Product Portfolio Scope Optimization Based on Features and Goals

Joseph Gillain
University of Namur
jgillain@fundp.ac.be

Stephane Faulkner
University of Namur
sfaulkner@fundp.ac.be

Patrick Heymans
University of Namur
phe@info.fundp.ac.be

Ivan Jureta
University of Namur
ijureta@fundp.ac.be

Monique Snoeck
Katholieke Universiteit Leuven
monique.snoeck@econ.kuleuven.be

ABSTRACT

In this paper we propose a mathematical program able to optimize the product portfolio scope of a software product line and sketch both a development and a release planning. Our model is based on the description of customer needs in terms of goals. We show that this model can be instantiated in several contexts such as a market customization strategy or a mass-customization strategy. It can deal with Software Product Line development from scratch as well as starting from a legacy software base. We demonstrate its applicability with an example based on a case study.

Categories and Subject Descriptors

D.2.13 [**Reusable Software**]: Domain engineering

General Terms

Economics

Keywords

Software product line; scoping optimization; product portfolio; release planning

1. INTRODUCTION

Even if it provides customized software products, Software Product Line Engineering (SPLE) is currently regarded as an efficient approach to achieve large scale reuse. By adopting SPLE, an organization can help achieve different objectives, e.g., reduce their cost, provide adapted solutions for a variety of customers or decrease the time-to-market of software products.

A software product line (SPL) is described as *a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way* [1]. When developing a SPL, one of the main factors impacting the previously cited objectives is the SPL scope. As described by P. Clements, scoping is defined as the "*activity that bounds a system or set of systems by defining those behaviors or aspects that are in and those behaviors or aspects that are out*" [2]. Determining the scope is mainly a trade-off question. On the one hand, if the scope is too large (i.e. the SPL includes many products which cover a large set of different markets), product members vary too much and the commonality is reduced. Consequently, economies of scales eventually drop and the time-to-market reduction cannot be achieved. On the other hand, if the scope is too narrow, the core asset base (i.e. the common part) does not satisfy the needs of enough customers and the return on investment never materializes.

According to Schmid [3], there are three types of scoping. *Product portfolio scoping* aims at identifying the particular products that should be developed as well as the features they should provide. *Domain scoping* is the task of bounding the domains that are relevant to the SPL. *Asset scoping* aims at identifying functional parts of the SPL that should be developed in a reusable manner. Three goal levels for each scoping type can be distinguished: *identification*, *evaluation* and *optimization* of scope. As highlighted by recent reviews, current scoping methods fail in providing optimization methods. Some provide optimization for the product portfolio scope but there are no domain or feature optimization methods yet [4][5].

In this paper we present a mathematical model based on the joint use of goals and features. We show how this approach can help to optimize the SPL scope, especially product portfolio and assets scoping. We suggest that during scoping, identification and evaluation of the three types of scoping can be performed separately. Nonetheless, finding a global optimum for the SPL scope requires that the optimization of the three scoping types are performed in an integrated model. Deciding which features will be included in each product, determining which features will be reusable and selecting domains are tasks that cannot be executed independently. Otherwise the optimization would likely return a local optimum.

In order to integrate those three considerations into a single model we decided to use both goals and features in a SPL life cycle profit optimization. The reasoning behind such an approach is that the purpose of all SPL objectives is to increase profits. Moreover, we assume the customers' willingness-to-pay (WTP) for a system-to-be is function of

SPLC '12 , September 02 - 07 2012, Salvador, Brazil

the utility it provides and this utility is function of the satisfied goals and not of the provided features. Indeed, some features have value only when used with others, e.g., because of feature interactions. Goal models are thus used as a problem-oriented approach while features are solution-oriented aspects. The proposed scoping method optimizes the matching between both aspects. This approach has to be considered as a complementary method with other current approaches such as PuLSE-Eco [3].

The next section covers the use of both goals and features and we discuss the profit maximization objective. The mathematical model is presented in Section 3. In Section 4, we underline that further commonality and variability analysis is required. The adaptation of the model to different contexts is described in Section 5. In Section 6, we show that this model is particularly useful in iterative development. We then illustrate the applicability of our model by applying it on the *market maker* case study from [6]. We finally discuss related work and conclude in Section 8 and 9.

2. FEATURES AND GOALS FOR SCOPE OPTIMIZATION

As highlighted by K. Schmid [3] and D. Nazareth et al. [7], advantages resulting from software reuse in SPLE can take several forms. It is important to highlight that such advantages are mainly raised by economic considerations rather than technical factors. These potential reuse benefits are:

- reduction in development cost,
- reduction in time-to-market,
- increase in programmer productivity,
- improvement in software quality,
- improvement in maintainability,
- improvement in project planning.

Determining the scope of the product portfolio will have an impact on those benefits. However, assessing and predicting those benefits is difficult since varying the scope can have different effects on those benefits. Determining an optimal scope regarding all those advantages could require the use of multiple objectives decision framework. However, such process raises several difficulties such as expressing the preferences between those objectives.

Our method is based on the assumption that all the above objectives are driven by the same general purpose: profit maximization. For example, reducing the time-to-market aims at getting revenue earlier and benefit from market opportunities. Considering only this single objective allows to design a mathematical model taking into account the previously cited considerations. Generally speaking, the SPL profit is the difference between its revenue and its cost. The SPL scope is an important factor influencing profit. By increasing the scope, an organization can increase its revenue since the set of potential customers will increase due to a larger set of products. However, as discussed in Section 1, it will increase the variability and decrease the commonality between product members implying a reduction of the economies of scale on the side of the development cost.

As discussed above, identification and evaluation of SPL scopes have to performed before optimization. Regarding our model, identification will result in a set of goal models for different customers or domains, a set of relevant features to be included in the future products and an identification of reusability of these features. The second step, namely the evaluation phase, will consist in assessing the value of goals, costs of features and costs of reuse. The final step is the optimization of the SPL scope on the basis of the previously identified and evaluated artifacts. We briefly discuss the identification and evaluation steps in this section but a complete description of those first two steps is out of the scope of this paper. We will focus on the third step of scoping in the next section of this paper.

