RED-PL, a Method for Deriving Product Requirements from a Product Line Requirements Model

Olfa Djebbi[12], Camille Salinesi[1]

[1] CRI, Université Paris 1 – Sorbonne, 90, rue de Tolbiac, 75013 Paris, France
[2] Stago Instruments, 136 avenue Louis Roche, 92341 Gennevilliers, France
olfa.djebbi@malix.univ-paris1.fr, Camille.Salinesi@univ-paris1.fr, odjebbi@stago.fr

Abstract. Software product lines (SPL) modeling has proven to be an effective approach to reuse in software development. Several variability approaches were developed to plan requirements reuse, but only little of them actually address the issue of deriving product requirements. Indeed, while the modeling approaches sell on requirements reuse, the associated derivation techniques actually focus on deriving and reusing technical product data.
This paper presents a method that intends to support requirements derivation. Its underlying principle is to take advantage of approaches made for reuse PL requirements and to complete them by a requirements development process by reuse for single products. The proposed approach matches users' product requirements with PL requirements models and derives a collection of requirements that is (i) consistent, and (ii) optimal with respect to users' priorities and company's constraints. The proposed methodological process was validated in an industrial setting by considering the requirement engineering phase of a product line of blood analyzers.

Keywords: Requirements, Derivation, Product Line.

1 Introduction

As defined by the Software Engineering Institute (SEI), "*a software product line (SPL) is a set of software-intensive systems that share a common, managed set of features satisfying 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*".

Software Product Line Engineering is rapidly emerging as a viable and important software development paradigm allowing companies to realize order-of-magnitude improvements in time to market, cost, productivity, quality and flexibility.

These new outcomes can be attributed to *strategic software reuse*. Software product line techniques explicitly capitalize on commonality and formally manage the variations among products in the product line. As a result, the main effort to design a product from the product line is due to the variations and the impact of the choices made for the required product.

Compared with conventional techniques, companies that manage a software product line report success stories in which they decreased their time-to-market for

new products by factors of 2 to 50, reduced defect rates as high as 96% and multiplied productivity by a factor of 2 to 3 [1].
As Fig. 1 shows it, software products are developed, in the context of product line engineering, according to a two-stage process: the domain engineering stage and the application engineering stage [2]. Domain engineering involves implementing commonalities between product family members through a set of shared software artifacts, while preserving at the same time the ability to vary the products. During application engineering, individual products are derived from the product family, i.e. constructed using a subset of the shared software artifacts.

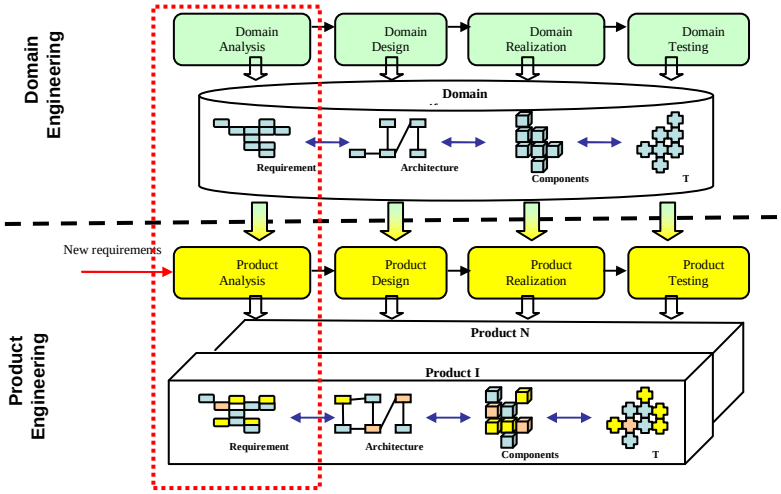


Fig. 1. Requirements Engineering challenges in a Software Product Line context (SEI).

In this particular context, Requirements Engineering (RE) processes have two goals: to define and manage requirements within the product line and to coordinate requirements for the single products. To achieve the latter goals, product requirements must be elicited by matching the product line requirements with customers' initial requirements (fig.1).

Some recommendations can be found to manage requirements in the context of SPL, but they always need to be customized [3] [4] [5] [6] [7]. Existing approaches rely on a requirements variability modeling process followed by a requirements selection process to retrieve a requirements collection specifying the single product to build.

Our experience showed us that, as stated by [8] [9], this way of working has several limits:

- *Requirements are solution-driven*: the selection among pre-defined product line requirements models that most often correspond to features already implemented in existing products, can influence stakeholders and skew their choices. They will naturally establish links between their problem and the existing solutions, adopt features with marginal value, and naturally forget about important requirements that are not present in the PL requirements model. As a result, the

focus is on model elements that implement the solution rather than on the expression of actual needs.

