Bringing Evolutionary Computation to Industrial Applications with GUIDE

Luis da Costa, Marc Schoenauer

▶ **To cite this version:**

Luis da Costa, Marc Schoenauer. Bringing Evolutionary Computation to Industrial Applications with GUIDE. GECCO 2009, ACM, Jul 2009, Montréal, Québec, Canada. inria-00375419

Bringing Evolutionary Computation to Industrial Applications with GUIDE

Luis Da Costa[1], Marc Schoenauer[1,2]

[1]Team TAO, LRI (UMR CNRS 8623)
INRIA Saclay - Île-de-France
Bat 490, Université Paris-Sud
91405 Orsay Cedex, France

[2]Microsoft Research–INRIA Joint Centre
Parc Orsay Université
28, rue Jean Rostand
91893 Orsay Cedex, France

luis.dacosta@inria.fr, marc.schoenauer@inria.fr

ABSTRACT

Evolutionary Computation is an exciting research field with the power to assist researchers in the task of solving hard optimization problems (*i.e.*, problems where the exploitable knowledge about the solution space is very hard and/or expensive to obtain). However, Evolutionary Algorithms are rarely used outside the circle of *knowledgeable practitioners*, and in that way have not achieved a status of useful enough tool to assist "general" researchers. We think that part of the blame is the lack of practical implementations of research efforts reflecting a unifying common ground in the field.

In this communication we present GUIDE, a software framework incorporating some of the latest results from the EC research community and offering a Graphical User Interface that allows the straightforward manipulation of evolutionary algorithms. From a high-level description provided by the user it generates the code that is needed to run an evolutionary algorithm in a specified existing library (as of March 2009, EO and ECJ are the possible targeted libraries). GUIDE's GUI allows users to acquire a straightforward understanding of EC ideas, while at the same time providing them with a sophisticated research tool. In this communication we present 3 industrial case studies using GUIDE as one of the main tools in order to perform software testing on large, complex systems.

Categories and Subject Descriptors

D.2.2 [**Design Tools and Techniques**]: Software libraries;
D.1.7 [**Programming Techniques**]: Visual Programming;
H.5.2 [**User Interfaces**]: Graphical Interfaces

General Terms

Algorithms

Keywords

Evolutionary Computation; Software Development; Algo-

rithms.

1. INTRODUCTION AND MOTIVATION

In this article we present GUIDE, a programming framework that comes with an easy-to-use Graphical User Interface that allows the easy, efficient, and economical manipulation of evolutionary algorithms (EAs). From the high-level description of the user's problem, generated through the GUI, it derives the code of a complete Evolutionary Algorithm in a specified existing library, and compiles it into an executable program that can be executed as a normal computer program. The GUI also offers an original global point of view of the algorithm that results in a straightforward improvement of the user's understanding of EC ideas. Finally, it allows any GUIDE user to benefit from recent results from the research community as soon as they are implemented in its kernel.

The development of GUIDE was born out of our concern of bringing the methods of our domain, Evolutionary Computation, closer to *general, non-expert,* users. It started to be developed under the **DREAM** FP5 European Project (www.dcs.napier.ac.uk/~benp/dream/dream.htm), from where it took its acronym ("**G**raphical **U**ser **I**nterface for **D**REAM **E**xperiments") and has come a long way since. The need for such a "democratization" of the field has appeared along with the practical success of Evolutionary Algorithms to solve hard optimization problems. Indeed, EAs consistently perform well approximating solutions to a large number of *types* of problems (see references [20, 10] for a long list and explanations of these problems), largely because they do not make any assumption about the underlying search space. Research groups on EC are very active around the world, with several high-level conferences regularly held and an important number of scientific journals dedicated to the field. However, their adoption by non-expert users has not followed. We think there are several reasons for such a state of affairs, the most important ones being:

1. **A lack of practical support for work:** in spite of the publication of Michalewicz' seminal book [20], and much more recently of the two unifying books by Eiben and Smith [10] and DeJong [8], it is very difficult to get a clear picture of the field, as there still is not even a common terminology between all papers published in the domain. This terminology is still much influenced by the history of the models that have been proposed

over the years. However, a user, advanced or not, is mainly concerned about the best possible way to solve her/his problem, so the historical differences that exist in the terminology should not matter [5]. However, even though all models share a common structure, no existing software package allows the user to actually shift from one model to another.

