented only by one Feature will facilitate the traceability and the indirect selection of components, once a goal configuration is selected and also a possible derivation of a first PL Feature model from the PL Goal model.

4 MDA based Feature to UML Models Transformation

The first consideration is, as already mentioned, to answer the question about the meta-model compatibility of the different models. It is clear that the application model is a UML model and therefore the meta-model is the UML meta-model. In the case of the asset base, a UML model (some authors propose extensions or profiles to complete the information about the variability) is also used. The feature models (and sub-models) are built using other concepts but several studies have specified their meta-model using MOF. Two recent approaches have been selected. The Massen proposal [30] is used initially, a partial version of which is shown in figure 7. The Czarnecki meta-model [10] is analyzed later.

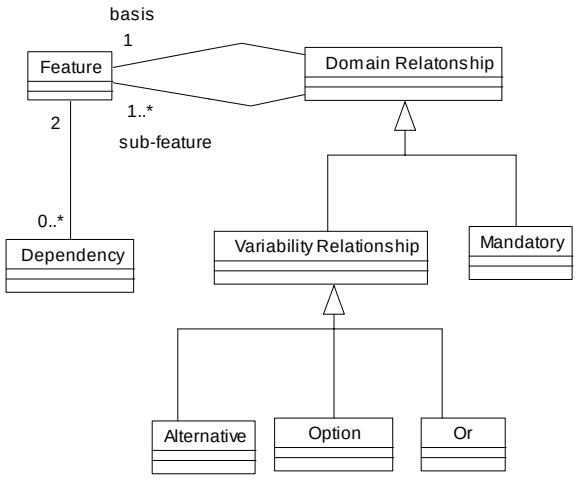


Fig. 7 Feature meta-model, adapted from [30]

As we have a sufficiently precise definition of the meta-models, the kind of transformation we have chosen is based on the meta-model mapping approach [9]. The work consists of defining a set of transformations between the elements of variability in the feature models and the architectural solutions (really each kind of variability in the feature model can be implemented by more than one technique [8]). An example can be seen in figure 8: a typical example of a feature model being transformed into a model that represents a simple framework

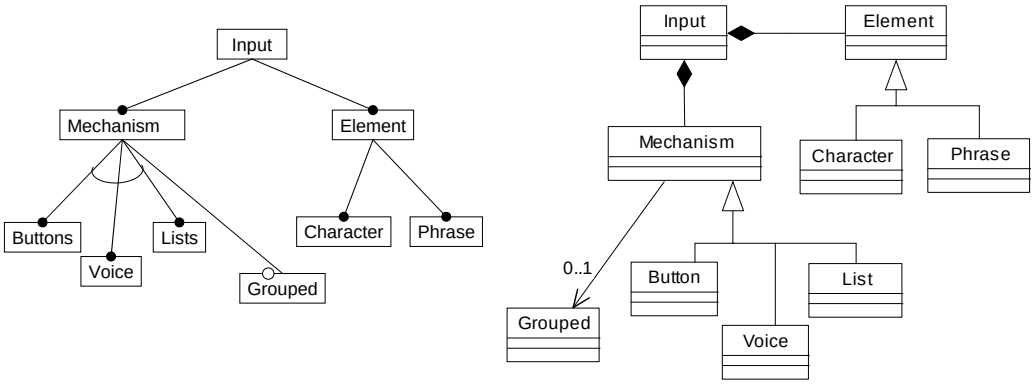


Fig. 8 Feature model and a possible solution using composition, association and specialization

The way to define a transformation is: select an element of the feature meta-model and give one or several equivalences in the UML meta-model. This implies that an annotation is needed in the feature model to select one of the possible design mechanisms. The main idea, consistent with [29], is:

a) To transform each feature into a class

b) To transform mandatory relationships into aggregation/composition UML relationships
c) To transform optional relationships into UML 0..1 associations
d) To transform alternative relationships into UML single generalization
e) To transform or relationships into a UML association multiplicity equal to the number of features and using single UML generalization (this is subtle because the and/or relationship influences the multiplicity of an association between two already existing classes)
f) Finally, to transform *require* and *mutual-exclusive* relationships into OCL expressions

The form of expressing these possibilities is provisionally (and until the QVT OMG standard [27] is approved) the language proposed by Kleppe et al. [20]. For example, in the first example of a meta-model, a mandatory feature transformation and an alternative feature can be expressed:

```
Transformation MandatoryFeature(FM, UML){
  source
    feature:   FM::Feature;
  source condition
    feature.kind = 'mandatory';
  target
              class: UML::Class
              composition: UML::association;
  target condition
     composition.end.first.composition = true;
     composition.end.first().class.name=feature.parent.name
     composition.end.last().class.name = feature.name;
  unidirectional
mapping
  feature.name <-> class.name;
  'a' + feature.name <-> composition.end.first().name;
}

Transformation AlternativeFeature(FM, UML){
  source
              feature:   FM::Feature;
  source condition
    feature.kind = 'alternative';
  target
              class: UML::Class
              generalization: UML::Generalization;
  target condition
   generalization.parent.name = feature.parent.name
   generalization.child = class;
```

```
  unidirectional
  mapping
   feature.name <-> class.name;
}
```

These are two samples of the type of declaration that are needed but are only indicative of the available possibilities. The final approval of the OMG QVT standard and the support of CASE tools for this kind of languages will allow the process to be automated.

The alternative meta-model, proposed recently by Czarnecki et al. [10], change significantly the transformation method. In this approach, the distinctive property of the relationships is the cardinality. In the meta-model of figure 9, the relationships are implicit and the source of the transformation must be the cardinality attribute of the features and group of features. The transformation implies:

a) To transform the Feature model in a Package
b) To transform each Feature (all the subtypes) into a class and, additionally, associate each SolitaryFeature with the class generated from the owner feature with multiplicity equal to the featureCardinality
c) To transform each FeatureGroup into a super-class of the set of classes generated by its owned GroupedFeature instances and generate an association with the class generated from the owner Feature with multiplicity equal to the groupCardinality

The strategy is based in the three subtypes of Feature. The root of every tree in a Feature model (RootFeature) is transformed in a class and a recursive transformation of Solitary Features and Feature Groups linked to every feature is carried out. The presence of a group implies a class associated to the parent feature that is specialized into several subtypes (one per alternative feature). Figure 9 shows the above-mentioned Feature meta-model and Figures 10 and 11 the part of the UML meta-model used in the transformation. The UML meta-model is the 1.5 version, as published by the OMG and related to the 1.2 version of XMI. There is only a implementation mechanism of variability for each Feature type and the traceability is missing.

