Vers une fabrique de logiciels pour les algorithmes génétiques

Abdelghani Alidra
Université 20 Août 55
Skikda, Algérie
alidrandco@yahoo.fr

Mohamed Tahar Kimour
Laboratore LASE, université de Badji Mokhta
-Annaba,Algérie
mtkimour@hotmail.fr

Résumé— **Les Lignes de produits logiciels (software product line ou SPL) sont un ensemble de procédés et de techniques qui permettent d'accroitre la productivité et la qualité des projets informatiques pour une famille de logiciels. Dans le présent article nous proposant une nouvelle approche basée sur les lignes de produits logiciels pour l'implémentation des algorithmes génétiques (AG). Notre approche est motivée par la grande diversité mais également la grande similitude entre des algorithmes génétiques destinés à résoudre des problèmes différents. Notre approche tire tout le parti des SPL et de l'ingénierie dirigée par les modèles. Par ailleurs, elle ouvre de nouvelles perspectives à l'exploitation des mécanismes des lignes de produits dynamiques (Dynamiques SPL) pour l'adaptation en cours d'exécution du comportement et des paramètres de l'algorithme génétique afin d'en accroitre la puissance.**

Mots Clés : algorithmes Génétiques, réutilisation, Ingénierie dirigée par les modèles, Lignes de produits logiciels.

I. INTRODUCTION

Les algorithmes génétiques sont des algorithmes d'optimisation stochastique fondés sur les mécanismes de la sélection naturelle et de la génétique [27]. Ils sont connus pour leur simplicité et leur efficacité à converger assez rapidement vers des solutions proches de l'optimum notamment quand ils sont combinés à d'autres techniques de recherche locale [18][3]. Ils sont aujourd'hui utilisés avec beaucoup de sucés dans divers disciplines tel que l'économie, le traitement de signal, l'aérodynamique, l'industrie automobile...

Les algorithmes génétiques sont une famille de programmes dans le sens où ils partagent de nombreux aspects (points communs) et diffèrent par d'autres. En effet, tous les algorithmes génétiques travaillent sur une population d'individus qui représentent des solutions potentielles, utilisent une (ou plusieurs) fonction Fitness pour évaluer la qualité des solutions et utilisent les opérateurs de croisement et de mutation pour faire évoluer les solutions vers des zones de recherche prometteuses. Mais, deux implémentations des algorithmes génétiques utiliseront des représentations différentes des solutions, des fonctions fitness spécifiques et des opérateurs de croisement et de mutations adaptés.

Il en ressort que les opportunités de réutilisation entre les algorithmes génétiques peuvent être très considérable. Ces opportunités ont été exploitées par le passé au niveau du code; le code de l'algorithme génétique été manipulé manuellement pour incorporer les modifications nécessaires. L'expérience montre toutefois que cette manière de procéder est non-productive par ce qu'elle est compliquée, couteuse et qu'elle augmente le risque d'introduire de nouvelles erreurs.[26].

Nous proposons au contraire d'opérer la réutilisation au niveau modèle grâce au paradigme de lignes de produits logiciels et des modèles des fonctionnalités (feature models ou FM) [7][11][12]. Notre approche permet d'exploiter les avantages des SPL et de l'ingénierie dirigée par les modèles en termes d'amélioration des coûts, de la réutilisation de code et de la qualité des programmes produits ainsi que l'exploitation des récents progrès dans le domaine du raisonnement automatique sur les features models.

Ainsi que nous l'expliquerons plus loin, une ligne de produits logiciels est le processus permettant de concevoir et de gérer un ensemble de produits logiciels "apparentés" i.e. partageant une partie de leur code et destiné à un fragment du marché bien spécifique. La sélection d'ensembles distincts de composants permettra la génération de produits distincts répondants aux besoins d'applications différentes. Des résultats empiriques rapportés dans [7] montrent que les lignes de produits sont d'autant plus efficaces que le nombre de produits est important. Ceci est notamment le cas des algorithmes génétiques ce qui justifie pleinement notre travail.

L'approche proposée utilise les features model pour représenter explicitement les points de variation entre les différents AG. Aux features nous associons des composants logiciels (classes Java). En sélectionnant et en configurant les features appropriées, le développeur générera automatiquement un produit particulier.