2.1 Software Product Line Revenue

In order to generate revenue (e.g. by being sold), products from a SPL have to satisfy customers' requirements. Before deciding which product will be part of the SPL, we need to know what are the requirements of potential customers. In this context, goal models present two advantages compared to other requirements methods such as object-oriented requirements modeling. Firstly, it allows us to analyze alternative ways to satisfy the same requirements. E.g., two customers from different domains can have several alternative solutions satisfying their requirements, two of which are satisfiable with a common SPL. Comparing alternative solutions makes it possible to find the largest possible core asset base for the planned SPL. Secondly, goal models capture the purpose of the software, not the way it will be implemented. Those requirements are more stable which is important because determining the scope needs to be performed on quite stable considerations. Indeed, it is more difficult to modify the scope than a feature.

Moreover, as previously discussed, goals are good utility indicators. Customers are willing to pay for a system which satisfies their goals, as this is how the client can understand the value of the system-to-be, rather than considering the specific features. For example, financial organizations intrinsically do not care if aggregated financial data are aggregated by the system or received from an external source, as long as they are effectively aggregated. Which of the features will be selected will depend on softgoals (i.e., nonfunctional goals), which may require data to be reliable or privacy of the data recipient to be ensured. Thus, the requirements problem can be stated as follows: finding a set of tasks (i.e., ways of achieving goals) such that under some domain assumptions, stakeholders goals are satisfied [8].

Using the CORE ontology for requirements engineering [8], the requirements problem can be formulated as: *"Given the elicited domain assumptions, goals, quality constraints, softgoals and tasks, some of which being optional or mandatory, the engineer has to find tasks and domain assumptions which satisfy all mandatory goals, quality constraints, and ideally also satisfy at least some of the optional goals and quality constraints."* This is expressed by the following formalization: $K, T \mid\sim G, S, Q$, where the symbols respectively stand for the sets of domain assumptions, the tasks, the defeasible consequence relation, the goals, the softgoals and the quality constraints. Given a requirements problem, SPL engineers are interested in finding a product (i.e. a valid set of features) able to realize tasks of the requirements problem, formally $K', p \mid\sim T$ where K' states the conditions of tasks realization by the product and $p \in \mathcal{P}(F)$[1]. Consequently,

[1] $\mathcal{P}(F)$ is the powerset of a set of features F

the extended requirements problem for a particular customer becomes $K'', p \nvdash\sim G, S, Q$ where $K \cup K' = K''$. Then, the process of developing a SPL consists in finding a SPL such that for a given set of n customers with $i = (0 \dots n)$, there is a product p_i satisfying $K_i'', p_i \nvdash\sim G_i, S_i, Q_i$ with p_i being a product member of the SPL.

As highlighted by Müller, methods such as the conjoint analysis can be used to evaluate a customer's WTP [4]. However, further considerations about the identification process of customer needs and the method to be used in order to evaluate them is out of the scope of this paper.

2.2 Software Product Line Cost

In this paper, we consider that a product member from a SPL is defined by the set of features F it is made of. This description requires a clear definition of what a feature is. Among the different feature definitions [9], our model is compliant with Batory's definition which states that a feature is an increment of product functionalities [10]. Those features will be used to model the *software variability* which refers to the ability of a software system or artifact to be efficiently extended, changed, customized or configured for use in a particular context [11].

We influence the scope by deciding which features we include. They are a means to realize tasks, and thereby assess the cost of a SPL needed to satisfy a particular set of tasks. Developers can more easily give an estimation of the development cost of a feature than of the cost to satisfy a goal since the latter can have alternative solution. On the contrary, they are no good measurements for customers' WTP customers regarding a product. For reasons mentioned above, we see goals as a better means than features to evaluate revenue that can be generated. When evaluating a software product, customers will not assess features in an isolated way, but rather how features can help to satisfy their requirements, i.e. their goals, softgoals and quality constraints.

For assessing a SPL cost, we have to underline that SPLE consists of two main processes, namely domain engineering and application engineering. The former is the process of SPLE in which the commonality and the variability of the SPL are defined and realized while the latter is the process in which the applications of the SPL are built by reusing domain artifacts and exploiting the SPL variability [12]. Böckle et al. introduce a cost function for SPLs based on this distinction [13].

$$C_{org} + C_{cab} + \sum_{n}^{i=1} C_{unique}(p_i) + \sum_{n}^{i=1} C_{reuse}(p_i) \qquad (1)$$

where C_{org} is the cost to an organization for adopting SPLE. C_{cab} is the cost to develop a core asset base (CAB) suited to support the planned SPL. $C_{unique}(p_i)$ is the cost to develop the unique software part of the product p_i that is not in the core asset base while $C_{reuse}(p_i)$ is the cost to reuse the core assets for the product p_i.

Since the core asset base and the products are described in terms of features, assessing the above cited costs in our framework will require that we evaluate features' cost. We assume that C_{org} does not depend on the scope of the considered SPL and we therefore do not use it in our optimization. C_{cab} is the cost for developing the set of features which can be reused as part of the SPL. Developing a feature as part of the CAB introduces extra costs for making it more generic or putting it in a register.

Table 1: Description of decision variables.

Indices	
m	goal models $m = (1, \dots, M)$
k	features $k = (1, \dots, K)$
t	periods $t = (1, \dots, T)$
Parameters	
π_{a_m}	revenue from goal a_m
c_k	effort for developing feature k as a unique feature expressed in required development "man-weeks"
Δ	cost factor for generic development
δ	cost factor for CAB feature reuse
c_w	cost of a development man-week
r	discount rate
Decision variables	
Goal Model	
$g_{a_m,t}$	is equal to 1 if the goal a from goal model m with $a_m = 1, \dots, A_m$ is satisfied during the period t, it is 0 otherwise
$i_{b_m,t}$	is equal to 1 if the inference node b from goal model m with $b_m = 1, \dots, B_m$ is satisfied during the period t, it is 0 otherwise
$t_{c_m,t}$	is equal to 1 if the task c from goal model m with $c_m = 1, \dots, C_m$ is realized during the period t, it is 0 otherwise
Software Product Line	
$f_{k,m,t}$	is equal to 1 if the feature k is used in goal model m during the period t, it is 0 otherwise
$f_{k,t}^{\text{cab}}$	is equal to 1 if the feature k has been integrated to the CAB during the period t, it is 0 if not
$f_{k,m,t}^{\text{r}}$	is equal to 1 if the feature k was reused from the CAB for the product satisfying the goal model m during the period t, it is 0 otherwise
$f_{k,m,t}^{\text{u}}$	is equal to 1 if the feature k was uniquely developed for the product satisfying the goal model m during the period t, it is 0 otherwise
w_t	represents the number of development weeks allocated during period t

2.3 Limited Resources and Discount Rate

The development of a SPL is a long term process. Due to limited resources, this development would likely take several years. Taking into account the cost of time and the limited resource is of primary importance to determine the SPL scope but more specifically to determine priorities in the development and release planning.