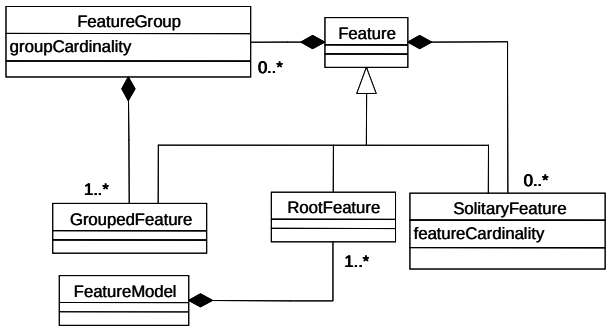


Fig. 9 Feature simplified meta-model, adapted from Czarnecki et al. [10] and a transformation definition as a class

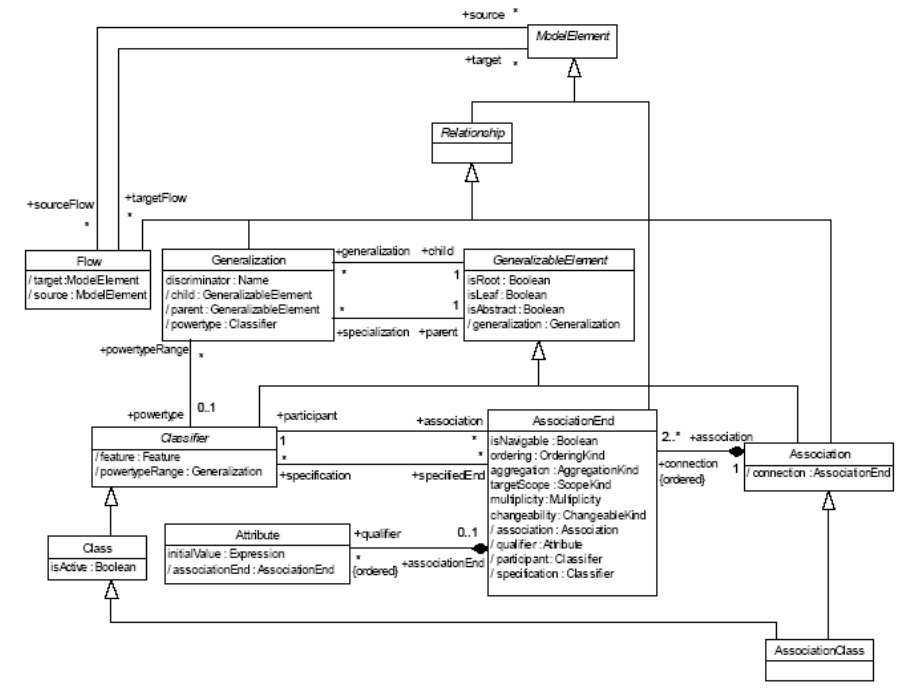


Fig. 10 UML 1.5 meta-model

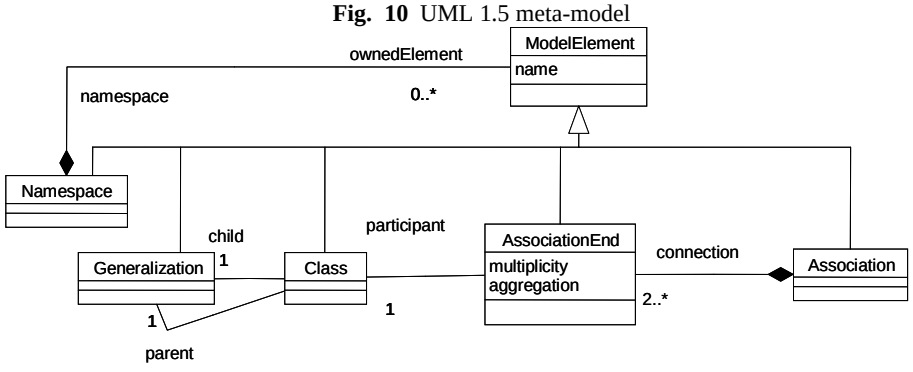


Fig. 11 Essential UML/XMI 1.5 meta-model as used in the transformation

Figures 12 to 14 represent the transformation from Feature models to UML models, graphically expressed using the last QVT submission graphical syntax [27].

