Goals to Features: a Model Driven Proposal in I-GANDALF

(GIRO Technical Report 2009/01-v0.4)

Bruno González-Baixauli[1], Elena Navarro[1], Miguel A. Laguna[1], Julio Cesar Sampaio do Prado Leite[1]

[1] Department of Computer Science, University of Valladolid, Spain
{bbaixauli, mlaguna}@infor.uva.es
[2] Department of Computer Science, University of Castilla La Mancha, Spain
Elena.Navarro@eclm.es
[1] Department of Computer Science, University PUC do Rio de Janeiro, Brazil
http://www.inf.puc-rio.br/~julio/

Abstract. Product lines are one of the most successful approaches to develop quality software in a changing environment. This is achieved by defining core assets that will be common to several products. In such environment, the analysis of commonality and variability emerges as key activity.
The main problem with this approach is that requirements engineering effort to obtain the features that the system should have is, usually, over-simplified. These features are organized in Feature Models that are already focused on the solution domain.
In this paper, we propose to adapt techniques from Requirements Engineering to Product Line, specifically the Goal Oriented approach, that models the problem from an intentional viewpoint (the whys). Goal Models have been selected because: their structure is similar to features, simplifying the transition between them; and they proved useful in dealing with non-functional requirements. In addition, they can be used to provide a way to select the desired product from the problem domain and not from the solution domain.
The main contribution of this paper is the description of a process that derives a product line architecture from goals, using features to bridge the gap between them. We named this process: I-GANDALF (Intentional Goal ANalysis Directed by Architectural Features), using a Model-Driven approach. In this article, the overall proposal is presented, but we focus on the early stages: from goals to features, defining the models, explaining what the relationships between them are, and providing the definition and description of the transformation between Goal and Feature models using QVT.
We use MORPHEUS, a graphical environment for the description of the different models, to support the approach, and Medini QVT to implement the QVT transformation. The evaluation of the approach is carried out using a case study in the domain of e-commerce.

Keywords: software product lines, goal modeling, feature modeling, model-driven engineering, non-functional requirements, QVT

1 Introduction

Different alternatives have been defined up to date in the Software Engineering field to face the challenge of developing quality software in a changing environment. Product Line Software Engineering (PLSE) is one which emphasizes the idea of software reuse. This paradigm guides the software development process, from the requirements to deployment stage, by defining and developing the core *features* that make up the product line, so that products are developed by reusing them instead of from scratch. A feature is a "prominent or distinctive user-visible aspect or characteristic in a domain" [12]. Features are organized in a graph called *Feature Model*: "A Feature Model represents the standard features of a family of systems in the domain and relationships between them." [12]. This reuse oriented approach paves the way for reducing effort and increasing software quality since the shared assets are specified in detail, and validated accordingly. In order to define these core assets, PLSE uses two key concepts: *commonality* and *variability,* that is, what parts are common to all products, to someone or even to only one [8].

Requirements Engineering (RE) is the process of discovering system purpose by identifying stakeholders and their needs, and documenting these in a way that is amenable to analysis, communication, and subsequent implementation [20]. However, when this process is considered in the context of Product Lines (PL) development, its complexity and difficulty is much higher since there are several products to consider, being necessary to find relationships of commonality and variability. Unfortunately, in PLSE this process is usually reduced to find a set of features, grouped in a Feature Model, that products of the PL must or can have, that is, to deal with the *early variability* or *essential variability* [9]. Moreover, in practice these Feature Models go down to functionality details that are not so user-visible, even proposing a programming paradigm (Feature Oriented Programming [21]). Therefore, the main difference between Features and traditional RE approaches is the idea of purpose, while the former focus on systems capabilities (solution domain), the latter focus on what the system is and why it is need (problem domain).

Recently, more systematical approaches have appeared that adapt or use RE techniques such as Use Cases [9], Problem Frames [22] or Ontologies [1], to deal with the *essential variability*. In this context our proposal has been defined, by exploiting Goal Models to specify variability because the advantages they provide. We understand that these advantages are threefold. First, they have a similar structure to Feature Models so that a smooth transition can be obtained from the problem domain to the solution domain. Second, they have demonstrated to be one of the best choices to deal with non-functional requirements and the system intentionality from early stages [4]. And third, Goal Satisfiability Analysis can be used to reason about what product to choose according to both a set of selected goals and prioritized softgoals [7].

In this work, we present a proposal, using ideas from Model Driven Engineering (MDE) [24], that helps the development of PLs from the very beginning of their specification until the deployment of the products that make up them. Several Models have been identified in its description: Goal Models because they provide a mean to analyze the product line requirements; Feature Models because they have been proved useful for variability management, but from a more functional viewpoint, linking with the architecture; Architectural Models because they provide facilities to generate the final products of the PLs.

Main goal of this document is to define the transformation in the early stages, between Goal Models to Feature Models. We use the standard Query View Transformation (QVT) [18] from the Object Management Group (OMG).

The structure of the rest of the paper is: first, we present the case study that will illustrate the approach, then we introduce the intentional PLSE focusing in early stages, that is the use of Goal Analysis and Feature Modeling in product line. Section 4 describes the different models used in the proposal. Section 5 defines the transformation between goals and features using QVT, and Section 6 describes how MORPHEUS provides support to the proposal. Finally, in Section 7 we discuss other related works, and conclude in Section 8.

2 Case Study: e-commerce systems

The ideas proposed in this paper have been applied at e-commerce systems domain (see Fig. 1). This domain was selected because of the wide range of documentation available, such as [15], [3] or [26]. Authors of these works have highlighted the high variability of these systems and the unquestionable benefit that the introduction of PLSE in its development means in terms of cost and quality of the final products. These benefits are convincing enough as to achieve a seamless transition from academia to industry. In addition, e-commerce systems form an important part of many business strategies as companies expand their presence to the Internet; this guaranteed applicability of the research, which was another reason why the domain was an attractive choice. We have focused this work on a partial description in order to enhance the comprehensibility of both the proposal and the domain.

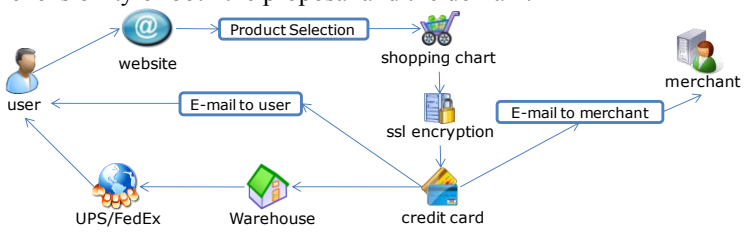

Fig. 1. A sketched view of the e-commerce domain

3 Intentional Product Line Engineering

One of the main problems for the adoption of PLSE is the high complexity and the effort that it demands. This is where our proposal, called I-GANDALF (Intentional *Goal Analysis Directed by Architectural Features*[1]) comes to foreground taking into account two objectives: first, providing guidance for its application throughout the PL development process; second, introducing good practices from early stages of development, mainly related to Requirements Engineering and, more specifically, to non-functional requirements (NFR). With regard to the first goal, we propose a seamless guided process starting from an initial modeling of the problem until the code development. Concerning the second objective, we use Goal-Oriented Requirements Engineering (GORE) [14], a well-known approach that has proven its utility in requirements phase. GORE propose an explicit modeling of the intentionality of the system (the *whys*) so that they model the problem with a more client-focused approach; taking into account the goals of the clients instead of a possible solutions to these goals. In addition, it is a useful mechanism to analyze non functional requirements because they are modeled and analyzed considering their relation to the functional ones [4].

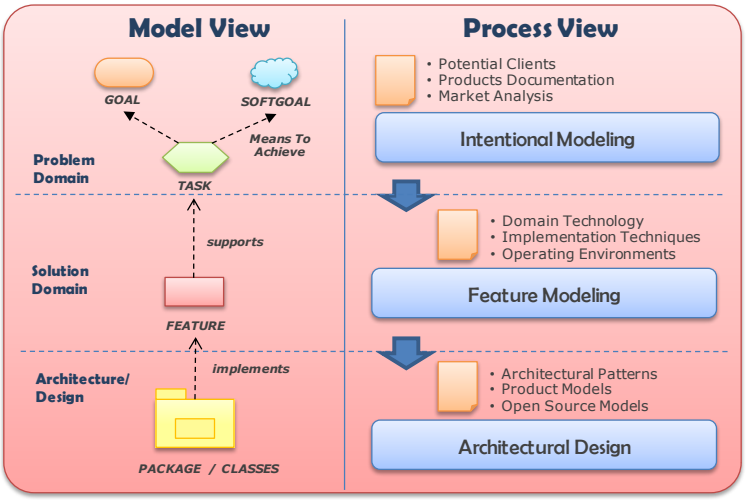

Fig. 2. A sketched view of I-GANDALF

Fig. 2 shows both the Process View and the Model View of I-GANDALF. It can be observed that it entails three main activities:

[1] The string Gandalf has been used for naming different types of things or characters: in software engineering it was used by one of the first software environments[10]

