A Java-Based Code Generator for Parallel Evolutionary Algorithms

Jackson Amaral da Silva
Computer Engineering Department
Universidade Estadual do Maranhão
São Luis – MA – Brazil
jackson.amarals@gmail.com

Omar A. Carmona Cortes, Josenildo Costa da Silva
Informatics Academic Department
Instituto Federal do Maranhão
São Luis – MA – Brazil
{omar, jcsilva}@ifma.edu.br

***Abstract*—Evolutionary Algorithms (EAs) are able to find out solutions in many fields and complex disciplines. Parallel Evolutionary Algorithms (PEAs) solve many kinds of problems, as well; moreover it overcomes problems with run time constraints when the problems being solved are much more complex. Thereby, we can state that PEAs can be efficient and faster than a regular EA. On the other hand, parallel programming brings new substantial problems to the developers that have to deal with synchronization and a different debugging process, increasing the learning curve and the programming efforts. In this context, we developed a web-based software that automatically creates java code for parallel genetic algorithms and parallel evolutionary strategies, reducing the required time to develop this kind of application. Two parallel models were implemented, the island model and master-slave. Further, the design patterns strategy and observer were applied in the generation of the PEAs code in order to increase the software maintainability and legibility. An experiment was conducted in order to show how our software can reduce the time consuming of an EA.**

Keywords—Parallel Evolutionary Algorithms, Automatic Code Creation, Design Patterns.

I. Introduction

Evolutionary Algorithms (EAs) are search algorithms based on natural evolution [1], which can find out good solutions in a reasonably amount of time. However, as they solve harder problems there is an increase in the running time in order to find adequate solutions. As a consequence, there have been multiple efforts to make EAs faster, and one of the most promising choices is to use parallel implementations [2].

The concept of Parallel Evolutionary Algorithms (PEA) is quite simple. Despite this, the use of parallel computing demands challenges such as the synchronization between demes and the proper exploration of parallel algorithms. Furthermore, the lack of experience in developing parallel applications might impact directly in the productivity of parallel evolutionary applications.

In order to overcome the problems raised by parallel computing, Jaquie [3], Malacarne [4], Passini & Dotti [5] and Rodrigues et. al. [6] present some tools that produce parallel code automatically. However, users have to implement their specific code. In this sense, Hawick & Playne [7] create an approach for generating parallel code to solve partial differential equations (PDEs); Linford et. al. [8] present a general analysis and code generation tool that achieves significantly reduced time-to-solution for chemical kinetics kernels on three multicore platforms. In Cahon et. al. [9] is presented the ParadisEO, which is a framework that provides a generic set of classes for implementing parallel and distributed metaheuristics. Further, ParadisEO offers different models of parallelization using MPI, PVM and PThreads. Inspite of this, the user has to implement the specific classes of his application and learn how to use those classes properly. To the best of our knowledge, there is no work that automatically produces parallel code for evolutionary algorithms.

In this context, this work presents the Java Parallel Evolutionary Algorithms Generator (JPEAG), which is a template-based parallel code generator that produces parallel evolutionary algorithms in Java, where users have to implement only the evaluation function because it is specific for each application. Nonetheless, this function can be inserted by means of the graphical interface. Thus, the code is ready to be compiled and executed. Currently, JPEAG can create code for parallel genetic algorithms and parallel evolutionary strategies, using real-coded individuals. However, extensions for generating other kinds of PEA can be easily implemented into the software.

The remaining of this paper is divided as following: Section II shows the basic concepts of evolutionary algorithms and parallel evolutionary algorithms; Section III presents briefly the theory about code generation; Section IV illustrates how the design patterns was used and applied into the codification of PEAs; Section V deals with the development of the JPEAG; finally, Section VI presents the conclusion of this work.

II. Evolutionary Algorithms

Basically, evolutionary algorithms try to find out the best solution for a specific problem processing a population of individuals, where each one represents a possible solution. The population has to undergo genetic operators on many iterations until reach some stop criteria as follows: (i) certain amount of iterations, (ii) there is no more evolution, or (iii) the algorithm gets to the known optimal. Fig. 1 shows the structure of an EA.

