Graph Modelling of a Refactoring Process for Product Line Architecture Design

Francisca Losavio, Oscar Ordaz
MoST, Escuela de Computación, Facultad de Ciencias,
Universidad Central de Venezuela
Caracas, Venezuela
{francisca.losavio, oscar.ordaz}@ciens.ucv.ve

Nicole Levy, Anthony Baïotto
CEDRIC, CNAM, Paris, France
nicole.levy@cnam.fr
anthony.baiotto@thalesgroup.com

Abstract—*Product Line Architecture (PLA)* is the main tangible element shared by all products of a *Software Product Line (SPL)*; it covers common functionality and the required variability of *SPL* products. Responding to industrial practice, this paper proposes a *reactive refactoring bottom-up process* to build a *PLA* from existing similar software product architectures of a domain, expressed by UML logical views. An *architecture* is represented by a connected graph or *valid architectural configuration (P, R)*, where *P* and *R* represent components and connectors of the product. This process constructs a graph *(RG)* for each product, organized by levels, containing intermediate valid configurations or connected induced sub-graphs of *(P, R)*. A candidate *PLA* is automatically constructed followed by an optimization process to obtain the PLA using the domain quality model. The refactoring process is applied to a case study in the robotics industry domain. Automatic parts of the process are tool supported.

Keywords — Software product line, product line architecture, refactoring graph, automatic detection of variation points, quality model

I. INTRODUCTION

A *Software Product Line (SPL)* is a set of software-intensive systems, sharing a common, managed set of features that satisfy the specific needs of a particular market segment and which are developed from a common set of core assets. These assets are reused in different products that form a family [1]. The key issue in *SPL* development is the construction of a common architecture from which new products can be derived. The *SPL* approach favours reusability and claims to decrease costs and time-to-market. *Software Architecture* is defined in [2] as "a collection of computational components - or simple components - together with a description of the interactions between these components, the connectors". *Product Line Architecture (PLA)* is defined by [3], [4] as a core architecture that captures the high level design for all the products of the *SPL* family, including variation points and variants documented in the variability model. Two main axes for *SPL* engineering are followed: the *proactive top-down design* which considers constructing *PLA* from domain knowledge and the *reactive or extractive bottom-up design* which develops the *PLA* from systematic refactoring of existing products [5]. In this paper the reactive design is followed because in practice, many industrial organizations do not have a *PLA*; they only dispose of products constructed over time and by different developer teams. A generic architecture according to the business domain is required to reduce costs, time-to-market, and increase products evolution. In this case, several similar products must be examined, using reverse engineering techniques to identify commonalities and variation points. The *PLA* design is a complex process that is in general poorly described in literature and left to incomplete case studies [6]; the details of methods and approaches are difficult to follow due to the lack of standards. Moreover, existing traditional architectural methods and evaluation techniques for single-systems are reengineered and not specifically designed for *SPL* [7], [8]. In the proactive approach, reengineering is used mostly to maintain and improve the *PLA*, and in the reactive approach it is required to recover architectural knowledge from existing products. Reengineering techniques are *Reverse Engineering* (the examination) that helps clarifying the structure by extracting information providing also high-level views on the subject system, and *Refactoring* (the alteration) that modifies software or software artefacts to improve some of its quality properties, such as reliability or maintainability.

An important question that commonly appears in the literature is what needs to be done to ensure a suitable choice of architecture for the family of products of the *SPL* [9]. This paper contributes to provide an answer to this question, proposing a semiautomatic *refactoring process* to build the *PLA* as a main goal.

The basic input required by this approach are *architecture descriptions* of similar existing products that will conform to the *SPL* family. In practice, a documented description of the architecture is often missing, and these descriptions are obtained by reverse engineering of the products, showing a logical view of the architecture. However, since the products have been already constructed considering the experience of a software architect and his knowledge of the system domain, the architecturally relevant non-functional information related to the architectural style, is already implicitly contained in the architecture of the existing product.

On the other hand, in the reactive approach, *domain architectural quality requirements* are used mostly to reengineer an existing *PLA*. In our case, they will be used as input to optimize the candidate *PLA*, obtained by the functionality-based refactoring; the ISO/IEC 25010[1] standard quality model will be used to specify these architectural quality requirements. The proposed refactoring process is a straightforward process that is easy to use and apply to a case study, since the construction of an initial or candidate *PLA,* with commonality and variation points, is tool-supported, and also from an academic point-of-view, it can be used didactically to show a complete construction process of a *PLA*. A prototype tool implementing the process algorithms supports the automatic construction of the candidate PLA. However as it is just a prototype, it will not be described here; we have only shown the graphics produced by the tool. This reactive bottom-up process is based on the identification of different connected architectural intermediate configurations, starting from existing similar product architectures that will be part of the *SPL* family. The inspiration for this paper came from [10] with a case study in the robotics industry domain, where a *PLA* is manually developed. The model supporting our process is a graph or *Refactoring Graph*, denoted by *RG*. The bottom-up process to construct *RG*, whose activity diagram is shown in Figure 2, follows a strategy based on situation planning proposed in Carlos Matus [13]. Algorithms for all process steps are provided and a prototype tool supports the automatic steps.

In our approach, the initial or candidate *PLA* obtained automatically contains the most relevant functionalities of the domain, according to [15]; it must undergo a manual optimization process to construct the final *PLA,* which must respond also to the domain architectural requirements. In general, reengineering techniques and semi-automatic processes are used for the evolution or construction of *PLA*, however complete automation has not yet been achieved, because the expertise of the software architect is still considered in practice the key issue to select convenient architectural solutions [9]. In consequence, our *PLA* optimization step, which involves a refactoring of the candidate *PLA* to take into account architectural quality requirements, is still performed manually. In this context, the ISO/IEC 25010 standard quality model is a software asset capturing the domain knowledge on the quality requirements of the family of products [25]. It has been selected among other product quality standards, because it is the updated version of the ISO/IEC 9126-1, well known by the software community. It is used as a main standard quality-based scenario [30] to specify these quality requirements, unifying also software quality terminology. Architectural evaluation techniques in general offer limited and non-justified quality scenarios [9], [29]. The standard quality model provides a complete view of the domain quality requirements offering eight high-level quality characteristics, allowing their refinement until the quality attributes or measurable elements and their metrics are attained, to facilitate and document the justification of the selection of the convenient architectural solution. However, the ISO/IEC 25010 quality model is a framework that must be customized or adapted for a particular domain using the experience of a quality engineer or architect.

This paper is structured as follows, besides this introduction: the second section proposes the graph model supporting the refactoring process and describes the case study of the robotics industry domain. The third section contains the detailed steps of the complete refactoring process activities, and each step is illustrated by the application to the case study. The fourth section discusses the related works. Finally, conclusions and perspectives are presented.

II. Construction and Properties of *RG* – Description of the Case Study

Definitions, terminology and properties of the *RG* associated to each product are presented in what follows; the case study is described at the end of this section.

A. *Definitions, terminology and properties of RG*

Architectural configuration or *configuration* is a graph *(P, R)*, where *P* is a set of *components* or software assets and *R* is the set of relations (*connectors*) between each pair of components. Notice that the *P* quality requirements are implicitly considered in these relations, because the architect took them into account when he constructed the product architecture. The *cardinality of P* is defined as the *order* of the configuration; to avoid trivial cases, $order \geq 2$. Note that assets, which are configurations of order 1, constitute a trivial case.

