Representation of Software Product Line Architectures for Search-Based Design

Thelma Elita Colanzi[1,2], Silvia Regina Vergilio[1]
[1]Computer Science Department Federal University of Paraná (UFPR)
CP: 19081, CEP: 19031-970, Curitiba, Brazil
[2]Computer Science Departament of State University of Maringá (UEM)
CEP: 87.020-900, Maringá, Brazil
{thelmae, silvia}@inf.ufpr.br

***Abstract*—The Product-Line Architecture (PLA) is the main artifact of a Software Product Line (SPL). Search-based approaches can provide automated discovery of near-optimal PLAs and make its design less dependent on human architects. To do this, it is necessary to adopt a suitable PLA representation to apply the search operators. In this sense, we review existing architecture representations proposed by related work, but all of them need to be extended to encompass specific characteristics of SPL. Then, the use of such representations for PLA is discussed and, based on the performed analysis, we introduce a novel direct PLA representation for search-based optimization. Some implementation aspects are discussed involving implementation details about the proposed PLA representation, constraints and impact on specific search operators. Ongoing work addresses the application of specific search operators for the proposed representation and the definition of a fitness function to be applied in a multi-objective search-based approach for the PLA design.**

***Index Terms*—architecture modelling, software product line, multi-objective search-based approach.**

I. Introduction

A Software Product Line (SPL) is comprised by core assets that have explicit common and variable features [1]. Features are attributes of a software system that affect directly the final users and they are usually represented in feature models. A product of the SPL is given by a combination of its features.

Several SPL development activities can be benefitted by search-based approaches. The design of software Product Line Architectures (PLAs) is one of them. PLAs entail a design that is common to all the products derived from the SPL [1]. The design of a PLA should encompass the components realizing all the mandatory and varying features in a domain [2]. So, a PLA is a key asset in SPL Engineering since it allows the large-scale reuse. The focus of this work is the PLA design, represented by UML class diagrams, since this kind of model is commonly used to model software architectures in the detailed level [3].

The presence of many inter-related variation points, the modularity maximization, the fulfillment of quality requirements, etc., turn the PLA design into a people-intensive task. Search-based approaches can provide automated discovery of near-optimal PLAs and make its design less dependent on human architects. The PLA design can be optimized to fulfill, in addition to basic design principles like coupling and cohesion, other quality requirements such as modularization of the features, extensibility, complexity, etc. Independently of the used search-based technique, it is important to have a suitable representation to PLAs. Besides our previous work [4], in which we present lessons learned about the PLA design optimization, we could not to find, in the literature, studies where the PLA design is the target of search-based optimization. So, from the best of our knowledge, there is not proposed representations for PLA design.

In the context of software architecture design, several studies adopted Search-Based Software Engineering (SBSE) techniques to object-oriented software design, service-oriented software design, software modularization and software refactoring [5]. Most studies adopt representations that are not natural and not able to represent variabilities and features, such as integer vector [3] and supergene representations [6]. Furthermore, the use of such representations may turn complex the search operators implementation.

Metaheuristic approaches, such as evolutionary algorithms, usually have standard representations (bit strings, real-values vectors, etc.). To apply them to a complex search space, usually a genotype-phenotype mapping is necessary, i.e., a mapping between the standard representation and the represented solution [7]. On the other hand, it is possible to apply search operators directly on the natural representation of a solution, for instance, class diagrams. It is noticeable that some related works adopt direct representations for the software architecture design and other ones perform a genotype-phenotype mapping. In the direct representation the natural representation of the architecture is the target of search operators.

Given the aforementioned context, an open question can be posed:*"What is the most suitable PLA representation? Can we use a direct PLA representation or do we need a genotype-phenotype mapping?"*. Such questions are addressed in this paper in the following way. First, we review existing architecture representations proposed by related work. They are compared briefly. After this, we make some comments about the use of such representations for PLA. Based on the performed analysis we introduce a direct PLA representation for search-based optimization.