In the first step, the population is initialized at random, normally using auniform distribution. Then the population is evaluated in order to determine how each individual fits to the solution. The better is the fitness, the stronger is the individual within the population, so might be higher the probability of an individual to be selected for genetic operators or to go to the next iteration (generation), that depends on which EA is running on.

```
Initialization;
Evaluation;
Whilestop criteria is not reached
Selection;
Genetic Operators;
Evaluation;
End-While;
Returnbest solution or individual;
```

Fig. 1. Structure of an EA.

Typically, the genetic operators are selection, crossover and mutation. Selection is the process of choosing individual to undergo to genetic operators or to go to next generation. Parents exchange information (genes) between themselves in order to create one or more offspring, in the crossover operator. Ideally, when two strong individuals exchange genes the offspring is stronger than its parents [10] and then, spread its genes in next generations. On the other hand, this behavior can lead to a premature convergence of the solution because the population can be trap into local optima. The mutation operator has the purpose of avoid the premature convergence applying some modifications to one or more genes. In other words, the process of using genetic operators tends to improve the solution quality as new generations carry on [11].

Taking this operators into account, we can notice that several EA share similar features. For instance, genetic algorithms and evolutionary strategies may use all those genetic operations. However, the sequence that these operators are used is different. Evolutionary strategy applies first the genetic operators then it uses the selection operators, whereas genetic algorithms select the individuals and thereafter perform the genetic operators.Moreover, evolutionary strategies need two vectors for representing an individual instead of only one of genetic algorithms. Details about theseEAscan be seen in Michalewicz [11], Cortes [12] and Herrara [10].

A. Parallel Evolutionary Algorithms (PEAs)

The main idea behind the parallel computing is to divide a task into smaller pieces and solve those using different processing units. In this context, EAs can be considered because they have an intrinsic parallelism [13]. There are many ways to explore their parallelism such as: find out different solutions for the same problem, explore several points in the search space at the same time, and reduce the computation time to get a solution [14].

According to Cantu-Paz [2] there are three basic models to explore the parallelism of EAs: master-slave, cell and island. Combining these models we can obtain a hybrid model as well. Different names might be used for these models, however their characteristics remain the same. In this work we implemented the master-slave and the island model.

Regarding the master-slave model, a PEA maintains a population in a master processor that delegates the function evaluation or the applications of the genetic operators to slaves. Commonly, the parallelization is done distributing the function evaluation to slaves [2]. In the cell model, all processors works on the same population, where each individual is put into a grid, so the genetic operators can be done only with their neighborhood in the grid. Finally, independent populations are processed at the same time in the island model, introducing the concept of migration, where one or more individuals can be frequently exchanged between the populations. This last model also introduces new parameters such as the number of individuals being exchanged, how frequent the migration has to be done and the island topology which represents how islands can communicate with other ones. Researches indicate that the proper migration process contributes in the population diversity and enhance the quality of solutions [2]. Regardless the model being considered, all communication between processor can be synchronous or asynchronous. In the first one, if a processor wants to communicate, it has to wait until all of involved processors be ready. On the other hand, in the asynchronous communication, the execution and the communication do not depend on the other processors, i.e., if a processor wants to communicate with another one it send the information and continues with its execution.

III. Automatic Code Generation

Code generation can be defined as a technique in which we write programs that create another programs. According to Herrington [15], creating code presents many benefits such as:

- Agile software development – code generators go toward completion faster than a hand-made code, reducing the cost of development as well.
- Consistency – code generators maintain the standard in both design patterns and code conventions, avoiding breaches introduced by programmers.
- One point for gathering knowledge – a change made in a definition file can be propagated to all files created previously, whereas programmers have to do this file by file in a hand-made coding

There are some strategies for coding generation. We are focus on templates which represents a pre-defined piece of software, i.e., an unfinished software that may be completed using variables [16]. In other words, a software replaces some elements present in the template file [17]. The substitution has to be performed by a template processor considering a set of inputs.

The approach based on templates was chosen because a significant amount of code for GAs and ESs are similar. Moreover, we noticed the same characteristics in the parallel models, especially in the island and master-slave models.Thus, when these similarities were identified we could write the templates containing the common parts. Besides, the template approach allows to modify any part of the generated code changing only the proper template, becoming easier the software maintainability.