2. **The difficult tuning of an EA:** The second problem, and probably the most important one, is related to the specific tuning of an evolutionary algorithm. The description of a specific EA contains its components (the choice of representation, selection, recombination, and mutation operators) thereby setting a framework while still leaving quite a few items undefined [11]. For instance, in a simple GA, while the representation is a simple bitstring with its associated operators, further details have to be given (population size, choice of crossover and mutation operators, and probabilities of application of these operators) before having a running version of the algorithm. These algorithm parameters greatly determine whether the algorithm will find an optimal or near-optimal solution, and whether it will find such a solution efficiently [11]. Choosing such operators and parameters is difficult, no theoretical guidelines yet exist, and a great deal of knowledge about EC is needed to make a sensible choice. The growing field of Automatic Parameter Tuning is based on the idea that is possible to make the choice of the parameters in an automatic way, with minimal user intervention, but is at the moment only described in recent papers (a good review of the state of the art in the field can be found in reference [18]) and is not yet available in existing EC software packages.

GUIDE is a complete software environment that proposes a solution for these two problems: first, it offers a framework with a GUI, establishing in a practical and easy-to-visualize way a clear paradigm to work with any EA. Second, it incorporates the latest results from current research on Adaptive Parameter Tuning. This paper introduces a description of the system (Section 2), and presents three current industrial applications that use GUIDE under the frame of the Evotest European Project: two from the automobile industry and one from the software development industry (Section 3).

2. GUIDE

In order to *Cross the Chasm* [21] and bring the benefits of EC to the research and industry communities it is imperative to find a way to let the user easily interact with a research tool. In the case of an EC tool, this idea amounts to "asking" the user only for what they know about: (1) *what is the problem?* (*i.e.*, how can the problem be encoded?) and, (2) *how to recognize a good solution for the problem?* Other than that, they needn't be concerned with any of the details that advance practitioners in the field of EC have to know about.

2.1 Defining an Evolutionary Algorithm

GUIDE lets the user intuitively define an EA, either programatically through an API (as in the case where a third-party application interfaces with it) or by means of a GUI.

We show in this Section hints of the GUI interaction; interfacing with the API is completely equivalent, from a functional point of view. This definition has 3 parts: the *representation*, or how to describe a possible solution of the problem at hand, the *fitness function*, or how to evaluate the quality of possible solutions, and the *Evolution Engine*, or how to possibly describe the different selection mechanisms of an EA.

2.1.1 Representation

Rationale: Solving a given problem with an EA starts with specifying a representation of the candidate solutions (the "*individuals*" of the population). There has been for some time now a body of research dedicated to searching for a problem-independent representation: a certain representation that could be easily instantiated and directly used independently of the problem. This search is focused on finding regularities or properties of a general search space, and then reaching conclusions valid for any representation mapping into that search space.

Early work that has been published in that respect start with John Holland's *Schema Theory* [14]. The Schema Theory argues for the optimality of binary encoding (*i.e.*, *bitstring encoding is the most efficient encoding for any problem*), which is proven for finite search spaces. It was heavily argued (in [1, 22, 25, 23, 24]) and later proved (in [32, 33]) that no single representation can be better than any other for all problems. Then the search for a universal representation had to be switched to the more realistic goal of guiding the user towards the specification of a sensible representation for their problem.

An extension of Holland's ideas is the *Forma Theory*, originally developed by Radcliffe [22, 25, 23, 24] and continued later with Surry [26, 27, 28, 30, 31]. It is both an attempt to generalize the Schema Theory, and, more importantly, to link it more tightly with the problem at hand. Radcliffe and Surry do not claim there exists *one* optimal representation for *any* problem, but instead they build their ideas on the existence of a *family* $\mathcal{F}$ of representations that are optimal. Radcliffe and Surry's ideas were later extended by Droste and Wiesmann [29, 9], essentially discussing conditions needed to achieve convergence on the search.