Given a configuration *(P, R)* and *Q* a subset of *P*, the *sub-graph induced by Q*, i.e., *(Q, R')*, where *aR'b iff aRb*, is defined as an *intermediate configuration* of *(P, R)*. $Q \subseteq P$ will denote the fact that *(Q, R')* is an intermediate configuration of *(P, R)*. Intermediate configurations that will be considered must to be *valid configurations*, i.e. representing an architecture that could make sense. We will consider that a configuration is *valid* when it does not contain isolated components or groups of components, i.e., it is a connected graph of components and connectors modelling the structure of the architecture [14]. A *product* is a software system that is represented by a connected or valid configuration. Let us note that a more restrictive definition of validity, considering some additional constraints, could be considered later on during the process.

Given a product *(P, R)*, its *refactoring graph RG* is defined as follows: the *nodes* are intermediate valid configurations of *(P, R),* distributed in levels. There are as many levels as components in the product. Intermediate valid configurations constructed with *i* components are placed in level *i*, L_i. The bottom-up process to construct *RG*

[1] *ISO/IEC 25010. Systems and software engineering - (SQuaRE) -- System and software quality models," ISO/IEC JTC1/SC7/WG6, 2011.*

starts from the last level with only one node, corresponding to the complete product configuration. Each node or intermediate valid configuration of L_i, $i \geq 2$, originates as many nodes in L_{i-1}, as valid configurations exist from i possible combinations. There exists an *arc* between two nodes in consecutive levels, if one of them is obtained from the other by adding a new component. That is to say, all precedent nodes of L_i are placed in L_{i-1}.

Given two nodes c_{i-1}, c_i, belonging to levels L_{i-1} and L_i respectively; the arc represents the *transformation* to obtain c_i from c_{i-1}, by adding a new component to c_{i-1}. A triplet indicating the *starting node*, the *added component* and its associated *weight*, labels each arc. The weight associated to each component of the product is used to select main functionalities or archetypes. According to the J. Bosch's architectural design method [15], archetypes constitute an initial architecture based on functionality. The scale of weights ranking the relevance of the component with respect to the architecture is set to: high = 3, medium = 2, and low = 1 (see Table I).

A *path* between two nodes c_i in L_i and c_j in L_j with $i<j$, is a sequence of transformations and intermediate configurations to obtain c_j from c_i, and can be considered a *construction strategy*, as it describes how an architecture is constructed from another by composition of transformations. Another way to denote a path between two nodes is to give the sequence of nodes forming the path; in this case the transformation is implicitly defined. Notice that there is a path between two nodes Q and Q' iff $Q \subseteq Q'$; therefore, the configuration node representing the product can be obtained from any node following a path between them. The *weight of the construction strategy* can be defined as a function of the weights of the arcs forming the path.

The following properties of *RG* hold because *(P, R)* is a connected graph:

- All levels in *RG* are non empty sets.
- Levels L_i, $i \geq 2$ are sets constituted by intermediate valid configurations of *(P, R)* with *i* components, i.e., they are induced connected sub-graphs of product *(P, R)*.
- Each level L_i is constituted by all intermediate configurations of *(P, R)* of order *i*, $i \geq 2$.
- Each *RG* describes the different ways to build a product *P* from given software assets, by adding components one by one, such that each addition produces a valid configuration of immediate increasing order.

Given *RGs* associated to products P_k, a *variation point VP*, is defined as equal configuration nodes on the same level L_q, $q \geq 2$, in each *RG* of P_k, with the maximum number of components. Notice that if more than one *VP* is present, by definition, they are on the same level L_q.

B. Case Study in the Robotics Industry Domain

An interesting work of Koziolek, Weiss and Doppelhamer [10] describes a reengineering approach to build the *PLA* to develop PC software for a robotics industry, based on the refactoring of three existing products. However, it lacks the description of a systematic process, being limited to expertise considerations; the *PLA* that is produced satisfies established architectural quality goals for the domain. Considering this case study [10], we propose in this work a systematic and repeatable derivation process, based on a refactoring approach and the use of the graph model described in section II.A, to construct a *PLA*. The bottom-up derivation of the *PLA* from the individual products is performed analysing automatically the *RG* associated to each product, and using a heuristic proposed by I. Bosch [15], on the choice of the main common functionality. Figure 1 shows three similar products of the robotics industry domain specified in UML [11], corresponding to the case study described in section II.B, adapted from [10]. Notice that UML 2.0 is used here as an ADL (Architecture Description Language); however, only static design aspects, a logical view of the architecture, are discussed in this work [12].

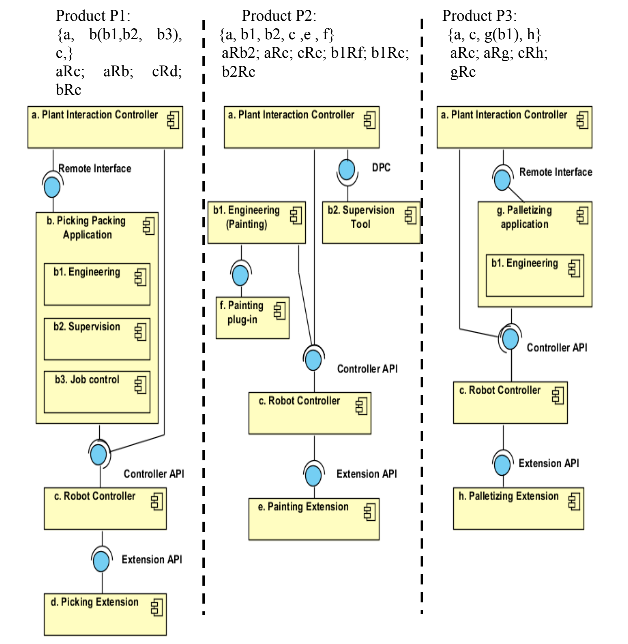

Fig. 1. Architecture logical view of 3 products of a Robotics Industry [10].

In Figure 1, *aRb* means that component *a* is connected to component *b* by some type of connector *R*, and for example for the first product: $P_1 = \{a, b, c, d\}$; sub-components are denoted by a list, *b (b1, b2, b3)*. In fact, a component b containing other components b1, b2, b3 can be considered also as a set of 4 components {b, b1, b2, b3} and relations b R b1, b R b2 and b R b3; however, we have not considered this composition relation here. The "functional relevance" of each component with respect to the architectural configuration of each product is expressed by their weights, provided by the architect, as shown in Table I, according to [15].

TABLE I. COMPONENT WEIGHTS FOR EACH PRODUCT

P_i\w	a	b	b1	b2	b3	c	d	e	f	g	h
P_1	3	3	2	2	2	3	1				
P_2	3		3	3		3		1	1		
P_3	3		2			3				3	1

However, other architectural metrics could have been chosen [30], such as for example the SPL compound metrics of [31] or the SPL structural metrics of Rahman [32] based on the reusability and modularity offered by the component. In our case, the components shared by the 3 initial architectures seemed to be the most functionally relevant and therefore have a weight equal to 3. The weight 2 is when they appear in 2 architectures and otherwise, their weight is 1. Product architectures shown in Figure 1 had been recovered in [10] using reverse engineering techniques, which are not discussed here. However, component names with similar semantics were unified in this paper for the three products, considering internal name and content similarity according to [22].

In consequence, intermediate valid configurations in different products containing the same component names are considered equivalent. This requires a preceding step where the similar components of the initial architectures have been recognized and unified to form a single reusable component. This preceding step is not detailed in this paper, but it is really crucial and… difficult to achieve.

III. REACTIVE DESIGN PROCESS TO BUILD THE *PLA*

Figure 2 shows the main activities or sub-processes of the refactoring process to construct the *PLA*.