- *Intentional Modeling* is in charge of considering the problem domain in order to determine what the domain goals are. With this aim, the Goal Model uses two kinds of elements: (*hard*)*goals* that are states that the stakeholders want the system to achieve, and whose satisfaction can be clearly determinate; and, *softgoals* that are used to specify NFR and whose satisfaction cannot be clearly determinate so that it is only possible to state that they are enough or partially satisfied [4]. Hardgoals represent hard restrictions in client requirements, meanwhile softgoals can be achieved in several degrees and decide between different possible solutions for the selected goals as show in [7]. The use of softgoals is very useful to model and analyze NFRs since this type of requirement is fuzzy by nature. Goals and softgoals can be decomposed in sub-goals until *tasks*, that is, means to achieve them, are specified. In GORE, alternative goals and tasks are analyzed to determine the optimal ones to be implemented in the system-to-be. In PLSE, all the goals and tasks should be taken into account in order to explore the different alternatives. This allows one to scope properly the PL, determining what tasks should be supported by the products of the PL, and to gather domain concepts by studying the goal descriptions. In addition, a wider scope also implies more constraints on scalability; therefore our proposal also aims to improve scalability of Goal Models. Section 0 describes the I-GANDALF Goal Model.
- *Feature Modeling* carries out the representation of the solution domain. It uses as input: the Goal Model specified previously along with information on domain technology, implementation techniques, and operating environments. With this aim the Feature Model is specified by determining the *features* of the PL. When the Feature Model is specified it must be taken into account that the high level features must support the tasks identified in the Goal Model (see Fig. 2), domain concepts relevant to determine points of variability, and softgoals (NFR) as means to select alternative products with similar hardgoals but with different qualities. Those high-level features are refined in more detailed ones, even taking into account implementation techniques. During this activity, a more complete variability analysis is performed, determining what features can be at the same time in one product, which ones require others, or the cardinalities, using *cardinalities in feature decomposition* and restrictions. Section 4 describes the I-GANDALF Feature Model.
- *Architectural Design*. As features are the basis of PL architecture definition [2], this activity is oriented to the semi-automatic generation of the architecture using the features previously specified. This generation is performed following ideas of MDE so that traceability can be easily managed. We have presented in [13] how this activity is carried out by using *UML packages* as the elements that implements the specified features (see Fig. 2), and *package merge* relationship to incrementally add details. In this sense, packages and merge relationship provide a mechanism to design

the features, maintaining the variability information in the Feature Model. Therefore, goals are related to the architecture, and if the architecture is linked with the components, even to the code. This is very important to deal with one key part of PLSE know as *Application Engineering*, that is, the derivation of specific products but optimizing them from the client needs (hardgoals), and desires (softgoals).

In this article, we focus on the Model View of I-GANDALF, that is, on describing the Goal Model and Feature Model along with guidelines necessary for their description.

4 Models Supporting I-GANDALF

As was stated above, MDE paradigm has been used to describe the Metamodels that I-GANDALF entails. One of its cornerstones is the exploitation of Model-To-Model transformation techniques to generate the Models downstream of the process development automating a task that could be cumbersome and error prone. An additional advantage is that as the Models are generated, the traceability links between the source Model and destination Model are created automatically as well. These traceability links can be used later to configure the relation between goals and features, features and architecture, and even between architecture and code. Therefore we can obtain the configured product from the needs and desires of the client (goals and softgoals). In the following sections the Metamodels of I-GANDALF are introduced.

4.1 I-GANDALF Goal Model

Our goal oriented approach is initially based on V-Graph [26], a particular type of Goal Model as an extension of the NFR Framework [4]. The main problem of this model is the lack of a complete definition of the different components, mainly about relationships and their targets/sources. To solve this problem, our goal model uses the same type of elements, but clarifies the relations by specifying the Metamodel illustrated in Fig. 3.

Goal, *softgoal* and *task*, introduced previously in section 2, are the main elements of the I-GANDALF Goal Model and are modeled as Metaclasses. Note the difference between *Goal* / *Task* and *Softgoal* Metaclasses, the formers have clear-cut and binary values (*satisfy* or not and *isVariantMember* or not), and are grouped in an abstract Metaclass called *HardElement*. Meanwhile softgoal has a fuzzy nature with no clear-cut criteria (*softsatisfaction*). Each one is described by a *type* that constitutes its main description, and optionally a number of *topics* that give contextual information to the main description. Fig. 3 shows that *type* and *topic* are associations from *DescribedElement* to *DescriptionElement* which is inherited by *goal*, *softgoal, task, and aspect.* Fig. 4

shows some of the identified goals and tasks for e-commerce domain where it can be appreciated that, for instance, one of the goals is *Manage[Order],* being *Manage* the type and *Order* the topic.

As Fig. 3 shows, all the goal elements are structured according to the concern they belong to. With this aim, concern definition must be performed as first step of the process to find the main goals (hard and soft) that will be the root of the initial concerns. In the example, *Manage[Order]* or *Secure[Order]* are some of these initial goals. The concerns are defined by these initial goals and their decomposition into sub-goals, and finally into tasks, as done in traditional Goal Modeling, but trying to separate as much as possible different concerns. To help to systematize this separation, a rule is applied to enforce that goals/tasks be unique to each concern. If the rule can not be enforced, it is then modeled as an aspect (as will be detailed later on this section) or leads to the creation of a new concern.

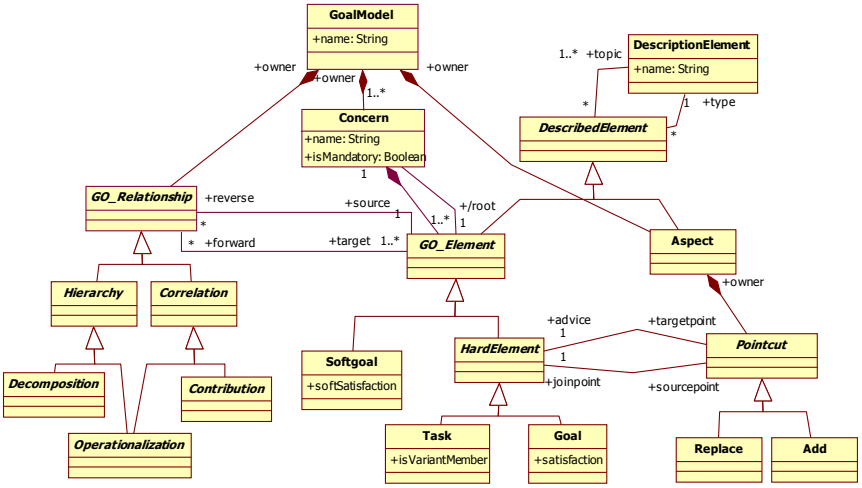

Fig. 3 Metamodel for Goal Models

Regarding relationships, they are grouped in an abstract Metaclass called *GO_Relationship* (Fig 4. shows the possible relations used in GANDALF). As shown in Fig. 3, there are three kinds of GO relationships: *Decomposition*, *Operationalization* and *Contribution*. *Decomposition* relationships, that is, *And/Or* relationships, are employed to specify the Goal Model hierarchically so that more abstract goals and softgoals are decomposed into simpler one until tasks can be identified. When the satisfaction of the Goal Model is analyzed it must be considered that the root goal is satisfied if every leaf (at least one leaf) is satisfied when an And(Or) relationship is established among them. Fig. 4 shows how the goal *Manage[Order]* is decomposed into one goal Manage*[ShoppingChart]* and two tasks, *Confirmating [Order]*, *Making[Order]* meaning that every one of them must be satisfied to satisfy the root.

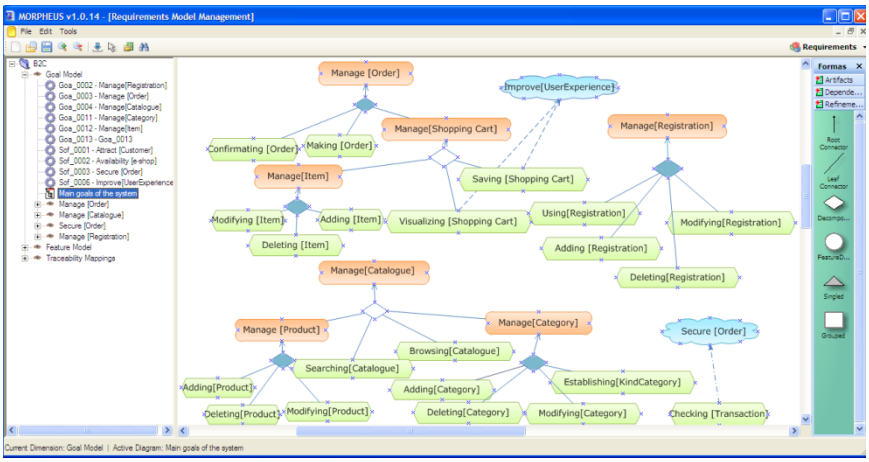

Fig. 4 A partial view of a I-GANDALF Goal Model for e-commerce. *Hardgoals* are the orange shapes, *Tasks* are green and *Softgoals* are the blue ones. Diamonds represent *decomposition* relationship: filled for *And* and outlined for *Or*. Dotted lines are *Contributions*.

One of the main differences, regarding the V-Graph, is the introduction of *Contribution* relationship (see Fig. 3) unifying *correlation* and *contribution* to simplify the analysis. In PL context, the analysis of the Goal Models focuses on the selection among alternatives for products without caring if they affect directly or not, being relevant only the kind and degree. Although it is not shown in Fig. 3, this relationship is specialized as *Make(++)/Help(+)/Unknown/Hurt(-)/Break(--)* in order to denote the contribution strength. Fig. 4 illustrates an example of this relationship where tasks *Visualizing [ShoppingChart]* and *Saving [ShoppingChart]* are contributing positively (*Help*) to achieve the goal *Improve [UserExperience]*.

Another relationship introduced in our Goal Model, taken from the NFR Framework that does not exist in V-Graph, is *Operationalization*. It has at the same time a double nature (see Fig. 3): as decomposition is used to specify a way to achieve a softgoal; as contribution is employed to specify that a task helps to or is sufficient to achieve the softgoal it is related with, because of the not clear-cut satisfaction criteria of the softgoals. Although it has been not shown in Fig. 3, this relationship is specialized as Strong(++)/Weak(+) to denote the strength it helps to achieve a determined softgoal. Fig. 4 shows an example of this relationship where the task *Checking [Transaction]* is describing a solution to the goal *Secure [Order]*.

As was stated in section 3, in order to specify PLs, a requirement that any proposal must satisfy is the ability to deal with large specifications. Although Goal Models have been defined to manage alternatives for a product, it is especially relevant when they are used to specify PL because they have also to manage alternatives between all the products of the PL and relations between them. This means that scalability is a must of the proposal. With this aim, we

have borrowed ideas from Aspect-Oriented Software Development (AOSD), specifically, the use of aspects to improve the modularity of the specification. In our proposal, they are employed to improve the Goal Model modularity and, therefore, their scalability. It is worth noting that this constitutes a difference with the original use of V-Graph [26], whose aim is to find aspects by means of the analysis of how tasks contribute to softgoals in a stable Goal Model. Another difference with [26] is that, since there are several products in a PL, some products will focus on a set of softgoals, meanwhile others will focus on another, perhaps hurting them. Therefore, conflicts must be allowed and maintained in stable PL models.