TABLE I
EXISTING REPRESENTATIONS USED IN RELATED WORK

	Study	Context	Representation	Applied to SPL
Genotype-Phenotype Mapping	Grunske [8]	Deployment Architectural Refactoring	Vector of integers	No
	Martens et al. [9]	Deployment Architectural Refactoring	Vector of characters	No
	Räihä [6]	Application of design patterns to architecture design	Vector of supergenes	No
	Bowman et al. [3]	Class Responsibility Assignment	Vector of integers	No
Direct Representation	Simons et al. [10]	Class Responsibility Assignment	Object-oriented representation	No
	Etemaadi et al. [7]	Application of patterns and antipatterns to architecture design	ADL	No

The paper is organized in sections. Section II presents the representations used in related work. Section III addresses the use of the existing representation to search-based PLA design. Section IV introduces a direct representation. Finally, Section V concludes the paper.

II. ARCHITECTURE REPRESENTATIONS: A REVIEW IN THE SEARCH-BASED CONTEXT

As mentioned before, we have not found works addressing PLA representations for search-based design. The subject of the most related existing works is on software architecture optimization. Räihä has recently published a *survey* describing studies on search-based software design [5]. Some of them apply SBSE techniques to object-oriented design and they are briefly reviewed below, emphasizing the adopted representation.

Some studies focus on the refactoring of existing architectures aiming at improving their structural quality [11]–[14]. They represent the architecture in class diagrams but optimize the architectural design indirectly since the search operators are applied on graphs that represent the architectural transformations. Hence the search space is composed by the alternatives of transformation sequences.

However, the architectural transformation sequence cannot be so relevant in the search-based architecture design outset, so the optimization of the architecture is more natural despite more complex. Another point to be considered in this case is to maintain the consistency of the solutions after the search operators application.

In several studies the architectural transformations (through search operators) are applied directly to the software design. That is, the possible variants of the design form the search space. To apply metaheuristics, some studies perform a genotype-phenotype mapping and apply the search operators on strings or vectors [3], [6], [8], [9]; and other ones perform the search using the natural representation of the design [7], [10].

Some of them are concerned to quality requirements optimization in the software architecture [8], [9]. Other ones focus on the architectural design outset by addressing two problems related to software design: the Class Responsibility Assignment (CRA) [3], [10] and the application of design patterns to software design [6], [7]. Table I presents the kind of representations adopted by related works that optimize the own software architecture. The following sections present more details about the representations used.

A. Genotype-Phenotype Mapping

1) Deployment Architectural Refactoring: Grunske [8] introduces a multi-objective strategy based on architecture refactoring that performs architectural transformations and identifies alternatives considering functional and quality requirements. Two objectives are adopted: to reduce development cost of satellites and to improve the reliability on the two main functions of the satellites. The representation consists of a vector of integers where each position represents a component and its value identifies the number of redundant components used. The strategy allows automatic architectural transformations in a deployment view to improve the quality requirements without impact on the system behaviour.

Martens et al. [9] propose a multi-objective approach to optimize the performance, reliability and cost of component-based architectures in a deployment view. They employ the metaheuristic NSGA-II and represent each chromosome by a vector of chosen design options.

2) Architectural design outset: Bowman et al. [3] present a multi-objective and evolutionary approach to optimize the CRA problem, considering coupling and cohesion measures. The approach works with the class diagram and the algorithm SPEA2. Aiming at obtaining a better domain model, the approach moves methods, attributes and relationships from one class to another. In addition, the algorithm adds and deletes classes. The chromosome representation tracks the attributes and methods and the class to which they belong. So, attribute and method are assigned to a gene within the chromosome. Chromosomes simply consist of integer values, where the position of the gene within the chromosome represents the class member (attribute or method) and the gene's integer value denotes their class assignment. Dependency information is represented by a dependency matrix, stored separately since the dependency information does not change during the search.

Räihä et al. [6] introduce an approach to synthesize software architecture by using genetic algorithms (GA). The approach applies architectural design patterns for mutations and two quality metrics for evaluating individual architectures: modifiability and efficiency. The fitness function is expressed in terms of coupling and cohesion metrics, and crossover operation can be performed by merging two architectures without breaking

existing pattern instances. The initial architecture obtained from the functional requirements is provided as the input to the GA. The architecture is encoded in a chromosome that consists of a vector of supergenes to represent all the operations (responsibilities) of the system. Each supergene is a vector of integers that contains information about one operation such as: name, type, the class it belongs to, the interface that it implements, design patterns associated, invoked operations, etc. It is an operation-driven representation due to the patterns applied are concerned to operations. Raiha et al. state that using such representation makes it easier to keep the architecture consistent, as no responsibility can be left out of the architecture at any point, and there is no risk of breaking the dependencies between responsibilities.