To take into account the cost of time, we suggest to discount revenues and costs with the weighted average cost of capital (WACC). This rate calculates an organization's cost of capital in which each category of capital is proportionately weighted. It is often used to discount cash flows and determine the net present value (NPV) of a project.

Regarding the limited resources, we assume that the main resource of a SPL provider is its development team.

3. OPTIMIZING THE SPL SCOPE

In this section we describe the mathematical model for finding the optimal scope of a SPL. We showed in Section 2.1 that the requirements engineering problem of SPLs formulated with CORE consists of three steps. Firstly, we have to determine who the relevant customers (from different possi-

ble domains) are and what their needs are. This requires the identification and evaluation of $K_i, T_i \mid\sim G_i, S_i, Q_i$ for different customers. The second step is defining what the products are constituted of. Answering this question implies that we need to identify and evaluate a set of features F which can be used to derive product members, i.e. $p \in \mathcal{P}(F)$. Thirdly, we need to identify conditions for the product to realize the tasks, i.e. K'. The description of our model follows this distinction. After presenting the objective function, we discuss the constituting features. Then, we deal with goal models to describe customer needs. Finally, we state constraints modeling conditions of tasks realization. The whole set of indexes, parameters and decision variables are described in Tab.1.

3.1 The Product Line

Equation (2) is the objective function maximizing the profit which is the difference between the revenue resulting from satisfied goals and the development costs which is evaluated by salary charge. Equation (3) states the development is allocated to either domain engineering or application engineering following the cost function previously discussed. A feature used in a product for a particular segment will have to be reused from the CAB or developed exclusively for this product. This is stated in equation (4). However before reusing a feature, it has to be integrated in the CAB (5).

$$\max \sum_{t=1}^{T} \left(\sum_{m=1}^{M} \sum_{a_m=1}^{A_m} \frac{\pi_{a_m t}}{r^t} g_{a_m,t} - \frac{c_w}{r^t} w_t \right) \tag{2}$$

s.t.

$$\forall t \quad : \quad \sum_{k=1}^{K} \Delta c_k f_{k,t}^{\text{cab}} + \sum_{m=1}^{M} \sum_{k=1}^{K} \left(c_k f_{k,m,t}^{\text{u}} + \delta c_k f_{k,m,t}^{\text{r}} \right) \leq w_t \tag{3}$$

$$\forall m,k,t \quad : \quad f_{k,m,t} \leq \sum_{j=1}^{t} (f_{k,m,j}^{\text{u}} + f_{k,m,j}^{\text{r}}) \tag{4}$$

$$\forall m,k,t \quad : \quad f_{k,m,t}^{\text{r}} \leq \sum_{j=1}^{t} f_{k,j}^{\text{cab}} \tag{5}$$

Additional considerations about features need to be stated. Feature dependencies and interactions are important constraints when developing a SPL. We can identify some patterns when considering feature dependencies. On the one hand, we can identify common features which are features belonging to all product members from the SPL. On the other hand, variable features fall into three categories: alternative, OR and optional features [14]. Moreover, there are some dependencies among features. These can take several forms such as "include" or "exclude" links. Due to some operational dependencies, some of those dependencies and interactions have to be identified since they will have significant implications in the development of systems. Here we present the instantiation of some feature interactions into linear constraints.

The "exclude" constraint expresses that some features can not be active at the same time. E.g., consider a security system. Because of some interferences between waves, radar motion detectors can not work at the same time as wireless networks. In terms of linear constraints, we have to express that a set of features indexed with $X \subseteq \{1, \ldots, K\}$ can not be used in the same product. Then, if we consider that we have j such sets:

$$\forall j,t,m : \sum_{x \in X_j} f_{xmt} \leq 1 \tag{6}$$

We can also model the "mutually required" constraint between sets of features

$$\forall j,t,m : f_{z,m,t} - \frac{1}{|X_j \setminus z|} \sum_{x \in X_j \setminus z} f_{x,m,t} = 0 \tag{7}$$

where $z \in X_j$.

If a feature f_i requires the set of features X to be part of the product while the features from X can be selected independently from f_i, we can model this constraint as following:

$$\forall t,m : f_{i,m,t} - \frac{1}{|X|} \sum_{x \in X} f_{x,m,t} \leq 0 \tag{8}$$

3.2 Goal Models

We assume that requirements are formalized as in Techne [15]. We use this goal-oriented requirements modeling language because firstly it provides simple reasoning rules and secondly, as it is based on the CORE ontology for requirements it is compatible with other goal-oriented requirements modeling languages. In Techne, a requirement is:

- a domain assumption, if it states a belief of stakeholders,
- a goal if the proposition expresses a desire, i.e., it states a property that we want to see holding; e.g., "Order books are traded",
- a quality constraint if it places a constraint on desirable values of a quantifiable property,
- a softgoal if it refers to a desirable value of a property in such a way that it is not possible to identify exactly which value, or range of values of that property it refers to; e.g., "Trading is reliable",
- a task if the proposition in it says what to do; e.g., "Aggregate financial data".

An example of a goal model is depicted in Fig.1. Requirements can be in three binary or n-ary relations. The triangle `I` in Fig.1 refers to the inference relation where incoming arrows are linked to the premises and the outcoming to the conclusion: for example, since we allow modus ponens in Techne, we can deduce g_1 from either $\{g_2, g_3, g_4\}$ or $\{g_5, g_6\}$. Such construction models goal refinement.