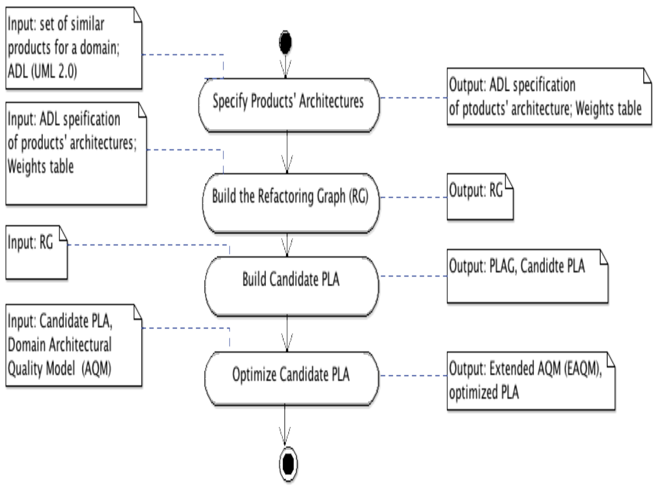
Input: set of similar products for a domain; ADL (UML 2.0)
Specify Products' Architectures
Output: ADL specification of ptoducts' architecture; Weights table
Input: ADL speification of products' architectures; Weights table
Build the Refactoring Graph (RG)
Output: RG
Input: RG
Build Candidate PLA
Output: PLAG, Candidte PLA
Input: Candidate PLA, Domain Architectural Quality Model (AQM)
Optimize Candidate PLA
Output: Extended AQM (EAQM), optimized PLA

Fig. 2. Activity diagram for the *PLA* Reactive Design Process.

In what follows, each step of the *PLA* Refactoring Process will be detailed and applied to the case study.

A. *Specify products' architectures.*

- *Process to specify products' architectures*

Actor: Software Architect
Input: Description of the products' architectures obtained from the domain knowledge, using reverse engineering techniques

```
For each product
Begin
        Study names and contents of components; study connections.
        Unify components' names for all products considering internal similarity of name and content.
        Assign a weight to each component, according to its relevance to the architecture.
        Specify the architecture of the current product using an ADL.
End
```

Output: Specification of products' architectures in ADL and weights table.

- *Application to the case study*

Figure 1 shows the UML specification of the three products for robotics industry considered. Names for components and their connections are specified as follows: *P_1, P = {a, b (b1, b2, b3), c, d), R = {aRc, aRb, cRd, bRc}; P_2, P = {a, b1, b2, c, e, f}, R = {aRb2, aRc, cRe, b1Rf, b1Rc, b2Rc}; P_3, P = {a, c, g (b1), h}, R = {aRc, aRg, cRh, gRc}*. The weights of each component were shown in Table I.

B. *Build the Refactoring Graph (RG).*

- *Process to construct RG for each product*

Actor: Refactoring Tool
Input: ADL specification of the architecture for each product *(P, R)*

```
Begin
let RGV being the set of vertices of RG
let RGE being the set of edges of RG
  Begin
    RGE = Ø
    RGV = G
    Level(N) = G
    for n = N-1 to 1 (step - 1)
          We create the set of LevelE(n) containing all
          valid configurations of (P, R) of order n such
          that for each configuration G′ in LevelE(n+1)
             for each Component C in configuration G′
               if (R' - C) is a valid configuration then
                 we add this configuration to LevelE(n):
                 LevelE(n) = RGV U (G′-C)
                 we register the couple ((G′-C), G′) as a
                 transformation in RG
                RGE = RGE U {(G′ -C) --> G}
              End if
            next Component in configuration G′
          next configuration in LevelE(n+1).
         We add the current level to the RG:
         RGV = RGV U LevelE(n) next level
    End
End
```

Output: the *RG* of *(P, R)* following *P: (RGV, RGE)*

- *Application to the case study*

Figure 3 shows the *RG* for product P_1. The node *abcd* represents the original architectural configuration of product P_1 on level L_4 (see Figure 1), and *RG* shows that *abcd* can be obtained by three alternative ways from the preceding level L_3, either from valid configuration *abc* by adding *d* or from valid configuration *bcd* by adding *a*, and finally from *acd* by adding *b*. Figures 4 and 5 show the refactoring graphs *RG* for products P_2 and P_3, respectively. In these graphs, each node represents (part of) an architecture.

C. *Build Candidate PLA.*

The *Candidate PLA* is an architecture that contains the variation points and those architectural components that are functionally most relevant to the domain, according to the heuristic stated in [15]; this *heuristic* ensures the presence of the complete architectural configuration of at least one of the products, guaranteeing in this way that a domain architectural style will be also included in the candidate *PLA*.

Let us note that the *PLA* quality requirements will be considered during the optimization step of the Candidate *PLA*. Any variation point can be considered as the *initial PLA*. In the case study, for example, the common configuration with the maximum number of components in the *RGs* of the three products is *ac*. From this initial *PLA*, the Candidate *PLA* will be constructed, according to the proposed heuristic. To do so, we need to find the more "*convenient path*" to obtain at least one of the products, among different existing paths containing the variation point. This path will contain the most relevant functionality of the architecture (see Table I). For example in product P_1, *(ac, b, 3)* means that component *b* is added to configuration *ac*, to conform configuration *abc;* since $w=3$, *b* (Picking/Packing application) is a relevant functionality for product P_1 of the robotics industry.

Two main automatic sub-processes are used to build the Candidate *PLA*: *Construct the Product Line Architecture Graph (PLAG)* and *Construct the Candidate PLA*; we proceed as follows:

- *Construction of the PLAG*

On the level (containing at least one of the products) *nearest* to the level containing the *VP*(s), a valid configuration Q_i containing the *VP(s)* is selected for each *RG*, according to the heuristic which considers given weights ranking the relevance of a component w.r.t. the product architectural configuration. The *PLAG* is a graph defined by levels; it contains the *VP(s)* and the paths starting from L_1 including these *VP(s)* and ending in the configuration Q_i; these paths are the "convenient paths".

- *Construction of the Candidate PLA*

It performs the automatic fusion, respecting component connections, of all the Q_i architectural configurations, will be a *candidate PLA*. According to our heuristic, this first *PLA* configuration contains all possible *VP*(s) and most relevant common functionalities considered for the domain. This architecture has to be optimized in the final step, considering domain quality requirements, to obtain the final *PLA*.