In our proposal, goals and tasks are specified in different graphs using the relationships described in Fig. 3 and according the concern they are related to, as can be observed in Fig. 4. However, using aspects those concerns can be weaved in order to describe properly the PL. This weaving proceeds by replacing or adding specific goals or tasks from the source concern on specific goals or tasks of the target concern. With this aim the Metamodel includes several concepts: *Concern, Aspect,* and *Pointcut*. Goal Models are composed by *Concerns* that help to modularize the model by representing the key concepts of the system. This concept is represented by the root goal of the different trees that make the Goal Model. *Aspects* specify the solution that wants to be achieved which is made up by a set of *Pointcuts* (see Fig. 3). Every pointcut specify part of the solution by *replacing* with (or *adding*) tasks or goals, identified by the *advice* relationship, (to) other goals or tasks identified by *joinpoint* relationship (see Fig. 3). Fig. 5 depicts an example of how AOSD concepts are put into practice, where a task *Checking [Transaction],* which was defined in a different tree (see Fig. 4), is added to tasks *Making [Order]* and *Confirmating [Order]*. In the example, *Securing [Transaction]* is the aspect that helps to weave the involved concerns describing the solution.

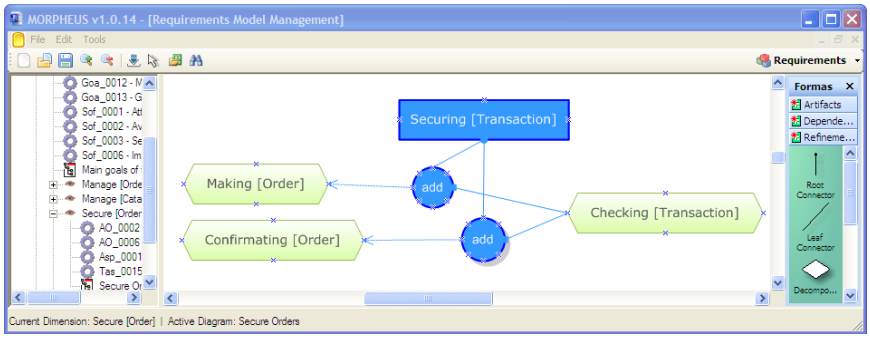

Fig. 5 An example of aspectual relationship

Using this approach, we understand that it is easier to define separated concerns, to add or remove Goal Models or even to define them separately, improving scalability and therefore making Goal Modeling more suitable for PL

analysis. Also, it improves V-Graph since it provides a better separation of concepts defining concerns, but facilitating the relation between functional and non-functional requirements with aspects.

4.2 I-GANDALF Feature Model

In Goal Models, variability is analyzed, but it is not done in detail, leaving out concepts such as *cardinality* or restrictions as important as *mutual exclusive* or *requires* [9]. Goal Models focus on modeling client needs and desires and ways to achieve them. However, Feature Model have proved being both a suitable technique to specify the variability of the PL, and an appropriate initial step to develop the PL architecture. In addition, Feature Models specification determines a transition to the solution domain from the problem domain described by the Goal Models, because the analysis of features that the PL must support is carried out in terms of the tasks defined in the Goal Model. For these reasons, a Feature Model has been included in the description of I-GANDALF, facilitating both a deeper analysis of the variability of the PL and the generation of the software architecture during the activity *Architectural Design* of I-GANDALF (see section 0).

Fig. 6 shows the Metamodel for Feature Models that is based on Czarneski et al [5], but simpler. *Features* are the main elements that made up Feature Models, and the I-GANDALF Feature Model in particular. Originally, a feature was defined as "a prominent and distinctive user-visible characteristic of a system" [16]. Therefore, according to this definition features can be a tool for user interaction or product configuration. However, features are widely used to design PL architecture, so that the most accepted definition includes not only users nowadays, but stakeholders as well, what includes software architects or designers (amongst others). In this work, an architectural-oriented view is applied, since the next activity of I-GANDALF, *Architectural Design*, takes as input the Feature Model to generate an initial version of the architecture; the user-oriented view is offered by the Goal Model.

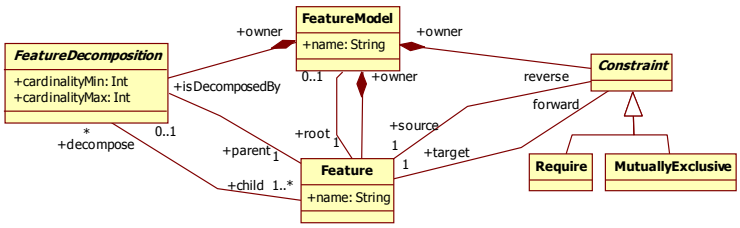

Fig. 6 Metamodel for Feature Modeling using a cardinality approach

Fig. 6 depicts the identified relationships in the I-GANDALF Feature Model as well. As can be observed a Feature Model is built by a progressive refinement, by decomposing more abstract and complex features until simpler one can be defined. With this aim, the *FeatureDecomposition* relationship has been defined so that a feature (parent) can be decomposed into one or several (children) features. The Metamodel makes no restriction among *Root*, *Solitary* or *Grouped Features* as done in [5], but they are implemented using this relationship. Therefore, an n-ary relationship is used instead of using different types of features, so that Metaclasses are employed for defining models, elements and relationships, what help in transformations between models. In addition, references are not necessary to specify that a feature can have several parents.

FeatureDecomposition relationship is highly relevant because it helps us to denote variability points in the specification of the PL by using two attributes, *cardinalityMin* and *cardinalityMax,* as they help us to specify what child features are mandatory/optional/alternative. Let *n* be the number of child features for a *FeatureDecomposition* relationship:

- Child features will be mandatory whenever cardinalityMin and cardinalityMax are set to n, what means that all of them must be supported by all the products of the PL. Fig. 7 shows (by filled rectangles) that confirmate, make and ShoppingCart are mandatory features for e-commerce domain.
- Child features will be optional whenever cardinalityMin < cardinalityMax ≤ n, determining that depending on the product the features that must be present in the PL will be variable. Fig. 7 shows (by grey rectangles) that Manage Item, Visualize and Save are alternative features for e-commerce domain.
- Child features will be alternative whenever cardinalityMin = cardinalityMax=1, determining that only one of the child features can be simultaneously when concrete products of the PL are being specified.

FeatureDecomposition is an abstract relationship that has been specialized as *Grouped* or *Singled* in order to specify when cardinality applies to a group of features or to several instances of the same feature. This differentiation is important as stated in [5] to avoid redundant representation and the need for group normalization. It can be observed in Fig 7, that most of the relationships are grouped decomposition (shown by rectangles), being only the decomposition of transition a singled one.

Another important part of Feature Models are the relationships used to specify constraints between elements as they are key to analyze properly the PL [9]. In the Metamodel two kinds of constraints can be defined: *MutuallyExclusive* is defined among two features if they cannot be simultaneously in a product; and *Require* is specified among two features if one needs the other to be present in case it is selected when configuring a product. It can be observed in Fig. 7 that this relationship has been established from *Confirmate* and *Make* to *Check* in order to achieve secure transactions.

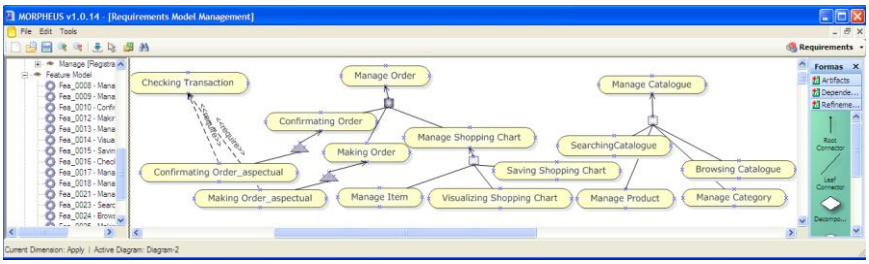

Fig. 7. Partial view of the Feature Model generated from Fig. 4

4.3 Mapping Goal Model and Feature Models

One of the key characteristics of any proposal called MDE is the ability to deal with traceability from the requirements models till the architecture and code. This is due to the fact that traceability assists to reconcile the changes in user's models with the software, decrease costs of acquiring critical knowledge, assess consequences and impact of a change, etc. Taking into account these ideas, the Metamodel described in Fig. 8 was defined. As can be observed, a new kind of relationship called *Support* has been included that links tasks and features. This relationship is employed to describe what features of the Feature Model are required to perform what tasks of the Goal Model. Therefore, the analysis of the Feature Model will proceed by determining that every task in the Goal Model has at least one feature in the Feature Model in order to be supported by the products of the PL.

It can be observed also in Fig. 8 that a relationship called *Contribution* has been introduced to describe the relation between features and softgoals. This relationship improves the analysis process of the Feature Model, because NFR can be considered defining what features will have better behavior in terms of NFR, and at configuration time be selected for its inclusion in the products of the PL. In addition, it also facilitates the configuration of these products taking into account to what extent these features contribute to the satisfaction of the softgoals. It must be highlighted that as I-GANDALF is a MDE proposal, it can make good use of Model-to-Model transformation languages, specifically QVT [18], to not only generate that traceability relationships but also the features in the Feature Model.