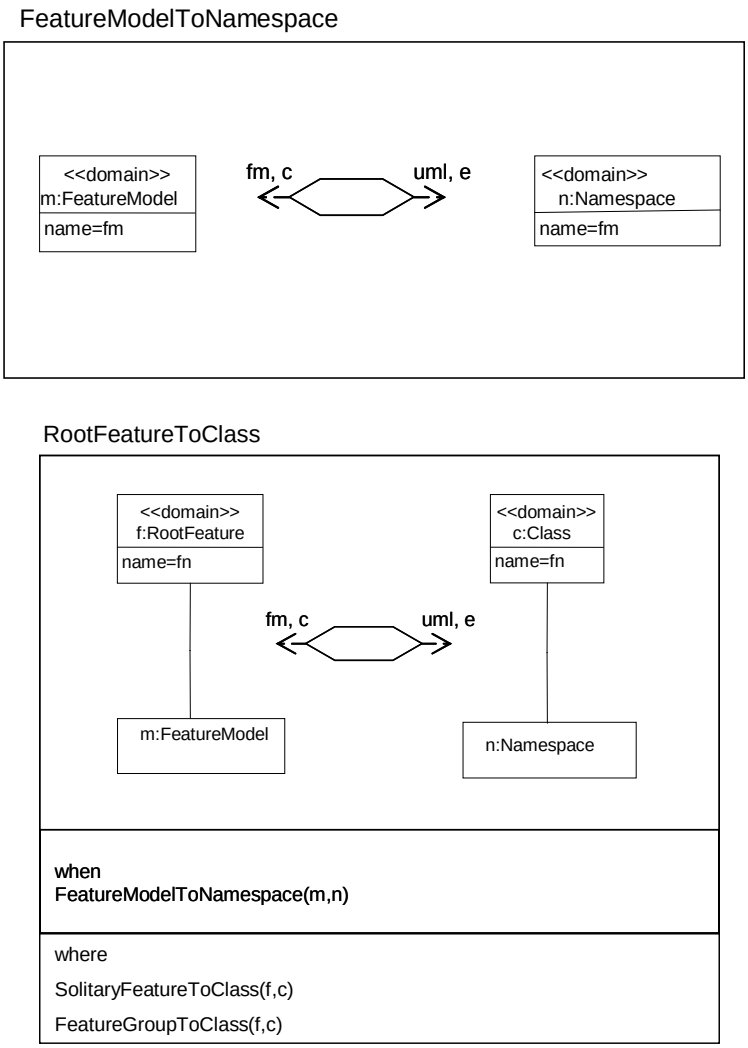
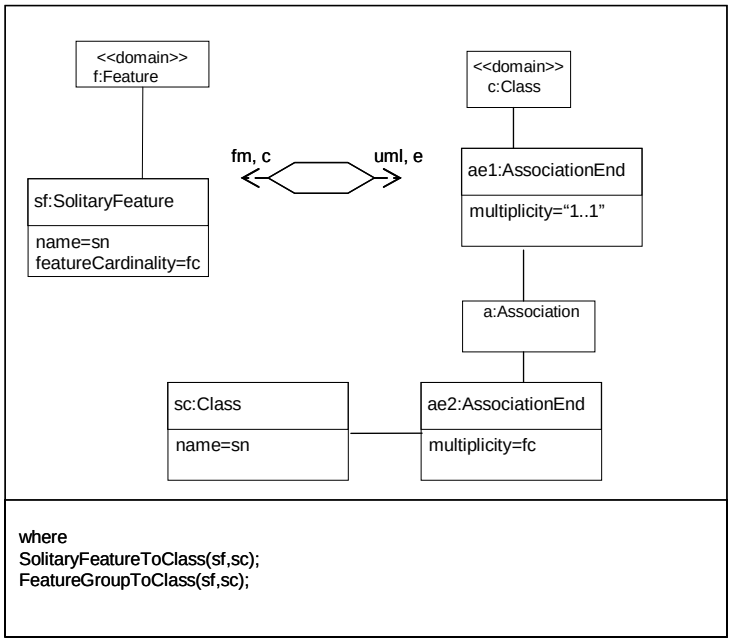


Fig. 12 Feature transformation definition

SolitaryFeatureToClass



FeatureGroupToClass

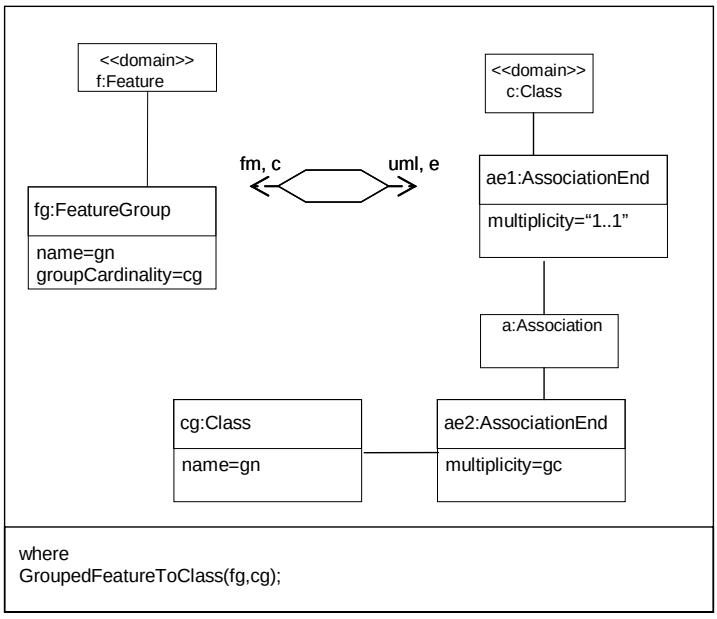


Fig. 13 Feature transformation definition

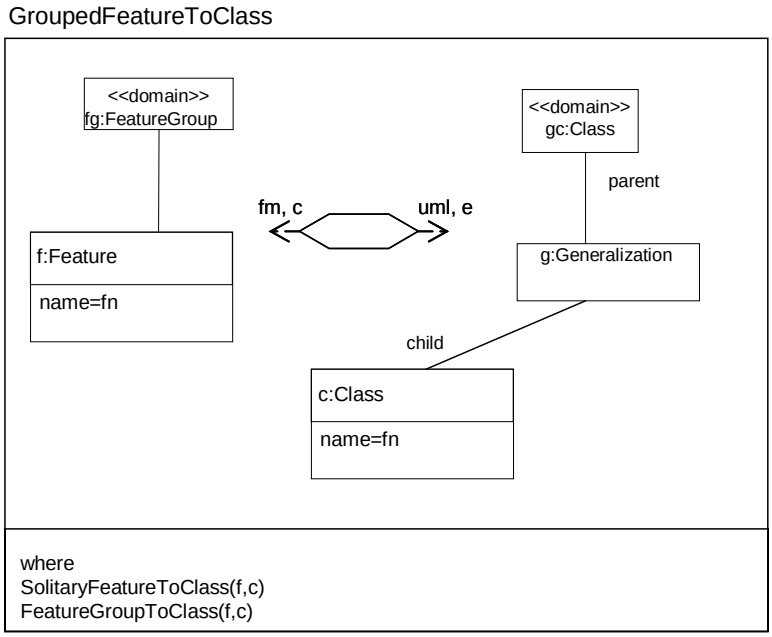


Fig. 14 Feature transformation definition

5 Implementation of the transformation

The specification of the transformation is relatively independent of the way it is implemented. Several authors have classified the different categories [9] of transformation. In our case a Model-to-Model approach is needed and in this category some possibilities are recommended. The most common representation mechanism of feature models is based in XML. In the UML side, the use of XMI is more and more frequent, in spite of compatibility issues. Therefore, XML must be the basis of the representation of both models and the mechanism of the transformation will be influenced by this reason. The most straightforward implementations will use XLST style sheets or java tools able to translate XML files directly or via DOM trees.