Inference nodes can be model into a linear program as follows. Consider a set of inference nodes i_x (with $x = (1 \ldots X)$) refining a goal g, the premises being noted as p_{j_x} (with $j_x = (1 \ldots J_x)$ and the conclusion being written as c (i.e. the goal). The inference nodes are then modeled as two sets of constraints:

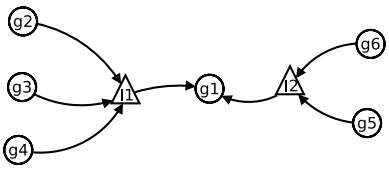

Figure 1: Example of a goal refinment

$$c \leq \sum_{x=1}^{X} i_x \quad (9)$$

$$(\forall x) i_x \leq \frac{1}{J_x} \sum_{j_x=1}^{J_x} p_{j_x} \quad (10)$$

Actually, this constraints structure is used for goal refinement, goal operationalization (i.e., decomposition of goals into tasks) and tasks realization (i.e. realization of tasks thanks to particular features). Consequently, the conclusion c can be instantiated by a goal or a task and the premises can be goals, tasks or features. In order to illustrate those constraints, consider the following constraints instantiating the goal refinement from Fig.1.

$$g_1 \leq i_1 + i_2 \quad (11)$$
$$i_1 \leq (g_2 + g_3 + g_4)/3 \quad (12)$$
$$i_2 \leq (g_5 + g_6)/2 \quad (13)$$

A requirement can be mandatory, optional, or neither. A mandatory goal means that it must be satisfied while an optional goal means that it is more desirable that it is satisfied, than not, but we will still accept a system which fails to satisfy it. Consequently, additional constraints have to be applied to each goal model. E.g., we assume that a product will be relevant to a customer if all mandatory goals are satisfied. This is expressed by the following constraint. Consider that a goal model is composed of a set of mandatory goals g_i with $i \in Y$ and where Y is the set of indices of mandatory goals, then the constraint is equal to:

$$g_j - \frac{1}{|Y \setminus j|} \sum_{i \in Y \setminus j} g_i = 0 \quad (14)$$

with $j \in Y$. Then, optional goals can be satisfied only if all mandatory ones have been satisfied:

$$\forall o \in O : g_o \leq g_j \quad (15)$$

where O is the set of indices of optional goals and g_j is any mandatory goal. We need to define this constraint with only one mandatory goal as they are mutually required because of (14).

3.3 Tasks realization

Modeling task realizations follows the same pattern as goal refinement. For example modeling the following task realization $(f_1 \wedge f_2) \vee (f_1 \wedge f_3) \rightarrow t_1$ (which states that the task t_1 will be realized either with the features f_1 and f_2 or with f_1 and f_3) will use the following equations:

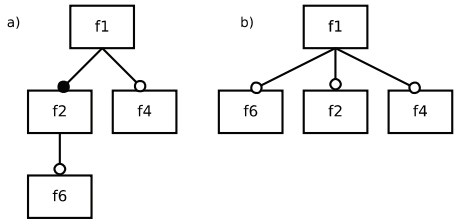

Figure 2: Example of various feature diagram definitions

$$t_1 \leq or_1 + or_2 \quad (16)$$
$$or_1 \leq (f_1 + f_2)/2 \quad (17)$$
$$or_2 \leq (f_1 + f_3)/2 \quad (18)$$

where or_i variables are introduced to represent disjunctions.

4. FURTHER COMMONALITY AND VARIABILITY ANALYSIS

The output of the mathematical model consists among others of different sets of features satisfying each market segment. The model determines if those features are developed during domain whether application engineering.

Actually, the software products satisfying the different market segments (i.e. goal models) could not be members of the SPL for two reasons. Firstly, they can include features developed exclusively for these market segment. Consequently, the product has more features than possible SPL members have. Secondly, it is not mandatory to base the feature constraints definition of the SPL on the strict interpretation of the model output. For example, assume that the model determines the four following products: $p_1 = \{f_1, f_2, f_3, f_6\}$, $p_2 = \{f_1, f_2, f_4, f_6\}$, $p_3 = \{f_1, f_2, f_5\}$ and $p_4 = \{f_1, f_2, f_4, f_7\}$. Moreover, it considers that $\{f_1, f_2, f_4, f_6\} \subseteq$ CAB. A strict definition of the SPL described in a feature diagram could be one such as in Fig.2(a). However, SPL engineers could decide to eventually design a SPL such as in Fig.2(b) which also allows the definition of $p_x = \{f_1, f_6, f_4\}$.

While the features and the features constraints identified before the optimization described the *software variability*, the different decisions taken about the feature constraints in this commonality and variability analysis (as well as possible feature aggregation, addition of feature compound...) will result in the definition of the *Software Product Line variability* which describes the variation between the systems that belong to a SPL in terms of properties and qualities. This analysis is required to disambiguate the two types of variability [16]. Acher et al. suggested a method able to extract feature diagrams from products description [17] and consequently able to support this analysis.

5. CONTEXT-AWARE CONSIDERATIONS

In this section, we describe how to adapt the mathematical model to various SPLE situations. First, we make a distinction between market strategies. Then, we describe

how to take into account the clear separation between domain and application engineering teams. Finally, we show that our model can, to some extent, introduce legacy components considerations.

5.1 Mass-Customization or Mass-Markets

K. Schmid identified two product strategies: the *mass-customization* and the *mass-markets* [18]. Depending on the strategy, constraints of the mathematical model will be different.

In a mass-market situation, the product is mainly defined by the producing organization in order to sell the product in a market segment to a large number of customers during a certain span of time. Consequently, if goal values are an annual expected revenue, revenues from the model can be added each year. Moreover, this product can be improved with new features and a new release can be sold to this segment. This product strategy is the default situation in our mathematical model. Each goal model represents the aggregated needs of a particular market segment and goal values are prediction of yearly sells.