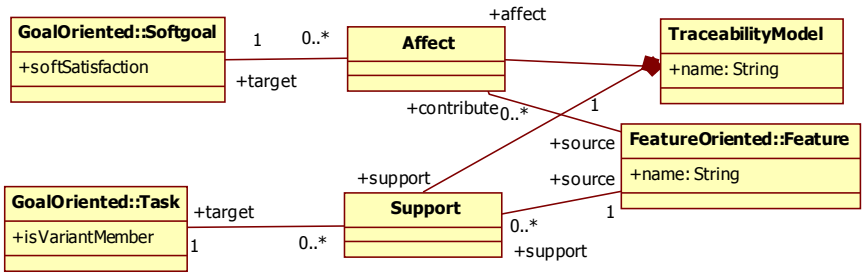

Fig. 8 Mapping relationships between Goal Metamodel and Feature Metamodel

One of the key points is how to carry out the transition from the Goal Model till the Feature Model, that is, how these traceability relationships can be determinate. In this sense some guidelines have been defined to guide the analyst in such a task that use feature taxonomies as starting point, specifically, that presented by Kang et al. [12]. They classify features in four layers:

1. The *capabilities*, from the user perspective, of the application in a specific domain. They are characteristics visible by the user that can be identified as services, operations, or non-functional characteristics. This kind of features is directly related to tasks of the Goal Models, since services and operations (product capabilities) need a task as a means to achieve a goal. Services, as high level features, are concerned with complex tasks, and in a similar way, operations as parts of these complex tasks. However, features are not powerful enough to deal with most of the non-functional requirements because features are not able to describe different degrees of quality for different products. In our approach, contributions are modeled in Goal Models, but also in Feature Models, what can be used to analyze how different features affect to the NFR. Also, softgoal decomposition in Goal Models provides task to improve them, that is, operationalizations that will be reflected in the Feature Model.
2. The *technology* of the application domain. This is based on the made decisions about requirements, including laws, standardization, and business rules. This kind of features is specific to the domain, including high level features, so that they can be linked to tasks to be done, but also to low-level solutions that focus on solving high-level ones. Therefore they are more related to architecture than to the user goals. As I-GANDALF starts by defining goals it provides the high-level features view that will be decomposed later in the low-level ones. We separate goals and features, more related to user and architecture respectively avoiding to complicate the user view with architectural details.
3. The *operating environments* (hardware and software platforms), in which applications are going to be used and operated, and the implementation techniques (algorithms and data structures) layers. This two kind of features, focus on solution domain, describing, for instance, what kind of operating

environment supports high-level features, or even what implementation techniques can be used to support them. There is not direct relationship between these kinds of features and goals but it is only evaluated to what extent the features affect the NFR.

It is worth noting that these high-level features, which have traceability relationships with tasks of the Goal Model, are refined into sub-features according to low level domain technologies, implementation techniques or operating systems. This is highly relevant as these sub-features bridge the gap between the Feature Model and the later description of the architecture, that is, the *Architectural Design* activity of I-GANDALF. How this process is performed has been already presented in [13].

5 M2M transformation: generating the feature model

As I-Gandalf has been defined as MDD proposal, one of the advantages is that Model-To-Model transformations (M2M) can be used to generate new models from existing ones, as well as to check the consistency between them. With this aim, a M2M transformation has been defined (see Appendix A for a whole description) using QVT (Query-View-Transformation) a proposal of the OMG (Object Management Group). This transformation allows us to generate a first draft of the feature model from the goal model, having an improved definition. In addition, one of the advantages of using of QVT is that a trace class is automatically generated for each relation, facilitating that the traceability between the elements of the candidate models can be easily maintained.

In QVT, a transformation is described by means of a set of *relations* that establish how the matching between several *candidate models* is carried out. A candidate model is any model that conforms to one of the metamodels identified when the transformation is specified. In Fig 9 is shown the starting part of transformations definition. As can be seen, the transformation *goalToFeature* identifies three involved metamodels: *IGGoal*, *IGFeature* and *IGTraceability*, where IG means I-GANDALF. Each one is typing one domain of the transformation, that is, *iggoal*, *igfeature*, and *igtraceability*.

```
transformation goalToFeature (iggoal:IGGoal,
igfeature:IGFeature, igtraceability:IGTraceability)
{
key IGFeature::FeatureModel {name};
key IGFeature::Feature{name};
key IGFeature::FeatureDecomposition {parent, child};
key IGFeature::Constraint{source, target};

key IGTraceability::Affect{source, target};
key IGTraceability::Support{source, target};
```

Fig. 9 Initial part of transformation *goalToFeature*

Every relation is defined by two or more of these domains. For instance, the relation *GoalModelToFeatureModel* (look at Appendix for the complete definition) is defined by three domains: *iggoal*, *igfeature*, and *igtraceability*. Each domain describes a pattern that the elements of the candidate models must match by creating, modifying or deleting them in order to satisfy the relation. In addition, each relation can have the conditional clause *when* and *where*. The former identifies what conditions must be satisfied in order to hold the relation. The latter specifies what conditions must satisfy the elements of the candidate models participating in the relation. In addition, every relation is described as either a top-relation (by using the keyword *top* before its name) or non-top relation. Every top-relation must be hold when a transformation is executed, whereas non-top relations can be hold or not depending if they are invoked or not from the *where* clause of another relation.

Second type of elements in Fig 9 are the *keys*, that set what attributes identify the elements (Metaclasses). These keys are usually defined for the elements in the domains where the elements will be created or modified. In our transformation, the keys for *FeatureModel*, *Feature*, *FeatureDecomposition* and *Constraint* are defined for *IGFeature* domain, and for *Affect* and *Support* in *IGTraceability*.

The main element in the transformation is the *GoalElement*. As explained above in section 2, Features should support Tasks in Goal Models, but taking into account the variability described in the Goal Model. Variability can be specified in Tasks (by means of Or decomposition), but also in Goals and Softgoals (by using Or decomposition and Operationalization). Also, the Concerns (that represent concepts of interest in the system and are decomposed independently) can be mandatory or not. Finally, the Aspects, what relate elements from different Concerns are inherently optional.

When defining a model transformation, two are the main approaches to describe them according to the core element: to use elements or to use relationships being the latter the easier approach. Since features are focused in variability, to improve scalability, our policy is to group as much *GoalElements* as possible in Features. Therefore, rules description should start from *GoalElements* viewpoint because there is not a 1-1 relationship and an easy way to generate features from relationships. Considering this issue we have defined the following four groups of rules

- **Initial rules.** They are 3 rules, the first two start the transformation and the third one groups the rules to apply on each *GoalElement*. These rules are shown in Fig. 10:

```
top relation GoalModelToFeatureModel
// Maps Goal to Feature models, and creates root feature
{
  tmn, gmn: String;

   checkonly domain iggoal gm:GoalModel {name=gmn};
```

```
   enforce domain igfeature fm:FeatureModel { name=gmn,
                   root=f:Feature { name=gmn, owner=fm}};
   enforce domain igtraceability tm:TraceabilityModel{name=tmn};
  where {tmn='Traceability Model of ' + gmn}
}

top relation ConcernToServiceFeature
// Maps Concerns to main or service features
{
   cn: String;
   minCardinality: Integer;

   checkonly domain iggoal c:Concern {owner=gm:GoalModel {},
name=cn, root=rge:HardElement {} };
   enforce domain igfeature cf:Feature
              {owner=fm:FeatureModel {root=rf:Feature}, name =
cn,
               decompose=fd:FeatureDecomposition { owner=fm,
 target=rf,
 cardinalityMin=minCardinality,
 cardinalityMax=1} };
   enforce domain igtraceability tm:TraceabilityModel{};

   when {
      GoalModelToFeatureModel (gm, fm, tm);
   }
   where {
      minCardinality = if (c.isMandatory) then 1 else 0;
      GoalRelationshipsToFeatureRelationships (rge, cf, tm);
   }
}

relation GoalRelationshipsToFeatureRelationships
// Intermediate relation to group following ones
{
   checkonly domain iggoal ge:GoalElement {};
   enforce domain igfeature f:Feature {}
   enforce domain igtraceability tm:TraceabilityModel{};

   where {
      ORDecompositionToFeatureDecomposition(ge, f, tm);
      ANDDecomposition(ge, f, tm);
      OperationalizationToFeatureDecomposition(ge, f, tm);
      TaskToTraceability(ge, f, tm);
      CorrelationToTraceability(ge, f, tm);
      AdviceToRequireTarget(ge, f, tm);
   }
}
```

Fig. 10 Initial relations for goalToFeature transformation

– *GoalModelToFeatureModel*: Creates a new FeatureModel and TraceabilityModel from the GoalModel. It is a starting rule, and therefore a top relation. It also creates the root feature of the

FeatureModel that represents the complete Product Line. Root feature will group the set of trees in GoalModel that represents different Concerns. The names of the models, and the Root Feature are the same that the source GoalModel, and the name of the traceability model is similar to the goal model. Note that this must be done with an intermediate variable (defined in the starting part of the relation definition).

- *ConcernToService*: in I-Gandalf, concerns are mapped to higher level features (Services in feature classification), that is, children of the Root Feature. Here, we create as new Features and FeatureDecompositions as Concerns. The cardinality of the FeatureDecompositions will be 1..1 provided Concern is mandatory, or 0..1 otherwise. These FeatureDecompositions relate the root feature (source) to the new features. This relation is top, but only is executed when a GoalModelToFeatureModel has been executed, and invokes GoalRelationshipToFeatureRelationship to perform the mapping for each Concern.
- *GoalRelationshipsToFeatureRelationships*: this rule relates GoalElements to Features and allows taking into account the different relationships that have as target or source the GoalElement. This relation will go through all the GoalElements of the candidate GoalModel since each GoalElement decompose a Concern, but features will only be created in some cases. Therefore the relation can have several GoalElements for the same Feature (but not vice versa).