Le reste de cet article est organisé comme suit : les sections 2 et 3 rappellent les concepts de base traités par notre approche à savoir les algorithmes génétiques et les lignes de produits logiciels. La section 4 explique notre approche pour l'implémentation des AG en utilisant les SPL. Les travaux connexes sont exposés dans la section 5. Enfin, nous concluons notre travail dans la section 6.

II. LES ALGORITHMES GENETIQUES

Les algorithmes génétiques sont des méthodes méta-heuristiques pour la résolution de problèmes d'optimisation. Ils

s'inspirent du processus de la sélection naturelle dans un environnement défavorable selon la théorie de l'évolution proposée par C. Darwin. Selon cette théorie, les "meilleurs individus" tendent à vivre plus longtemps pour se reproduire et donner naissance à d'autres individus potentiellement de meilleure qualité alors que les plus faibles ont tendance à disparaître (survival of the fittest).

Par analogie avec l'évolution naturelle, des individus dans un AG représentent les solutions candidates du problème à résoudre. A Chaque individu est associée une valeur appelée fonction d'adaptation (fitness) qui mesure la qualité de la solution. Cette fonction reflète en fait la fonction à optimiser. L'algorithme génétique fait alors évoluer l'ensemble initial des solutions par les mécanismes de croisement et de mutation et une nouvelle population de solutions est produite pour la génération suivante. Ce processus est répété jusqu'à la vérification d'un critère d'arrêt [23][24][9][27], ainsi que le montre la figure 1.

Nous rappelons dans les paragraphes suivants les principales étapes des algorithmes génétiques.

A. REPRESENTATION DES SOLUTIONS

Les AG utilisent une représentation des solutions (parfois appelées phénotypes) d'un problème donné sous forme de chromosomes ou génotypes. Une manière commune de représenter les solutions est d'utiliser un tableau binaire simple. Toutefois d'autre codage sont possible tel que le codage de Gray, l'utilisation de tableaux à taille variable ou le codage par entiers ou par nombres réels. Le choix du codage est un paramètre décisif car il conditionne souvent l'utilisation des mécanismes d'évolution et le calcul des fonctions d'adaptation. Par exemple, si un codage binaire est adopté, l'opérateur de mutation doit être lui même de type binaire.

B. GENERATION DE LA POPULATION INITIALE:

Le point de départ d'un AG est un ensemble d'individus que l'algorithme fait évoluer génération après génération vers des individus plus adaptés. Cette ensemble d'individus appelé population initiale est généralement générer de façon aléatoire. Il existe toutefois d'autres méthodes de génération de la population initiale qui visent à débuter avec les régions les plus prometteuses afin d'accélérer la convergence à des solutions optimales.

C. LA SELECTION:

Dans les AG, la génération de nouveaux individus (appelés fils) s'opère par l'évolution d'individus dits parents. Le processus de choix des individus parents est appelé sélection, ce processus se base sur la valeur de la fonction d'adaptation (fitness). La force des AG provient en partie de cette politique de survie des meilleures solutions.

Il existe dans la littérature plusieurs algorithmes de sélection. Citons à titre d'exemple : la sélection par tournoi, la sélection élitiste, la roue de la fortune...

D. LE CROISEMENT

Le croisement consiste en la création de nouveaux chromosomes par l'échange de bloc d'informations entre deux parents appariés afin de produire des enfants potentiellement de meilleure qualité. Les algorithmes de croisement les plus courants sont le croisement en un point, le croisement en deux points, le croisement uniforme et semi-uniforme...

E. LA MUTATION

Le but de l'opérateur de croisement est de converger vers un optimum local. Au contraire, le rôle de l'opérateur de mutation est d'introduire des variations génétiques afin d'empêcher l'AG de stagner dans un optimum local. La mutation est exécutée sur un seul chromosome. Elle représente l'altération aléatoire et occasionnelle de la valeur d'un gène de l'individu. Plusieurs algorithmes de mutation existent, citons par d'exemple ; la mutation uniforme, limitée, flip…

F. LE CRITERE D'ARRET

Le processus de génération de nouveaux individus se poursuit jusqu'à la satisfaction d'une certaine condition appelée critère d'arrêt. Ce critère d'arrêt peut être exprimé en nombre d'itérations effectué par l'algorithme génétique, le temps alloué au calcul des solutions optimales, la génération d'un ensemble de solutions d'une qualité acceptable...