The diversity and volatility of feature meta-models is an important problem because probably in the near future it will be necessary to adapt the transformations to new meta-models. For this reason we have selected the XFeature[1] tool, described in [4] as the builder of feature models. The XFeature tool has some advantages, as the use of standard technology (XML and Eclipse) or the customizability of the feature meta-model (the tool allows users to

1 available in http://www.pnp-software.com/XFeature/

define their own meta-models). The tool configuration is defined by a set of configuration files, in particular an XML Schema representing the feature meta-model. The tool is delivered with three sets of configuration files representing three different feature meta-models. One of them is based on the feature meta-model we use in the transformation definition of previous section.

The resulting UML model must be XMI-based and the meta-model the UML standard, therefore any UML compliant case tool could be a valid option. A first implementation, using a XML stylesheet, is given in Appendix A. The figure 15 shows the original feature model and the figure 16 the class diagram obtained from the resulting XMI file, after the application of the sylesheet.

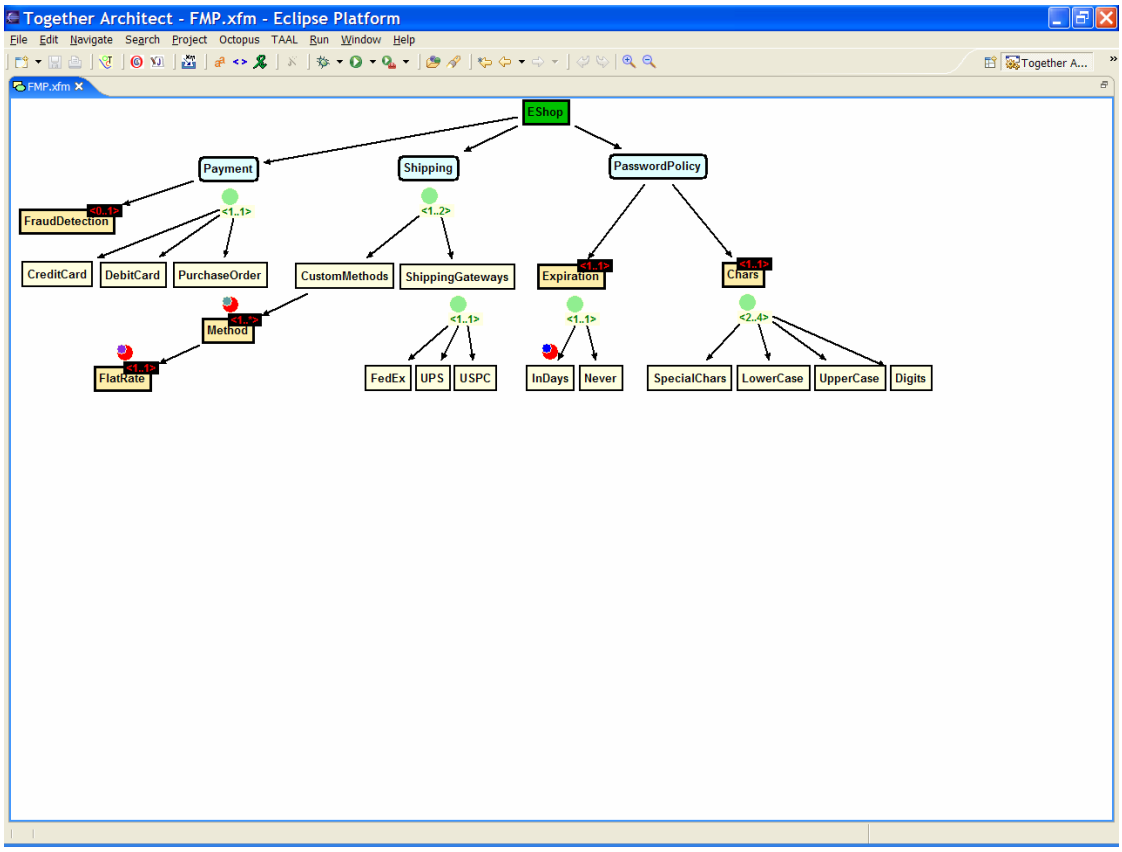


Fig. 15 A Feature model in Xfeature tool

In practice, many problems of compatibility with XMI are well-known. In consequence, as an alternative, we have selected a concrete tool, Eclipse Modeling Framework[2] (EMF) [3], that is also based in XML and Eclipse. EMF is a MDA-based modeling framework and code generation facility for building applications based on a UML model. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor.

The core EMF framework includes a meta-model (Ecore, see figure 17) for describing models that was initially an implementation of the Meta Object Facility (MOF). EMF is an open source project that enhances the MOF 2.0 model and restructures its design in a way that is easy for the user.

