Towards a Software Factory for Genetic Algorithms

Abdelghani Alidra and Mohamed Tahar Kimour

***Abstract*—Genetic algorithms (GA) are a class of powerful metaheuristic search methods that solve complex and highly nonlinear problems. However, reuse opportunities have been underexploited because reuse was made at the code level. We argue that this is inefficient because it is complex and error prone. At the opposite, we propose the use of Software Product Lines engineering (SPLE) because it offers an effective way to easily manage commonalities at the model level and efficiently customize and derive a relevant product from a family of products. Another important feature of our approach is that it opens the door to the exploitation of dynamic Software product line techniques for dynamically evolving a genetic algorithm during execution.**

***Index Terms*—Genetic algorithms, reuse, software engineering, software product line, software factory.**

I. Introduction

A genetic algorithm is "a stochastic search-based technique for finding solutions to optimization and search-based problems" [1]. In genetic algorithms, the process of natural evolution is imitated by simulating natural selection and genetic evolution. Many works has shown that genetic algorithms can be extremely efficient in exploring the search space and rapidly converging to good solutions for complex and highly nonlinear problems especially when combined with local search methods [2], [3]. They are successfully used in many disciplines such as signal and image processing, avionics, automotives, and so on.

Genetic algorithms are rather family of algorithms that share some aspects (commonalities) and differ in some others (variations). Indeed, all genetic algorithms use a population of individuals that encode candidate solutions in the search space, use fitness functions to determine the quality of each individual and use this fitness information, along with the processes of selection, crossover, and mutation, to direct the search process towards promising parts of the search space (commonalities). But each genetic algorithm uses its own adapted mechanisms that best fit the specific domain of application (variabilities). For example, each genetic algorithm uses a different domain-specific fitness function, individual encoding, and a particular crossover and mutation operators.

A. Reuse in Genetic Algorithms

From this perspective, it appears that there are many parts of genetic algorithms that are shared between different programs and that can be reused. Traditionally, the reuse was made at the code level; the code of the genetic algorithm was modified to incorporate changes. However, experience shows that this is complex, time consuming and error prone [4]. At the opposite, we propose the use of Software Product Line Engineering techniques [5]-[7], to efficiently manage the set of reusable assets and final software derivation. Our ultimate objective is to implement a software factory for the domain of genetic algorithms.

Product line engineering is a concept comprising methods, tools, and techniques for the development of product lines [8]. A software product line is the process aiming at designing and managing a set of related software products that target a specific domain and share some parts of their code. By selecting varying sets of assets, different products (a.k.a. variants), fulfilling different requirements of a specific application, can be generated.

The basis of SPLE is the explicit modeling of what is common and what differs between product variants [9]. Feature Models [10], [11] are frequently used at this aim.

In practice, the more the software products are represented in the SPL the more efficient the SPL will be. Empirical results reported in [5] show that the cost of developing multiple SPL based products is effective up to 3 to 4 products. This is especially the case of genetic algorithms.

Our approach uses the feature model to explicitly represent commonalities and variabilities. Features are bind to software components (java classes). The developer can then select and customize the desired features in the feature model to automatically derive a specific product.

We believe that the automatic derivation of code from the feature model and the existing set of components brings many advantages in terms of gain in time, costs and test and documentation management. Besides, our approach allows developers to benefit from existing technologies for verification and test on feature models [12], [13]

The rest of the article is organized as follows: Section II and III recalls some concepts that we use in our approach, namely, genetic algorithms and Software Product Lines. Section IV introduces our proposed approach for deriving genetic algorithms using SPL. Related works are reviewed in section V. Finally we conclude our work in section 6.

II. Genetic Algorithms

Genetic algorithms are a class of techniques that are very often used to solve optimization problems where the search space can be very large. They are inspired by biological evolution of chromosomes. This includes mutation, recombination, and selection [14]-[16]. The main idea behind genetic algorithms is to gradually evolve an initial set of (possibly random) solutions for an optimization problem, to

Manuscript received March 11, 2013; revised July 15, 2013.

A. Alidra is with the computer science department, 20th August 1955 University - Skikda, Algeria (e-mail: alidrandco@yahoo.fr).