Lors de l'implémentation d'un algorithme génétique pour la résolution d'un problème spécifique, les développeurs réutilisent souvent des blocs de code écrit pour un autre AG dans le cadre d'un autre projet. Par exemple, le code de l'opérateur de croisement en deux points, de la mutation uniforme et de la sélection élitiste est réutilisé. Afin de permettre cette réutilisation, une phase d'adaptation du code est souvent nécessaire afin de prendre en considération par exemple, le changement de la codification des solutions, de la longueur des chromosomes ou des contraintes de validité de ceux-ci.

Toutefois, cette démarche est souvent fastidieuse, couteuse en temps et en ressources et augmente le risque d'introduire de nouveaux bug dans le code. De plus des inconsistances peuvent se glisser si les modifications apportées à un module ne sont pas correctement propagées au reste de l'application. Enfin la gestion d'un grand nombre de fragment code peut vite se révéler problématique. [28]

Afin de répondre à ces préoccupations, nous proposons l'utilisation des lignes de produits logiciels pour une meilleurs gestion des points communs et des points de variation dans la famille des algorithmes génétiques. Cette approche permettra une réutilisation plus efficace du code partagé entre les AG.

III. LES LIGNES DE PRODUITS LOGICIELS

Une ligne de produits logiciels est un ensemble de techniques, procédés et outils qui permettent de générer des programmes partageant certaines fonctionnalités et répondant aux besoins spécifiques d'un segment de marché et développé à partir d'un ensemble commun de composants logiciels suivant une démarche bien établie. Une SPL permet donc aux développeurs de se focaliser sur les aspects spécifiques d'un

seul produit au lieu de se préoccuper de ceux communs à tous. Les bénéfices en termes de coûts de production, de temps vers le marché et de réactivité aux fluctuations du marché sont immédiats. Ainsi différentes expériences basés sur les Lignes de produits logiciels ont étaient conduites avec succès dans divers secteurs industriels (voire par exemple [12] et [13])

Le concept central de l'ingénierie des lignes de produits logiciels (Software Product Lines Engineering ou SPLE) est la représentation explicite des aspects communs et des aspects singuliers des produits (appelés aussi points de variation). En ingénierie des SPL, le cycle de développement est séparé en deux phases : l'ingénierie du domaine et l'ingénierie d'application. La première phase est dédiée à l'analyse du domaine et au développement des composants de base. La seconde vise à dérivé un produit spécifique par la sélection et la configuration des composants de base appropriés.

Afin de permettre une gestion efficace des composants de base de la ligne de produit, les modèles de variabilité ont été introduits. Le modèle de variabilité le plus commun est le modèle de fonctionnalités (Feature model ou FM) que nous présentons plus loin.

Une fonctionnalité logicielle (ou feature) est une caractéristique distinguant une entité logicielle [14]. Un FM est un ensemble hiérarchisé de fonctionnalités et la relation entre elles qui déterminent les règles et les contraintes de compositions ainsi que des informations supplémentaires tel que les recommandations ou la justification de sélection de features[15].

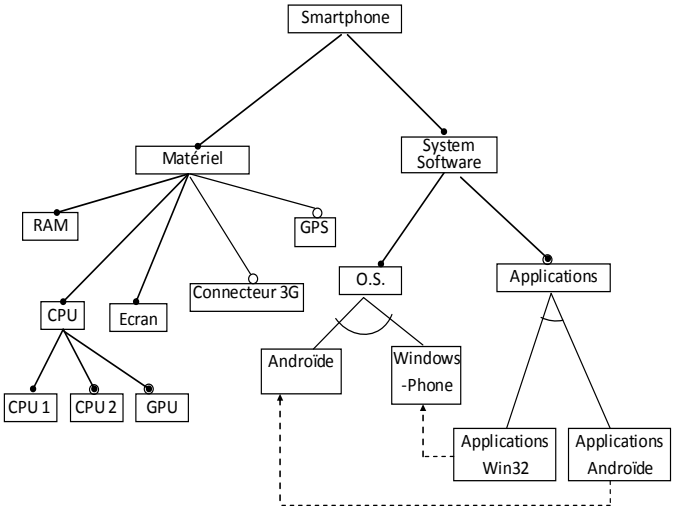

Fig.2. Un exemple de modèle de fonctionnalités (Feature model)