IV. Design Patterns in PEAs

Regardless which EA are being implemented, to find out the optimal configuration for its parameters is not a trivial task due to the huge range of possible combinations between parameters [18]. Thereby, the code of PEAs have to be flexible, extensible and readable, allowing changes in a stress-free way. The use of design patterns can guarantee these features.

Design patterns are general solutions to well-known problems in object oriented programming. All solutions have been tested and are considered elegant by software designers [18]. Further, design patterns bring benefits such as: modularization, low coupling and state change awareness, besides, it does not add significant complexity to the code. In this context, we applied two design patterns for generating PEA code: **strategy** and **observer**.

The main reason of using strategy pattern is to provide modularization to the genetic operators and to the stop criteria, being easy to add new features and reuse the available operators in new EAs. The observer pattern was used because it offers a simple and elegant manner of synchronizing the communication between threads.

The strategy pattern defines a group of algorithms encapsulating them by means of interfaces, allowing variations regardless it uses in the clients. This pattern is used, for example, to hide the implementation of genetic operators and to assure that any changes in the operators does not affect other parts of the code [19]. Further, this patterns permits that the genetic operators could be used in other kinds of EAs, for instance, in the future producing of parallel hybrid algorithms. The stop criteria was implemented using the strategy pattern in this work, as well.

The observer pattern defines a *1-to-n* relationship between objects. This relationship consists of one object being observed by many other ones with low coupling. When an observed object has its status modified all observers receive a notification being automatically updated [19]. This pattern was used for synchronization purposes between the objects. Being specific, a process receives information about the other processes, so it can decide whether await for the other processes, in the case of migration for example, or carry out with its own execution.

V. The Java-based Code Generator

Even though the idea behind PEAs is simple, its implementation is not a trivial task because involves challenges from parallel computing such as synchronization problems, the proper uses of parallel resources/models and a hard procedure of debugging. So, the lack of experience in parallel computing might significantly reduce the productivity in the development of PEAs code. Moreover, parallel computing programming might demands the learning of a new frameworks, a new library or even a new language.

Alternatively, tools for generating parallel code of general purpose can be used, however the specific code for the EA has to be implemented anyway. In this context, a specific tool for creating parallel code for EAs called JPEAG is proposed. Currently, two parallel evolutionary algorithms are available: genetic algorithms (GA) and evolutionary strategies (ES), both using a real-coded representation. Table I shows the available operators for each one. The stop criteria are the same for both EAs, i.e., certain number of generations. Besides, the user can choose if the problem to be solved is a maximization or a minimization one.

TABLE I. Available EAs and their genetic operations

EA	**Genetic operators**	**Methods**
Genetic Algorithm	Selection	Roulette Tournament
	Crossover	One point Heuristic Linear
	Mutation	Uniform or Random Creep

Evolutionary Strategy	Selection	ES-(μ,λ)
		ES-(μ+λ)

	Crossover	Intermediary
	Mutation	Gaussian

A. JPEAG: Requirements

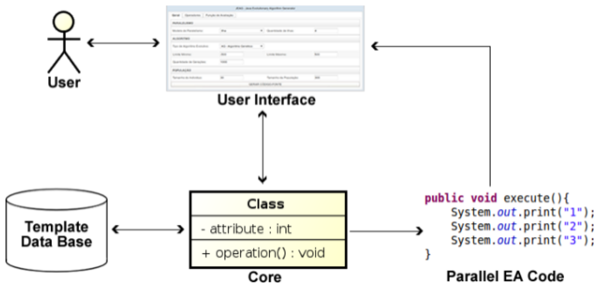
The JPEAG is a graphic web-based application developed in Java that follows the MVC pattern. It provides a graphic interface to create code for PEAs also in Java. Almost the entire for the AEP code is created with this application, excepting the evaluation function. Nevertheless, the function can be easily inserted in the application producing a complete code ready to be compiled and executed. The application has developed obeying the following requirements:

- Object oriented – nowadays the object oriented concepts are important, especially in order to apply pattern designs into the code.
- User friendly – a user friendly application is easy to learn and might increase the code productivity.
- Hiding the complexity of the parallel computing – users with no expertise in parallel computing are able to create PEAs, because the application hides the aspects involved in parallel models.
- Code flexibility – The JPEAG is able to generate the entire code for a PEAs in Java, saving it in a text file that users can easily modify, adapt and extend.
- Using of design patterns – applying design patterns we guarantee that tested solutions are being used. Besides, the PEA code becomes extensible and readable, improving its quality.