In a mass-customization situation, each product is designed for the specific needs of a customer. Consequently, a product is sold only once and estimated revenue can be obtained only once. It means that each goal model captures the needs of a particular customer, which implies that it can be satisfied only once. However, we can distinguish two situations. In the first case, once a product is derived and deployed customer needs are considered as satisfied and no more changes will be accepted. We will then add these sets of constraints:

$$\forall m, a_m : \quad \sum_{t=1}^{T} g_{a_m,t} \leq 1 \tag{19}$$

$$\forall m, a_m : \quad \sum_{t=1}^{T} g_{a_m,t} \leq p_{m,t} \tag{20}$$

$$\forall m : \quad \sum_{t=1}^{T} p_{m,t} \leq 1 \tag{21}$$

where $p_{m,t}$ is equal to 1 if a product for the market m is delivered at time t and (19) states that a goal can be satisfied only once through the whole set of considered periods and (20) and (21) state that there is one single release of the product dedicated to the customer m.

In the second situation, a product can be updated and a new release can be deployed. In this case, we can consider that unsatisfied goals can still be satisfied later in time. In this case we have to add only the set of equations (19).

5.2 Separation of Domain and Application Engineering Teams

Organizations can for various reasons [6] decide to separate or merge domain and application engineering teams. Those distinct situations can be integrated in our mathematical model. Merged teams is the default situation where w_t in (3) represents the available development resources for one period dedicated to both domain and application engineering.

In order to make a distinction between domain engineering and application engineering, we have to replace the decision variables w_t by w_t^D and w_t^A which represent respectively development weeks for domain engineering and application engineering. Then, (3) has to be replaced by:

$$\forall t : \quad \sum_{k=1}^{K} \Delta c_k f_{k,t}^{\text{cab}} \leq w_t^D \tag{22}$$

$$\forall t : \quad \sum_{m=1}^{M} \sum_{k=1}^{K} \left(c_k f_{k,m,t}^{\text{u}} + \delta c_k f_{k,m,t}^{\text{r}} \right) \leq w_t^A \tag{23}$$

Further situations can also be modeled, e.g. two basis team dedicated to domain and application engineering helped by a flexible team which could be working on both domain and application engineering.

5.3 Encapsulation of Legacy Systems

Developing a SPL on the basis of legacy systems by wrapping some components is a common situation in SPLE. For instance, as described by F. van der Linden, a legacy component can be first wrapped and integrated to a SPL before it will be replaced by a new component [6].

This situation is modeled by adjusting the cost of integrating a feature to the CAB. For example, consider a financial calculation module which was present in a legacy system. We can then consider two features realizing the same tasks where f_1 is the legacy module and f_2 is a similar module developed from scratch. Then we can deduce the cost to integrate f_1 to the CAB will be lower than the cost to integrate f_2. We also have to fix the variable $f_{1,m,t}^{\text{u}} = 0$ since the legacy module can not be developed only for a single product.

Nonetheless, a comprehensive approach of this context should consider extra maintenance cost introduced by legacy components in the CAB. Those extra costs have to be integrated in the cost function.

6. CHANGE AND ITERATIVE DEVELOPMENT

We advocate to apply this model in an iterative and incremental development process and not in a predictive way. Since all products will not be developed at once and since the CAB will incrementally grow as new products are developed, it would be useful to frequently reevaluate the scope.

This results from two facts. First, developing a SPL is a long term process and the SPL life cycle is often largely longer than for a simple software product. So all products can not be developed at the same time. Secondly, it is impossible to exactly predict and evaluate customer needs several years in advance. Environment changes are frequent and can affect previously stated customer needs. Additionally the probability that changes occur is a function of the considered time span. Considering a too long time will obviously result in unexpected change.

Our suggested method can not be considered as an exact predictive method to be applied in the early phase of a SPL life cycle without any later changes. It has to be used in an iterative development process and results have to be considered as a sketch for further investigation. Those considerations justify our use of goal models since they allow to model high-level requirements (which are less likely to change). Instantiated with an incremental and iterative development method, our model can help to control the development and release planning since we can integrate environment changes

in goal models and we can model the development progress of the CAB. E.g. a feature already developed and integrated to the CAB can get a zero development cost and a positive reuse cost.

7. EXAMPLE OF APPLICATION ON A CASE STUDY

In this section, we apply our method to the case study of *market maker* Software AG [1, 6]. In the first part of this section, we present the organization and we state both problem and mathematical model. In the second part, we present the result of the SPL scope optimization.

7.1 Problem Statement

market maker is an organization providing applications able to collect, display and manage financial data. At the end of 2004, the SPL development team consisted of around 25 people and the annual revenue was €5 million per year. For this organization, SPLE was regarded as a key strategic element in addressing new market segments. In 1999, "*when markets were boiling and the demand for innovative products was immense*", they started the development of a SPL for web-based applications, called i*ProductLine nearly from scratch. This SPL aimed at developing web-applications able to collect, validate, store, analyze, aggregate, repackage and distribute financial data. We show in the remainder of this section how our mathematical model can be applied to the market marker case study and how it can determine the SPL scope as well as sketching a development and a release planning.

In 2004, i*ProductLine was instantiated for various markets: information systems for asset managers in banks, market data servers integrated in brokerage systems for on-line ordering, specialized data display services for metal traders and grains and oilseed traders and, content provisioning for financial web portals. A small goal model is depicted for each market segment in Fig.3. For the rest of this example, we assume some potential annual revenue for each goal. Those revenues are presented in thousands of Euro near each related goal. Mandatory goals have bold circles.

On the basis of the case study description, we identified a set of 13 coarse-grained features able to realize tasks identified in each goal models. They are described in Tab.2. We did not consider legacy systems. The CAB integration cost and the reuse cost were calculated by applying respectively a design-for-reuse factor and a design-with-reuse factor on the traditional development cost. We assumed those factors to be respectively $\Delta = 1.5$ and $\delta = 0.2$. Some feature dependencies were identified before the optimization of the product portfolio. Firstly, we expressed that requesting financial data from various xignite web services, i.e. features f_9,f_{10},f_{11} and f_{12}, required that the feature f_8, a data import module, was present in the product. In a propositional logic statement, it is expressed as $f_9 \vee f_{10} \vee f_{11} \vee f_{12} \rightarrow f_8$. We also add the following constraint: $f_4 \vee f_5 \vee f_6 \rightarrow f_3$. Secondly, we stated that if the interface is designed using Java Servlets (i.e. f_2), and that the product uses a database (i.e. f_3), then the product needs to use the feature f_{13} which is JDBC. This constraint was formalized as follows:

$$\forall m, t : f_{2,m,t} + f_{3,m,t} \leq 1 + f_{13,m,t} \quad (24)$$

Additional constraints had to be considered. Indeed, the management demanded that a first product would had been developed within the first 12 months. They also decided to create a complete new team dedicated to the SPL development. There was no team separation between domain and application engineering.