La figure 2 représente le FM (simplifié) d'une SPL dans le domaine des smart-phones. Cet exemple, inspiré de [8], montre les différentes relations de variabilité tel que : optionnel, indispensable, choix exclusif ou multiple ainsi que des contraintes de sélection tel que l'implication ou l'exclusion entre fonctionnalités. [8]

Ainsi par exemple, tous les Smartphones partagent les deux catégories de composants : logiciels et matériels. Le matériel inclut un ou plusieurs processeurs, un écran, une RAM et parfois une antenne 3G et un GPS. Le logiciel implique un système d'exploitation qui est un choix exclusif entre Windows et Androïde et un ensemble d'applications, qui peuvent être soit des applications Win32 requérant le système Windows-phone soit des applications Androïde requérant le système Androïde.

Pour plus de détails concernant les FM, le lecteur peut se référé à [15,16,19, 20]

Le Feature model est un modèle simple qui permet de modéliser les points communs et les points de variations ainsi qu'une bonne gestion des composants logiciels réutilisables rendant la réutilisation plus efficace. En plus, un effort considérable a été fait sur la sémantique formelle des FM autorisant le raisonnement automatique sur le modèle tel que la vérifications de la validité des configurations ou la génération de suites de test [8,10,21,22]

IV. UTILISATION D'UNE LIGNE DE PRODUIT POUR LES ALGORITHMES GENTIQUES

Ainsi que nous l'avons montré plus haut, la réutilisation est un aspect important dans l'implémentation des AG. Dans le but d'en améliorer l'efficacité, nous proposons une politique de gestion des points communs et variables en utilisant l'ingénierie des lignes de produits logiciels. Dans cette perspective, nous allons procéder comme suit :

1. Construction du model de fonctionnalités (feature model)

2. Mise en correspondance des fonctionnalités avec les composants logiciels

3. Dérivation des algorithmes génétiques à partir de la ligne de produits.

A. CONSTRUCTION DU MODELE DE FONCTIONNALITES

La figure 3 représente partiellement le FM que nous avons proposé pour le domaine des algorithmes génétiques. La figure montre ainsi que tous les algorithmes génétiques incluent une représentation des solutions sous forme de chromosomes, un algorithme d'initialisation, un ensemble de mécanismes de d'évolution, une méthode de sélection, un ensemble de fonction de fitness, un critère d'arrêt et parfois une méthode de validation par un ensemble de contraintes spécifiques.

D'autre part, le feature model montre la hiérarchie des points de variations pour chaque fonctionnalité. Par exemple, l'encodage des solutions est caractérisé par la structure d'encodage (tableau de taille fixe ou variable), et les valeurs des gènes (valeurs binaires ou réels). L'évolution s'opère soit par l'opérateur de croisement soit par celui de la mutation. Le croisement est un choix exclusif entre les opérateurs de croisement des chromosomes ordonnés, des chromosomes à