- *Customer dissatisfaction*: the customer requirements can be different from the ones identified in the PL requirements models. Selecting among existing requirements can lead to miss out important requirements.
- *Innovation damping*: the RE process is inherently characterized by insight-driven evolution episodes. It fosters opportunistic exploration of the conceptual space and promotes creative thinking within the system requirements. On the opposite, selecting among predefined requirements restricts considerably creativity and search for innovative ways to deal with problems, hence reducing the added value of the new products to be developed.
- *Lack of guidance*: customers and marketing people are most often on their own to elicit the requirements for new products. Existing approaches provide little guidance (notation, process, rules …) to assist them in eliciting consistent product requirements, neither are developers guided in adding new requirements to the PL requirements model.
- *Customer training*: interactions between customers and variable requirements models imply that users should make an additional effort to understand the PL models and to seek their requirements in these models.
- *Customer overwhelm*: customers should not have to consider the complete collection of PL requirements as they are only interested in the requirements for a current product. Overwhelmed by a huge amount of data, customers lose track of the initial mission and are naturally lead to inquire about, comment, and even ponder over “requirements” that do not correspond to real needs.

These limits engendered by the requirements selection processing have many impacts on the project processes and artifacts, namely:

- *Quality of the requirements documents*: when stakeholders select requirements from the PL models, the resulting documents consist in a copy of a PL requirements model extract. When, on the contrary, stakeholders come up with new requirements, specifying these independently from the PL requirements model is inefficient. We believe, there is a need for guiding the merge between variability requirements specifications with requirements documentation for single products. Furthermore, product requirements specifications can be inconsistent since PL RE methods do not propose processes to verify the consistency and the compatibility of the new requirements with the older ones.
- *Quality of the resulting product*: it is quite well documented that the outcomes of projects with poor requirements management drive to poor product quality. This applies to products developed in the context of PL as for any other kinds of products even though reuse is facilitated.
- *Project management*: training customers to understand the PL requirements models and to discuss about them is a waste of time and creates an ambiguity between the roles of analysts and customers that inevitably leads to conflicts. This, associated with poor requirements definition in early project phase, generates rework in later phases of the project, extra costs, deadlines overrun, and difficult project management.
- *Strategic objectives of the company*: stopping innovation and market anticipation with new products may harm the company strategic objectives. Besides, applied

methods leading to customer dissatisfaction may even threaten the survival of the company.

To overcome these shortcomings of existing methods, we believe there is a need for a product requirements derivation approach that satisfies the following characteristics:

- *Requirements oriented*: customers should be able to express their real needs, and the built product should answer to these needs.
- *Product line based*: the developed product should take advantage of the PL platform and reuse elaborated requirements that are already linked, traced and validated.
- *Unified into the whole product line development cycle*: it should provide means to ensure traceability with the remainder development phases for both the product line and the single product being developed.
- *Easy to apply*
- *Supported by a CASE tool that is integrated into existing toolkits*: appropriate tool support is mandatory to facilitate automate handling of the method processes and artifacts, and hence his large adoption by developers' community.
- *Scalable*: the method should allow modeling large-scale systems.

This paper presents a method that intends to support the requirements listed above. The study was undertaken with the collaboration of the AFIS[1] association and the method was developed by application to a product line of a French company named Stago -a medical company that produces blood analyzers [10]. The experience consisted in gradually introducing basic PL management principles while meeting practical issues in the RE phases of a new product creation project. The selection of these basic principles resulted from extensive bibliography research. Based on this experience, we developed a method, named RED-PL (Requirements Elicitation & Derivation for Product Lines) that guides the elicitation of product requirements by derivation from the PL requirements specification. The approach takes into account both the company's environmental and technical constraints and the specific product requirements as expressed by customers.

RED-PL is based on already existing PL requirements notations. The originality of the RED-PL method is that (i) it is user-oriented and (ii) it guides product requirements elicitation as a decision making activity. Indeed, RED-PL makes it possible to users to express their needs using classic requirements engineering techniques. Then, mechanisms are used to convert these needs and match them with the PL requirements specification. Negotiation and arbitration are finally supported in RED-PL to elicit optimal product requirements while maximizing reuse.

The paper is organized as follows. Section 2 outlines the challenges faced by Stago and the problems encountered while performing RE activities within its SPL management context. Section 3 presents the RED-PL method which was developed to meet these challenges. The methodological process is illustrated using the Stago data that were initially used to develop it. Section 4 provides an overview of existing

[1] French Association on Systems Engineering, affiliated to INCOSE (International Council on Systems Engineering) http://www.afis.fr/

methods and discusses how they deal with these challenges. Finally, conclusions are given in section 5.

2 Problem statement in Stago's context

Stago Instruments [10] is a company that produces analytical instruments for the haemostasis diagnosis. These instruments are embedded and real-time systems. They are used in hospitals and laboratories in the context of routine analysis or biologic researches.

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

In general, instruments make tests on patients' products (total blood, plasma) and return results that are then interpreted by doctors.