Table 2: Description of identified features and related effort required for unique development in terms of development weeks

	Feature	C_{unique}
f_1	HTML Interface	150
f_2	Java Servlet Interface	250
f_3	DataBase	400
f_4	Portfolio Module	300
f_5	User Management Module	300
f_6	Data Storage	250
f_7	Data Processing	450
f_8	Data Import	300
f_9	xignite metals	50
f_{10}	xignite commodities	50
f_{11}	xignite stocks real-time	50
f_{12}	xignite stocks delayed	50
f_{13}	JDBC	150

The selected discount rate was the weighted average cost of capital (WACC), that is the rate that a company is expected to pay on average to all its security holders to finance its assets. Due to really high variation of the WACC in the software industry [19], we applied an average WACC calculated for small-medium organizations from 1996 to 2006. It is equal to 17.14% [19].

Once goals and features were identified and evaluated, we had to express how those features could realize the tasks from the goal models. Such assessment is depicted in Tab.3.

In this problem, we consider 6 years of concern. Development was allocated to the first 4 years and revenues of the 5^{th} and 6^{th} year were a discounted projection of the 4^{th} year.

7.2 Results Presentation

The entire model was written and resolved with the GNU Linear Programming Kit[2] (GLPK). Results are depicted in Tab.4, Fig.4 and Fig.5.

During the first period, the model plans to develop products for trading market and web portal market. They consist respectively of features $\{f_2, f_8, f_9, f_{10}\}$ and $\{f_2, f_8, f_9\}$. Features $\{f_2, f_8, f_9\}$ will integrate to the CAB. Those products satisfy goals g_1, g_2, g_3, g_4 of traders and goal g_1 of web portals.

During the second period, the CAB will be extended with the feature f_3. This feature will be reused for starting the development of the banks product. It is also planned to reuse it in order to extend the trader product. This product will be further extended with features f_6 and f_{13} (developed uniquely for this product). This new release should satisfy goals g_5 and g_6.

In the third period, it is planned to develop f_{13} again and integrate it to the CAB. This feature will be used in products for web portal and brokers. The feature f_4 will also be integrated into the CAB and it will extend products for web portals and banks. Finally, the product for banks is released (after starting its development in the second period)

[2]http://www.gnu.org/software/glpk/

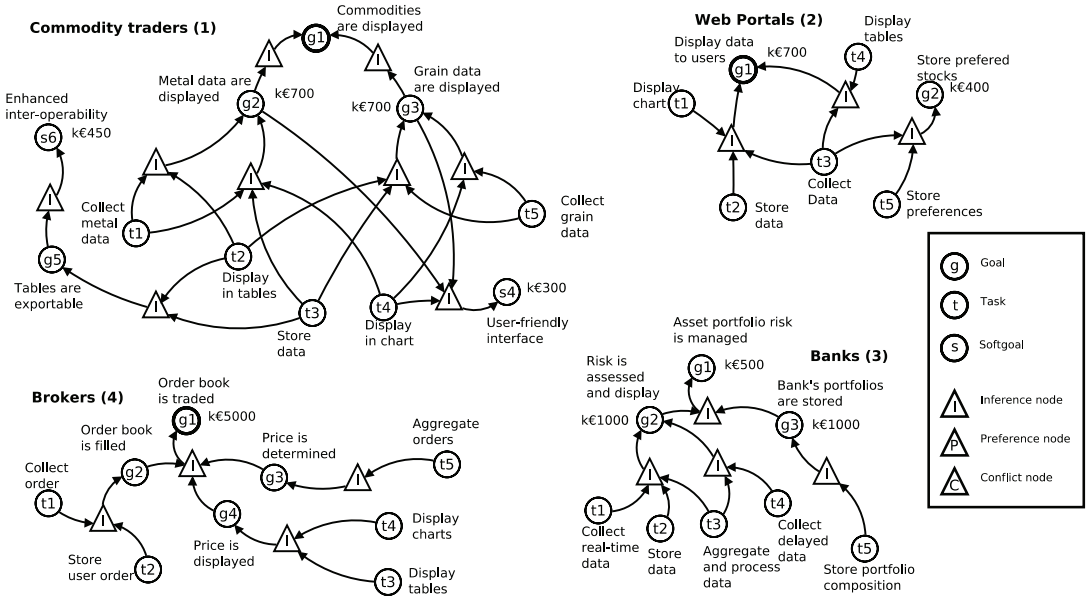

Figure 3: The 4 goal models used to design *market maker* customer segments

Table 3: Task realization description

Web Portals	Commodity Traders	Brokers	Banks
$(f_2 \rightarrow t_1)$	$(f_9 \rightarrow t_1)$	$(f_8 \rightarrow t_1)$	$(f_{11} \rightarrow t_1)$
$(f_6 \rightarrow t_2)$	$(f_1 \vee f_2 \rightarrow t_2)$	$((f_4 \wedge f_5 \wedge f_6) \rightarrow t_2)$	$((f_5 \vee f_6) \rightarrow t_2)$
$((f_9 \vee f_{10} \vee f_{11} \vee f_{12} \rightarrow t_3)$	$(f_6 \rightarrow t_3)$	$((f_1 \vee f_2) \rightarrow t_3)$	$(f_7 \rightarrow t_3)$
$(f_1 \vee f_2 \rightarrow t_4)$	$(f_2 \rightarrow t_4)$	$(f_2 \rightarrow t_4)$	$(f_{12} \rightarrow t_4)$
$((f_4 \wedge f_5) \rightarrow t_5)$	$(f_{10} \rightarrow t_5)$	$(f_7 \rightarrow t_5)$	$(f_4 \rightarrow t_5)$

and it should satisfy goal g_3.