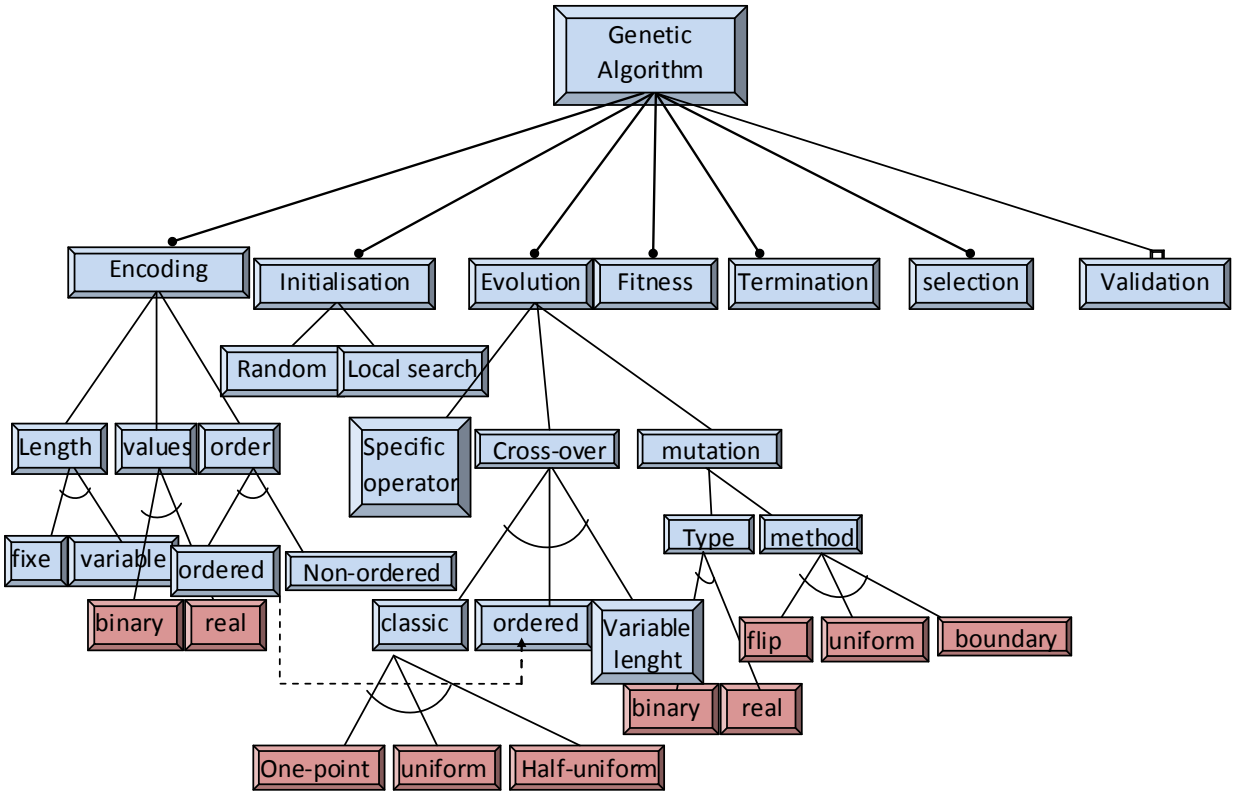

Fig.3 Le modèle des fonctionnalités de la famille des algorithmes génétiques

taille variable ou le croisement classique. La mutation quant à elle est caractérisée par l'algorithme de mutation (uniforme, flip...) et par le type de mutation (binaire ou réelle)...

Enfin des contraintes sur les configurations valides sont représentées, ainsi par exemple:

- le type d'encodage des individus détermine le type de mutation.
- L'utilisation d'une représentation ordonnée nécessite l'utilisation d'un opérateur de croisement ordonné.
- Si plus d'une fonction de fitness est utilisée, une méthode de sélection multi-objectifs est nécessaire.
- ...

B. ASSOCIATION DE COMPOSANTS LOGICIELS AUX FEATURES DANS LE FM

L'utilisation des SPL et des FM apporte l'avantage de manipuler un modèle (typiquement personnaliser, sélectionner et dé-sélectionner des features) en minimisant le besoin de modifier le code de l'application. Afin de permettre à la ligne de produit de généré le code correspondant à la configuration du développeur, nous proposons d'associer des composants logiciels (des classes Java) aux différentes features représentées dans le FM.

Nous définissons à cet effet une relation d'héritage entre les classes Java du dépôt de la ligne de produits. La relation d'héritage est basée sur la relation parent-fils dans le feature model. Ainsi, une feature abstraite représentée par un nœud intermédiaire (boite bleue dans la figure 3) est associée à une classe « abstraite » qui implémente uniquement les fonctionnalités partagées entre les sous-features.

Par exemple, la fonctionalité Cross-over est associée à une classe de même nom qui définie une méthode unique public chromosome[] DoCrosseOver(chromosome [] Chromosomes)

D'autre part, les features concrètes représentée par un noeud terminal ou feuille (boite rouge dans la figure 3) sont associées à des classes Java concrètes qui étendent les classes correspondants à leurs features parents avec des fonctionnalités spécifiques. Ainsi par exemple, la feature uniform-CrossOver est associée à la classe UniformCrossOver qui étend la classe Cross-over par l'ajout du code spécifique de l'opérateur de croisement uniforme à la méthode DoCrosseOver

C. DERIVATION D'ALGORITHMES GENETIQUES A PARTIR DE LA LIGNE DE PRODUITS

Afin de dériver une implémentation de l'algorithme génétique de la ligne de produit, le développeur doit procéder à la sélection et à la configuration des features.