• **Hierarchy rules.** These rules go through the Concern decomposition and creates Features only if there is variability, that is, the Hierarchy is an OR Decomposition or it is an Operationalization. Note that each Hierarchy requires two rules, one to create FeatureDecomposition and another to deal with children elements (create features in OR, do nothing in AND, and create feature and Affect traceability relationship in Operationalization). This separation also allows dealing with Aspect as will be seen later. . Hierarchy rules are shown in Fig. 10.

```
relation ORDecompositionToFeatureDecomposition
// OR Decomposition -> Create new Feature Decomposition, and
// (in ORDecomposedToFeature) new Features
{
  or_rel: IGGoal::OR;
  maxCardinality: Integer;
  checkonly domain iggoal ge:GoalElement {
                                        reverse=or_rel:OR {}
                                        };
  enforce domain igfeature f:Feature {
                       owner=fm:FeatureModel {},
                       isDecomposedBy=fd:FeatureDecomposition {
                                   owner=fm,
                                   cardinalityMin=1,
```

```
                                    cardinalityMax=maxCardinality}
                                   };
  enforce domain igtraceability tm:TraceabilityModel{};

  where {
    maxCardinality = or_rel.target->size();
    ORDecomposedToFeature (or_rel, fd);
    DecomposedWithAddJoinPointToFeature (or_rel, fd);
    DecomposedWithReplaceJoinPointToFeature (or_rel, fd);
  }
}

relation ORDecomposedToFeature
// Creates Features for each OR target
{
  featureName: String;
  tge:IGGoal::GoalElement;
  chf:IGFeature::Feature;

  checkonly domain iggoal or_rel:OR {
                               target=target->including(tge)
                               }{tge.joinpoint->isEmpty()};
  enforce domain igfeature fd:FeatureDecomposition {
                              owner=fm:FeatureModel{},
                              child=child->including(chf)};
  enforce domain igtraceability tm:TraceabilityModel{};
  where {
    chf:Feature{owner=fm, name=featureName};
    featureName = DescriptionToFeatureName(tge.type, tge.topic);
    GoalRelationshipsToFeatureRelationships(tge, chf);
  }
}

relation ANDDecomposition
// Integrates the GoalElement with ancestor following in
// decomposition, but without creating new Features, grouping them
// in upper feature (excepting if some And target is the target of
// a joinpoint, where the And target will be created
{
  checkonly domain iggoal ge:GoalElement {
                              reverse=and_rel:AND {}
                              };
  enforce domain igfeature f:Feature {};
  enforce domain igtraceability tm:TraceabilityModel{};

    where {
    ANDDecomposedToNext (and_rel, f);
    ANDDecomposedWithJoinPointToFeature (and_rel, f);
  }
}

relation ANDDecomposedToNext
// Goes to next GoalElement to analyze if no joinpoints
{
  tge:IGGoal:GoalElement;
```

```
    checkonly domain iggoal and_rel:AND {
                    target=target->including(tge:GoalElement {})
                    } {tge.joinpoint->isEmpty()};
  enforce domain igfeature f:Feature {};
  enforce domain igtraceability tm:TraceabilityModel{};

  where {
    GoalRelationshipsToFeatureRelationships(tge, f);
  }
}

relation OperationalizationToFeatureDecomposition
// Maps Operationalizations to Feature Decomposition,
// creating new Feature, and traceability (contribution)
{
  featureName: String;
  sgt:IGGoal::Softgoal;
  tf:IGFeature::Feature;
  fd:IGFeature::FeatureDecomposition;

  checkonly domain iggoal sg:Softgoal {
                            reverse=op:Operationalization {}
                           };
  enforce domain igfeature f:Feature {
           owner=fm:FeatureModel{},
           isDecomposedBy=isDecomposedBy->(fd:FeatureDecomposition{
                       cardinalityMin=0,
                       cardinalityMax=1
                       })
        };
   enforce domain igtraceability tm:TraceabilityModel{};

   where {
      OperationalizationToFeature(op, fd);
      DecomposedWithAddJoinPointToFeature (or_rel, fd);
      DecomposedWithReplaceJoinPointToFeature (or_rel, fd);
   }
}

relation OperationalizationTaskToFeature
// Maps Operationalizations to Feature Decomposition,
// creating new Feature, and traceability (contribution)
{
  featureName: String;

  checkonly domain iggoal op:Operationalization {
                              source=sgt:Softgoal{},
                              target=tt:Task{}
                              } {tt.joinpoint->isEmpty()};
  enforce domain igfeature fd:FeatureDecomposition {
                            owner=fm:FeatureModel{},
                            target=tf:Feature {name=featureName,
                                               owner=fm
                                              }
                           };
  enforce domain igtraceability tm:TraceabilityModel{};
```

```
  where {
    featureName = DescriptionToFeatureName(tt.type, tt.topic);
    GoalRelationshipsToFeatureRelationships(tt, tf);
  }
}
```

Fig. 10 Hierarchy relations for goalToFeature transformation

- *ORDecompositionToFeatureDecomposition*: creates a new FeatureDecompostion with cardinality 1 to number of GoalElements decomposing the initial GoalElement. To calculate this number we use size() function.
- *ORDecomposedToFeature*: creates, for each decomposed GoalElement, a new Feature related to the FeatureDecomposition created in previous relation. Finally, it goes to GoalRelationshipsToFeatureRelationships with the decomposed GoalElement, and the new Feature.
- *ANDDecomposition*: this rule deals with AND decomposition where, as it does not create new variability, no new feature is created. In this relation just go through the Decomposition to execute ANDDecomposedToNext
- *ANDDecomposedToNext*: as said before, in AND Decomposition there is no creation of Features / FeatureDecomposition, so this relation only go to GoalRelationshipsToFeatureRelationships with the decomposed GoalElement and the previous Feature. In this way, AND decomposed GoalElements are integrated in previous features. Traceability relationships will be created later in TaskToTraceability relation.
- *OperationalizationToFeatureDecomposition*: Operationalization also provides some variability, since as they solve a Softgoal in certain degree and Softgoal fuzzy nature, they can appear or not. Main differences respecting ORDecomposition is that it is a 1-1 relationship (one Softgoal to one Task), that they are optional (can appear or not) and that an Affect traceability must be created. In this relationship, the FeatureDecomposition is created with cardinality 0..1, and following rule is executed.
- *OperationalizationTaskToFeature*: here, a new Feature is created with the name generated from the Operationalizated Task, and the process follows by going back to the rule GoalRelationshipsToFeatureRelationships. Note that the traceability relationship Affect will be created later in CorrelationToTraceability.

Note that we do not talk about the other rules on the where clausule of the first relations that deals with each relationship type. Thos rules are used to deal with Aspects. Also, second relations has a constraint in iggoal domain, this contraint is used to limit the execution only when the target GoalElement is not the target of a Joinpoint. This will also explaned in Aspect rules.

- **Traceability.** These rules deal with the creation of traceability relationships. They are very simple, but to fulfill with QVT specification, it is necessary to define intermediate variables for the elements to relate. Note that these rules are executed in GoalRelationshipsToFeatureRelationships, that is executed each time that either GoalElement or Feature to be analyzed change. . Relations are shown in Fig. 11.

```
relation TaskToTraceability
// Creates traceability relationships between Feature and Tasks
{
  tt:IGGoal::Task;
  sf:IGFeature::Feature ;


  checkonly domain iggoal t:Task { };
  checkonly domain igfeature f:Feature { };
  enforce domain igtraceability s:Support {target=tt,
                                           source=sf};
  where {
    tt=t;
    sf=f;
  }
}
relation CorrelationToTraceability
// Creates affect traceability links between features and softgoals
// using the GoalElement correlations (Contributions and
// Correlations)
{
  tsg:IGGoal::Softgoal;
  sf:IGFeature::Feature;

  checkonly domain iggoal ge:GoalElement {
              reverse=reverse->including(c:Correlation {
                                        target=sg:Softgoal} )
              };
  checkonly domain igfeature f:Feature {};
  enforce domain igtraceability c:Affect { target=tsg; source=sf };

  where {
    tsg=sg;
    sf=f;
  }
}
```

Fig. 11 Traceability relations for goalToFeature transformation

- *TaskToTraceability*: this relation creates a new Support relationship between a Feature and a Task. Note that since the same Feature can group several Task, Support relationship will be created for each Task.
- *CorrelationToTraceability*: here, we create the Affect relationship between a Feature that deal with a GoalElement that has a Correlation (Contribution or Operationalization) with a Softgoal.