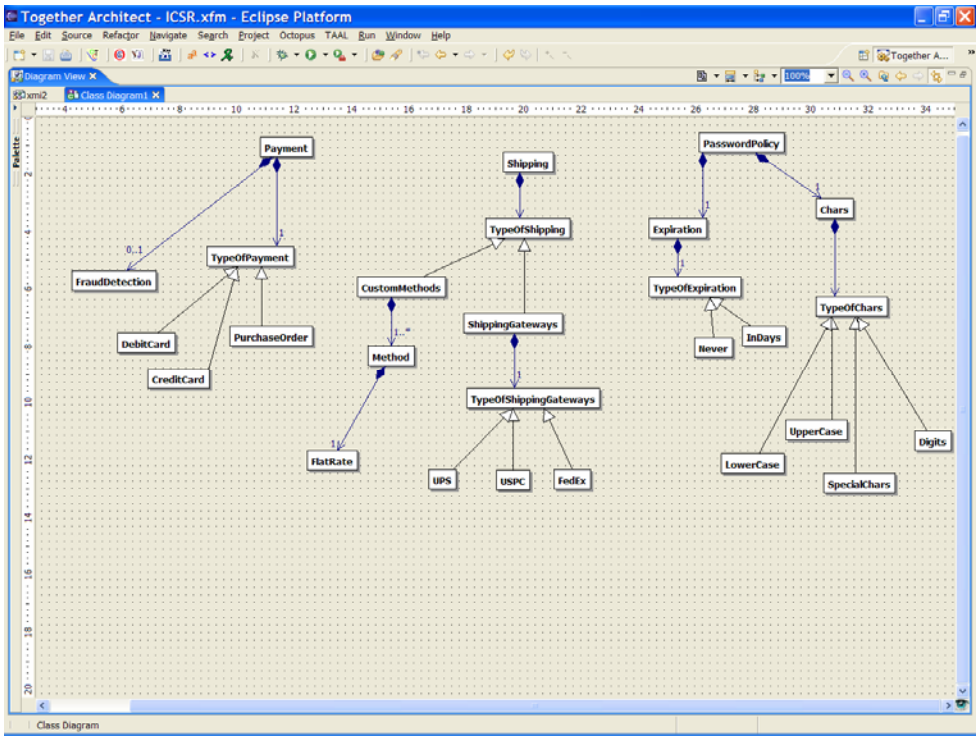


Fig. 16 Framework for the feature model of the previous figure, automatically obtained using the XML stylesheet of appendix A

[2] http://www.eclipse.org/emf

6 Conclusions and future work

In this article, the possibilities provided by the new technologies for the problem of requirements elicitation and analysis in the context of product line development are discussed.

Nevertheless, when a model PIM is created, it is accompanied by other requirements of quality, security, etc. These non-functional requirements influence the type of architectural solution and technologies that must be applied. The conclusion that arises is that already there is a hidden set of RNF that are not specified in the PIM.

The solution that we envision would happen in several steps:

a) To separate different aspects of a PIM in an explicit form, using goals and *soft-goal*s models to build the product line feature model

b) To transform this set of PIM into a new PIM that represents the initial asset base (in the form of an object-oriented framework) and complete it manually with design details

c) Using goal/*soft-goal* as a reference (and with a tool like that described in [14]), to derive an optimal sub-graph of features to solve a concrete problem in the product line

d) From the features sub-graph, to rebuild the architectural PIM for the new application.

Steps b and d are open to consideration in future work. In some product lines, especially in small organizations, it may be preferable not to develop a complete framework. Instead, an elevation of the abstraction level of the product line supposes that the real asset will be the set of goal/*soft-goal*/feature models that will be transformed directly into application PIMs, therefore saving and reusing many of the individual assets of previous applications.

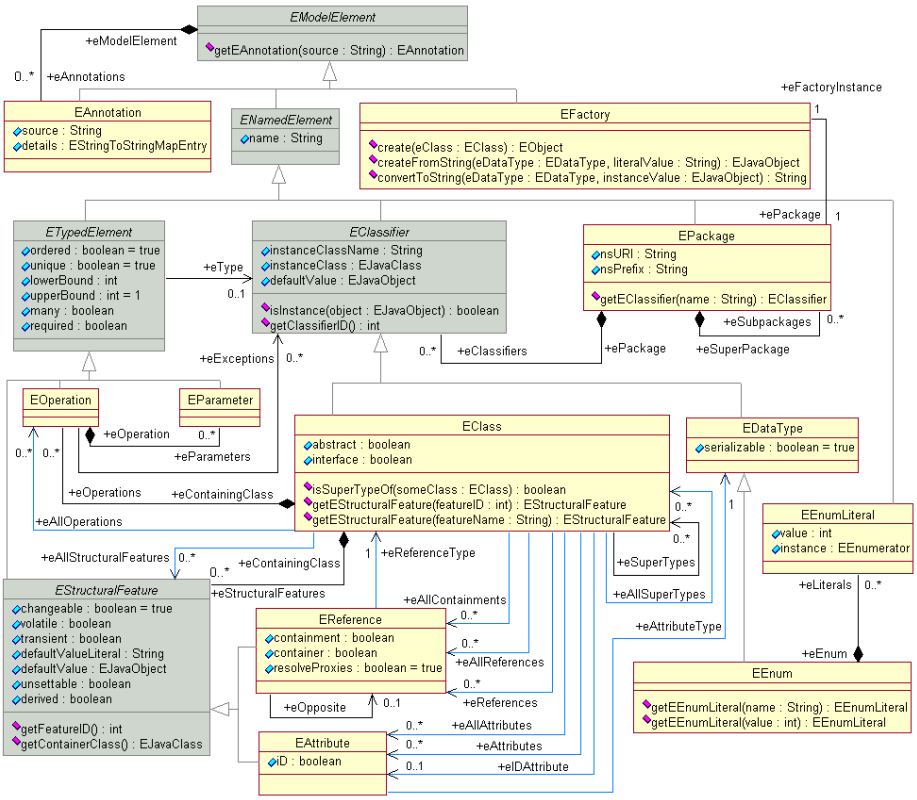


Fig. 17 EMF Ecore meta-model

Acknowledgements

This work has been supported by the Spanish MEC and EU FEDER Funds (project TIN2004-03145).