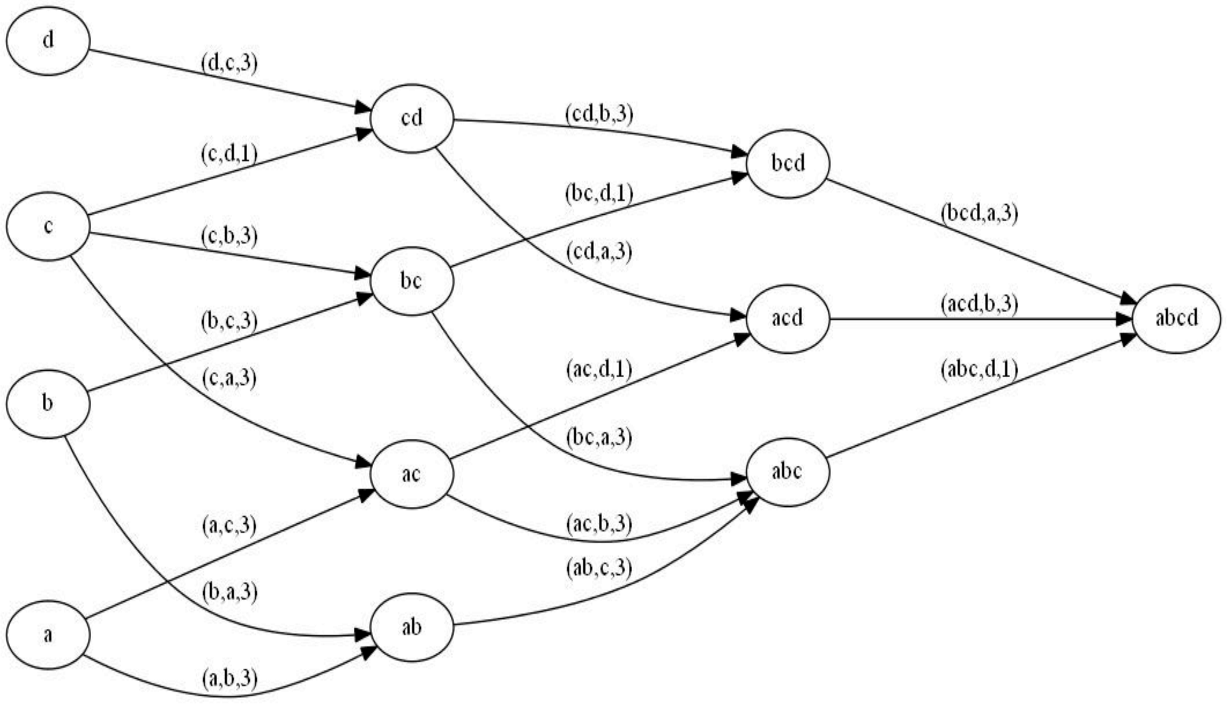

Fig. 3. *RG* of product *P1*.

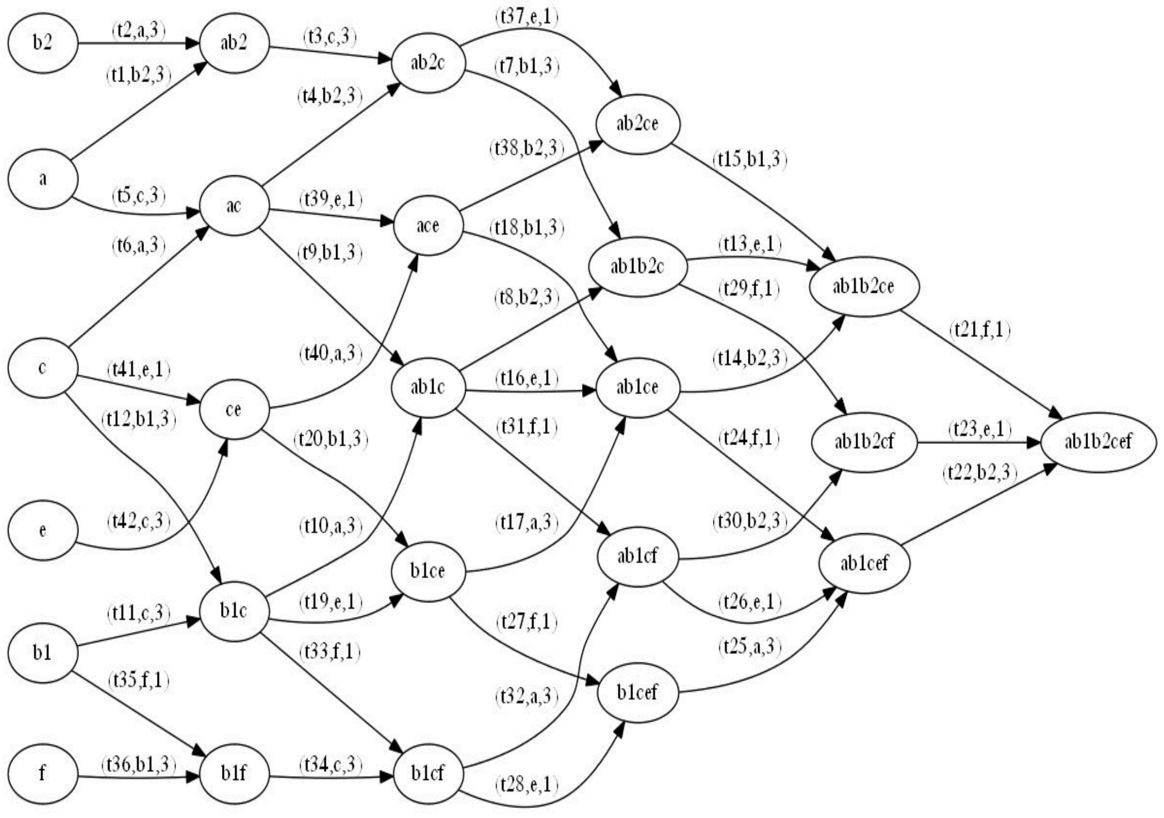

Fig. 4. *RG* of product *P2*.

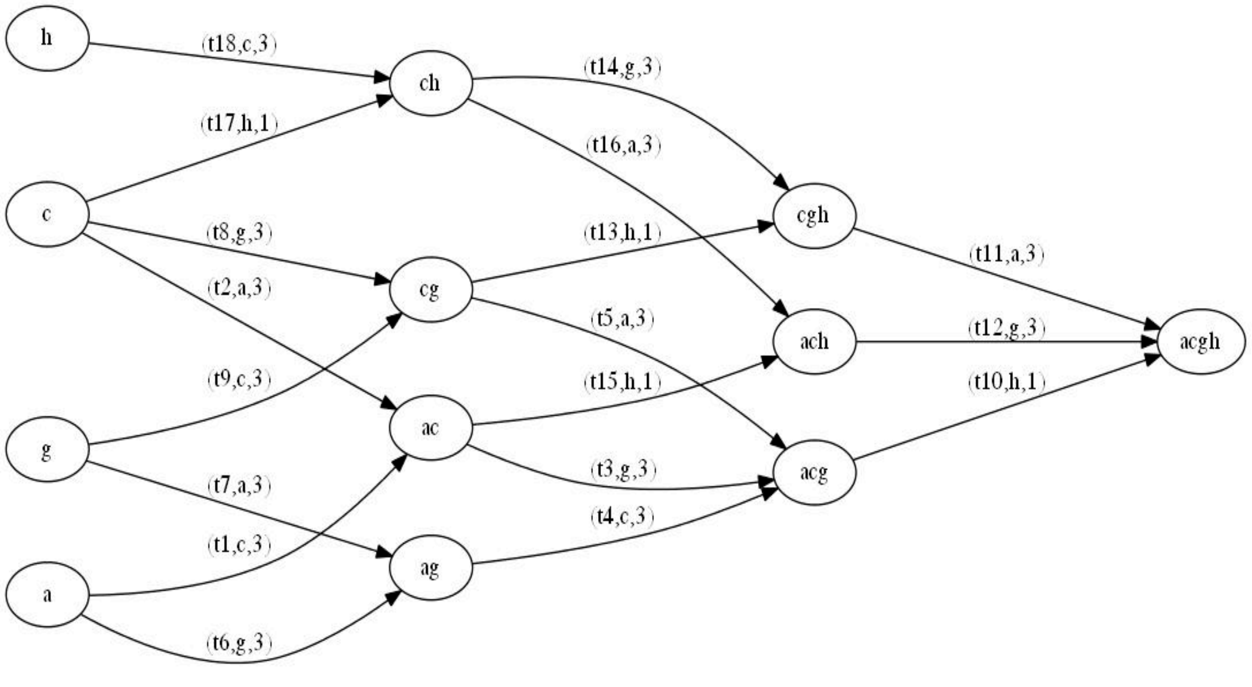

Fig. 5. *RG* of product *P3*.