M. T. Kimour is with LASE Laboratory, Badji Mokhtar-Annaba university, Algeria (e-mail: mtkimour@hotmail.fr).

newer ones in a way that their fitness is improved from generation to generation.

Fig. 1 represents the most relevant steps of a genetic algorithm. The next few sections give a brief description of them.

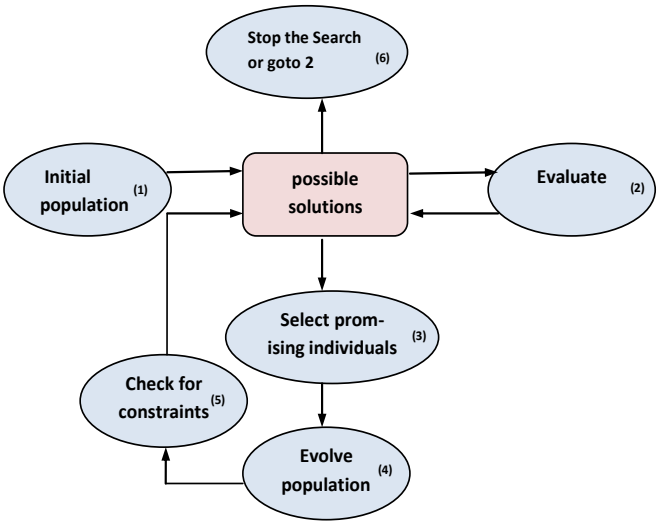

Fig. 1. The general steps of genetic algorithms

A. Solutions Encoding

Genetic algorithms are based on a population of solutions also called *individuals* or *phenotypes*. Each individual encodes a candidate solution in a *chromosome* (*genotype*). A common way to represent chromosomes is an array of bits. However other types and structures can be used, for example natural values may replace the bits or variable length of the array may be in use. The solutions encoding is an important issue in genetic algorithms because it influences other choices like the crossover and mutation operators. For example, a binary mutation operator is needed if the binary encoding is adopted.

B. Initial Population

The starting point of a genetic algorithm is a set of solutions that the algorithm will gradually evolve to better ones. This initial set of solutions is called *initial population*. Usually, the initial population is generated randomly; sometimes however, specific methods may be used to fasten the search by starting with promising areas.

C. Selection

In Genetic Algorithm a generation is a round of generating new chromosomes (referred to as children) from older chromosomes (referred to as parents). The choice of parent chromosomes is known as *selection* and is based on a *fitness* function that is used to evaluate the quality of the solutions. With this survival of the fittest strategy, Genetic Algorithms often converge to a (near) optimal solution for optimization problems.

Their exist in the literature many selection algorithms amount others: tournament selection, eletist selection, roulette-wheel selection.

D. Cross-Over

Genetic algorithms use crossover to exchange building blocks between two fit individuals, hopefully producing off-springs with higher fitness values. The most common forms of crossover are one-point, two-point crossover, "cut and splice,"Uniform Crossover and Half Uniform Crossover, Crossover for Ordered Chromosomes …

E. Mutation

Crossover aims at converging at (local) optima. In contrary, the role of mutation is to introduce genetic variation that may have been lost through-out the population as a result of the crossover operator [13]. Mutation takes an individual and randomly changes parts of its encoded solution based on some specified mutation rate. Mutation algorithms are uniform, boundary, flip…

F. Termination Criterion

Essentially, the process of generating new chromosomes based on the previous chromosomes is repeated until a specific condition is verified. This might be: an acceptable set of answers are developed, a maximum number of generations is reached, the maximum computing time is consumed.

When developing a new algorithm to solve a specific problem, developers often reuse the code that has been written in another project. For example, the code of the Uniform Crossover, the flip mutation and the tournament selection code is reused. Usually, an adaptation phase is needed then the code is merged into the final product. However, proceeding this way can be complex, time consuming and error prone because modifying the code manually introduces new errors. Besides inconsistencies may appear if changes are not properly propagated or incompatible techniques are used together. Finally, managing a large number of code fragments can quickly become problematic.

In order to overcome these limitations, we propose the use of product line engineering techniques in such a way that commonalities and variabilities are better managed and products are more efficiently generated.