B. The JPEAG architecture

The JPEAG architecture is presented in Fig. 2. Its operation is described as following: the interface is a web-based interface where users configure all features of the PEAs, including the evaluation function. When all parameters are set the user send it to the core. The core is responsible for selecting the proper template in the templates data base and use it to create the parallel EA code. Then, the parallel code is packed in a zip file and send it back to the user via download because is a web-based application.

Fig. 3 shows some details about the classes that compose the core of the JPEAG. The *AEConfiguration* class has a set of attributes that store the configuration of a PEA inserted by the user in the graphic interface. This process is done by the ZK framework [20] by means of AJAX requisitions.



```
public void execute(){
    System.out.print("1");
    System.out.print("2");
    System.out.print("3");
}
```

Fig. 2. JPEAGarchitecture

The *TemplateProcessor* class is responsible for processing the templates and creating the parallel code. The templates are processed by a framework called Apache Velocity [21] which implements an engine for template processing, and defines a language (VTL – Velocity Template Language) for creating it. The main advantage of the Apache Velocity is to have methods for processing templates and creating code for any textual language.

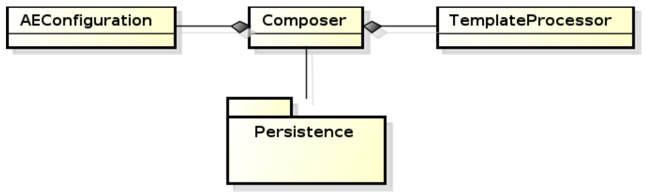


Fig. 3. Main classes of JPEAG

The templates are Java code combined with code written in VTL, which indicates where the JPEAG has to fill in the code according to the parameters defined by the user in the graphic interface. All this code is replaced when the template is processed in the core of the JPEAG. Fig. 4 shows an example of VTL code.

```
#if ($parallelismModel == "Island")
      ${EAModel}${parallelismModel} ea = new
      ${EAModel}${parallelismModel}($islandNumber,
      $migrationRate, $migrationFrequency);
#end
```

Fig. 4. Example of VTL code used by JPEAG

VLT directives starts with the character "#" and is executed when the template is processed. For instance, the #if directive in the Fig. 4 means that this part of code will be processed only if the user chose the island model. Variables begins with the character "$" and will be filled in according to the configuration done by the user in the graphic interface. As a result, the user will receive a pure Java code as shown in Fig. 5, where the parameters are the number of islands (4), the migration rate (5) and the migration frequency (10). Finally, the application is controlled by the Composer class, receiving all events whose the source is the visualization layer. Its name is suggested by ZK framework.

```
GAIsland ea = new GAIsland(4, 5, 10);
```

Fig. 5. Java code after a template processing

The other classes are in the package Persistence and implement the application model using the Java Persistence API (JPA) and the Hibernate framework. Basically, their function is to provide the persistence layer retrieving and saving information about parallel models, EA models and genetic operators. At this time, all the persistence is done in a PostgreSQL database.

C. Parallelism and Synchronization in PEAs

JPEAG is capable of create PEAs in the models master-slave (with evaluation function parallelization) and island in the current version. Besides, the parallelism is based on threads with no remote execution. In the master-slave model the evaluation is done in different threads. On the other hand, in the island model each deme evolves in a different thread. All communication is synchronized by a monitor object on both approaches.

Regarding to master-salve, there is a main thread where all population is held and the genetic operators are executed. The evaluation functions are executed in the slave threads. When the main process needs to perform the evaluation stage the population is divided into pieces and sent to the slaves. As soon as a slave ends the evaluation stage a notification is sent to the monitor object. If all slaves had done their jobs then the monitor object send a message to the master job that carry on with its execution.

Concerning the island model, each island runs an independent EA in each thread and individuals are exchange in a migration stage. The migration rate and the migration frequency are defined by the user in the graphic interface. When the time comes the *n* best individual, corresponding to the migration rate, are sent to the other islands. The received individuals replace the *n* worst ones in the population. As soon as an island ends the sending process the monitor object is notified and each island waits for an acknowledgment from the monitor to continue its execution.