- *Process to Construct the Product Line Architecture Graph (PLAG)*

Actor: Refactoring tool
Input: RG of each product P_k, *k=1, …, n.*

```
Begin
For each RG
  Begin
    Begin
      Find the VP(s), V, i.e., equal configuration nodes on the
      same level $L_q$, $q \geq 2$, in each RG of $P_k$, with the max number
      of components. Let $d(V, P_k)$ for some V, be the number of
      transformations required to reach $P_k$ from V; $M = \min d(V, P_k)$, $k = 1, …, n$.
      For each intermediate valid configuration Q on $L_{M+q}$ of
      each RG of $P_k$, such that $V \subseteq Q$, compute the number of
      incidences of the weight values of the Q components,
      according to the weights of $P_k$ components.
      The PLAG is the graph constituted by levels $L_1$, $L_2$, ..,
      $L_q$, $L_{q+1}$,…, $L_{M+q}$, each $L_i$ contains configurations with i
      nodes.
      VP(s), V, are placed on the PLAG on $L_q$.
      For each V in $L_q$
       Begin
         For each $P_k$
          Begin
            Let Q on $L_{M+q}$ such that $V \subseteq Q$ with the greatest
            Number of incidences each weight value, by
            default.
            Select a path from $L_1$ to Q in RG of $P_k$
            containing the variation point V and those
            intermediate configurations of Q.
            Place the selected convenient path on the PLAG
            levels.
          End
       End
    End
End
Output: PLAG
```

- *Application to the case study*

The resulting *PLAG* obtained for the case study is given in Figure 6. It describes how to construct the *SPL* common architecture for the Robotics Industry and how to construct each of the different products conforming the *SPL* family. In this case, there is only one variation point, but there could be several *VP(s)* denoting the different decisions to be taken when designing a product. Notice that in Figure 6, all the paths to obtain the three products were considered.

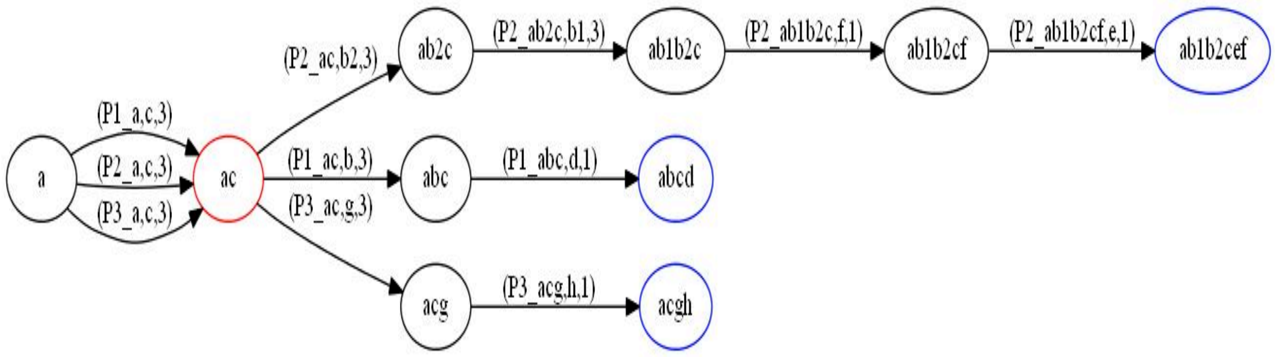

Fig. 6. *PLAG* for the Candidate *PLA* for Robotics Industry.

- *Process to construct the Candidate PLA*

Actor: Refactoring tool
Input: PLAG

```
Begin
    The fusion of all configuration nodes on level $L_{M+q}$ of the
    PLAG, which includes all variation points, and respecting
    connections, constitutes the Candidate PLA.
End
Output: Candidate PLA
```

- *Application to the case study*

The *PLA* is constituted by the fusion of the configuration nodes *abcd, ab1b2c, acgh* on L_4 belonging to the selected path, from *ac* to P_1, P_2, P_3 respectively (see Figures 6, 7, 8).

D. Optimize Candidate PLA.

This optimization step is a manual refactoring process performed using the expertise of the architect and the domain engineer to transform iteratively the Candidate *PLA* into the final *PLA*, responding to specific architectural quality properties of the domain. It considers three main sub-processes: *build AQM* (*Architectural Quality Model*), *build EAQM* (*Extended Architectural Quality Model*), and *refactor the Candidate PLA*; we proceed as follows:

- *Build AQM*

The software architect will take into account the knowledge of the domain (*DD*) to build the *AQM,* which contains the *PLA* quality requirements; these architectural quality goals drive the whole optimization process. It is specified according to the ISO/IEC 25010 standard. Notice that *AQM* should be part of the *PLA* asset repository; otherwise it should be built accordingly, using the available domain knowledge. The quality attributes to be considered in a particular domain, together with their metrics have to be defined.

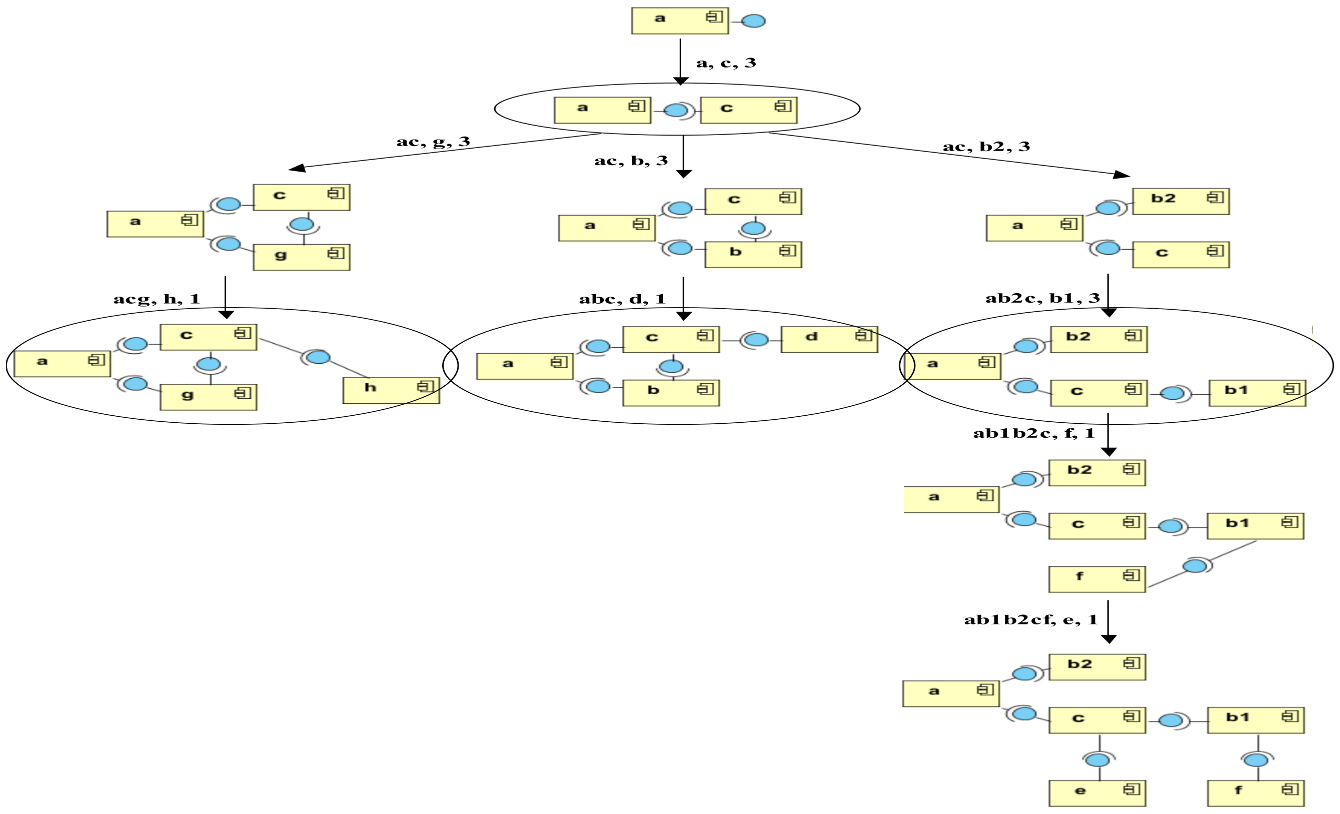

Fig. 7. UML static view of *PLAG* for Robotics Industry, showing main components of Candidate *PLA*.