B. Direct Representation

Simons et al. [10] propose a search-based approach to find the best object-oriented conceptual software design. This approach encompasses a direct, novel object-oriented representation and genetic operators for evolutionary search. The genetic operators change methods and attributes from one class to another, and also add new classes. The fitness of each solution is obtained from coupling and cohesion metrics and the algorithm used was NSGA-II. The representation comprises classes, methods, and attributes. The design search space thus comprises a set of attributes and a set of methods, derived from use cases. Classes are represented as groupings of methods and attributes.

Etemaadi et al. [7] introduce a new hybridization process for solving the architecture design problem, in which it uses quality improvement heuristics within an evolutionary algorithm. This approach uses a direct representation within a modeling language, and specific search operators that can be implemented by the model-to-model transformation language. So, the solution is represented in a system model representation, such as an ADL (Architecture Description Language), instead of genotype-phenotype mapping approach. Authors state that existing patterns and antipatterns can be implemented as search operators to optimize the system architecture design.

We can observe that the mentioned works successfully optimize software architectures, however, they have not been applied in the SPL context in order to optimize PLAs. Furthermore, they do not consider specific characteristics of SPLs, such as variabilities, features assigned to architectural elements, and so on. In this sense, it is necessary to define a novel representation specific to PLAs. Such a representation is introduced in the next section.

III. An analysis considering PLA design

Aiming at applying search-based approaches to optimize the PLA design it is necessary to define an appropriate representation since it impacts on the search operators implementation. As no related work uses PLA representation at class diagram level, we have analysed the adaptation possibilities of each representation used in related work to the PLA design context.

Representations based on trees of the genetic programming or directed graphs of the evolutionary programming seem interesting representations, but due to the non-hierarchical structure of the software architectures such representations do not fit well [10]. Such graphs are adequated to represent the architecture transformation sequences used in [11], [13], [14], as mentioned in Section II. However we are interested in the optimization of the architecture design instead of transformation sequences.

The representation used in [3] with vector of integers is adaptable to PLA design. However it is necessary to find ways to represent variabilities and its variants and features assigned to architectural elements by using integers. In the same way, the supergene adopted in [6] can be used despite it needs to be complemented to accommodate variabilities and features. In both cases, information about dependencies and features assigned to architectural elements would be represented in a matrix separated from the chromosome. However, information regarding to the variabilities, their variation points, possible variants, and the places where each of them should be inside the chromosome since they can be changed during the search process. Such adaptation would increase the complexity of the representation becoming the application of standard search operators infeasible and making their implementation more complex.

Similarly to the approach proposed by Etemaadi et al. [7], we could evolve the PLA design specified by an ADL. However, the ADLs that support SPL specification do not encompass representation of classes [15]–[19]. They are limited to model PLAs by using only the component-connector model. In addition to this, only LightPL-Acme [18], Koala [19] and xADL-SML [17] provide automated support.

In this sense, the object-oriented representation [10] seems to be more suitable and natural than others. It is easier to extend it in order to represent variabilities and features. So, we proposed a novel representation that comprises all the needs regarding to SPL aiming at PLA optimization. This representation is described in next section.

IV. The proposed PLA representation

Considering the analysis of the existing representations, we introduce in this section a direct PLA representation. It consists on a metamodel, presented in Figure 1. This metamodel is based on the work of Chang et al. [20].

A PLA contains architectural elements (components, interfaces, operations and their inter-relationships) besides to be conform to some architectural style or design pattern. Each architectural element is assigned to feature(s) by using UML stereotypes and it can be common to all SPL products or variable, being present only in some product(s). Variable elements are associated to variabilities that have variation points and their variants.