- **Aspect rules.** These rules are the most complex ones. In our approach, Aspect relates HardElements in different concerns by adding (with Pointcut type Add) an Advice to a Joinpoint (both HardElements), or by replacing (with Pointcut type Replace) the Joinpoint with the Advice. Aspects are considered by default as optional, therefore they create new variability since Features must to consider when the Aspect is applied or when it is not. Also, since feature models are trees, these adding or replacing cannot be done by adding or replacing the feature related to the Advice because that feature will have several parents. To solve this problem, we create a new Feature with a Require constraint to the Advice Feature. Therefore always this new feature is selected, the advice also will selected. Fig. 12 shows these relations. More details are given for each rule.

```
relation DecomposedWithReplaceJoinPointToFeature
// Creates Features for each Decomposition target that is target of
// a Replace Pointcut
{
  featureName, aspectualFeatureName, abstractFeatureName: String;
  maxCardinality: Integer;
  sf, af: IGFeature::Feature;

  checkonly domain iggoal dec:Decomposition {
                            target=tge:GoalElement {
                                 joinpoint=p:Replace {
                                        advice=sge:GoalElement{}
                                 }
                           }{tge.oclIsTypeOf(HardElement) and
                             tge.joinpoint->notEmpty()};
  enforce domain igfeature fd:FeatureDecomposition {
                owner=fm:FeatureModel{},
                child=chf:Feature{owner=fm,
                    name=abstractFeatureName,
                    source=nfd:FeatureDecomposition {owner=fm,
                       cardinalityMin=1,
                       cardinalityMax=1,
                       target=target->including(sf)->including(af)
                    }
                };
  enforce domain igtraceability tm:TraceabilityModel{};

  where {
    featureName = DescriptionToFeatureName(tge.type, tge.topic);
    aspectualFeatureName = DescriptionToFeatureName(tge.type,
      tge.topic) + '-' + DescriptionToFeatureName(a.type, a.topic);
    abstractFeatureName = DescriptionToFeatureName(tge.type,
      tge.topic) + '_aspectual';
    adviceName=DescriptionToFeatureName(sge.type, sge.topic);
    sf= sff:Feature {owner=fm, name=featureName };
    af=aff:Feature {owner=fm, name=aspectualFeatureName,
                    reverse=r:Require{
                         target=tarF:Feature{owner=fm,
                                name=adviceName
                                }
```

```
                        }
                    };
    GoalRelationshipsToFeatureRelationships(tge, sf);
    //Continue descompostion from sf (corresponding feature to tge)
   }
}

relation DecomposedWithAddJoinPointToFeature
// Creates Features for each Decompositon target that is target of
// an Add Pointcut
{
  featureName, aspectualFeatureName, adviceName: String;
  chf:IGFeature::Feature;

  checkonly domain iggoal dec:Decomposition {
                               target=tge:GoalElement {
                                   joinpoint=p:Pointcut {
                                     advice=sge:GoalElement{}
                                   }
                               }{tge.oclIsTypeOf(HardElement) and
                                 tge.joinpoint->notEmpty() and
                                 p.oclIsTypeOf(Add)};

  enforce domain igfeature fd:FeatureDecomposition {
                             owner=fm:FeatureModel{},
                             child=chf:Feature{owner=fm,
                                  name=featureName,
                                  source=nfd:FeatureDecomposition {
                                      owner=fm,
                                      cardinalityMin=0,
                                      cardinalityMax=1,
                                      target=sf:Feature {
                                         owner=fm,
                                         name=aspectualFeatureName,
                                         reverse=r:Require{
                                              target=tarF:Feature{
                                                 owner=fm,
                                                 name=adviceName
                                              }
                                         }
                                      },
                                  }
                            };
  enforce domain igtraceability tm:TraceabilityModel{};

  where {
    featureName = DescriptionToFeatureName(tge.type, tge.topic);
    aspectualFeatureName =
         DescriptionToFeatureName(tge.type, tge.topic) + '-' +
         DescriptionToFeatureName(a.type, a.topic);
    adviceName=DescriptionToFeatureName(sge.type, sge.topic);
    //Continue decompostion from chf (corresponding feature to tge)
    GoalRelationshipsToFeatureRelationships(tge, chf);
   }
}
```

```
relation AdviceToRequireTarget
// Creates advice part of a Pointcut To Require Constraint
{
  p:IGGoal::Poinctcut;

  checkonly domain iggoal hg:GoalElement {
                            targetpoint=pc:Pointcut{
                                 jointpoint=sge:GoalElement{}
                                 }
                            } {hg.oclIsTypeOf(HardElement) and
                               hg.advice->size()>0 };
  enforce domain igfeature fd:FeatureDecomposition {
                            owner=fm:FeatureModel{},
                            child=chf:Feature{owner=fm,
                                   name=adviceName,
                                   reverse=r:Require{
                                      target=tf:Feature{
                                          name=jointpointname
                                          }
                                      }
                                   }
                           };
  enforce domain igtraceability tm:TraceabilityModel{};

  where{
    adviceName=DescriptionToFeatureName(hg.type, hg.topic);
    jointpointname=DescriptionToFeatureName(tf.type, tf.topic);
    }
}

relation ANDDecomposedWithJoinPointToFeature
// Creates a new FeatureDecomposition for each AND target that is
// target of a Pointcut and goes to
// DecomposedWithAddJoinPointToFeature
{
   featureName, aspectualFeatureName: String;
   tge:IGGoal:GoalElement;
   fd:IGFeature::FeatureDecomposition;

   checkonly domain iggoal and_rel:AND {
                         target=target->including(tge:GoalElement {
                              joinpoint=p:Pointcut {
                                   owner= a:Aspect} )
                         } {tge.oclIsTypeOf(HardElement) and
                            tge.joinpoint->notEmpty()};
  enforce domain igfeature f:Feature {
                         isDecomposed=isDecomposed->including (fd)
                         };
  enforce domain igtraceability tm:TraceabilityModel{};

  where {
      DecomposedWithAddJoinPointToFeature (and_rel, fd);
      DecomposedWithReplaceJoinPointToFeature (and_rel, fd);
   }
}
```

Fig. 12 Aspect relations for goalToFeature transformation

- *DecomposedWithAddJoinpointToFeature*: this relation is executed when a decomposed Feature is going to be created, but it is target of an Add. Since we have not created the decomposed Feature, it is created, but also a new Feature (sf) and a FeatureDecomposition to relate them. Since sf is optional, the cardinality is 0..1. Finally, a Require relationship is defined with sf feature as source.
- *DecomposedWithReplaceJoinpointToFeature*: this relation is a bit more complex. In this case, we must allow choosing between the decomposed Feature, and the new Feature. So, we need to create another Feature to represent this new variability point. Relation creates that new Feature (chf) that links with the previous FeatureDecomposition. Also, a new FeatureDecomposition is created (nfd) with cardinality 1..1 (alternative) that links with the decomposed Feature (sf) and a new Feature (af) that is the source of a new Require.
- *AdviceToRequireTarget*: this relation create the relationship between the Require created because of the Joinpoint and the Advice. The advice is also created if not previously done. To do this, when the FeatureDecomposition is analyzed, it is checked if it is target of a Advice relationship, and then the reverse relation (inverse to Require.target as shown in Fig.4) to Require created.
- *ANDDecomposedWithJoinPointToFeature*: first rules are used when the decomposed Feature is going to be created, but in AND Decomposition, a priori, the Feature is not created. This rule is an intermediate step that creates a FeatureDecomposition with cardinality 1..1 (mandatory since it has just one target). Once this step is done, previous rules can be performed.

Last element is a Query (DescriptionToFeatureName), that allows us to generate Feature names from the GoalElement type and topics. This simple version just concatenates Type and the set of Topics, but could be adapted for each domain.

6 MORPHEUS: Tool Support

In order to deal with the complexity brought by the I-GANDALF modelling, automation is a must. With this aim MORPHEUS[2] was selected to provide support to the early stages of I-GANDALF process.

MORPHEUS is structured in three different environments. Specifically, the *Requirement Environment* [19] was used for our proposal. It provides analysts with a requirements metamodelling work context, shown in Fig. 9, for

[2]Interested readers can download some demos of MORPHEUS from http://www.dsi.uclm.es/personal/elenanavarro/research_atrium.htm.

describing metamodels customized according to the project's semantic needs. This environment automatically provides another work context for the description and analysis of Requirement Models according to the active metamodel. Fig. 7 shows what MORPHEUS looks like whenever this context is active. It can be observed that it has a browser that allows the user navigates throughout the active model, and some stencils on the right that makes available the active metamodel for modelling purposes, so that the model is defined only by dragging and dropping the necessary metaelements on the drawing surface.

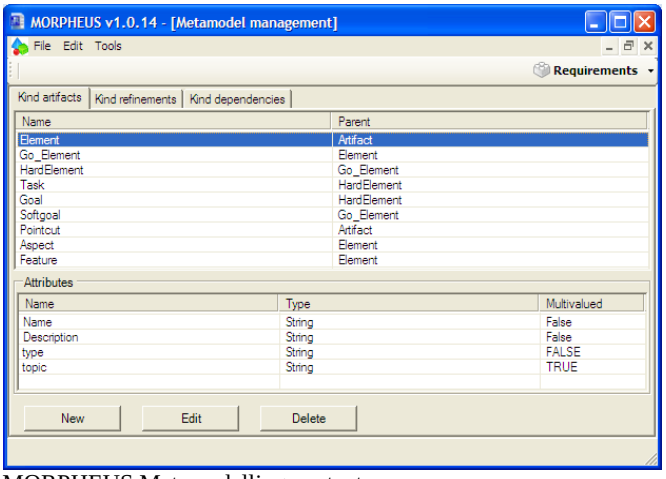

Fig. XXX MORPHEUS Metamodelling context

MORPHEUS is able to execute QVT transformations by using external engines. For this work, we use Medini QVT[3] that uses the Eclipse framework[4] to implement QVT Relations over e-core models, but also can be integrated in other applications. E-core models are a UML subset defined in Eclipse. MORPHEUS generates these e-core models for each meta-model. Therefore, integration is easy.

In addition, to the metamodelling and modelling facilities, MORPHEUS was used because of it has a powerful analysis module that can be customized according to the active metamodel and the specific needs [19].

7 Related Work

There are several proposals that link Goal Models with aspects ideas, but they are intended to find aspects in early stages instead of improving modularity as is

[3] http://projects.ikv.de/qvt
[4] http://www.eclipse.org/

presented in this work. In [26] authors define V-Graph model for the description of intentional nodes, *goals* and *softgoals*, and operational ones, *tasks*. This proposal describes a "manual" process to systematically guide the analyst in the refinement of the Goal Model but having as main aim the early detection of aspects and not its exploitation to improve the modularity.

An extension of V-Graph is presented in [17] where the language Q7 is defined using a BNF grammar. Although BNF grammars have the advantage of being easily parseable, their understandability is quite limited. In this sense, the MDE approach, as it is used in this work, makes the relationships more explicit by using a Meta-model.

Silva developed in [23] an interesting proposal to add aspectual concepts and composition mechanism to V-Graph. Although it is very powerful, it adds too much complexity during the early modeling of the domain. This complexity results in problems to graphically display the models as only the existing crosscutting is represented, hiding how the aspectual relationship is.

Yu et al. [25] propose a technique to perform a smooth transition from Goal Models to design view (Feature Models, state-charts or even architectural ADLs). Specifically, the transition from goal to features is performed by annotations that set if a goal is non-system or if an OR-decomposition relationship is exclusive. By analyzing the combinations of annotations and traditional relationships, they determine mappings to feature relationships, meanwhile goals are directly mapped to features (except non-system ones). We understand that this transformation leaves out the differentiated characteristics of features and cardinalities, which usually depends more on feature nature than on the goal/task it supports.

Another work that relates features to goals, specifically softgoals, is [11], where the authors extend FODA with NFR Framework [4] concepts. They use SIG (Softgoal Interdependency Graph) using features as functional elements, and apply the satisfaction propagation algorithm to study how NFRs are affected. Note that this is also supported by our approach since features can contribute to softgoals, but based on V-Graph, using more recent research in satisfaction propagation ([6],[7]), and exploiting a deeper integration of goal-oriented ideas.

8 Conclusions and ongoing work

We have presented in this work a process called I-GANDALF to perform the transition from goal models to feature models and architecture. This process has been defined by following the ideas of MDE, establishing clearly the different models involved in each stage of the process along with their traceability links. The explicit modeling of these models improves understanding of previous results.