The last period of development is dedicated to complete the product for brokers. f_2, f_3, f_4 and f_8 are reused from the CAB while f_5, f_6 and f_7 are uniquely developed for this product.

We see in Fig.5 that break even is reached during the third period. The expected discounted operating profit over the 6 periods is about € 12.300.000 and the expected discounted revenue of the last period is about € 4.200.000.

Fig.4 shows a possible feature diagram for describing the SPL variability after an analysis of the variability and commonality between products. Two compound features were added (i.e. *i** and *data*).

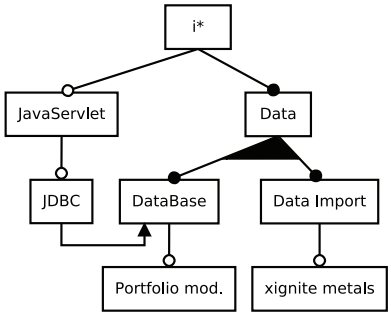

Figure 4: A possible feature diagram for i*ProductLine

8. RELATED WORK

Although several SPL scoping methods have been designed, few address the optimization goal of the product portfolio scoping on the basis of the profitability concern.

Recently, J. Müller suggested a mathematical program to optimize the product portfolio on the basis of a profit maximization objective [4]. His model is based on previous work in product lines (i.e. not software) and on conjoint analysis. Although the idea to use a mathematical program to optimize the SPL scope is similar to ours, the underlying assumptions of the two models are different. First, he assumes a SPL and all its valid configuration products and it optimizes the selection of the product-segment pair given that a configuration products can be implemented by a various set of asset components. His method allows to determine the optimal price and take competitors into account. However, it does not introduce any time considerations what prevents any possibility of setting development priorities or release planning which is however a primary concern in SPLE and assume an a priori complete SPL specification.

Several survey of Software Product Line Scoping have been performed. In their survey [5], I. John and M. Eisenbarth identify 16 scoping approaches but they highlight that scoping optimization has only been partially addressed. Moreover none of them integrate the optimization of the three types of scoping.

Table 4: The product portfolio planning - Increments between periods are bolded, r = reuse, u = uniquely developed

Per.	Segment	Satisfied goals	Product composition													CAB
			f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	
1	traders	$\{g_1, g_2, g_3, g_4\}$		**r**						**r**	**r**	**u**				$\{f_2, f_8, f_9\}$
	web	$\{g_1\}$		**r**						**r**	**r**					
	banks	$\emptyset$														
	brokers	$\emptyset$														
2	traders	$\{g_1, g_2, g_3, g_4, g_5, g_6\}$		r	**r**			**u**		r	r	u			**u**	$\{f_2, f_3, f_8, f_9\}$
	web	$\{g_1\}$		r						r	r					
	banks	$\emptyset$			**r**											
	brokers	$\emptyset$														
3	traders	$\{g_1, g_2, g_3, g_4, g_5, g_6\}$		r	r			u		r	r	u			u	$\{f_2, f_3, f_4, f_8, f_9, f_{13}\}$
	web	$\{g_1, g_2\}$		r	**r**	**r**	**u**			r	r				**r**	
	banks	$\{g_3\}$			r	**r**										
	brokers	$\emptyset$														
4	traders	$\{g_1, g_2, g_3, g_4, g_5, g_6\}$		r	r			u		r	r	u			u	$\{f_2, f_3, f_4, f_8, f_9, f_{13}\}$
	web	$\{g_1, g_2\}$		r	r	r	u			r	r				r	
	banks	$\{g_3\}$			r	r										
	brokers	$\{g_1, g_2, g_3, g_4\}$		**r**	**r**	**r**	**u**	**u**	**u**	**r**				**r**		

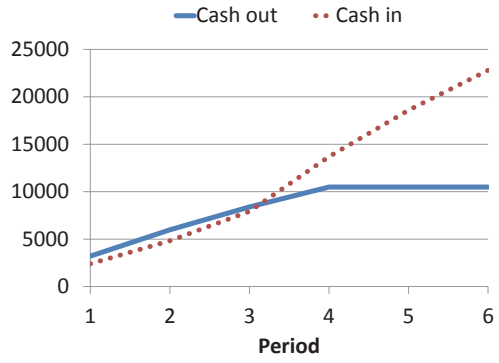

Figure 5: Expected discounted cumulated cash flows

Regarding the five methods addressing the product portfolio analysis [20, 21, 12, 22, 23] only [20] also addresses asset scoping. Moreover as highlighted by J. Müller none of those five methods integrate market and cost perspectives which prevents any profitability consideration.

The Quality Function Deployment Product Portfolio Planning suggested by Helferich [21] elicits required features of products of a SPL and asks engineers about technical feasibility. It also allows to identify customer segments. However it does not consider revenue and cost aspects and therefore cannot be used for profit maximization. However, this method can usefully be regarded as complementary to ours since it identifies features and can be used to match them with goal models.

Niehaus et al. present a Kano model which allows to design customer-oriented SPL [12]. It is focused on the product portfolio planning but does not consider any asset scoping.

The issue of a SPL release planning is addressed by several authors but they do not address neither composition of product portfolio nor asset scoping [22, 23]. In single software engineering, Denne et al. already highlighted the importance of "*optimizing the time at which value is returned to the customer, instead of concentrating only on controlling risk and cost.* [24]" They suggest a method to decompose the system into Minimum Marketable Features (i.e. units of customer-valued functionality) and determine the sequence of incremental release which optimizes the NPV of the system. Their research results are complementary to our method.

9. CONCLUSION AND FURTHER WORK

In this paper we suggested that when scoping, identification and evaluation of the three scope types (i.e. product portfolio, domain and assets scoping) can be performed separately but the optimization of those scopes has to be performed in an integrated model. Consequently, we proposed a mathematical program able to optimize the SPL scope and to sketch both a development and a release planning. Our method is based on the assumption that all SPL objectives are eventually driven by profit maximization. Revenue is a function of the customer satisfied needs and cost is a function of the feature development effort.