- *Build EAQM*

EAQM extends the *AQM* attaching a priority to each quality sub-characteristics (low, medium or high), an architectural solution for each sub-characteristic, and a comment justifying the selection. *EAQM* represents a quality-based scenario, such as those used in [9], [29], [30]. Notice that a complete architectural evaluation process, such as [26] or inspired in scenario-based evaluation methods, can be used in the selection step to justify the solution choice, which implies adding new or removing existing components. These components can be predefined architectural design patterns taken from catalogues, which are part of the *PLA* software assets. Table III shows the architectural solutions proposed for the case study, adding/removing components to take in charge the achievement of the quality; for example, to be able to recover the system efficiently in case of failure. Notice that *EAQM* could be mapped to a Software Interdependency Graph (SIG), where softgoals correspond to ISO/IEC 25010 quality characteristics and sub-characteristics are refinements of softgoals; operationalizations are represented by the architectural solutions given in Table III.

- *Refactor Candidate PLA*

To refine or improve the selection of the architectural solution, the contributions technique from [16] is used. Contributions for each quality characteristic with respect to each other have been defined in Table IV for the case study, where +, -, 0 means positive, negative, indifferent contribution respectively. For example, if maintainability is achieved by introducing a new functionality, a user interface component to separate concerns and hence decrease coupling can contribute positively (+) to s*ecurity* if it includes authentication functionality, and also to *performance*, since coupling among components is reduced. The satisfaction of the security issue contributes negatively (-) to performance because extra time checking is increased, for example during the login functionality. Notice that more sophisticated metrics for each quality attribute, according also to the particular domain can be provided to enrich Table II.

- *Process to Optimize Candidate PLA*

```
Optimization process
Actors: Software Architect Domain Engineer
Input: Domain Description (DD), Candidate PLA, Asset
         Repository (AR) including components on level $L_1$, ISO/IEC
         25010 standard to specify the AQM, Architectural
        Evaluation Method (optional)
Begin 'optimization of Candidate PLA'

Build the AQM
Input: DD, ISO/IEC 25010
Begin 'building AQM'
   Elicitate domain architectural requirements from the DD.
   For each architectural requirement
  Begin
      State the architectural solutions
      For each architectural solution,
          Begin
               Specify quality requirements as ISO/IEC 25010 quality
               characteristics and sub-characteristics.
               List quality characteristics and sub-characteristics
               For each sub-characteristic
                  Begin
                     Identify quality attributes and metrics to define the
                    AQM for the domain, in terms of ISO/IEC 25010.
                  End
           End
     End
  End
End 'building AQM'
Output: AQM

Build the EAQM
Begin 'building EAQM'
    Begin
    For each sub-characteristic of the AQM, specify a priority
    end
   Begin
     For each priority (high, medium, low), starting from the
     sub-characteristic with the highest priority
     Begin
          Evaluate the value of each quality on the Candidate PLA;
               for example, evaluate the coupling attribute using the
               metrics "number of connections" of a component.
          Provide an architectural solution (addition/deletion
          of components);
          if more than one architectural  solution is present, then a
                scenario-based AEM could be used to select the
                convenient solution to help the architect in his choice.
          End if.
             Note that some quality requirements of minor priority
             can be achieved while satisfying higher priority ones;
                the study of contributions of each quality characteristic
                can be useful to handle these trade-offs (see Table 3).
             Reconfigure the Candidate PLA with the selected
             architectural solution.
             Provide a textual justification of the decision taken.
      End
   End
End 'building EAQM'
Output: architectural solutions, EAQM
Refactor Candidate PLA
Input: architectural solutions, EAQM, PLA
Begin 'refactoring of Candidate PLA'
   Complete Candidate PLA with the selected architectural
   solutions identified in the previous step, respecting connections
   between components.
   Study the AR to complete the PLA, if some components are still
   missing; add the missing components respecting connections.
End 'refactoring of Candidate PLA'
Output: PLA

End 'optimization of Candidate PLA'
Output: AQM, EAQM, PLA
```

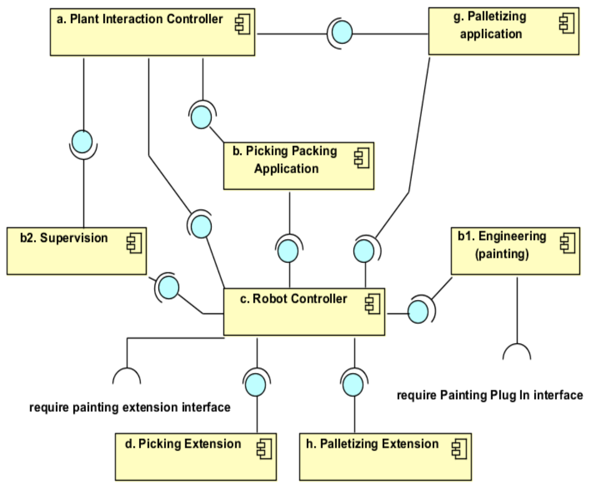
a. Plant Interaction Controller
g. Palletizing application
b. Picking Packing Application
b2. Supervision
b1. Engineering (painting)
c. Robot Controller
require painting extension interface
require Painting Plug In interface
d. Picking Extension
h. Palletizing Extension

Fig. 8. Candidate *PLA* for Robotics Industry.

- *Application to the case study*

Table II presents the *AQM* for our case study: *reliability, maintainability, usability, security and performance*; the sub-characteristics of *maintainability* that have been selected for the robotics industry *AQM* are: *modularity* (the degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components), and *modifiability* (the degree to which a product can be effectively and efficiently modified without introducing defects or degrading performance). *Reusability* (the degree to which an asset can be used in more than one system, or in building other assets) is a global property that *PLA* has to accomplish as an asset and was not included in *AQM*; we preferred to enforce the fact that systems in the robotics industry domain must support evolution expressing *scalability* and *sustainability* (as a sub-sub-characteristic of modifiability), as it was stated in [10].

TABLE II. *AQM* FOR ROBOTICS INDUSTRY

Quality Characteristics	Quality sub-characteristics	Quality Attributes	Metrics
Reliability	- Availability	Time to recover from failure	MTTR (1)
		Mean time between failures	MTBF (2)
Maintainability	- Modularity - Modifiability - Scalability - Sustainability	Coupling (4)	Number of connections
Usability	- Appropriateness - Accessibility	Presence of a mechanism	Yes/no
Security	- Authenticity	Presence of a mechanism	
Performance	- Time behaviour	Response time	Latency (3)

(1) MTTR = average time that a device take to recover from any failure;
(2) MTBF = average time between the difference of time spent failing and time spent recovering
(3) Latency is a measure of the time delay experienced in a system; it can be computed roughly at architectural level, by summing up for each component the time spent in requiring/providing data, in a given scenario.
(4) This metric was proposed in [10]; other measures could certainly be used here; however, a low coupling is a sign of a well-structured system and a good design, and when combined with high cohesion, supports the general goals of high readability and maintainability.