In order to make tests, biologists load tubes of patients' products as well as reagent tubes in the instrument. While loading, tubes have to be identified. The biologist must then choose an analysis methodology and launch the tests. A methodology is a series of steps that simulate corpus reactions. Methodologies differ following test types (TP, TCA, etc.), but comprise necessarily a mix step and an incubation step. They may also use mixing and heating steps. Researchers can compose their own methodologies.

The instruments treat tubes, accomplish analyses according to specified methodologies, make measurements, and return the results to the biologist.

Products are loaded by batch. Nevertheless, the instrument is able to interrupt current tests in order to load and treat urgent tubes. Before launching tests, tubes must be treated to separate their constituents. Two processes of separation exist: centrifugation and micro-filtration. All instruments are able to implement theses processes however only one of them is implemented at a time in a given instrument.

There are three kinds of measurements: chronometric, colorimetric and immunologic. Instruments can implement several measure techniques, but an instrument that implements micro-filtration should not implement the photometric measure.

Test results are provided to the biologists in gross unit (Sec, D.O/min, Δ D.O), as well as in calculated unit (INR, µg/ml, UI/ml). To establish correspondences between units, the instrument must support calibrations. Besides, the instrument can view results on the screen, print them, and/or transfer them to the hospital or laboratory' host and put them into the patient case historic.

During projects, Stago teams manage in parallel the requirements documentation for the product line (common requirements) and for the single products (variable requirements).

Fig. 2 presents a model that was developed to document the most important requirements of the Stago instruments product line. The PL requirements are modeled using a Feature-oriented notation.

The figure shows a tree in which nodes are the features that correspond to PL requirements and links describe feature decomposition. There are three types of requirements: mandatory (e.g 'Load products'), optional (e.g. 'Separate constituents') and alternative (e.g. 'Centrifuge' and 'Micro-filter'). A mandatory requirement is common to the PL and must be included in every product of the PL. An optional requirement may, or not, be chosen for the considered product. Alternative requirements are collections of requirements from which some can be selected and others not. A UML-cardinality is associated to the collection to indicate the minimum and maximum number of requirements to be chosen. Additional dependency links between requirements, namely the 'requires' and the 'mutex' relationships, can be defined to specify additional constraints in requirements selection.

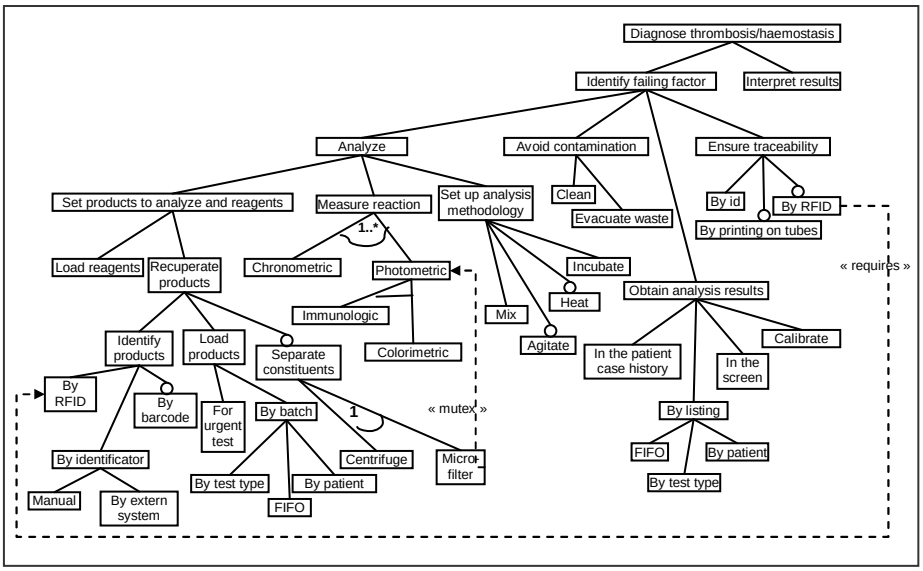


Fig. 2. Requirements model of Stago's product line

Since users are free in their way to express requirements, it happens that some requirements already exist in the PL requirements documentation, but with a different form. Users also insist on some requirements and ignore their impacts on other ones, or on the project progress itself. Users also often forget about important requirements and ignore opportunities offered by the product line.

In this context, Stago raised priority questions namely: (i) how to ensure the satisfaction of the real user's needs? and (ii) how to derive an optimal and consistent collection of product requirements that meet users needs and that cost little to the company? The RED-PL approach was developed and tried out on a Stago project to answer these questions.

2 The RED-PL approach

In contrast to the traditional 'Selection' approach, requirements derivation for PLs must take into account stakeholders' original needs. As depicted in Fig. 3, RED-PL consists of:

- eliciting user requirements,
- matching users' requirements with PL requirements. This activity leads to establish the set of requirements that the PL subsumes and that satisfy users' needs. They correspond to a set of possible products to build.
- deriving the optimal set of product requirements, taking into account users' and company's constraints.

These processes are respectively described in the three following sub-sections.