Figure 2 depicts an example of a PLA for the SPL Arcade Game Maker (AGM) [21] that can be represented by the proposed metamodel. AGM encompasses three simple arcade games and it was created to support learning about and

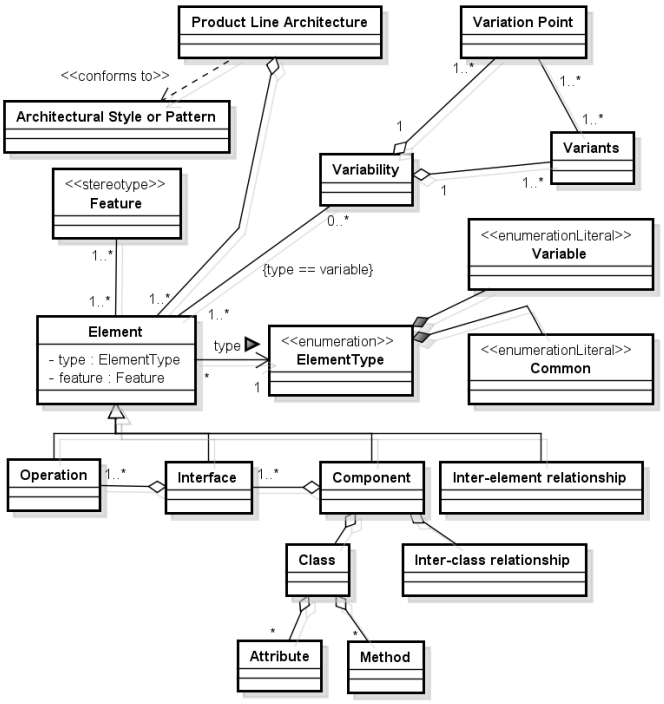

Fig. 1. PLA Representation by Metamodel

experimenting with SPL. In this example, components are represented by packages and attributes and operations were omitted in classes and interfaces. It is possible to see features assigned to some architectural elements by using stereotypes, such as action, rule, configuration and service. We are using SMarty [22] to represent variabilities in PLAs. According to the SMarty notation, there are four variabilities presented in notes assigned to classes. Variation points are labeled with the stereotype $<< variationPoint >>$ and variants are labeled with stereotype $<< alternative\_OR >>$. Mandatory variation points are also labeled with the stereotype $<< mandatory >>$.

Adopting a direct PLA representation implies in to propose specific search operators and to establish constraints to maintain the consistency of the PLA design during the search process. Though if we had extended any standard representation probably it would be necessary to adapt the standard search operators as well. In the next section we discuss some implementation aspects of using the proposed PLA representation.

A. Implementation Aspects

In our research, we have decided to start the optimization of PLA design by applying the direct PLA representation with multi-objective algorithms since they have achieved good results in SBSE. We are instantiating the framework JMetal to the context of our problem.

1) Implementing the Representation: We are modeling a PLA by using the tool Poseidon[1] and exporting the model in a XMI[2] file. In our search-based approach [23], the PLA representation is generated from this XMI file and following the proposed metamodel (Figure 1). During the XMI processing to generate the PLA representation, for each XMI element identified an object of the metamodel is instantiated according to the type of XMI element. So, the PLA representation will contain objects to represent all ar chitectural elements, their inter-relationships, all variation points, variants and features assigned to each architectural element.

During the evolutionary process the search operators are applied on the PLA representation, and, at the end, a set of PLA representations is generated as output. This set of PLA representations should be converted in a legible view to the architect, i.e., XMI file to be seen in Poseidon. One alternative that prioritizes some objective(s) can be adopted as the PLA according to the organization priorities.

We are using the framework UML2[3] of the Eclipse Foundation to generates XMI files containing the PLA design alternatives obtained from the search process. However, Poseidon cannot read the XMI files generated by UML2 due to some specific details that Poseidon includes in its XMI files. Hence we could not read the XMI files generated as output of our implementation in Poseidon. Nowadays, we are trying to adapt our existing code to work with XMI files generated by the modeling tool Papyrus[4] that is integrated with XML2 in order to create and to read the input and output of the search-based approach, respectively.