References

1. Bass, L., Clements, P., Donohoe, P., McGregor, J. and Northrop, L. "*Fourth Product Line Practice Workshop Report*". Technical Report CMU/SEI-2000-TR-002 (ESC-TR-2000-002), Software Engineering Institute. Carnegie Mellon University, Pittsburgh, Pennsylvania 15213 (USA). 2000.
2. Bosch, J. "*Design & Use of Software Architectures. Adopting and Evolving a Product-Line Approach*". Addison-Wesley. 2000.
3. Frank Budinsky, David Steinberg, Ed Merks, Ray Ellersick, Timothy Grose. "Eclipse Modeling Framework", Addison Wesley Professional, 2003.

4. V. Cechticky, A. Pasetti, O. Rohlik, W. Schaufelberger, *"XML-Based Feature Modelling"*, Lecture Notes in Computer Science, Volume 3107, Pages 101–114, Jun 2004.
5. Chastek, G., Donohoe, P., Kang, K. C., Thiel, S. *"Product Line Analysis: A Practical Introduction"*. Technical Report CMU/SEI-2001-TR-001 ESC-TR-2001-001, Software Engineering Institute (Carnegie Mellon), Pittsburgh, PA 15213
6. Chung, L., Nixon, B., Yu, E. and Mylopoulos, J. Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers 2000.
7. Clements, Paul C. and Northrop, Linda. *"Software Product Lines: Practices and Patterns"*. SEI Series in Software Engineering, Addison-Wesley. 2001.
8. Krzystof Czarnecki and Ulrich W. Eisenecker, *"Generative Programming: Methods, Tools, and Applications*", Addison-Wesley, 2000
9. Krzysztof Czarnecki, Simon Helsen. *"Classification of Model Transformation Approaches"*. OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture, 2003.
10. K. Czarnecki, S. Helsen, and U. Eisenecker. *"Staged Configuration Through Specialization and Multi-Level Configuration of Feature Models"*. To appear in Software Process Improvement and Practice, 10(2), 2005.
11. Sybren Deelstra, Marco Sinnema, Jilles van Gurp, Jan Bosch, *"Model Driven Architecture as Approach to Manage Variability in Software Product Families"*, in Arend Rensink (Editor), Model Driven Architecture: Foundations and Applications, CTIT Technical Report TR–CTIT–03–27, University of Twente, avalaible in http://trese.cs.utwente.nl/mdafa2003
12. García, F. J., Barras, J. A., Laguna, M.A., and Marqués, J. M. *"Product Line Variability Support by FORM and Mecano Model Integration*". In ACM Software Engineering Notes. 27(1);35-38. January 2002.
13. Bruno González-Baixauli, Miguel A. Laguna, Julio Cesar Sampaio do Prado Leite, *"Análisis de Variabilidad con Modelos de Objetivos*". VII Workshop on Requirements Engineering (WER-2004). Anais do WER04, pp 77-87, 2004.
14. González-Baixauli, B., Leite J.C.S.P., and Mylopoulos, J. *"Visual Variability Analysis with Goal Models*". Proc. of the RE'2004. Sept. 2004. Kyoto, Japan. IEEE Computer Society, 2004. pp: 198-207.
15. Jacobson I., Griss M. and Jonsson P. *"Software Reuse. Architecture, Process and Organization for Business Success*". ACM Press. Addison Wesley Longman. 1997.
16. Jacobson, I., Booch, G., Rumbaugh, J. *"The Unified Software Development Process*". Object Technology Series. Addison-Wesley, 1999.
17. Kang, K. C., Kim, S., Lee, J. y Kim, K. *"FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures*". Annals of Software Engineering, 5:143-168. 1998.
18. K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. *"Feature-Oriented Domain Analysis (FODA) Feasibility Study"*. Technical Report, CMU/SEI-90-TR-21, Software Engineering Institute (Carnegie Mellon), Pittsburgh, PA 15213
19. G. Kiczalez, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J-M. Loingtier, J. Irwin, *"Aspect Oriented Programming"*, in Proceedings of 11th European Conference on Object Oriented Programming, pp. 220-242, Springer Verlag, 1997.
20. Anneke Kleppe, Jos Warmer, Wim Bast. *"MDA Explained: The Model Driven Architecture: Practice and Promise"*. Addison Wesley, 2003.
21. Miguel A. Laguna, Bruno González, Oscar López, F. J. García, *"Introducing Systematic Reuse in Mainstream Software Process"*, IEEE Proocedings of EUROMICRO'2003, Antalya, Turkey, 2003.
22. Lee, K., Kang, K. C., Chae, W. y Choi, B. W. *"Feature-Based Approach to Object-Oriented Engineering of Applications for Reuse*". Software: Practice and Experience, 30(9):1025-1046. 2000.
23. S.J. Mellor, M. J.Balcer, *"Executable UML A foundation for the Model-Driven Architecture*", Addison Wesley Professional, 2002

24. J. Mylopoulos, L. Chung, and E. Yu. *"From object-oriented to goal-oriented requirements analysis"*. Communications of the ACM, 42(1):31–37, Jan. 1999.
25. Object Management Group, *"MDA Guide Version 1.0"*, 2003
26. Object Management Group (OMG), *"Reusable Asset Specification (RAS)"*, ptc/04-06-06, 2004.
27. Object Management Group and *QVT-Merge Group , "Revised submission for MOF 2.0 Query/View/Transformation version 2.0"* Object Management Group doc. *ad/2005-03-02, 2005.*
28. van Lamsweerde, A. *"Goal-Oriented Requirements Engineering: A Guided Tour*", Proceedings of the 5 IEEE Int. Symp. on Requirements Engineering, 2001, pp:249-262
29. A. van Deursen and P. Klint. *"Domain-Specific Language Design Requires Feature Descriptions"*. Journal of Computing and Information Technology 10(1):1-17, 2002.
30. Thomas von der Massen, Horst Lichter, "RequiLine: A Requirements Engineering Tool for Software Product Lines". In Software Product-Family Engineering, PFE 2003, Siena, Italy, LNCS 3014 pp 168-180, 2003.
31. E. S. K. Yu and J. Mylopoulos. *"From E-R to A-R – modelling strategic actor relationships for business process reengineering"*. Int. Journal of Intelligent and Cooperative Information Systems, 4(2–3):125–144, 1995.