D. The Parallel Code of EAs

The considered models, master-slave and island, are created using different class diagrams. Fig. 6 presents de class diagram for a master-slave PEA created by JPEAG and Table II describes the main ones: *GAMasterSalve*, *Evaluator* and *Evaluators*.

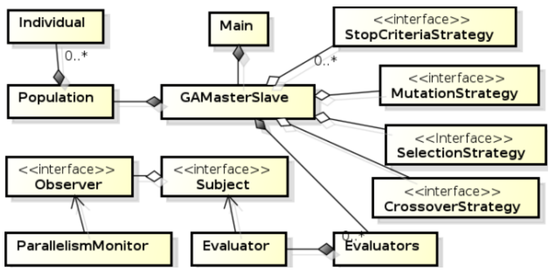


Fig. 6. Master-slave PEA class diagram created by JPEAG

The class diagram for the other model available in JPEAG, the island model, is depicted in Fig. 7 and Table III describes the main classes: GAIsland, GA and Evaluator. Actually, those cases (Figures 6 and 7) are a particular one concerned to a

Genetic Algorithm. Otherwise, taking an Evolutionary Strategy into account, these classes would begin with ES.

TABLE II. THE MAIN CLASSES OF MASTER-SLAVE PEA

Class	Description
GAMasterSlave	This class implements the EA. It maintains the objects that implements the operators according to the Strategy Pattern.
Evaluator	It implements all methods that use the evaluation function defined by the user. This class implements the interface *Java.lang.Runnable*, allowing that each evaluator runs into a different thread.
Evaluators	It maintains a data structure with the available evaluators, being similar to a list, allowing the master process to send individuals to the evaluators. In fact, the master send individuals to an evaluators object whose function is send it to the slaves.

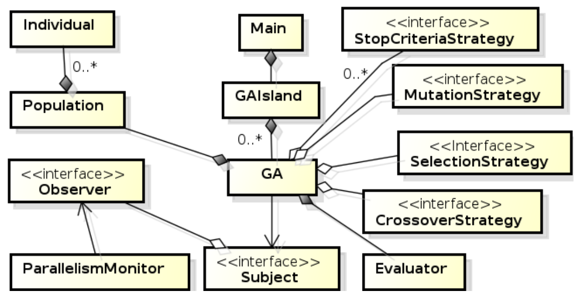

Individual
0..*
Population
Main
GAIsland
0..*
<<interface>> StopCriteriaStrategy
0..*
<<interface>> MutationStrategy
<<Interface>> SelectionStrategy
GA
<<interface>> Observer
<<interface>> CrossoverStrategy
ParallelismMonitor
<<interface>> Subject
Evaluator

Fig. 7. Island PEA class diagram created by JPEAG

TABLE III. THE MAIN CLASSES OF ISLAND PEA

Class	Description
GAIsland	It maintains a data structure similar to a graph indicating how the island can communicate each other. In fact, this class instances each island and execute it.
GA	This class maintains the objects that implement the genetic operators according to the strategy pattern. Also, this class implements the interface *Java.lang.Runnable*, allowing each island be executed in a different thread.
Evaluator	It implements the methods for the evaluation function defined by the JPEAG user.

The classes described in Table IV are common to both models with the same implementation or with small variations.

TABLE IV. COMMON CLASSES IN BOTH PARALLEL MODELS

Class	Description
Main	It is used for instancing the PEA and start its execution. The difference is in the instantiated class that has to be adequate tothe parallel model.
Individual	It is the class for instantiate individuals, having the same implementation for both PEAs.
Population	It is a list of individual containing methods for its manipulation.
ParallelismMonitor	Its function is to maintain the synchronization on the parallel models. It controls the migration process in the island model and the evaluators in the master-slave model.

E. Experiment

In order to demonstrate how the software can be used to improve the speedup of an EA, we present an experiment based on the Griewank function, which is showed in Equation 1, where $x_i \in [-600,600]$ and n is equal 30, representing an individual with 30 real-coded genes. This function was introduced in 1981 and is well known in the literature [22].

$$\min f(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) \qquad (1)$$

The experiment was conducted in an Intel i5 2.3 Ghz, 4GB of RAM with two physical cores, using Ubuntu Linux, JDK 1.7.04. A Parallel Genetic Algorithm was built using the configuration presented in Table V, where the first 9 parameters are common in both models and the last three ones are related only to the Island Model.