2) Search operators: We have implemented five mutation operators for software architecture based on mutation operators used in related work [3], [24] plus one mutation operator and one crossover operator specific for PLA design optimization proposed by us. The proposed operators are driven to PLA feature modularization, considering that a feature usually is solved by an architectural elements group. The mutation operators are described below:

- **Move Method**: moves a method to other existing class into the component. The class that receives the moved method becomes client from the original class of the method.
- **Move Attribute**: moves an attribute to other existing class into the component. The class that receives the moved attribute becomes client from the original class of the attribute.
- **Add Class**: moves a method or an attribute to a new class into the component. The addition of classes is necessary since optimal class assignments may require additional classes that were not identified in the design outset. The new class becomes client from the original class of the moved method/attribute.

[1] $< http://www.gentleware.com/ >$
[2] $< http://www.omg.org/spec/XMI/2.1.1/ >$
[3] $< www.eclipse.org/uml2/ >$
[4] $< http://www.eclipse.org/modeling/mdt/papyrus/ >$

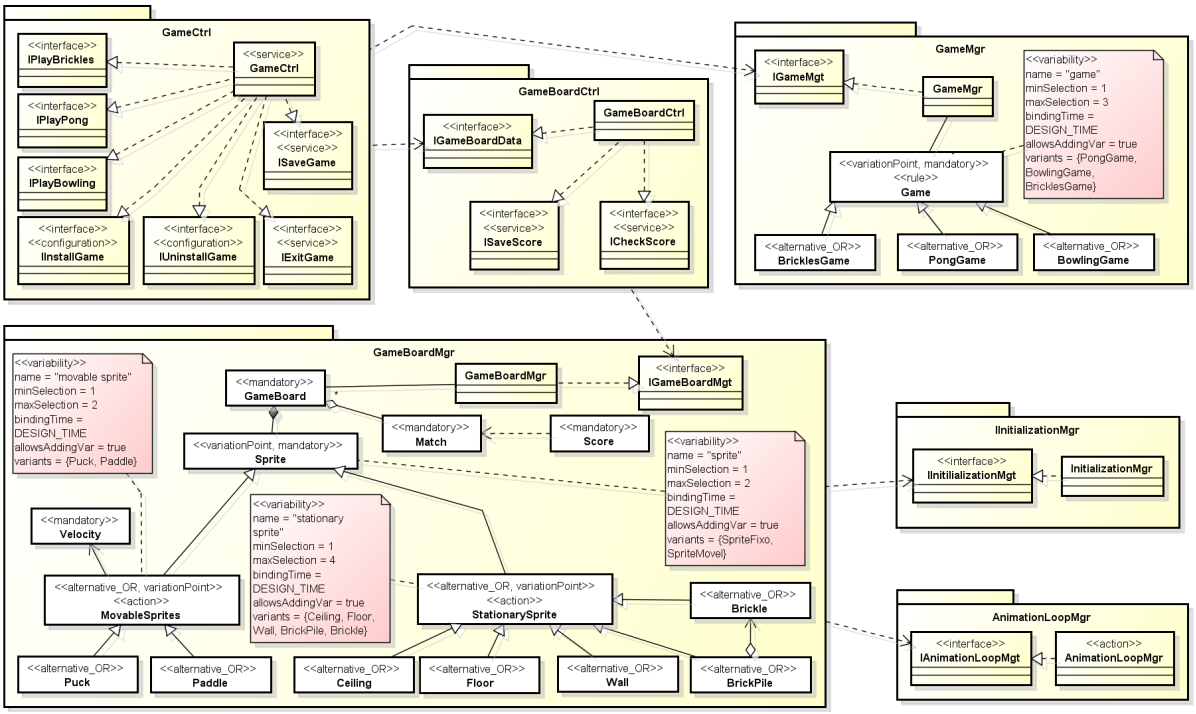

Fig. 2. Example of a PLA

- **Move Operation**: moves an operation from an interface to another. The component that receives the moved operation becomes client from the original interface of the operation.
- **Add Component**: similar to *Add Class*, it creates a new component and a new interface for this component, and then moves an operation to the new interface. The new component becomes client from the original interface of the operation.
- **Feature-driven Mutation**: it aims at modularizing a feature tangled with others in a component. It selects an arbitrary component (c_x), and, if c_x have architectural elements (interfaces, classes, operations, methods and attributes) assigned to different features, an arbitrary feature (f_x) is selected to be modularized in a new component (c_z). All architectural elements from c_x assigned to f_x are moved to c_z. c_z becomes client from the original components from the moved architectural elements.