III. Software Product Lines

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 developed from a common set of core assets in a prescribed way" [6] . A software product line enables developers to focus on issues that are specific to a single product rather than those that are common to all products. The benefits in terms of production and maintenance costs, time-to-market and flexibility to market changes are immediate and several experiences with SPL Engineering have been successfully conducted in various industry sectors (see for example [7] and [17])

In SPLE the development cycle is separated into two phases: domain and application engineering. The domain engineering phase is dedicated to the development of reusable artefacts, a.k.a core assets. Application engineering in the other hand refers to the phase where these core assets are used to derive a specific product by means of configuration. The configuration process is the activity that

allows developers to select artefacts that match the requirements of the desired product [13].

At the heart of SPLE is the management of reusable artefacts. Managing artefacts means rigorously documenting the variability in the SPL. This is often done by mean of a variability model. The feature model is one of the most popular variability models.

A software feature is "a distinguishing characteristic of a software item" [18]. A feature model is a hierarchically arranged set of features, and the relationships among them that determine the composition rules and the cross-tree constraints, and some additional information, such as trade-offs, rationale, and justifications for feature selection [19].

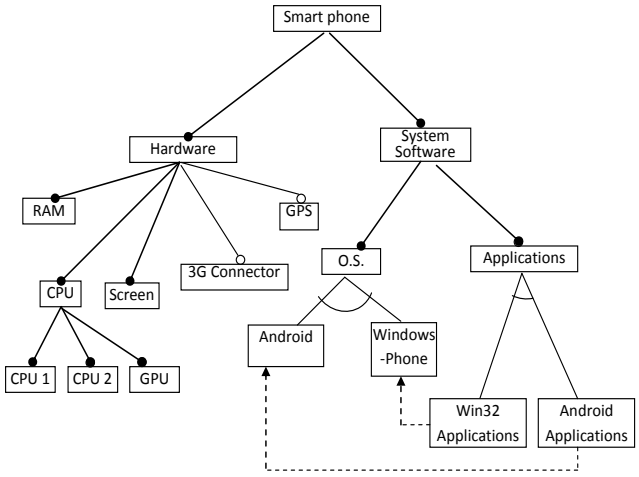

Fig. 2. A Simplified FM of Smart Phone Product Line

Fig. 2 depicts a possible (much simplified) feature model of an SPL in the domain of a smart-phone product family. This example is partially inspired by [12]. Features are hierarchically linked in a tree-like structure through variability relationships such as optional, mandatory, single-choice, and multiple-choice but also the cross-tree relations (i.e., requires, excludes, and cross-tree constraints). [12]

For instance, all smart-phones need to include the two components: hardware and software. Hardware including one or more processors, a screen, a RAM memory and sometimes a 3G connector and a GPS. Software would imply an OS whether Windows or Android and a set of applications that can be Win32 applications (requiring the Windows OS) or Android applications (requiring the Android OS).

For more details on FM notations semantic, reader can refer to [19]-[22].

The feature model is a simple way to model commonalities and variabilities and manage the reusable assets making the reuse more efficient. Besides, considerable effort was made on the feature model formal semantic [12], [13], [23] allowing automatic reasoning on the model. For example, authors in [24] proposed a technique to derive automatic test suites for Software Product Lines from a feature model.

IV. Using the Product Line

Reuse is an important mater in genetic algorithms. In order to improve reuse in the field of genetic algorithms we propose a better management policy of variabilities and commonalities of genetic algorithms based on the product lines engineering techniques. To this end, we will carry out the following steps:

1) building the feature model
2) mapping features to software components
3) deriving genetic algorithms from the product line

A. Building the Feature Model

Fig. 3 represents a partial feature model for the genetic algorithms software family that we have based ourselves on.

The feature model in Fig. 3 states that every genetic algorithm includes a solution encoding, an initialization algorithm, evolution mechanisms, a selection method, set of fitness functions, a termination criterion and sometimes a validation by a set of domain dependant constraints.