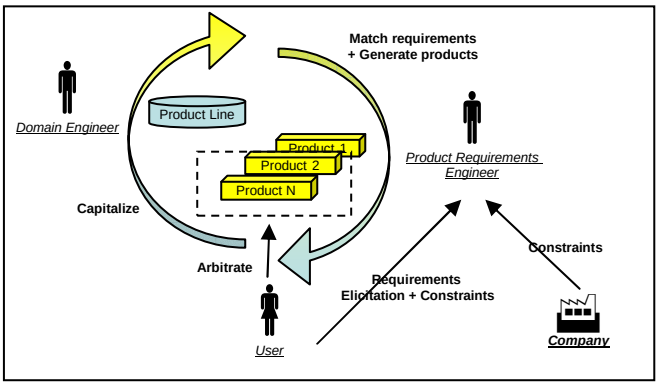


Fig. 3. Processes of the RED-PL approach

3.1 The matching process

The matching process is an iterative process that consists in interpreting users' requirements in terms of the PL requirements. It results in a collection of requirements that shall be implemented in the product (named 'product requirements'). The matching process aims at: (i) eliciting new users' requirements, (ii) avoid missing possible requirements, (iii) refining progressively the final product requirements, and (iv) updating the PL assets.

In the matching process, users' needs can be elicited using classical methods. Then, rules must be applied to construct a valid (i.e. unambiguous, consistent, traceable and verifiable) collection of product requirements. Once this is achieved, users' requirements can be fetched and marked in the PL model.

If users' requirements can not be found in the PL requirements model, then either (i) they are new requirements and they should be added to the PL model as well as links among them and in relation with old requirements, or (ii) they are the same requirements expressed differently, and then consensus should be made on the requirement formulation.

Requirements' matching is guided by using similarity analysis techniques. Two kinds of similarity analysis techniques can be used: surface level and deep level. First techniques are based on lexical similarity where two requirements are considered similar when they use the same term or the same linguistic structures. Conversely, deep level techniques use a structural and a semantic proximity. These techniques need more sophisticated tools such as dictionaries and linguistic parsers. Our similarity analysis approach also uses refinement, as suggested by goal modeling, to progressively improve the quality of the matching and to focus on requirements that are considered more important [11].

Our approach exploits the 30 generic similarity metrics adapted to Dice, Jaccard and Cosine's ratios. As shown below, similarity can be automatically computed by applying a weighted ratio between a number of similarities found between two requirements and the number of elements that define these requirements.

$$S_D^m(A,B) = \frac{\sum_A \underset{B}{MAX}\left[SIM(Termes_A, Termes_B)\right] + \sum_B \underset{A}{MAX}\left[SIM(Termes_A, Termes_B)\right]}{\left|\{Termes_A\}\right| + \left|\{Termes_B\}\right|}$$

(Formula 1) Adapted Dice ratio

After similarity study, marked requirements and all the associated requirements can then be retrieved from the PL model. This collection of requirements should correspond to a fragment of the PL requirements model, i.e. a sub-tree of requirements that satisfy users' requirements. However, the PL requirements model also contains requirements that are not yet marked. These requirements may be either (i) undesired, they must then be explicitly marked as such, (ii) mandatory then they must be considered in the collection of product requirements, or (iii) variable (optional/ alternative). As long as the tree contains unmarked optional and alternative requirements, a decision must be made on which additional PL requirements to select for the product. Arbitrations must therefore be investigated and discussed with users, as explained in the next sub-section.

3.2 The arbitration process

The output of the matching process consists in a PL requirements model composed of wanted/unwanted mandatory, optional and alternative requirements. The model fragment composed of desired requirements represents a set of possible releases as it can also contain optional and alternative requirements.

Only wanted optional and alternative requirements are considered in the following to express preferences since mandatory requirements must anyway be included in the collection of product requirements.

Preferences can be expressed by users under the form of weights associated to optional and alternative requirements. A 0 weight means that the requirements should not be selected, a 1 weight means that it should be included in the product requirements collection. The sum of weights of a bunch of alternative requirements must be equal to 1. Implicitly, each mandatory requirement has a 1 value weight.

Users can indicate their constraints on each requirement in terms of costs and benefits. Likewise, managers can state their development constraints on each requirement in terms of human resources, revenues, costs, and

implementation/integration time. Although we knew they are important, other constraints such as skills of development teams, team transfers, deadline extension, external resources, were voluntarily ignored because they were too difficult to evaluate and we didn't know if they would really influence arbitration significantly.

Once requirements, priorities and constraints are completely defined, they are formalized using an Integer Linear Programming (ILP) notation. The Akkar approach [12] was selected and adapted to solve the problem at hand. The adapted version allows to define the subset of requirements that composes the optimal release while doing a what-if analysis on a dashboard. The ILP approach generates a collection of requirements that satisfies the constraints values, and is optimal with respect to the optimization criterion.