The *EAQM* in Table III proposes architectural solutions for each quality requirements of the AQM, with their priority, where *modularity* and *modifiability* have the highest priority; according to our process, they will be considered first. *Modularity* is achieved by introducing a new Remote User Interface (RUI), and with the introduction of a new Engineering component*,* the *modifiability (scalability* and *sustainability)* requirement is accomplished; both solutions contribute to decrease the coupling. Moreover, the RUI component also improve the system usage, hence it will also directly contribute to achieve *usability*, with medium priority, which according to Table IV, could affect negatively the *performance* (-), with also medium priority. However, the *security* mechanism for *authenticity,* of low priority, that affects also negatively the *performance* (-)*,* increases the system's *reliability* (+), with medium priority, with respect to access control, and it is also accomplished by the RUI component. On the other hand, the Engineering component includes a *Job Controller* sub-component*,* which improves the *performance (time behaviour)* with a reduced *response time* and the *reliability* will finally also benefit (+) from the reduced coupling because the *recovery time* could be reduced, satisfying also the *availability* requirement of medium priority. In consequence, in this case study, the fact of satisfying the quality characteristics with highest priority has as well contributed to the satisfaction of the lower priority quality requirements.

The optimized *PLA*, obtained from the Candidate *PLA*, is shown in Figure 9. The Job Controller component is not shown, since it is included in the Engineering component. However, it can be adapted from the existing *b3* component of the Picking/Packing Application of product P_1, and added between the Plant Interaction Controller (a) and the Robot Controller (c) components to further increase *availability* and even some *performance*. The Engineering component is now responsible for the painting, palletizing and picking/packing applications.

Notice that components *e* and *f* were still missing in the candidate *PLA* of Figure 8, however they are required by components *c* and *b1* to perform the required functionality. According to the process to optimize the Candidate *PLA*, in the Refactor *PLA* sub-process, these components can be added from the asset repository, respecting the connections. In Figure 9 only the painting extension, component *e*, has been shown to illustrate this aspect.

TABLE III. *EAQM* FOR ROBOTICS INDUSTRY

Quality sub-characteristics	Priority	Architectural solution	Justification
- Availability	Medium	- Add a new Engineering component: (b3) Job Controller	To improve recovering time in case of failure
- Modularity	High	- Add a new RUI component: i	To decrease coupling
- Modifiability - Scalability - Sustainability		- Add a new Engineering component: (b3) Job Controller	To decrease coupling
- Appropriateness - Accessibility	Medium	- Add a new RUI component: i	To improve also the system usage by the end user
- Authenticity	Low	- Add a new RUI component: i	With an additional authentication functionality
- Time behaviour	Medium	- Add a new Engineering component: (b3) Job Controller	To improve the response time

TABLE IV. CONTRIBUTIONS FOR EACH QUALITY CHARACTERISTIC OF THE PROPOSED ARCHITECTURAL SOLUTION

Quality Characteristic	Reliability	Maintain-ability	Usability	Security	Performance
Reliability		+	0	+	+
Maintainability	+		+	+	+
Usability	0	+		+	-
Security	+	+	+		-
Performance	+	+	-	-	

There are tools that support the partial automation of the presented sub-processes [16], [23], [24], [26], [27], however a complete automation is still difficult to achieve; the final selection of architectural solutions is generally provided by the architect [15], [23], [24].

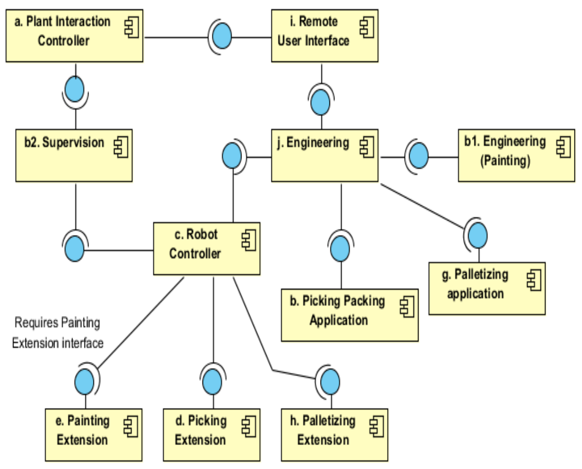
a. Plant Interaction Controller
i. Remote User Interface
b2. Supervision
j. Engineering
b1. Engineering (Painting)
c. Robot Controller
b. Picking Packing Application
g. Palletizing application
Requires Painting Extension interface
e. Painting Extension
d. Picking Extension
h. Palletizing Extension

Fig. 9. *PLA* for robotics industry.

IV. Related Works

Refactoring has been traditionally used to reconstruct legacy code and reverse engineering to recover or reconstruct documentation [17]. In the *SPL* context reengineering techniques have been used to modify existing *PLA* that in general have been built within a proactive top-down approach using domain knowledge; from this generalization, new evolutionary *PLAs* are built in [6], [18], and [19]; reverse engineering is also used in SPL to analyse feature models [20], [21]. On the other hand, reverse engineering and refactoring techniques are required to construct the *PLA* according to a bottom-up approach [5], which is the one followed in this paper. Works [10] and [22] have similarity with our approach, however none of them use a graph model to support the *PLA* construction; actually a graph model is a useful tool to make computations and our proposal is based on a graph algorithm. In [10] reverse engineering techniques were used to recover manually the architecture of three robotics industry products, from where the authors proceeded also manually to construct the *PLA* satisfying main domain architectural quality requirements; the trade-offs step was limited to an informal discussion and no standards were used to specify quality requirements.

More recent works [23], [24] propose the automation of the trade-offs step, based on multi-objective optimization, where objectives represent different quality attributes. The HAM (Hybrid Assessment Method) [9] is based on multi-criteria concepts and techniques for trade-offs analysis, in an architecture assessment process; however, quality standards are not used. In our case, the information about non-functional requirements, such as architectural styles and their quality, on one hand is contained in the candidate Architecture, since al least one of the *SPL* family products is included; on the other hand, in the optimization step, it is captured by the *AQM* with attributes and metrics and by the *EAQM*, which is used for the choice of the architectural solution, as a classic architecture evaluation scenario [30], providing a complete picture of the domain architectural requirements. However, HAM could be used to improve the assignment of the priorities of the quality properties in the *EAQM*.

Our paper inspired in [10], proposes a tool-supported automatic refactoring process to construct first a Candidate *PLA*; the final *PLA* is obtained by manual refactoring this candidate *PLA* to satisfy specific quality requirements; the ISO/IEC 25010 standard is used to improve communication among the work teams. The trade-offs step is performed manually, however goal-oriented techniques can be used to improve the selection of the architectural solution and more sophisticated tool-supported techniques could be introduced [9]. A semi-automatic *PLA* recovery approach is presented in [22], assuming that involved legacy products have similar designs and implementations. However, they do not deal specifically with the architectural configuration, as we do; measures are defined to detect class, code, and methods similarities; we use some of them in the first step of our process to unify components' names. In [28] legacy software products are systematically reengineered into *SPL*, based on automatic variability analysis. They propose a hybrid approach that consolidates feature knowledge from top-down domain analysis with bottom-up analysis of products' code similarities. The bottom-up analysis follows an approach similar to [22] with respect to similarity measures.

In general, we can appreciate the use of reengineering techniques and semi-automatic processes for the evolution and/or construction of the *PLA*, however complete automation has not yet been achieved, because the expertise of the software architect is still considered in practice the key issue to select convenient architectural solutions [9], [15]. Our main contribution is to produce automatically the candidate PLA architecture, using a graph model as a supporting structure for computations.