The Forma Theory approach would indeed provide a representation family containing *any* specific problem if **all** concrete representations could be proven to be in $\mathcal{F}$. But it has been shown that some widely used representations, like parse-trees (commonly used for Genetic Programming), cannot be expressed in this formalism.

In light of these research results, in GUIDE we have taken the approach of asking the user for some details about the representation. Note that, no matter what representation is chosen, there are a certain number of details that have to be defined in order to apply an EA: how to create (*initialize*) the population, and how it is modified (*mutated* and *combined*). In other words defining a representation for an EA not only requires the specification of the actual structure: it also involves the specification of the functions that modify this structure.

GUIDE provides three ways of helping with this specification: first, it has a GUI that a user can manipulate (or an API that can be used from within a program). Second, it provides a set of operators (dynamically loaded from an external repository) that apply to the specific types involved

in the definition of the problem. Third, it provides default values for the operators; that way, the definition of those operations becomes optional for the user.

For example, when a user chooses to use a real as the representation of her problem she has to specify how to initialize and modify each individual generated by an EA. GUIDE provides, for the chosen type real, some pre-defined functions with chosen defaults:

- **Initialization**: available are *Uniform* (taking a value uniformly in the interval of definition) and *Gaussian* (choosing a value by sampling a Gaussian function with mean the middle of the interval and with standard deviation a third of the length of interval of definition). Default is *Uniform*.

- **Mutation**: *Uniform* or *Gaussian*, defined as in the initialization description. Default is *Gaussian*.

- **Crossover**: *Exchange* (children exchange parents' values), *Linear* (children take a linear combination of their parents' values), *Middle* (children take the middle point of their parents' values) or *Gaussian* (children sample a Gaussian distribution defined by the parents' values). Default is *Gaussian*.

The same type of choices arises for every representation defined. Note that the operators needed for structured representations (*e.g.*, "vector of reals") can be defined using the operators present for basic representations. We give more details below.

Representation and operators in GUIDE: More precisely, an individual in GUIDE is defined through the use of **basic types** and **containers**, that can be recursively used to construct complex structures through the GUI. The corresponding 'structure' for the operators is built at the same time without any action from the user, as detailed below.

- **Basic types** are types that can't be defined using other types. Available in GUIDE are **enumeration**, **boolean**, **integer**, **real value** and **permutation**.

- **Containers** are places to hold different instances of other sub-structures in a recursive way. They are either heterogeneous or homogenous. The heterogeneous container, termed **tuple**, is a simple list of other sub-structures. Homogenous containers contain several instances of the same partial type, and are distinguished by the number of instances of the sub-structures they contain (either fixed or variable), and by whether or not the order of those sub-structures is relevant. This gives the 4 possibilities: 'ordered' containers are **vectors** (fixed length) and **lists** (variable length) and order-independent containers are called **bags** (fixed length) and **sets** (variable length).

  The main reason for distinguishing those 4 types (vectors, lists, bags and sets) is that their characteristics dictate the definition and application of the corresponding operators (initialization, crossover and mutation). For instance, the number of sub-structures to be initialized by the initialization operator is either fixed or uniformly drawn in a given interval.

  From there on, default operators can easily be defined for containers, that either act at the container level (*e.g.*, add or delete sub-structures for variable-length containers; 1-point crossover on vectors) or call the corresponding operator on the sub-structures (*e.g.*, the 'flip' mutation operator of any container sequentially calls the available mutation operator of each one of its substructures with a given probability – by default, $1/n$ where n is the number of sub-structures).

We want to stress here again that the user is only asked to provide a description of the structure of the possible solutions of the problem at hand. Default operators will be set up by GUIDE automatically. Though some of those operators seem to require some parameters, only those parameters that are part of the definition of the structure will be asked to the user.