The following presents our proposal for modeling PL requirements dependencies using Akkar's approach. In Akkar's approach, a requirement $x_a \in \{0,1\}$ with $x_a=1$ if x_a is selected, and $x_a=0$ otherwise. Five kinds of dependencies can be considered: composition, requires, optional composition, exclusion, and alternative. While the four former dependencies were already considered in Akkar's approach under the names 'combination', 'implication' and 'exclusion', the fifth kind had to be created to deal with the specific semantics of PL requirements modeling notations.

*Requires***.** If requirement x_b is selected, then requirement x_a must be selected too. In the ILP model, it must be ensured that: $x_b=1 \Rightarrow x_a=1$

The ILP model is extended by the linear inequality $x_b \leq x_a$ (x_a cannot be implemented without implementing x_b). In Akkar's initial approach, the corresponding dependency was 'implication'. In terms of PL requirements modeling, "requires" dependencies can be found from the alternative, the optional and the requires relationships.

$$x_b \leq x_a \quad (1)$$

*Composition***.** If two requirements x_a and x_b cannot be implemented separately, then it must be ensured that $x_a = x_b$. Composition dependencies can be found in the PL requirements models from composition relationships. In Akkar's terms, it corresponds to the combination dependency.

$$x_a = x_b \quad (2)$$

*Exclusion***.** If R_a and R_b cannot both be selected, in the ILP model then the inequality: $x_a + x_b \leq 1$ must be verified. In the PL modeling, exclusion dependencies can be found from "mutex" relationships.

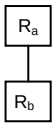
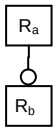
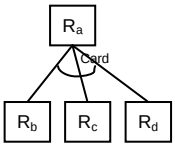
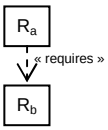
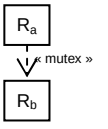
$$x_a + x_b \leq 1 \quad (3)$$

*Alternative***.** In PL engineering, a requirement can be realized by one or more requirements among a set. It is partly ensured by the implication relationship from a requirement x_a to its sub-requirements x_b.. x_k, but needs to be more detailed to model the relationship between sub-requirements themselves. So, the *alternative* dependency (which does not exist in Akkar's model) is defined in ILP model by the following inequality:

$$x_a * Card_{min} \leq x_b + .. + x_k \leq Card_{max} \quad (4)$$

The following table summarizes the mathematical formulae used to develop the ILP model.

Table 1. Recapitulation of requirements dependencies and their representation in the ILP

Dependency relationship	**Explication**	**Mathematical formula**
 (composition)	If a requirement is selected then all mandatory requirements composing it must be selected $R_a = 1 \Rightarrow R_b = 1$ | $R_a = 0 \Rightarrow R_b = 0$ $R_b = 1 \Rightarrow R_a = 1$ | $R_b = 0 \Rightarrow R_a = 0$	$R_a = R_b$ (Combination)
 (option)	If a requirement is selected then its optional sub-requirements may be selected $R_a = 1 \Rightarrow R_b \in \{0,1\}$ | $R_b = 1 \Rightarrow R_a = 1$ $R_a = 0 \Rightarrow R_b = 0$ | $R_b = 0 \Rightarrow R_a \in \{0,1\}$	$R_b \leq R_a$ (implication)
 (alternative)	If a requirement is selected then alternative sub-requirements must be selected respecting the specified cardinality $R_a = 1 \Rightarrow \begin{cases} R_{b..d} \in \{0,1\} \text{ and} \\ R_b + R_c + R_d \leq Card_{max} \text{ and} \\ R_b + R_c + R_d \geq Card_{min} \end{cases}$ $R_a = 0 \Rightarrow R_{b..d} = 0$ $R_{b..d} = 1 \Rightarrow R_a = 1$ $R_{b..d} = 0 \Rightarrow R_a \in \{0,1\}$	$R_{b..d} \leq R_a$ (implication) $R_a * Card_{min} \leq R_b + .. + R_d \leq Card_{max}$ (alternative)
 (requires)	If a requirement is selected then all required requirements must be selected $R_a = 1 \Rightarrow R_b = 1$ | $R_b = 1 \Rightarrow R_a \in \{0,1\}$ $R_a = 0 \Rightarrow R_b \in \{0,1\}$ | $R_b = 0 \Rightarrow R_a = 0$	$R_a \leq R_b$ (implication)
 (mutex)	If a requirement is selected then all requirements that are mutually exclusive with it must not be selected $R_a = 1 \Rightarrow R_b = 0$ | $R_b = 1 \Rightarrow R_a = 0$ $R_a = 0 \Rightarrow R_b \in \{0,1\}$ | $R_b = 0 \Rightarrow R_a \in \{0,1\}$	$R_a + R_b \leq 1$ (exclusion)

The ILP modeling approach presented in the former section was tested in a Stago project with satisfying results. The experience is reported in the next section.