Moreover, the feature model shows the hierarchy of variation points for each feature. For example, solutions encoding is characterized by the encoding structure (fixed or variable length array) and the encoding values (bits or real values). Evolution can be either a cross-over, a mutation or some specific evolution mechanism, where cross-over is an exclusive choice between ordered chromosomes cross-over, a variable length chromosomes cross-over or classic cross-over. Mutation at the other hand is characterized by the mutation algorithm (bit string, flip, uniform…) and the mutation type (binary or real mutation) …

Finally, some cross-tree constraints are represented. For example:

- The solutions encoding determines the mutation type.
- Using an ordered encoding of solutions requires the use of ordered cross-over only.
- If more than one fitness function is used, a multi-objectives selection method is required.

B. Mapping Features to Software Components

One of the major advantages of SPLE is that it is model driven. This means that the developer will manipulate the feature model (typically, customize, select and deselect features) then, the SPL will automatically generate the corresponding software. To this end, we map the features in the feature model to software components which consist in java elements (interfaces and classes). We define the relationship between the features and java entities in such a way that the parent-child relationship in the feature model represents object oriented inheritance between classes.

Subsequently, the abstract features (intermediate nodes that represent commonalities, in blue in Fig. 3) are mapped to abstract classes that implement only the shared functionalities between sub-features. For example, the *cross-over* feature is mapped to an interface *Cross-over* that define one method *CrossOver* : *public chromosome [] CrossOver(chromosome [] Chromosomes)*

Concrete features (terminals nodes that represent variation points, in red in Fig. 3) are bound to concrete java classes that extend the parent classes with the specific functionalities. For example, the *uniform cross-over* feature will be mapped to the class *UniformCrossOver* which implements the interface *CrossOver*

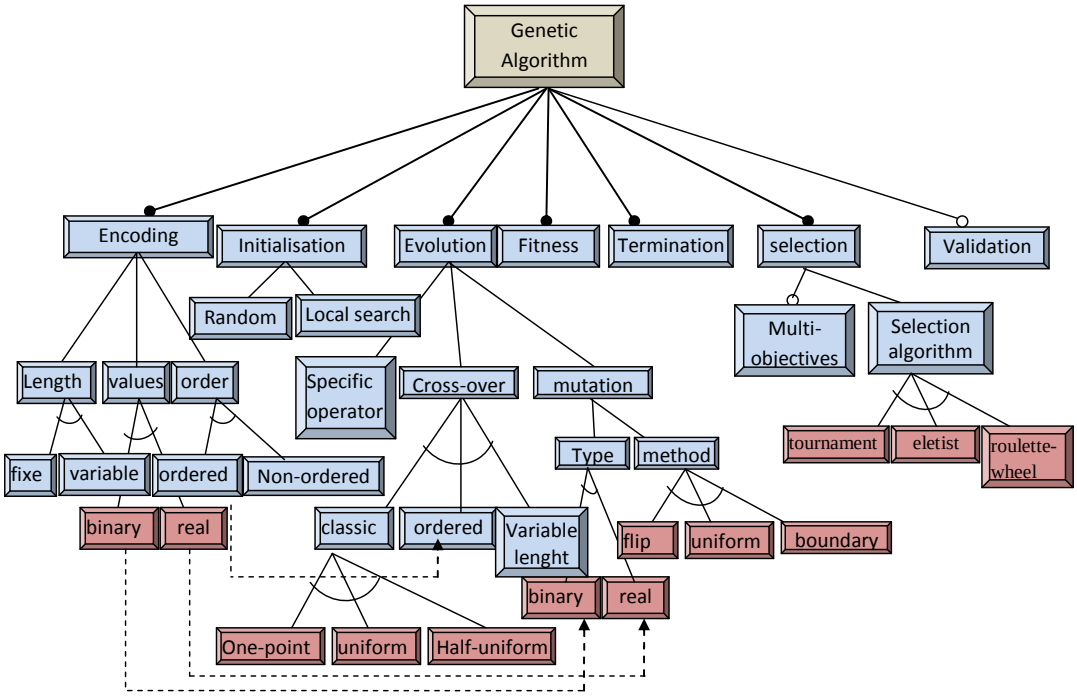

Fig. 3. A partial feature model for the genetic algorithm software family

C. Deriving Genetic Algorithms from the Product Line

The ultimate step in the process of deriving genetic algorithms from the software product line is the selection and configuration of features. The common architecture of genetic algorithms is as depicted in Fig. 1. Variation points are automatically inserted into code with respect to developer's choices. The mechanism of specialization is used to this end. The product line ensures the consistency of developer's choices by verifying cross-tree constraints before reverberating them on code. For example, if the uniform cross-over is selected, the SPL will add the line: *CrossOverObject = (UniformCrossOver) CrossOverObject;*

Then the uniform cross-over will be executed each time the crossover operator is called in the GA code.