TABLE V. PARAMETERS OF THE PEA

Evolutionary Algorithm	**Genetic Algorithm**
Selection Operator	**Tournament**
Tournament Size	**10**
Crossover operator	**Heuristic**
Probability of Crossover	**0.8**
Mutation Operator	**Uniform (Random)**
Probability of Mutation	**0.01**
Population Size	**360**
Stop Criteria	**2000 iterations**
Island Model	
Migration Rate	**5 individuals**
Migration Frequency	**200 iterations**
Topology	**Ring**

The performance, speedup and efficiency are presented in Table VI. The time is the average of 31 trials in milliseconds and the value between brackets depicts the standard deviation. The speedup is calculated based on the weak speedup [23], where the performance of the serial version is measured using the same code of the parallel version but running in only one thread. The efficiency represents how better a parallel version is when compared against a serial one. Finally, *t* represents a t-test [24] for identifying whether the differences are significant or not.

TABLE VI. PERFORMANCE, SPEEDUP AND EFFICIENCY

Master-Slave				
Threads	**Time(ms)**	**Speedup**	**Efficiency**	**t**
1	4572.3870 (37.52)	-	-	101.24
2	3778,1290 (22.35)	1,2102	60.5%	
Island				
1	4297.2258 (49.58)	-	-	59.07
2	3108.9354 (100.41)	1,3822	69.1%	

Regarding the differences between the parallel versions and the serials, they are significant, otherwise *t* should be in the range [-2.0017, 2.0017] with $\alpha=0.05$ in a two tailed test. In other words, the parallel version using the master-slave model is 60.5% faster than serial version, whereas the island model is 69.1% faster, with both models using 2 cores (threads).

Also according to the performance table we can realize that the island model overcome the master-slave model in 8.6% of the efficiency. Taking into account a t-test between them, the result *t=36.21* indicates that this difference is significant. This happens because the master-slave model has to deal with more synchronization aspects than the island model, i.e., there are more communications between threads in the master-slave model. However, in both cases there is a reduction in the time required to execute an evolutionary algorithm.

VI. Conclusions and future works

This paper proposed the Java Parallel Evolutionary Algorithm Generator (JPEAG), a graphic web-based application that allows users to create parallel evolutionary algorithms in a friendly way. The main advantage of our software is to generate parallel code ready to be compiled and executed. In other words, our application makes code much more quickly than a programmer doing the same job at-hand. Thereby, JPEAG contributes on the increasing of the programmer's productivity and reducing the cost of creating parallel code. Further, our tool demands knowledge only in PEA theory, dismissing knowledge in parallel computing, excepting if the user wants to change or adds more features into the parallel programming resources. Furthermore, the user does not have to deal with the complexity of some particular aspects such as design patterns, for instance. Therefore, our tool can also be used by less experienced programmers.

Regarding the design patters, experienced programmers are able to easily make changes and extend the code. The strategy pattern got show effective hence the implementation of genetic operators became independent with no coupling, allowing their use in extended versions or in new operators. Moreover, the observer pattern lets us to implement an elegant solution, created with low coupling in order to control the synchronization by means of an automatically information exchange between processes, in the parallel models.

Finally, the use of templates allows including new EAs and parallel models by changing the pre-existing templates and adding the specific features of each algorithm. Thus, parallel hybrid EA can easily attached to the JPEAG. However, if the new algorithms are completely different with absolutely no similarities with the existing ones new templates can be added.

A. Future Works

The implementation of JPEAG carries out. New features include: binary-coded for GA, more genetic operators (for GA and ES), the use of elitism in GA (ES ($\mu+\lambda$) is already available on ES), and the parallel cell model. Further, a study about how to perform the parallelization by means of asynchronous communication has to be taken into consideration. Furthermore, tests in different platform are required for a better evaluation of the parallelization efficiency.

Extensions of the JPEAG for generating distributed PEAs using remote objects or frameworks such as ProActive, and JavaMPI are also in consideration. Implementations of the PEAs in other languages such as C and Python are also in mind.

References

[1] A. E. Eiben and Smith, J. E. Introduction to Evolutionary Computing, Berlim: Springer Verlag,2003.