3.3 The case study

Once user requirements elicited, they were matched with PL requirements as recommended in the RED-PL matching process. The resulting requirements collection is a subset of the PL requirements model. The matching process revealed that users were decided neither on the measurement technique nor on whether the instrument to build should enable indoor constituents separation. Decisions had to be made to generate the optimal collection of requirements for a complete product. The arbitration process presented in section 3.2 was thus used to solve this problem.

First, the PL requirements model was analyzed and a ILP model was developed as defined in section 3.1. All the constraints were recorded in a Microsoft Excel spreadsheet, and analyzed with the Microsoft Excel solver (Fig. 4).

Two criteria were used to guide arbitration, namely cost and revenue. Revenue was evaluated by enquiring salespeople about the perceived value of the functionalities implementing the requirements. Cost evaluations were made by the engineering team who was asked to consider development and integration costs, need for resources (material and human), management costs, test costs, maintenance cost, and installation costs. These evaluations are an ordinary activity of salespeople and engineers, e.g in the context of risk analysis while elaborating the feasibility of the project. Several methods can be used to do this. Our approach does not focus on a particular one as it considers these evaluations as an input.

For confidentiality reasons, revenue and cost are defined in the next figure as relative values rather than under the form of the absolute values that were actually defined.

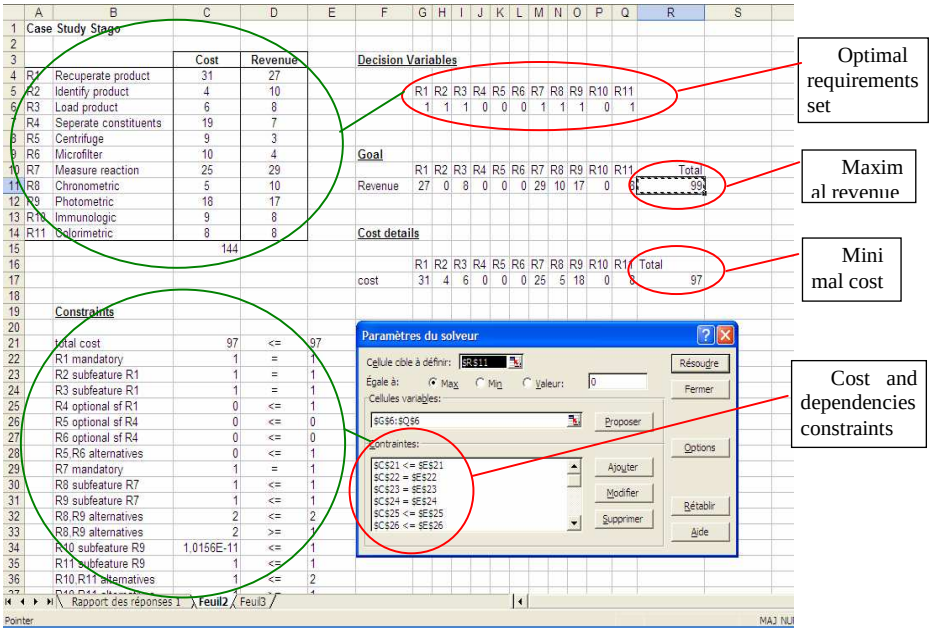


Fig. 4. Screenshot of Stago ILP problem after solving

Two goals were considered for optimization: either minimize cost while considering minimal revenue, or maximize revenue taking into account a global cost limitation. Sales and engineer teams agreed to focus on the second goal which is closer to their daily concerns. The collection of requirements generated by the solver using these parameters was found realistic in the sense that the resulting products did correspond to products already developed at Stago. Besides, the product did respond to the users' expressed at the beginning of the project and did correspond to products already identified as being of low cost. It was however difficult to assess if the generated product did really correspond to an optimal product or not.
Some difficulties were observed too while applying the method. First, the matching process was difficult to handle due to a lack of precision in the formulation of users' requirements. The difficulty was due not only to terminology, but also to a conceptual mismatch between users' requirements and the PL requirements (different levels of abstractions, different views). Besides, the ILP technique seemed to be not scalable to large systems, and is limited to optimization requests. We believe that this approach can be replaced by more adequate, flexible and scalable technique such as Constraint Programming. Further applications to other industrial projects are planed to enhance the method and favor its repeatability.

4 Related works

Many different methods interested in constructing SPL assets are available in literature [13] [14] [15] [16] [17] [18]. Product derivation methodologies are on the contrary rather scarce [4] [19] [20]. Besides, while derivation affects the whole product line artifacts, from requirements to code, the derivation issues are mainly addressed in terms of design and implementation [4] [6].

At the requirement engineering level, how to create the right requirements assets of the PL and dependencies among them to develop the right products have been extensively studied [7] [21] [22] [23] [24] [25], but understanding the derivation process itself has received little attention.