Our model is based on the description of customer needs in terms of goals. This use is justified because firstly, we showed that goals are better than features to capture the customer's WTP and secondly, they are able to capture the alternative solutions of a requirements problem what is useful to find large commonalities between market segments. More precisely, we use Techne because it allows automated reasoning and it is compatible with other goal-oriented requirements language.

We show that our mathematical model can be instantiated in several contexts such as a market customization strategy or a mass-customization strategy. It can deal with SPL development from scratch as well as from the basis of legacy software. The output of our model is a good basis for further commonalities and variabilities analysis. We also demonstrated its applicability with an example based on the *market maker* case study.

In further work we will consider the following extensions. Firstly, we estimated revenue on the WTP of satisfied goals. However, this assumption constraints the applicability of our model to monopoly setting. Further work should integrate customer decision in presence of competitors products. Secondly, we will study implications and difficulties of using conjoint analysis with goals instead of features. Thirdly, we limited our definition of features to Batory's definition

which allows to simplify the relation between feature and cost. However, a more complete feature definition would require to take into account the non-monotonicity between features and SPL cost function. Then, we need to integrate in our model both risk management (e.g. with stochastic models) and maintenance cost which should impact the CAB size and legacy components integration. Finally, we also started the development of a tool to support the application of this model that we intend to extend with a real case study.

10. REFERENCES

[1] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.

[2] P. Clements, "On the importance of product line scope," in *Software Product-Family Engineering* (F. van der Linden, ed.), vol. 2290 of *Lecture Notes in Computer Science*, pp. 102–113, Springer Berlin / Heidelberg, 2002.

[3] K. Schmid, "A comprehensive product line scoping approach and its validation," in *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, (New York, NY, USA), pp. 593–603, ACM, 2002.

[4] J. Müller, "Value-based portfolio optimization for software product lines," in *Proceedings of the 15th International Software Product Line Conference*, SPLC '11, (Washington, DC, USA), pp. 15–24, IEEE Computer Society, 2011.

[5] I. John and M. Eisenbarth, "A decade of scoping: a survey," in *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, (Pittsburgh, PA, USA), pp. 31–40, Carnegie Mellon University, 2009.

[6] F. J. v. d. Linden, K. Schmid, and E. Rommes, *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

[7] D. L. Nazareth and M. A. Rothenberger, "Assessing the cost-effectiveness of software reuse: A model for planned reuse," *Journal of Systems and Software*, vol. 73, no. 2, pp. 245 – 255, 2004.

[8] I. J. Jureta, J. Mylopoulos, and S. Faulkner, "Revisiting the core ontology and problem in requirements engineering," *IEEE International Requirements Engineering Conference*, pp. 71–80, 2008.

[9] A. Classen, P. Heymans, and P.-Y. Schobbens, "What's in a feature: a requirements engineering perspective," in *Proceedings of the Theory and practice of software, 11th international conference on Fundamental approaches to software engineering*, FASE'08/ETAPS'08, (Berlin, Heidelberg), pp. 16–30, Springer-Verlag, 2008.

[10] D. Batory, D. Benavides, and A. Ruiz-Cortes, "Automated analysis of feature models: challenges ahead," *Commun. ACM*, vol. 49, pp. 45–47, December 2006.

[11] M. Svahnberg, J. van Gurp, and J. Bosch, "A taxonomy of variability realization techniques: Research Articles," *Software Practice & Experience*, vol. 35, no. 8, pp. 705–754, 2005.

[12] K. Pohl, G. Böckle, and F. J. v. d. Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[13] G. Böckle, P. Clements, J. McGregor, D. Muthig, and K. Schmid, "A cost model for software product lines," in *Software Product-Family Engineering* (F. van der Linden, ed.), vol. 3014 of *Lecture Notes in Computer Science*, pp. 310–316, Springer Berlin / Heidelberg, 2004.

[14] K. Lee and K. Kang, "Feature dependency analysis for product line component design," in *Software Reuse: Methods, Techniques, and Tools* (J. Bosch and C. Krueger, eds.), vol. 3107 of *Lecture Notes in Computer Science*, pp. 69–85, Springer Berlin / Heidelberg, 2004.

[15] I. J. Jureta, A. Borgida, N. A. Ernst, and J. Mylopoulos, "Techne: Towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling," in *2010 18th IEEE International Requirements Engineering Conference*, pp. 115–124, IEEE, Sept. 2010.

[16] A. Metzger, K. Pohl, P. Heymans, P.-Y. Schobbens, and G. Saval, "Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis," *Requirements Engineering, IEEE International Conference on*, vol. 0, pp. 243–253, 2007.

[17] M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire, "On extracting feature models from product descriptions," in *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '12, (New York, NY, USA), pp. 45–54, ACM, 2012.

[18] K. Schmid, "An initial model of product line economics," in *Software Product-Family Engineering* (F. van der Linden, ed.), vol. 2290 of *Lecture Notes in Computer Science*, pp. 198–201, Springer Berlin / Heidelberg, 2002.

[19] S. Kalaiselvi, *Financial Performance in Software Industry*. New Delhi, India: Discovery Publishing House, 2009.

[20] G. Chastek, P. Donohoe, K. C. Kang, and S. Thiel, "Product line analysis: A practical introduction," 1998.

[21] A. Helferich, G. Herzwurm, and S. Schockert, "Qfd-ppp: Product line portfolio planning using quality function deployment," in *Software Product Lines* (H. Obbink and K. Pohl, eds.), vol. 3714 of *Lecture Notes in Computer Science*, pp. 162–173, Springer Berlin / Heidelberg, 2005.

[22] L. J. M. Taborda, "Generalized release planning for product line architectures," in *Software Product Lines* (R. Nord, ed.), vol. 3154 of *Lecture Notes in Computer Science*, pp. 153–155, Springer Berlin / Heidelberg, 2004.

[23] M. I. Ullah, G. Ruhe, and V. Garousi, "Decision support for moving from a single product to a product portfolio in evolving software systems," *J. Syst. Softw.*, vol. 83, pp. 2496–2512, Dec. 2010.

[24] M. Denne and J. Cleland-Huang, "The incremental funding method: Data-driven software development," *IEEE Softw.*, vol. 21, pp. 39–47, May 2004.