We also exploit one of the main advantages of MDE, that is, transformation model-to-model, providing the definition of the transformation in QVT. Using QVT, we get a initial Feature Model form the Goal Model, with the advantage that traceability links are automatically generated enhancing the quality of the final specification.

Another advantage of the proposal is that a clear separation is established between two different viewpoints: client-oriented view provided by the goal model that can be used to configure specific products; and, architecture-oriented view, that is provided by the feature model.

Other advantage is related to the description of V-Graph elements using types and topics that helps to identify domain concepts, limiting ambiguity of the specification, and even using more generic concepts. In addition, these characteristics assist in finding features, so it is important to analyze what types and topics are the best for each element.

Also, it must be pointed out how the introduction of AOSD ideas in the proposal helps to modularize goal models. Aspect relationships are added to the model as solutions from one concern to another so that they can be specified in a separated way simplifying their comprehension and specification. QVT transformation solves the step between aspectual goal models to non-aspectual feature models.

The different metamodels have been implemented with MORPHEUS giving support to the proposal as seen in Fig. 6. Currently, we are working to implement the transformation rules in the tool, using its analysis module. With the tool completed, next step is to conduct a complete Case Study.

Finally, we intend to integrate the tools already developed for the second stage (from features to architecture) with Morpheus, and therefore to take full advantage of the MDE approach.

Acknowledgements

This work has been funded by the Spanish Ministry of Education and Science under the National R&D&I Program, META Project TIN2006-15175-C05-01 and by Junta de Castilla y Leon (VA-018A07 project). Further funding comes from Research Network ELEPES (TIN2006-27690-E)

References

[1] T. Asikainen, T. Männistöa and T. Soininena, Kumbang: A domain ontology for modelling variability in software product families, Adv. Eng. Inf. 21(1), 23-40, (2007).

[2] H. de Bruin and H. van Vliet, Quality-driven software architecture composition, Journal of System and Software, vol. 66(3), 269-284, Elsevier Science, New York (2003).
[3] J. Castro, M. Kolp and John Mylopoulos, Towards requirements-driven information systems engineering: the Tropos project, Inf. Syst. 27(6), 365-389, (2003).
[4] L. Chung, B. A. Nixon, E. Yu and J. Mylopoulos, Non-Functional Requirements in Software Engineering, Kluwer Academic Publishing, (2000).
[5] K. Czarnecki, S. Helsen and U. W. Eisenecker, Staged configuration through specialization and multilevel configuration of Feature Models, Software Process: Improvement and Practice 10(2), 2005, pp. 143 – 169.
[6] P. Giorgini, J. Mylopoulos, E. Nicchiarelli and R. Sebastiani, Formal Reasoning Techniques for Goal Models, Journal of Data Semantics 1, 2003.
[7] B. González-Baixauli, J. C. S. P. Leite, J. Mylopoulos, Visual Variability Analysis for Goal Models, 12th IEEE Int. Conf. on Requirements Engineering (RE 04), 198-207 (2004).
[8] J. van Gurp, J. Bosch, and M. Svahnberg, On the notion of variability in software product lines, 45-54, IEEE/IFIP Conf. on Software Architecture, IEEE Computer Society, (2001).
[9] G. Halmans, K. Pohl, Communicating the Variability of a Software-Product Family to Customers. Software and System Modeling vol. 2(1), 15-36 (2003).
[10] A. N.Habermann, D. Notkin, Gandalf: software development environments. IEEE Trans. Softw. Eng. 12, 1117-1127 (1986).
[11] S. Jarzabek, B. Yang, and S. Yoeun, Addressing quality attributes in domain analysis for product lines, Software, IEE Proceedings 153(2), 61-73, (2006).
[12] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, A. Spencer Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study, CMU/SEI-90-TR-21, (1990).
[13] M. A. Laguna, B. González-Baixauli, J. M. Marqués: Seamless development of software product lines, 6th Int. Conf. Generative Prog. and Component Eng., 85-94, ACM (2007).
[14] A. van Lamsweerde, Goal-Oriented Requirements Enginering: A Roundtrip from Research to Practice, 12th IEEE Int. Req. Eng. Conf. (RE'04), 4-7, (2004).
[15] S. Lau, Domain Analysis of E-Commerce Systems Using Feature-Based Model Template, PhD Thesis, University of Waterloo, ECE Department, Canada, 2006.
[16] K. Lee, K. C. Kang and J. Lee, Concepts and Guidelines of Feature Modeling for Product Line Software Engineering, Software Reuse: Methods, Techniques, and Tools: Proceedings of the Seventh Reuse Conference (ICSR7), 62—77, Springer, Berlin, (2002).
[17] J. C. S. P. Leite, Y. Yu, L. Liu, E. S. K. Yu and J. Mylopoulos, Quality-Based Software Reuse, 17th Int. Conf. on Adv. Inf. Syst. Eng., LNCS vol. 3520, Springer, Berlin, (2005).
[18] MOF Query/Views/Transformations final adopted specification. OMG document ptc/05-11-01, (2005).
[19] E. Navarro and P. Letelier and D. Reolid and I. Ramos, Configurable Satisfiability Propagation for Goal Models using Dynamic Compilation Techniques, Advances in Information System Development, Springer, Berlin, (2007).
[20] B. Nuseibeh, S. Easterbrook, Requirements Engineering: A Roadmap, The Future of Software Engineering, 22nd Int. Conf. on Soft. Eng., 37-46, ACM, Washington (2000).

[21] C. Prehofer, Feature-oriented programming: A fresh look at objects, ECOOP'97 — Object-Oriented Programming, 419-443, Springer, Berlin, (1997).
[22] M. Salifu, B. Nuseibeh, L. Rapanotti T. Than Tun, Using Problem Descriptions to Represent Variability For Context-Aware Applications, First International Workshop on Variability Modelling of Software-intensive Systems, Lero Tech. Report 2007-01, (2007).
[23] L. F. Silva, An Aspect-Oriented Approach to Model Requirements, Doctoral Consortium, Part of 13th IEEE Int. Req. Eng. Conf., IEEE Computer Society, Washington, (2005).
[24] D. C. Schmidt, Guest Editor's Introduction: Model-Driven Engineering, IEEE Computer, 39(2), 25-31 (2006).
[25] Y. Yu, A. Lapouchnian, S. Liaskos J. Mylopoulos and J.C.S.P. Leite, From Goals to High-Variability Software Design, 17th Int. Symp. on Methodologies for Intelligent Systems, 1-16, LNCS 4994, Berlin, Springer (2008).
[26] Y. Yu and J. C. Sampaio do Prado Leite and J. Mylopoulos, From Goals to Aspects: Discovering Aspects from Requirements Goal Models, 12th IEEE Int. Requirements Engineering Conf., 38-47, IEEE Computer Society, Washington, (2004).