Appendix A

```
<?xml version="1.0" encoding="UTF-8"?>
<!--stylesheet for feature model to class package -->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance" xmlns:UML="//org.omg/UML/1.3"
xmlns:fm="http://www.pnp-software.com/XFeature/fmm">
                        <xsl:template match="fm:FeatureModel">
                           <XMI xmi.version="1.2" xmlns:UML="//org.omg/UML/1.3">
                              <XMI.header>
                                 <XMI.documentation>
                                       </XMI.documentation>
                                 <XMI.metamodel xmi.name="UML" xmi.version="1.4"/>
                              </XMI.header>
                              <XMI.content>
                                 <UML:Model>
                                    <xsl:attribute name="xmi.id"><xsl:value-of select="./@fm:value"/></xsl:attribute>
                                    <xsl:attribute name="name"><xsl:value-of select="./@fm:value"/></xsl:attribute>
                                    <xsl:attribute name="visibility">public</xsl:attribute>
                                    <UML:Namespace.ownedElement>
                                       <!--template for root feature     -->
                                       <xsl:apply-templates select="fm:RootFeature"/>
                                    </UML:Namespace.ownedElement>
                                 </UML:Model>
                              </XMI.content>
                           </XMI>
                        </xsl:template>
                        <!--
   for root feature
   -->
                        <xsl:template match="fm:RootFeature">
                           <UML:Class>
                              <xsl:attribute name="xmi.id"><xsl:value-of select="./@fm:value"/></xsl:attribute>
                              <xsl:attribute name="name"><xsl:value-of select="./@fm:value"/></xsl:attribute>
                              <xsl:attribute name="visibility">public</xsl:attribute>
                           </UML:Class>
                           <!--y tratamos todas las features que parten de la raiz  -->
                           <xsl:apply-templates select="fm:FeatureGroup"/>
                           <xsl:apply-templates select="fm:SolitaryFeature"/>
                        </xsl:template>
                        <!--
                           for solitary feature
    -->
                        <xsl:template match="fm:SolitaryFeature">
                           <UML:Class>
                              <xsl:attribute name="xmi.id"><xsl:value-of select="./@fm:value"/></xsl:attribute>
                              <xsl:attribute name="name"><xsl:value-of select="./@fm:value"/></xsl:attribute>
                              <xsl:attribute name="visibility">public</xsl:attribute>
                           </UML:Class>
                           <!--From Class  to parent Class -->
                           <UML:Association>
                              <xsl:attribute name="xmi.id"><xsl:value-of select="../@fm:value"/>To<xsl:value-of
select="./@fm:value"/></xsl:attribute>
                              <xsl:attribute name="name"/>
```

```
                        <xsl:attribute name="visibility">package</xsl:attribute>
                        <UML:Association.connection>
                           <UML:AssociationEnd>
                              <xsl:attribute name="xmi.id"><xsl:value-of select="../@fm:value"/>To<xsl:value-of
select="./@fm:value"/>1</xsl:attribute>
                              <xsl:attribute name="visibility">public</xsl:attribute>
                              <xsl:attribute name="isNavigable">true</xsl:attribute>
                              <xsl:attribute name="aggregation">composite</xsl:attribute>
                              <!-- isSpecification="false"  ordering="unordered"  targetScope="instance"
changeability="changeable"
                              <UML:AssociationEnd.qualifier>
                                 <UML:Attribute xmi.idref="S.10"/>
                              </UML:AssociationEnd.qualifier>-->
                              <UML:AssociationEnd.multiplicity>
                                 <UML:Multiplicity/>
                              </UML:AssociationEnd.multiplicity>
                              <UML:AssociationEnd.participant>
                                 <UML:Classifier>
                                    <xsl:attribute name="xmi.idref"><xsl:value-of
select="../@fm:value"/></xsl:attribute>
                                 </UML:Classifier>
                              </UML:AssociationEnd.participant>
                           </UML:AssociationEnd>
                           <UML:AssociationEnd>
                              <xsl:attribute name="xmi.id"><xsl:value-of select="../@fm:value"/>To<xsl:value-of
select="./@fm:value"/>2</xsl:attribute>
                              <xsl:attribute name="visibility">public</xsl:attribute>
                              <xsl:attribute name="isNavigable">true</xsl:attribute>
                              <xsl:attribute name="aggregation">none</xsl:attribute>
                              <!-- isSpecification="false"  ordering="unordered"  targetScope="instance"
changeability="changeable"
                              <UML:AssociationEnd.qualifier>
                                 <UML:Attribute xmi.idref="S.10"/>
                              </UML:AssociationEnd.qualifier>-->
                              <UML:AssociationEnd.multiplicity>
                                 <UML:Multiplicity>
                                    <UML:Multiplicity.range>
                                       <UML:MultiplicityRange>
                                          <xsl:apply-templates select="fm:Cardinality"/>
                                          <!--                        <xsl:attribute name="lower">1</xsl:attribute>
                                          <xsl:attribute name="upper">1</xsl:attribute>

                        <xsl:attribute name="lower"><xsl:value-of
select="./fm:Cardinality@fm:cardMin"/></xsl:attribute>
                                          <xsl:attribute name="upper"><xsl:value-of
select="./fm:Cardinality@fm:cardMax"/></xsl:attribute>   -->
                                       </UML:MultiplicityRange>
                                    </UML:Multiplicity.range>
                                 </UML:Multiplicity>
                              </UML:AssociationEnd.multiplicity>
                              <UML:AssociationEnd.participant>
                                 <UML:Classifier>
                                    <xsl:attribute name="xmi.idref"><xsl:value-of
select="./@fm:value"/></xsl:attribute>
                                 </UML:Classifier>
                              </UML:AssociationEnd.participant>
                           </UML:AssociationEnd>
                        </UML:Association.connection>
                     </UML:Association>
```

```
                    <xsl:apply-templates select="fm:FeatureGroup"/>