In existing approaches, the derivation of the product architecture, code or test artifacts from the product and the PL specifications is performed using the following techniques:

- *Model transformation*: static and dynamic models are instantiated for products from the PL models, using a model transformation language [4] [26] [27] [28].
- *Design patterns:* for instance the method introduced by Jezequel which consists in using the 'Abstract Factory' pattern as interface to create objects of each product in the product line [29] [30].
- *Variability bounding*: generative approaches (e.g. Generative Programming approach [19]) suggest automatic derivation by code generation. Selecting desired product features is sufficient to allow assembling correspondent SPL elementary reusable components and generate the application code. Other approaches introduce aspect programming techniques to assemble components by waving features [31] [32].

Mostly, derivation methods consider as input a collection of PL requirements selected from the SPL requirements model. However, industry experience suggests that simply having the right assets is not sufficient to facilitate its selection and assembly. So some works tried to propose guidelines to select the appropriate set of assets, but they are still reduced to technical levels.

Namely, the specific assets needed could be specified in a production plan which describes how the core assets are used to develop products [33]. Hunt considers software components and studies the optimal organization proceedings to facilitate finding and selecting them [34]. [35] discusses automating component selection using artificial intelligence techniques. [3] [5] provide a framework of terminology and concepts regarding product derivation as well as a generic software derivation process. It is organized on iterative phases in order to determine the final configuration of the derived product. Once again, the derivation process has by default as input a subset of requirements that originate from customers, legislation, hardware and product family organization. Details about how these requirements are aggregated are not given. [4] also establishes a derivation framework. It indicates that the product requirements derivation is made through a decision process. But, it does not include more details about this process.

Nevertheless, a necessary step in product derivation is to determine the set of requirements to use in order to build the particular product out of the possible products in the product line. This requires some description of the customer needs that allows it to be distinguished from others in the SPL. This description provides a set of product requirements. Someone must then find and select the assets that are needed to meet the product requirements. As presented in the existing approaches, it is often the product developer that makes these decisions as the product is assembled. The role of the user is dumped and mistreated.

While the focus provided by scoping develops mechanisms handling technical derivation, we are interested in instructing requirements derivation processes that originate from users needs, and involve users choices while tacking decisions; which is not typically available in general derivation approaches.

4 Conclusions and future work

A major addition to existing reuse approaches since the 1990s are software product lines that have been the long standing notion to solve the cost, quality and time-to-market issues associated with development of related software applications.

Over the past few years, domain engineering has received substantial attention from the software engineering community. Most of the researches, however, fail to provide detailed derivation processes namely for deriving requirements, which has been restricted to the selection of a requirements subset.

The idea behind the proposed approach is that the user, the main stakeholder to whom the final product is intended, should be involved in specifying product requirements, in a way that efforts expended in constructing the reusable requirements in domain engineering are outweighed by the benefits in deriving the right individual products that satisfy their mission.

RED-PL includes two processes that are the matching and the arbitration processes. The first establishes the set of possible requirements that meet users' needs. The latter, arbitrate on these requirements in order to derive a consistent requirements set that is optimal for a defined set of users and company constraints (e.g. revenue, cost, resources, time, etc.).

We have thought these processes (namely the mathematical model) based on feature models. But, it is obvious that it may be applied for the different PL modeling languages (Use cases, goals, UML, aspects). That is because these types of dependencies represent fundamental concepts that are implemented by all existing variability languages. Only visual representation is different depending on the language constructs (use cases, classes, etc.) and stereotypes. Besides, the approach viability was tested on real projects developing blood analyzers within a French company named Stago. Obtained results were verified and appreciated.

Further research will focus on the refinement of the approach processes. We aim at defining matching and arbitration processes of variable requirements in correlation with variable PL physical architecture. It is worthwhile in Stago context since it produces instruments where technical requirements impact heavily the decision on functional requirements depending on technology costs and revenues.

We intend next to implement a tool support that interfaces with existing modeling tools and enables such a matching and arbitration processes.

Moreover, the repeatability of the approach will be studied. The purpose is to define a systematic process allowing modeling PLs and deriving products suitable to different companies' contexts. We are confident that if the Integer Linear Programming is not scalable to large systems, it can be replaced by another more adequate Multi Criteria Decision Making method.

References

1. http://www.sei.cmu.edu/productlines/plp_hof.html. SEI Product Line Hall of Fame web page
2. Linden F.: Software Product Families in Europe: The Esaps & Café Projects. (2002)
3. Deelstra S., Sinnema M., Bosch J.: Product derivation in software product families: a case study. The Journal of Systems and Software (2004) 183–204
4. Haugen Ø., Møller-Pedersen B., Oldevik J., Solberg A.: An MDA®-based framework for model-driven product derivation. Software Engineering and Applications, USA (2004)
5. Sinnema M., Deelstra S., Nijhuis J., Bosch J.: COVAMOF: A Framework for Modeling Variability in Software Product Families. The 3rd Software Product Line Conference (2004)
6. Lee J., Kang K. C.: A Feature-Oriented Approach to Developing Dynamically Reconfigurable Products in Product Line Engineering. SPLC. (2006)
7. Halmans G., Pohl K.: Communicating the variability of a software-product family to customers. Proceedings of the Software and Systems Modeling, volume 2, Springer (2003)
8. Maiden N., Gizikis A., Robertson S.: Provoking Creativity: Imagine What Your Requirements Could Be Like. IEEE Software, Vol. 22, No. 5 (2004) 68-75
9. Michael G., Kang K. C.: Issues in Requirements Elicitation. Technical Report. (1992)
10. www.stago.fr. Diagnostica Stago Web page
11. Salinesi C., Etien A., Zoukar I.: A Systematic Approach to Express IS Evolution Requirements Using Gap Modelling and Similarity Modelling Techniques. CAiSE Conference, Springer Verlag, Riga, Latvia (2004)