La ligne de produit doit d'abord assurer l'absence d'inconsistances dans la configuration du développeur en respect des contraintes exprimées par le feature model. Par exemple, si la fonctionnalité du codage binaire est sélectionnée les opérateurs de mutation non binaires seront automatiquement désactivés.

Le code généré pour l'algorithme génétique est basée sur une architecture d'exécution commune tel que décrit dans la figure 1. Les points de variations sont insérer dans le code commun en respect des choix du développeur. A cet effet, nous exploitons le mécanisme orienté-objet de spécialisation. Ainsi,

dans le code commun à tous les algorithmes génétiques figurent des appels à l'opérateur génétique de croisement via un objet CrossOverObject qui est une instance de la classe "abstraite" CrossOver". Si le développeur choisi par exemple d'adopter le croisement uniforme, il suffit de spécialiser l'objet CrossOverObject invoqué dans l'algorithme génétique sur la sous classe UniformCrossOver par la ligne : CrossOverObject = (UniformCrossOver) CrossOverObject;

Toute invocation dans la suite du code à la méthode DoCrosseOver correspondra à l'exécution du code du croisement uniforme.

V. TRAVAUX CONNEXES

Le problème de la réutilisation a été sous-étudié dans le contexte des algorithmes génétiques. Dans [25], les auteurs ont proposé l'utilisation des patrons de conception (ou design patterns) afin de faciliter la réutilisation du code des algorithmes génétiques. Ils décomposent leur pattern en Trois packages : le package GAGenome qui inclut les classes avec les opérations sur les individus, le package GAPopulation qui inclut les classes avec les opérations de manipulation de la population et le package GAGeneticAlgorithm qui inclut les classes implémentant le processus de l'AG.

De la même manière les auteurs dans [26] ont proposé un Framework pour les algorithmes génétiques. Le but du Framework est de maximiser la réutilisation entre différents algorithmes évolutionnaires. Ils basent également leur travail sur l'utilisation des designs patterns. Le Framework est décomposé en trois classes principales : la classe geneticAlgorithm, la classe problème qui fournit les mécanismes de description du problème à résoudre et la classe abstraite plan qui représente le plan évolutionnaire qui servira à la résolution du problème à traité. Contrairement à nous, les approches dans [25] et [26] manipulent le programme au niveau code, nous proposons au contraire de manipuler un modèle du programme. Par ailleurs, l'utilisation des features model permet d'exprimer et de vérifier les combinaisons conceptuelles entre les fonctionnalités.

Ramirez et al [17] proposent une approche basée sur les aspects pour l'implémentation des algorithmes évolutionnaires. La principale contribution dans [17] est l'extraction automatique de préoccupations transverses (cross-cutting concerns) en aspects qui seront tissé pour aboutir à un Framework des algorithmes évolutionnaires au moment de la compilation. Les avantages de l'utilisation de la programmation par aspects est de garantir la consistance du système en propageant convenablement les changements, et de renforcer la modularité afin de facilité la réutilisation et la maintenance du code des applications. Toutefois, cette approche permet de manipuler les produits logiciels au niveau code. Notre approche en revanche se base sur un modèle du système par la sélection et la configuration graphique des différentes fonctionnalités. Dans notre cas, la modularité est renforcée grâce à l'organisation hiérarchique du feature model, et la consistance grâce aux contraintes du modèle.

VI. CONCLUSION

De plus en plus, la réutilisation est importante dans le monde des systèmes informatiques. Dans cet article nous avons montré que les algorithmes génétiques peuvent bénéficier des récents progrès dans les techniques de réutilisation de logiciels par ce qu'ils constituent une famille de programmes qui partagent de nombreuses fonctionnalités. A cet effet, Nous avons présenté une nouvelle approche basée sur les lignes de produits logiciels afin de faciliter et d'accélérer le développement des algorithmes génétiques par la manipulation au niveau model d'un ensemble commun de composants. Nous avons construit le modèle des fonctionnalités (Feature model) de la famille des algorithmes génétiques. Le feature model représente les points communs et les points de variations entre les différents algorithmes génétiques. Il inclut également un ensemble de contraintes de conceptions qui garantissent la validité des combinaisons conceptuelles entre fonctionnalités. Nous avons également proposé une correspondance entre les fonctionnalités du FM et les entités logicielles qui les implémentent. Le code de l'algorithme génétique est généré en respect des fonctionnalités sélectionnées dans le FM.