                    <xsl:apply-templates select="fm:SolitaryFeature"/>
                </xsl:template>
                <!--

                for FeatureGroup


-->
                <xsl:template match="fm:FeatureGroup">
                    <UML:Class>
                        <xsl:attribute name="xmi.id">TypeOf<xsl:value-of
select="../@fm:value"/></xsl:attribute>
                        <xsl:attribute name="name">TypeOf<xsl:value-of select="../@fm:value"/></xsl:attribute>
                        <xsl:attribute name="visibility">public</xsl:attribute>
                    </UML:Class>
                    <!--From Class  to parent Class -->
                    <UML:Association>
                        <xsl:attribute name="xmi.id"><xsl:value-of select="../@fm:value"/>ToTypeOf<xsl:value-
of select="../@fm:value"/></xsl:attribute>
                        <xsl:attribute name="name"/>
                        <xsl:attribute name="visibility">package</xsl:attribute>
                        <UML:Association.connection>
                            <UML:AssociationEnd>
                                <xsl:attribute name="xmi.id"><xsl:value-of
select="../@fm:value"/>ToTypeOf<xsl:value-of select="../@fm:value"/>1</xsl:attribute>
                                <xsl:attribute name="visibility">public</xsl:attribute>
                                <xsl:attribute name="isNavigable">true</xsl:attribute>
                                <xsl:attribute name="aggregation">composite</xsl:attribute>
                                <!-- isSpecification="false"  ordering="unordered"  targetScope="instance"
changeability="changeable"
                                <UML:AssociationEnd.qualifier>
                                    <UML:Attribute xmi.idref="S.10"/>
                                </UML:AssociationEnd.qualifier>-->
                                <UML:AssociationEnd.multiplicity>
                                    <UML:Multiplicity/>
                                </UML:AssociationEnd.multiplicity>
                                <UML:AssociationEnd.participant>
                                    <UML:Classifier>
                                        <xsl:attribute name="xmi.idref"><xsl:value-of
select="../@fm:value"/></xsl:attribute>
                                    </UML:Classifier>
                                </UML:AssociationEnd.participant>
                            </UML:AssociationEnd>
                            <UML:AssociationEnd>
                                <xsl:attribute name="xmi.id"><xsl:value-of
select="../@fm:value"/>ToTypeOf<xsl:value-of select="../@fm:value"/>2</xsl:attribute>
                                <xsl:attribute name="visibility">public</xsl:attribute>
                                <xsl:attribute name="isNavigable">true</xsl:attribute>
                                <xsl:attribute name="aggregation">none</xsl:attribute>
                                <!-- isSpecification="false"  ordering="unordered"  targetScope="instance"
changeability="changeable"
                                <UML:AssociationEnd.qualifier>
                                    <UML:Attribute xmi.idref="S.10"/>
                                </UML:AssociationEnd.qualifier>-->
                                <UML:AssociationEnd.multiplicity>
                                    <UML:Multiplicity>
                                        <UML:Multiplicity.range>
                                            <xsl:apply-templates select="fm:Cardinality"/>
```

```
                                    </UML:Multiplicity.range>
                                </UML:Multiplicity>
                            </UML:AssociationEnd.multiplicity>
                            <UML:AssociationEnd.participant>
                                <UML:Classifier>
                                    <xsl:attribute name="xmi.idref">TypeOf<xsl:value-of
select="../@fm:value"/></xsl:attribute>
                                </UML:Classifier>
                            </UML:AssociationEnd.participant>
                        </UML:AssociationEnd>
                    </UML:Association.connection>
                </UML:Association>
                <xsl:apply-templates select="fm:GroupedFeature"/>
            </xsl:template>
            <!--
                for solitary features a clases

 -->
            <xsl:template match="fm:GroupedFeature">
                <UML:Class>
                    <xsl:attribute name="xmi.id"><xsl:value-of select="./@fm:value"/></xsl:attribute>
                    <xsl:attribute name="name"><xsl:value-of select="./@fm:value"/></xsl:attribute>
                    <xsl:attribute name="visibility">public</xsl:attribute>
                    <UML:Namespace.ownedElement>
                        <UML:Generalization>
                            <xsl:attribute name="xmi.id"><xsl:value-of select="./@fm:value"/>To<xsl:value-of
select="../@fm:value"/></xsl:attribute>
                            <xsl:attribute name="name"/>
                            <xsl:attribute name="visibility">public</xsl:attribute>
                            <xsl:attribute name="isSpecification">false</xsl:attribute>
                            <xsl:attribute name="discriminator"/>
                            <UML:Generalization.child>
                                <UML:GeneralizableElement>
                                    <xsl:attribute name="xmi.idref"><xsl:value-of
select="./@fm:value"/></xsl:attribute>
                                </UML:GeneralizableElement>
                            </UML:Generalization.child>
                            <UML:Generalization.parent>
                                <UML:GeneralizableElement>
                                    <xsl:attribute name="xmi.idref">TypeOf<xsl:value-of
select="../../@fm:value"/></xsl:attribute>
                                </UML:GeneralizableElement>
                            </UML:Generalization.parent>
                        </UML:Generalization>
                    </UML:Namespace.ownedElement>
                </UML:Class>
                <xsl:apply-templates select="fm:FeatureGroup"/>
                <xsl:apply-templates select="fm:SolitaryFeature"/>
            </xsl:template>
            <!--
                for Cardinality

 -->
            <xsl:template match="fm:Cardinality">
                <UML:MultiplicityRange>
                    <xsl:variable name="var" select="./@fm:cardMax"/>
                    <xsl:attribute name="lower"><xsl:value-of select="./@fm:cardMin"/></xsl:attribute>
                    <xsl:choose>
                        <xsl:when test="./@fm:cardMax='*'">
```

```
                                    <xsl:attribute name="upper">-1</xsl:attribute>
                                </xsl:when>
                                <xsl:otherwise>
                                    <xsl:attribute name="upper"><xsl:value-of
  select="./@fm:cardMax"/></xsl:attribute>
                                </xsl:otherwise>
                            </xsl:choose>
                        </UML:MultiplicityRange>
                    </xsl:template>
</xsl:stylesheet>
```