A Appendix: Transformation Rules (QVT)

```
transformation goalToFeature (iggoal:IGGoal, igfeature:IGFeature,
igtraceability:IGTraceability)
{
key IGFeature::FeatureModel {name};
key IGFeature::Feature{name};
key IGFeature::FeatureDecomposition {parent, child};
key IGFeature::Constraint{source, target};

key IGTraceability::Affect{source, target};
key IGTraceability::Support{source, target};

top relation GoalModelToFeatureModel
// Maps Goal to Feature models, and creates root feature
{
  gmn: String;

  checkonly domain iggoal gm:GoalModel { name=gmn};
  enforce domain igfeature fm:FeatureModel { name=gmn,
                              root=f:Feature { name=gmn, owner=fm};
  enforce domain igtraceability tm:TraceabilityModel{};
}

top relation ConcernToServiceFeature
// Maps Concerns to main or service features
{
  cn: String;
  minCardinality: Integer;

  checkonly domain iggoal c:Concern {owner=gm:GoalModel{},
                                     name=cn,
                                     root=rge:GoalElement {}
                                         };
  enforce domain igfeature cf:Feature
                           {owner=fm:FeatureModel {root=rf:Feature},
                            name = cn,
                            decompose=fd:FeatureDecomposition
                                     {owner=fm,
                                      target=rf,
                                      cardinalityMin=minCardinality,
                                      cardinalityMax=1}
                          };
  enforce domain igtraceability tm:TraceabilityModel{};

  when {
     GoalModelToFeatureModel (gm, fm);
  }
  where {
     minCardinality = if (c.isMandatory) then 1 else 0 endif;
     GoalRelationshipsToFeatureRelationships (rge, cf);
  }
}

relation GoalRelationshipsToFeatureRelationships
```

```
// Intermediate relation to group following ones
{
  checkonly domain iggoal ge:GoalElement {};
  enforce domain igfeature f:Feature {};
  enforce domain igtraceability tm:TraceabilityModel{};

  where {
    ORDecompositionToFeatureDecomposition(ge, f);
    ANDDecomposition(ge, f);
    OperationalizationToFeatureDecomposition(ge, f);
    TaskToTraceability(ge, f);
    CorrelationToTraceability(ge, f);
    AdviceToRequireTarget(ge, f);
  }
}

//
// HIERARCHY RELATIONS
//

relation ORDecompositionToFeatureDecomposition
// OR Decomposition -> Create new Feature Decomposition, and
// (in ORDecomposedToFeature) new Features
{
  or_rel: IGGoal::OR;
  maxCardinality: Integer;
  checkonly domain iggoal ge:GoalElement {
                                        reverse=or_rel:OR {}
                                        };
  enforce domain igfeature f:Feature {
                       owner=fm:FeatureModel {},
                       isDecomposedBy=fd:FeatureDecomposition {
                                    owner=fm,
                                    cardinalityMin=1,
                                    cardinalityMax=maxCardinality}
                                   };
  enforce domain igtraceability tm:TraceabilityModel{};

  where {
    maxCardinality = or_rel.target->size();
    ORDecomposedToFeature (or_rel, fd);
    DecomposedWithAddJoinPointToFeature (or_rel, fd);
    DecomposedWithReplaceJoinPointToFeature (or_rel, fd);
  }
}

relation ORDecomposedToFeature
// Creates Features for each OR target
{
  featureName: String;
  tge:IGGoal::GoalElement;
  chf:IGFeature::Feature;

  checkonly domain iggoal or_rel:OR {
                               target=target->including(tge)
                               }{tge.joinpoint->isEmpty()};
```

```
  enforce domain igfeature fd:FeatureDecomposition {
                              owner=fm:FeatureModel{},
                              child=child->including(chf)};
  enforce domain igtraceability tm:TraceabilityModel{};
  where {
    chf:Feature{owner=fm, name=featureName};
    featureName = DescriptionToFeatureName(tge.type, tge.topic);
    GoalRelationshipsToFeatureRelationships(tge, chf);
  }
}

relation ANDDecomposition
// Integrates the GoalElement with ancestor following in
// decomposition, but without creating new Features, grouping them
// in upper feature (excepting if some And target is the target of
// a joinpoint, where the And target will be created
{
  checkonly domain iggoal ge:GoalElement {
                              reverse=and_rel:AND {}
                              };
  enforce domain igfeature f:Feature {};
  enforce domain igtraceability tm:TraceabilityModel{};

  where {
    ANDDecomposedToNext (and_rel, f);
    ANDDecomposedWithJoinPointToFeature (and_rel, f);
  }
}

relation ANDDecomposedToNext
// Goes to next GoalElement to analyze if no joinpoints
{
  tge:IGGoal:GoalElement;

  checkonly domain iggoal and_rel:AND {
                    target=target->including(tge:GoalElement {})
                    } {tge.joinpoint->isEmpty()};
  enforce domain igfeature f:Feature {};
  enforce domain igtraceability tm:TraceabilityModel{};

  where {
    GoalRelationshipsToFeatureRelationships(tge, f);
  }
}

relation OperationalizationToFeatureDecomposition
// Maps Operationalizations to Feature Decomposition,
// creating new Feature, and traceability (contribution)
{
  featureName: String;
  sgt:IGGoal::Softgoal;
  tf:IGFeature::Feature;
  fd:IGFeature::FeatureDecomposition;

  checkonly domain iggoal sg:Softgoal {
                            reverse=op:Operationalization {}
```

```
                          };
  enforce domain igfeature f:Feature {
           owner=fm:FeatureModel{},
           isDecomposedBy=isDecomposedBy->(fd:FeatureDecomposition{
                       cardinalityMin=0,
                       cardinalityMax=1
                       })
        };
   enforce domain igtraceability tm:TraceabilityModel{};

   where {
      OperationalizationToFeature(op, fd);
      DecomposedWithAddJoinPointToFeature (or_rel, fd);
      DecomposedWithReplaceJoinPointToFeature (or_rel, fd);
   }
}

relation OperationalizationTaskToFeature
// Maps Operationalizations to Feature Decomposition,
// creating new Feature, and traceability (contribution)
{
  featureName: String;

  checkonly domain iggoal op:Operationalization {
                             source=sgt:Softgoal{},
                             target=tt:Task{}
                             } {tt.joinpoint->isEmpty()};
  enforce domain igfeature fd:FeatureDecomposition {
                            owner=fm:FeatureModel{},
                            target=tf:Feature {name=featureName,
                                               owner=fm
                                              }
                           };
  enforce domain igtraceability tm:TraceabilityModel{};

  where {
    featureName = DescriptionToFeatureName(tt.type, tt.topic);
    GoalRelationshipsToFeatureRelationships(tt, tf);
  }
}

//
// TRACEABILITY RELATIONS
//

relation TaskToTraceability
// Creates traceability relationships between Feature and Tasks
{
  tt:IGGoal::Task;
  sf:IGFeature::Feature ;


  checkonly domain iggoal t:Task { };
  checkonly domain igfeature f:Feature { };
  enforce domain igtraceability s:Support {target=tt,
                                           source=sf};
```

```
  where {
        tt=t;
        sf=f;
  }
}
relation CorrelationToTraceability
// Creates affect traceability links between features and softgoals
// using the GoalElement correlations (Contributions and
// Correlations)
{
  tsg:IGGoal::Softgoal;
  sf:IGFeature::Feature;

  checkonly domain iggoal ge:GoalElement {
              reverse=reverse->including(c:Correlation {
                                        target=sg:Softgoal} )
              };
  checkonly domain igfeature f:Feature {};
  enforce domain igtraceability c:Affect { target=tsg; source=sf };

  where {
    tsg=sg;
    sf=f;
  }
}

//
// ASPECT RELATIONS
//

relation DecomposedWithReplaceJoinPointToFeature
// Creates Features for each Decomposition target that is target of
// a Replace Pointcut
{
  featureName, aspectualFeatureName, abstractFeatureName: String;
  maxCardinality: Integer;
  sf, af: IGFeature::Feature;

  checkonly domain iggoal dec:Decomposition {
                           target=tge:GoalElement {
                                joinpoint=p:Replace {
                                       advice=sge:GoalElement{}
                                }
                           }{tge.oclIsTypeOf(HardElement) and
                             tge.joinpoint->notEmpty()};
  enforce domain igfeature fd:FeatureDecomposition {
               owner=fm:FeatureModel{},
               child=chf:Feature{owner=fm,
                   name=abstractFeatureName,
                   source=nfd:FeatureDecomposition {owner=fm,
                      cardinalityMin=1,
                      cardinalityMax=1,
                      target=target->including(sf)->including(af)
                   }
               };
```

```
  enforce domain igtraceability tm:TraceabilityModel{};

  where {
    featureName = DescriptionToFeatureName(tge.type, tge.topic);
    aspectualFeatureName = DescriptionToFeatureName(tge.type,
      tge.topic) + '-' + DescriptionToFeatureName(a.type, a.topic);
    abstractFeatureName = DescriptionToFeatureName(tge.type,
      tge.topic) + '_aspectual';
    adviceName=DescriptionToFeatureName(sge.type, sge.topic);
    sf= sff:Feature {owner=fm, name=featureName };
    af=aff:Feature {owner=fm, name=aspectualFeatureName,
                    reverse=r:Require{
                         target=tarF:Feature{owner=fm,
                                name=adviceName
                                }
                         }
                   };
    GoalRelationshipsToFeatureRelationships(tge, sf);
    //Continue descompostion from sf (corresponding feature to tge)
   }
}

relation DecomposedWithAddJoinPointToFeature
// Creates Features for each Decompositon target that is target of
// an Add Pointcut
{
  featureName, aspectualFeatureName, adviceName: String;
  chf:IGFeature::Feature;

  checkonly domain iggoal dec:Decomposition {
                             target=tge:GoalElement {
                                 joinpoint=p:Pointcut {
                                   advice=sge:GoalElement{}
                                 }
                             }{tge.oclIsTypeOf(HardElement) and
                               tge.joinpoint->notEmpty() and
                               p.oclIsTypeOf(Add)};

  enforce domain igfeature fd:FeatureDecomposition {
                            owner=fm:FeatureModel{},
                            child=chf:Feature{owner=fm,
                                 name=featureName,
                                 source=nfd:FeatureDecomposition {
                                     owner=fm,
                                     cardinalityMin=0,
                                     cardinalityMax=1,
                                     target=sf:Feature {
                                        owner=fm,
                                        name=aspectualFeatureName,
                                        reverse=r:Require{
                                             target=tarF:Feature{
                                                owner=fm,
                                                name=adviceName
                                             }
                                        }
                                     },
```

```
                                }
                            };
  enforce domain igtraceability tm:TraceabilityModel{};

  where {
    featureName = DescriptionToFeatureName(tge.type, tge.topic);
    aspectualFeatureName =
         DescriptionToFeatureName(tge.type, tge.topic) + '-' +
         DescriptionToFeatureName(a.type, a.topic);
    adviceName=DescriptionToFeatureName(sge.type, sge.topic);
    //Continue decompostion from chf (corresponding feature to tge)
    GoalRelationshipsToFeatureRelationships(tge, chf);
   }
}

relation AdviceToRequireTarget
// Creates advice part of a Pointcut To Require Constraint
{
  p:IGGoal::Poinctcut;

  checkonly domain iggoal hg:GoalElement {
                            targetpoint=pc:Pointcut{
                                 jointpoint=sge:GoalElement{}
                                 }
                            } {hg.oclIsTypeOf(HardElement) and
                               hg.advice->size()>0 };
  enforce domain igfeature fd:FeatureDecomposition {
                            owner=fm:FeatureModel{},
                            child=chf:Feature{owner=fm,
                                   name=adviceName,
                                   reverse=r:Require{
                                      target=tf:Feature{
                                          name=jointpointname
                                          }
                                      }
                                   }
                           };
  enforce domain igtraceability tm:TraceabilityModel{};

  where{
    adviceName=DescriptionToFeatureName(hg.type, hg.topic);
    jointpointname=DescriptionToFeatureName(tf.type, tf.topic);
    }
}

relation ANDDecomposedWithJoinPointToFeature
// Creates a new FeatureDecomposition for each AND target that is
// target of a Pointcut and goes to
// DecomposedWithAddJoinPointToFeature
{
   featureName, aspectualFeatureName: String;
   tge:IGGoal:GoalElement;
   fd:IGFeature::FeatureDecomposition;

   checkonly domain iggoal and_rel:AND {
                        target=target->including(tge:GoalElement {
```

```
                              joinpoint=p:Pointcut {
                                   owner= a:Aspect} )
                            } {tge.oclIsTypeOf(HardElement) and
                               tge.joinpoint->notEmpty()};
  enforce domain igfeature f:Feature {
                            isDecomposed=isDecomposed->including (fd)
                            };
  enforce domain igtraceability tm:TraceabilityModel{};

  where {
      DecomposedWithAddJoinPointToFeature (and_rel, fd);
      DecomposedWithReplaceJoinPointToFeature (and_rel, fd);
   }
}

//
// QUERIES
//

query DescriptionToFeatureName (type:IGGoal::DescriptionElement,
topic:Set(IGGoal::DescriptionElement)):String
{
    if (topic->isEmpty()) then type.name
    else topic->first().oclAsType(DescriptionElement).name + ' ' +
DescriptionToFeatureName (type, topic->excludes(topic->first()))

}

} // END Transformation
```