V. Related Works

The problem of reuse has been under-studied in the context of genetic algorithms. In [25], authors proposed the use of design patterns to facilitate the reuse of genetic algorithms code. The pattern is decomposed in three packages: *GAPopulation* package that includes classes defining operators about colony, *GAGenome* Package that includes classes with operations on individuals and the *GAGeneticAlgorithm* which include classes that implement operations realizing the process of GA. In the same way, authors in [4] introduce a general purpose evolutionary computation framework. The aim of the framework is to maximize reuse between different evolutionary algorithms. To this end, the authors also proposed the use of design patterns. The framework is decomposed into three main classes: the *geneticAlgorithm* class, the *problem* class that provides the mechanisms defining the problem to be solved and the *plan* abstract class which represent the evolutive plan that will be used to solve the problem using GA. Unlike us the approaches in [25] and [4] manipulate the software at the code level, we propose at the contrary the manipulation of the software model. Moreover, the use of the feature model allows us to verify conceptual combinations of features (cross-tree constraints).

Ramirez et al [26] propose an aspect based approach to the implementation of evolutionary algorithms. The main contribution in [26] is the extraction of cross-cutting concerns into aspects that can be woven into the EC framework at compile time. The main benefits of using aspect programming are to guaranty the system consistency by properly propagating changes and enhancing modularity so that to facilitate maintenance and reuse of application code across different EC frameworks and approaches. Such approach can only manipulate products at the code level. In our approach, the feature model allows the manipulation of the system at the model level by graphically selecting and configuring the features. In the same way, modularity is enhanced thanks to the feature model hierarchical organization.

VI. Conclusion

Increasingly, reuse is important in software systems. In this article we established that genetic algorithms can benefit from recent advances in software reuse techniques because they are family of software that share a lot of common code and functionalities. Indeed, we presented a new approach, based on SPLE, to facilitate and fasten the development of genetic algorithms by the manipulation at the model level of a set of software artefacts. To this end we introduced the feature model of the genetic algorithms software family to represent commonalities and variabilities between genetic algorithms. The feature model also represents a set of cross-tree constraints that guaranty the validity of combinations of conceptual choices. We then suggested a

mapping of features to concrete software entities that implement the functionalities in the feature model. The code of the genetic algorithm is generated according to the selected and configured features.

We think that our approach will allow faster lifecycle for genetic algorithms. Beside, the use of SPLE will benefit from the model-oriented approach in terms of ease of use and verification.

Future works include the development of a workspace that supports our approach. Another important perspective is the evolution of the SPL to incorporate known hybridization algorithms. The implementation of the SPL may also be enhanced by the use of aspect oriented languages to improve the separation of concerns issue. Finally, we believe that our approach opens large perspectives to the use of Dynamic SPL techniques to manage the variability of GA at design time, as explained above, but also at run time to make the genetic algorithm adaptable during execution.

REFERENCES

[1] A. J. Ramirez, D. B. Knoester, B. H. C. Cheng, and P. K. McKinley, "Applying Genetic Algorithms to Decision Making in Autonomic Computing Systems," in *Proc. of ICAC'09*, June 15–19, 2009, Barcelona, Spain. ACM.

[2] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*, The MIT Press, December 1992.

[3] T. A. El-Mihoub, A. A. Hopgood, L. Nolle, and A. Battersby, *Hybrid Genetic Algorithms: A Review in Engineering Letters*, August 2006, vol. 13, no. 2, EL_13_2_11.

[4] N. Krasnogor and J. E. Smith, "MAFRA: A Java Memetic Algorithms Framework," in *Workshops Proc. the 2000 International Genetic and Evolutionary Computation Conference*, 2000.

[5] K. pohl, G. bockle, and F. J. V. D. Linden, *Software Product Line Engineering: Foundation, Principles and Techniques*, Springer_verlag 2005.