As the merging of two architectures seems not to be natural and there are evidences that an arbitrary crossover does not offer significant advantage in the software architecture optimization [24], [25], we have proposed an specific crossover operator based on modularization of SPL features. The **Feature-driven Crossover** randomly selects a feature (f_x) and then creates new individuals swapping the architectural elements (classes, interfaces, operations, etc.) that realize the feature f_x. From two parents ($P1$ and $P2$), two offsprings are generated as follow: $Child1$ contains the elements (and their relationships) that realize f_x from $P2$ and the other elements from $P1$; $Child2$ contains the elements (and their relationships) that realize f_x from $P1$ and the other elements from $P2$. Thus, it is possible to obtain individuals that were mutated and that better realize the feature f_x.

The manipulation of the PLA representation was natural and easy during the implementation of all operators. We believe that by using other kind of representation with genotype-phenotype mapping the implementation of the operators would be more difficult.

3) Constraints: During the search process, some constraints are applied aiming at preserving the consistency of each generated solution. They are:

- Classes and their subclasses cannot be separated.
- The moved element must maintain the original assigned feature even if it was moved to an element assigned to a different feature.
- Components, interfaces and classes cannot be empty in the design.

The two first constraints were implemented into the search operators. So, if a superclass was selected to be moved, its subclasses are moved together. We decided to repair the solutions that not satisfy the last constraint instead of excluding them from the population. By reparing a solution we can maintain the population diversity and satisfactory alternatives of design.

V. Concluding Remarks

In this paper we analysed the use of existing architecture representations proposed in the literature for the PLA context. From the performed analysis we conclude that both representations, direct and through genotype-phenotype mapping,

imply on the implementation of specific search operators and the existing representations need to be adapted to be applied in the search-based PLA design. So, we introduced a novel direct PLA representation for the application of search-based approaches to the PLA design since it is more natural and makes the implementation of specific search operators easier. We also discussed some implementation aspects regarding the proposed representation.

In addition to the proposed representation and specific search operators, other point to be investigated is related to metrics for fitness evaluation in PLA search-based design. Such subject was addressed in our previous work [4], but this still is an open question.

In this sense, our ongoing work involves the application of specific search operators and the definition of a fitness function to be applied in a multi-objective approach for the PLA design optimization.

ACKNOWLEDGMENT

This work is supported by CNPq.

REFERENCES

[1] F. v. d. Linden, F. Schmid, and E. Rommes, *Software Product Lines in Action - The Best Industrial Practice in Product Line Engineering*. Springer, 2007.

[2] A. C. Contieri Junior, G. G. Correia, T. E. Colanzi, I. M. de Souza Gimenes, E. A. Oliveira Junior, S. Ferrari, P. C. Masiero, and A. F. Garcia, "Extending UML components to develop software product-line architectures: Lessons learned," in *Proceeding of the 5th European Conference on Software Architecture*, ser. ECSA'11, I. Crnkovic, V. Gruhn, and M. Book, Eds., vol. 6903. Springer, 2011, pp. 130–138.

[3] M. Bowman, L. C. Briand, and Y. Labiche, "Solving the class responsibility assignment problem in object-oriented analysis with multi-objective genetic algorithms," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 817–837, 2010.

[4] T. E. Colanzi and S. R. Vergilio, "Applying search based optimization to software product line architectures: Lessons learned," in *Proceedings of the 4th International Symposium on Search Based Software Engineering*, ser. SSBSE'12, vol. 7515, 2012, pp. 259–266.

[5] O. Räihä, "A survey on search-based software design," *Computer Science Review*, vol. 4, no. 4, pp. 203 – 249, 2010.

[6] O. Räihä, K. Koskimies, and E. Mäkinen, "Generating software architecture spectrum with multi-objective genetic algorithms," in *2011 Third World Congress on Nature and Biologically Inspired Computing (NaBIC)*. IEEE, 2011, pp. 29–36.