Nous croyons que notre approche permettra d'accélérer le cycle de développement des AG. En plus l'utilisation des SPL permet de tirer profit des aspects de validation et de raisonnement automatique offerts par l'ingénierie dirigée par les modèles.

Les travaux futurs incluent le développement d'un environnement de développement qui supporte l'approche proposée. Une autre perspective importante est l'extension de la SPL pour incorporer les algorithmes génétiques hybrides. L'implémentation de la ligne de produit pourrai également bénéficier de l'utilisation des langages de programmation orientés aspect afin de renforcer la séparation des préoccupations transverses. Enfin, nous croyons que notre approche ouvre de larges perspectives à l'adoption des techniques de lignes de produits logiciels dynamiques (ou Dynamiques SPL) afin de permettre non seulement d'adapter le logiciel à la conception, tel que expliqué plus haut, mais également à l'exécution en rendant l'algorithme génétique automatiquement adaptable.

REFERENCES

[1] Andres J. Ramirez, David B. Knoester, Betty H.C. Cheng, Philip K. McKinley. Applying Genetic Algorithms to Decision Making in Autonomic Computing Systems, ICAC'09, June 15–19, 2009, Barcelona, Spain. ACM

[2] S. Apel and C. K¨astner. An overview of feature-oriented software development. Journal of Object Technology (JOT), 8(5):49–84, 2009.

[3] Tarek A. El-Mihoub, Adrian A. Hopgood, Lars Nolle, Alan Battersby. Hybrid Genetic Algorithms: A Review in Engineering Letters, 13:2, EL_13_2_11. August 2006

[4] Mark Dalgarno. Software Product Line Engineering with Feature Models, Design of applications and programs, Overload Journal #78 - Apr 2007

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

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

[7] klaus pohl, gruner bockle, and frank j. van der linden Software product line engineering : foundation, principles and techniques. Springer_verlag 2005

[8] Carlos Eduardo Alvarez Divo, Automated Reasoning on Feature Models via Constraint Programming, master thesis., June 2011

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

[10] Arnaud Hubaux. Feature-based Configuration: Collaborative, Dependable, and Controlled. Phd thesis, University of Namur, January 2012.

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

[12] 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, 20(6):52–60, 2003.

[13] F. van der 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.

[14] IEEE Std 829-1998. "IEEE Standard for Software Test Documentation". September 16th 1998.

[15] A. S. Karata¸s, H. Oguztüzün, and A. Dogru. "Global Constraints on Feature Models". Proceedings of Principles and Practice of Constraint Programming - 16th International Conference (CP-2010), Scotland 2010. Springer, vol. 6308, pp. 537-551. ISBN 9783642153952

[16] A. S. Karata¸s, H. Oguztüzün, and A. Dogru. "Mapping Extended Feature Models to Constraint Logic Programming over Finite Domains". Proceedings of Software Product Lines: Going Beyond - 14th International Conference, (SPLC-2010), South Korea 2010. Springer, vol. 6287, pp. 286-299. ISBN 9783642155789.

[17] Andres J. Ramirez, Adam C. Jensen, Betty H. C. Cheng: An aspect-oriented approach for implementing evolutionary computation applications. AOSD 2011: 153-164

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

[19] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortes. Automated reasoning on feature models. In 17TH Conference Advanced Information Systems Engineering, Springer, 2005.

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

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

[22] Faezeh Ensan, Ebrahim Bagheri, Dragan Gasevic: Evolutionary Search-Based Test Generation for Software Product Line Feature Models. CAiSE 2012: 613-628

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

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

[25] Zhuo Shi, Liu Chao and He 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, Volume 6, Numbers 1-2, Pages 209-217

[26] N. Krasnogor and J.E. Smith. MAFRA: A Java Memetic Algorithms Framework. In Workshops Proceedings of the 2000 International Genetic and Evolutionary Computation Conference (GECCO2000), 2000.

[27] Mme DRDI Leila, thèse de doctorat, INRS 2005

[28] Frédéric Esnault, L'usine de développement :du cauchemar à la réalité, génie logiciel numéro