[2] E. Cantú-Paz. E.A Survey of Parallel Genetic Algorithms.Department of ComputerScience and Illinois Genetic Algorithms Laboratory.Universityof Illinois at Urbana-Champaign – 1998.

[3] K. Jaquie. Extensão da Ferramenta de Apoio à Programação Paralela (F.A.P.P.) para Ambientes Paralelos Virtuais. Master Thesis – USP, São Carlos, SP, Brazil, feb 1999.

[4] J. Malacarne. Ambiente Visual para Programação Distribuída em Java. Master Thesis – UFRGS, Porto Alegre, RS, Brazil, feb 2001.

[5] F.Pasini, F. L. Dotti, Code Generation for Parallel Applications Modelled with Object-Based Graph Grammars, Electronic Notes in Theoretical Computer Science, v. 184, p. 113-131, 2007.

[6] A.W.O. Rodrigues, F. Guyomarc'h,J. Dekeyser, Y. Le Menach, Automatic Multi-GPU Code Generation Applied to Simulation of Electrical Machines, Magnetics, IEEE Transactions on , vol.48, no.2, pp.831,834, Feb. 2012

[7] K.A. Hawick, D.P. Playne, Automated and parallel code generation for finite-differencing stencils with arbitrary data types, Procedia Computer Science, Volume 1, Issue 1, May 2010.

[8] J. C. Linford, J.Michalakes, M.Vachharajani, A.Sandu,Automatic Generation of Multicore Chemical Kernels, Parallel and Distributed Systems, IEEE Transactions on , vol.22, no.1, pp.119,131, Jan. 2011

[9] S. Cahon, N. Melab, E.-G. Talbi. ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics. Journal of Heuristics, v. 10, n. 3, p. 357-380, Springer 2004.

[10] F. Herrera, M. Lozano, and J. L. Verdagay. Tack-ling Real-Coded Genetic Algorithms: Operators and Tools for BehaviouralAnalisys. Artificial Intelligence Review, 4(12):265–319, 1998.

[11] Z. Michalewicz. Genetic Algorithms + DataStructure = Evolution Programs. Springer-Verlag,New York, 3 edition, 1999.

[12] O. A. C. Cortes, R. H. C. Santana, M. J. Santana, O. R. S. Mendez, Análise de Operadores de Recombinação em Estratégias Evolutivas Aplicados no Refinamento de um Sistema Nebuloso. In: Simpósio Brasileiro de Automática Inteligente, 2005.

[13] J. Yao. Analysis of Scalable Parallel Evolutionary Algorithms. IEEE Congress on Evolutionary Computation Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada. July, 2006.

[14] E. Alba and M.Tomassini. Parallelism and Evolutionary Algorithms. IEEE Transactionson EvolutionaryComputation, Vol. 6, No. 5, October, 2002.

[15] J. Herrington. Code generation in action. Greenwich: Manning Publications, 2003.

[16] D. Lucrédio. Uma Abordagem Orientada a Modelos para Reutilização de Phd Thesis –USP, São Carlos, SP, Brazil, 2009.

[17] D. Manolescu. Pattern Languages of Program Design 5. Reading: Addison-Wesley Professional.

[18] U.Markowska-Kaczmar and F.Krygowski. The Influence of Using Design Patterns on the Process of Implementing Genetic Algorithms. Trends in Applied Intelligent Systems, p. 173-182, Springer, 2010.

[19] H. Feng, K. Li-shaff, C. Yu-ping. A Generic Design Model for Evolutionary Algorithms. Wuhan University Journal of Natural Sciences Vol.8, No 1b, p. 224-228, 2003.

[20] ZK Framework Site - http://www.zkoss.org - access: 4/10/2013.

[21] Apache Velocity Project Site - http://velocity.apache.org - access: 4/10/2013.

[22] M. Locateli,, A note on the Griewank Test Function, Journal of Global optimization, v. 25, p. 169-174, 2003.

[23] E. Alba, G. Luque, Evaluation of Parallel Metaheuristics, In L. Paquete, M. Chiarandini and D. Basso (eds.), Proceedings of the EMAA, pp. 9-14, Reykjavik, Islandia, 2006.

[24] W. C. Schefler, "Statistics: Concepts and Applications", The Benjamin/Cumming Publishing Company Inc., UK, 1988.