[7] R. Etemaadi, M. T. M. Emmerich, and M. R. V. Chaudron, "Problem-specific search operators for metaheuristic software architecture design," in *Proceedings of the 4th International Symposium on Search Based Software Engineering*, ser. SSBSE'12, 2012, pp. 267–272.

[8] L. Grunske, "Identifying "good" architectural design alternatives with multi-objective optimization strategies," in *Proceeding of ICSE'06*. New York, NY, USA: ACM, 2006, pp. 849–852.

[9] A. Martens, H. Koziolek, S. Becker, and R. Reussner, "Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms," in *Proceedings of the first joint WOSP/SIPEW International Conference on Performance Engineering (WOSP/SIPEW '10)*. New York, NY, USA: ACM, 2010, pp. 105–116.

[10] C. Simons, I. Parmee, and R. Gwynllyw, "Interactive, evolutionary search in upstream object-oriented class design," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 798 –816, nov.-dec. 2010.

[11] M. Amoui, S. Mirarab, S. Ansari, and C. Lucas, "A genetic algorithm approach to design evolution using design pattern transformation," *International Journal of Information Technology and Intelligent Computing*, vol. 1, pp. 235–245, 2006.

[12] M. Harman and L. Tratt, "Pareto optimal search based refactoring at the design level," in *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO'07)*. New York, NY, USA: ACM, 2007, pp. 1106–1113.

[13] A. C. Jensen and B. H. Cheng, "On the use of genetic programming for automated refactoring and the introduction of design patterns," in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO'10)*. New York, NY, USA: ACM, 2010, pp. 1341–1348.

[14] F. Qayum and R. Heckel, "Local search-based refactoring as graph transformation," in *Proceedings of the 2009 1st International Symposium on Search Based Software Engineering (SSBSE'09)*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 43–46.

[15] E. A. Barbosa, T. Batista, A. Garcia, and E. Silva, "Pl-aspectualacme: an aspect-oriented architectural description language for software product lines," in *Proceedings of the 5th European Conference on Software Architecture*, ser. ECSA'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 139–146.

[16] R. Bashroush, T. Brown, I. Spence, and P. Kilpatrick, "Adlars: An architecture description language for software product lines," *Software Engineering Workshop, Annual IEEE/NASA Goddard*, vol. 0, pp. 163–173, 2005.

[17] E. M. Dashofy, A. v. d. Hoek, and R. N. Taylor, "A comprehensive approach for the development of modular software architecture description languages," *ACM Transactions on Software Engineering Methodology*, vol. 14, no. 2, pp. 199–245, Apr. 2005.

[18] E. Silva and E. Cavalcante. (2012) LightPL-ACME Studio. [Available at: http://www.dimap.ufrn.br/lightplacmestudio/index.php. Acessed in 12/12/2012.].

[19] J. C. Trigaux and P. Heymans, "Software Product Lines: State of the art," Institut d'Informatique FUNDP, Technical Report, Sep. 2003.

[20] S. H. Chang, H. J. La, and S. D. Kim, "Key issues and metrics for evaluating product line architectures," in *SEKE*, K. Zhang, G. Spanoudakis, and G. Visaggio, Eds., 2006, pp. 212–219.

[21] S. S. E. Institute. (2013) Arcade Game Maker pedagogical product line. Http://www.sei.cmu.edu/productlines/ppl/.

[22] E. A. O. Junior, I. M. S. Gimenes, and J. C. Maldonado, "Systematic management of variability in uml-based software product lines," *Journal of Universal Computer Science*, vol. 16, no. 17, pp. 2374–2393, sep 2010.

[23] T. Colanzi, "Search based design of software product lines architectures," in *2012 34th International Conference on Software Engineering (ICSE), Doctoral Symposium*, june 2012, pp. 1507 –1510.

[24] C. L. Simons, "Interactive evolutionary computing in early lifecycle software engineering design," Ph.D. dissertation, University of the West of England, Bristol, UK, 2011.

[25] O. Räihä, K. Koskimies, and E. Mäkinen, "Empirical study on the effect of crossover in genetic software architecture synthesis," in *Proc. of NaBIC*. IEEE, 2009, pp. 619–625.