12. van den Akker M., Brinkkemper S., Diepen G., Versendaal J.: Flexible Release Planning Using Integer Linear Programming. Proceedings of REFSQ (2005) 257-272
13. Gomaa H.: Designing Software Product Lines with UML: From Use Cases to Pattern-based Software Architectures. Addison Wesley Object Technology Series (2004)
14. Bayer J., Flege O., Knauber P., Laqua R., Muthig D., Schmid K., Widen T., DeBaud J.-M.: Pulse: a methodology to develop software product lines. In Proceedings of the SSR (1999)
15. Clements P., Northrop L. M.: Software Product Lines: Practices and Patterns. Addison Wesley Professional (2001)
16. Bosch J., Florijn G., Greefhorst D., Kuusela J., Obbink H., Pohl K.: Variability Issues in Software Product Lines. The International Workshop on Product Family Engineering (2001)
17. Dobrica L., Niemelä E.: UML Notation Extensions for Product Line Architectures Modeling. Australasian Workshop on Software and System Architectures, Australia (2004)
18. Robak S., Franczyk B., Politowicz K.: Extending the UML for modelling variability for system families. International Conference on Algorithmic Mathematics and Computer Science (2002) 295–308
19. Czarnecki K., Eisenecker U.W.: Generative Programming: Methods, Tools, and Applications. Addison Wesley (2000)
20. Sinnema M., Deelstra S., Hoekstra P.: The COVAMOF Derivation Process. Proceedings of the 9th International Conference on Software Reuse (2006)
21. Thompson J., Heimdahl M.: Structuring Product Family Requirements for n-Dimensional and Hierarchical Product Lines. Requirements Engineering Journal, vol-8, Issue 1 (2002)
22. Streitferdt D.: Family-Oriented Requirements Engineering. PhD Thesis, Technical University Ilmenau (2003)
23. Kang K., Lee K., Lee J.: Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. Proceedings of the 7th International Conference on Software Reuse: Methods, Techniques, and Tools (2002) 62 - 77
24. Gibson J. P.: Feature Requirements Models: Understanding Interactions. In Feature Interactions, in Telecommunications IV, Montreal, Canada, IOS Press (1997)
25. Buhne S., Lauenroth K., Pohl K.: Modelling requirements variability across product lines. In 14th IEEE International Conference on Requirements Engineering (2005)
26. Perez Garcia J., A. Laguna M., Gonzalez-Carvajal Y. C., Gonzalez-Baixauli B.: Requirements variability support through MDD and graph transformation. International Workshop on Graph and Model Transformation, Tallinn, Estonia (2006) 171-183
27. Ziadi T. : Manipulation de Lignes de Produits en UML. PhD thesis, Université de Rennes 1, équipe IRISA-TRISKELL, directeur Jean-Marc Jézéquel (2004)
28. Ziadi T., Hélouët L., Jézéquel J-M.: Towards a uml profile for software product Lines. In the Fifth Internationl Workshop on Product Familly Engineering, Springer Verlag (2003)
29. Jézéquel J-M.: Reifying configuration management for object-oriented software. Proceedings of the 21th international conference on Software engineering (1998) 250–259
30. Jézéquel J-M.: Reifying variants in configuration management. ACM Transaction on Software Engineering and Methodology (1999) 294–305
31. Jansen A., Smedinga R., van Gurp J., Bosch J.: First class feature abstractions for product derivation. Special issue on Early Aspects: Aspect-oriented Requirements Engineering and Architecture Design, IEE Proceedings Software (2004) 197-207
32. Mezini M., Ostermann K.: Variability Management with Feature Oriented Programming and Aspects. Foundations of Software Engineering, ACM SIGSOFT (2004)
33. Chastek G., McGregor J. D.: Guidelines for developing a product line production plan. Software Engineering Institute, Technical Report CMU/SEI-2102-TR-006 (2002)
34. Hunt J.M.: Organizing the asset base for product derivation. In 10th SPLC (2006)
35. Asikainen T., Mnnist T., Soininen T.: Using a configurator for modelling and configuring software product lines based on feature models. Software Variability Management for Product Derivation - Towards Tool Support at International Workshop of SPLC (2004)