[6] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[7] A. Birk, G. Heller, I. John, K. Schmid, T. von der Masen, and K. Muller, "Product line engineering: The state of the practice," *IEEE Software*, vol. 20, no. 6, pp. 52–60, 2003.

[8] S. Apel and C. Kastner, "An overview of feature-oriented software development," *Journal of Object Technology*, vol. 8, no. 5, pp. 49–84, 2009.

[9] M. Dalgarno, "Software Product Line Engineering with Feature Models, Design of applications and programs," *Overload Journal* #78 – Apr. 2007.

[10] K. Kang *et al*., "Feature Oriented Domain Analysis Feasibility Study," Technical report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, 1990.

[11] K. Czarnecki and U. W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, 2000.

[12] C. E. A. Divo, "Automated Reasoning on Feature Models via Constraint Programming," M.S. thesis, Dept. Inf. Tech., Uppsala University Sweden, June 2011.

[13] A. Hubaux, "Feature-based Configuration: Collaborative, Dependable, and Controlled," Ph.D. thesis, Dept Comp.Sci., University of Namur, January 2012.

[14] K. D. Jong, "An analysis of the behavior of a class of genetic adaptive systems," Ph.D. Dissertation. Ann Arbor: The University of Michigan, 1975.

[15] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[16] J. H. Holland, *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge, MA, USA, 1992.

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

[18] IEEE Std 829-1998, *IEEE Standard for Software Test Documentation*, September 16th 1998.

[19] A. S. Karata, H. Oguztüzün, and A. Dogru. "Global Constraints on Feature Models," in *Proc. Principles and Practice of Constraint Programming - 16th International Conference*, Scotland, 2010, pp. 537-551.

[20] A. S. Karata¸ H. Oguztüzün, and A. Dogru. "Mapping Extended Feature Models to Constraint Logic Programming over Finite Domains," in *Proc. Software Product Lines: Going Beyond - 14th International Conference*, South Korea, 2010, Springer, vol. 6287, pp. 286-299.

[21] D. Benavides, P. Trinidad, and A. Ruiz-Cortes, "Automated reasoning on feature models," in *Proc. 17th Conference Advanced Information Systems Engineering*, Springer, 2005.

[22] D. Benavides, "On The Automated Analysis of Software Product Lines using Feature Models. A framework for developing automated tool support," Ph.D. dissertation, Sevilla, university, May 2007.

[23] D. Batory, "Feature Models, Grammars, and Propositional Formulas," in *Proc. the 9th Int. Conf. on SPLs*, 2005, pp. 7-20.

[24] F. Ensan, E. Bagheri, and D. Gasevic, "Evolutionary Search-Based Test Generation for Software Product Line Feature Models," *CAiSE 2012*, pp. 613-628.

[25] Z. Shi, L. Chao, and H. Ke-qing, "A software pattern of the Genetic Algorithm, A study on reusable object model of Genetic Algorithm," *Wuhan University Journal of Natural Sciences*, 2001, vol. 6, no. 1-2, pp. 209-217.

[26] A. J. Ramirez, A. C. Jensen, and B. H. C. Cheng, "An aspect-oriented approach for implementing evolutionary computation applications," *AOSD*, March 2011, pp. 153-164.

Abdelghani Alida was born in Algeria on June 26th, 1982. He is a teacher in the Computer Science Department in University of Skikda, Algeria. He has obtained his Magister Diploma in 2008 at University of Constantine. Now, he is inscribed to obtain his doctorate dilpoma in software architectures and data and knowledge engineering. Alidra's research interest is genetic algorithms, model driven engineering, reuse and online variability. In this subject he has published a paper in proceeding of ICSSE 2012 international conference.

Mohamed Tahar Kimour was born in Algeria on January 03, 1960. He is an associated professor in Computer Science Department at University of Annaba, Algeria. He has obtained his Doctorate Diploma in 2005 at University of Annaba. Mohamed Tahar's research interest is software model driven engineering and real time and embedded systems modeling. He has published many papers on real time and embedded systems modeling, Pr. Kimour is a head of project research on embedded systems engineering in Research Laboratory of Computer Science (LRI). He teaches courses in information systems, software engineering and organizational systems.