V. Conclusion

A semiautomatic reactive bottom-up process based on the *RG* graph model has been proposed to build a *PLA*. *RG* expresses all different ways to assemble a product starting by a component and adding one component at a time, such that connectivity is preserved in already assembled configurations. Moreover, *RG* allows the clear identification of *VPs* for all products since they belong to the same level, facilitating the fusion process, which completes the candidate *PLA* with the main common functionality, and including at least an architectural style of one of the given products, according to the heuristic provided. Combinatorial explosion in case of huge products is limited by the connectivity of valid configurations. The automatic construction of *RG*, *PLAG* and Candidate *PLA* is supported by a prototype computational tool, which can be used also for didactical purposes, to show the *PLA* construction process. The optimization of the Candidate *PLA* is still manual; the standard *AQM* is used as a main scenario-based quality requirements specification tool, offering the complete picture of the domain quality requirements; however the trade-offs analysis of quality requirements can be automated integrating existing tools from multi-objective optimization models and from goal-oriented engineering; for example *EAQM* can be mapped to a *SIG* that can be automatically generated and Model-Driven and Goal-Oriented Engineering techniques could be used. These are still on-going research trends. Let us note about the first step of the process, that the existence of the architectures of several products used to construct the *PLA* is not as simple as it may seem. It involves a considerable reverse engineering effort in documenting individual components and unifying their names and semantics using complex similarity measures. Our support tool is currently under construction; Figures 3, 4, 5 and 6 have been obtained using it. This paper considers only *VP(s)* common to all products. In the near future the case of partial variability will be considered.

ACKNOWLEDGMENT

This research has been partially sponsored by the PEII-Fonacit DISoft project N^{o} 2011001343, and the CDCH DesCCaP project Nr. PG-03-8014-2011, Universidad Central de Venezuela.

REFERENCES

[1] P. Clements and L. Northrop. "Software product lines: practices and patterns", 3rd edn., Readings, MA, Addison Wesley, 2001.

[2] M. Shaw, D. Garlan. "Software Architecture. Perspectives of an emerging discipline", Prentice-Hall, 1996

[3] K. Pohl, G. Böckle, F. van der Linden. "Software product line engineering - foundations, principles, and techniques". Springer IXXVI, 1-467, 2005.

[4] E. Y. Nakagawa, P. O. Antonio and M. Becker. "Reference architecture and product line architecture: a subtle but critical difference", Crancovic V., Grhun, and M. Book (Eds.): ECSA 2011, LNCS 6903, pp. 207-211, Springer-Verlag, Berlin, Heidelberg, 2011.

[5] P. Clements and C. Krueger. "Point-Counterpoint", *IEEE Software, 19(4),* 2002.

[6] A. Rashid, JC. Royer, A. Rummler (Eds). "Aspect-Oriented Model-Driven Software Product Lines. The AMPLE Way". Chapter 1, p. 4, Cambridge University Press, Cambridge, 2011.

[7] M. Matinlassi. "Comparison of software product line architecture design Methods: COPA, FAST, FORM, KobrA and QADA", Proc. of the 26th. Inter. Conference on Software Engineering (ICSE'04), 2004.

[8] E. J. Chikofsky and J. H. Cross, II. "Reverse engineering and design recovery: A taxonomy". IEEE Software, pp. 13–17, January, 1990.

[9] A. Rashid, JC. Royer and A. Rummler (Eds). "Aspect-Oriented Model-Driven Software Product Lines. The AMPLE Way". Chapter 2, p. 48, and Chap. 5 pp. 125-157. Cambridge University Press, Cambridge, 2011.

[10] H. Koziolek, R. Weiss, J. Doppelhamer. "Evolving industrial software architectures into a software product line: A case study". R. Mirandola, J. Gorton, and C. Hofmeister (Eds): QoSA 2009, LNCS 5581, pp. 177-193, 2009.

[11] P. Krutchen. "The 4+1 View model of software architecture". IEEE Software, 12(6) pp. 42-50, 1995.

[12] Object Management Group (OMG). "Unified Modeling Language Supersturcture", version 2.0 (formal/05-07-04), August. Available at: www.omg.org/spec/UML/2.0. 2005.

[13] C. Matus. "Planificación de situaciones". Cuadernos de CENDES, Universidad Central de Venezuela, Caracas, Venezuela, 1977.

[14] B. Westfechtel and A. von der Hoek. "Software architecture and software configuration management", in Software Configuration Management (SCM): ICSE Workshops, LNCS Vol. 2649, Portland, OR, Springer-Verlag, May, pp. 24-39, 2003.

[15] J. Bosch. "Design and use of software architectures - adopting and evolving a product-line approach". Addison-Wesley, 2000.

[16] L. Chung, B. Nixon and E. Yu. "Using non-functional requirements to systematically select among alternatives in architectural Design", in 1st Inter. Workshop on Architectures for Software Systems, pp. 31-42, Seattle, Washington, 1995.

[17] M. Fowler. 1999. *Refactoring. Improving the design of existing code*. Addison-Wesley.

[18] M. Critchlow, K. Dodd, J. Chou, and A. van der Hoek. "Refactoring product line architectures", in IWR: Achievements, Challenges, and Effects, pp. 23–26, 2003.

[19] D. Saraiva, L. Pereira, T. Batista, F.C. Delicato, P.F. Pires, U. Kulesza, R. Araújo, T. Freitas, S. Miranda, and A.L. Souto. "Architecting a model-driven aspect-oriented product line for a digital TV middleware: A refactoring experience". LNCS, Volume 6285, Software Architecture, pp. 166-181, 2010.

[20] M. Acher, A. Cleve, P. Collet, P. Merle, L. Duchien and P. Lahire. "Reverse engineering architectural feature models". ECSA'11, 2011.

[21] S. She, R. Lotufo, T. Berger, A. Wasowski and K. Czarnecki. "Reverse engineering feature models". ICSE'11: 461-470, 2011.

[22] Y. Wu, Y. Yang, X. Peng, C. Qiu and W. Zhao. "Recovering object-oriented framework for software product line reengineering". 12th Inter. Conf. on Software Reuse, ICSR'11, Pohang, South Korea, June 13-17. LNCS 6727, Springer, pp. 119-134, 2011.

[23] A. Martens, D. Ardagna, H. Koziolek, R. Mirandola and R. Reussner. "A hybrid approach for multi-attribute QoS optimisation, in component based software systems", 6th QoSA'10, vol. 6093 LNCS, pp. 84-101. Springer, June 2010.

[24] A. Koziolek, H. Koziolek and R. Reussne, "PerOpteryx. Automated application of tactics in multi-objective software architecture optimization", QoSA'11, Boulder, Colorado, USA, 2011.

[25] F. Losavio, A. Matteo, N. Levy, "Web services domain knowledge with an ontology on software quality standards." ITA'09, pp.74-85, CAIR, Glyndwr University, UK, Sept. 2009

[26] L. Zhu, A. Keaurum, I. Gorton and R. Jeffrey. "Trade-off and sensitivity analysis in software architecture evaluation using analytic hierarchy process", Software Quality Journal, 13, pp. 357–375, 2005.

[27] I.M. Murwantara. "Towards quality attributes decision modelling approach for a product line architecture", *IJCSNS* International Journal of Computer Science and Network Security, Vol.11 No.11, November 2011.

[28] Y. Xue. "Reengineering legacy software products into software product line based on automatic variability analysis". ICSE 2011: pp. 1114-1117, 2011.

[29] Clements P., Kazman R. and Klein M. "Evaluating Software Architectures: Methods and Case Studies", Addison-Wesley, 2002.

[30] Tan L., Lin Y., Ye H. "Quality-Oriented Software Product Line Architecture Design", Journal of Software Engineering and Applications, Vol. 5, 472-476, 2012.

[31] van der Hoeck A., Dincel E., Medvidovic N. Metrics to Assess the Structure of Product Line Architectures, MTRICS'03, 9th International Symposium on Software Metrics, p 298, 2003.

[32] Rahman A. Metrics for the Structural Assessment of Product Lines Architectures, Master Thesis, School of Engineering, Blekinge Institute of Technology, Ronneby, Sweden, 2004.