Another important feature of GUIDE is its *flexibility*: all the above-mentioned default values can be overridden by the user in the GUI, clicking on the corresponding tag. The curious user will gradually acquire deeper and deeper understanding of the principles of an EA by looking at the operators that are actually used in the corresponding of GUIDE, and start to change them, checking the effect on the behavior of the algorithm. Ultimately, the advanced user can even include his own hand-made operators. Local and specific improvements could hence be rapidly disseminated to all colleagues who do not want to know about EC in detail.

2.1.2 The Fitness Function

Once the representation for the problem has been chosen, a measure of quality for the solutions has to be defined. This is clearly related to the problem being solved: fitness-based selection implements the paradigm of *natural selection*, and is the force that represents the drive toward quality improvements in an EA. The *knowledge* the user brings from his(her) field of expertise into the resolution of their problem has to be somehow fed into the system that is solving it, and the fitness function very often is the only information about the problem in the algorithm.

GUIDE provides a basic screen allowing the user either to write her/his own fitness function, or to specify where the code or executable program for the fitness function is located (specification of the API is provided in this latter case). Details on the definition can be found in GUIDE's manuals (documents hosted in `guide.gforge.inria.fr`). It is worth noting that this is the only place where any kind of programming is needed in order to use GUIDE.

2.1.3 The Evolution Engine

After defining the representation and the fitness function, the framework for an EA is completed. However, there are still some undefined points in order to have a running version of the algorithm: the details about exactly how to carry out the operations defined by the representation definition. These are called the *parameters of the evolution engine*, and are illustrated by boxes and comments in Alg. 1

The evolution engine parameters define the fine-tuned functioning of the algorithm: how many individuals form the population (line 1), the stopping condition of the algorithm (line 3), and the details about the internal dynamics of a generation: the way the parents are selected to become genitors (line 4), the way those genitors are recombined in order to yield the offspring (line 5) and the way that offspring replaces the old population in order to form the new population (line 7).

Algorithm 1 Parameters of an Evolutionary Algorithm

1: **Initialize** population.{*How many* | individuals |*?*}
2: **Evaluate** (compute fitness of) all individuals
3: **while** | not STOP | **do** {*Good stopping condition?*}
4:   | **Select** genitors | from parent population {*How?* }
5:   Create offspring using | **variation** | operators on genitors {*stochastically?*}
6:   **Evaluate** newborn offspring
7:   | **Replace** | some parents by some offspring {*How?*}
8: **end while**

GUIDE has a very user friendly screen dedicated to the evolution engine. A snapshot is presented in Fig 1. This part was carefully designed for clarity, as a user's understanding of the relevant parameters is central to the use of our tool. A side effect, here again, is that it makes GUIDE a great tool for teaching Evolutionary Computation.

While the problem-specific points have to be specified by the users (as they are the only ones to *know* what the problem is about, in the end), we believe that the parameters can be chosen in automatic ways. There is a strong community of research in this area, exemplified by the workshops held regularly in EC conferences, or by the printing of the latest books (see, for example, [18]). GUIDE contains initial implementations of some of the latest results on the area: a method to perform basic dynamic choice of evolutionary operators based on past performance ([12]) and an initial implementation of the Racing method [3], which is a method that optimizes all parameters of an EA.

2.2 Features of GUIDE

Now that we have presented GUIDE from an user's point of view we would like to present the characteristics that we think are most important in the framework, along with the theoretical motivations behind them. GUIDE follows the idea of the first library implemented under the theoretical developments of representation-independence: EO [15]. Indeed, despite good theoretical advances, no practical implementation of the idea of finding a representation-independent EA existed other than a first effort by Radcliffe and Surry (they implemented a description language called **Reproductive Plans Language**, but their efforts stopped there). From this effort, Keijzer *et al.* [15] developed a representation-independent framework called EO (for *Evolving Objects* – eodev.sourceforge.net). Its design principles implement a syntax-based forma theory, differing from the full-fledged Forma Theory in that the equivalence relations (and therefore the formæ) are built purely from syntactic information from the structure of the representation.

However, EO still requires, for each new representation, a careful design of the corresponding initialization and variation operators. A very basic idea implemented in GUIDE is to provide generic basic types, with associated operators, that could be reused when constructing the final representations. In that way, all the skeletons of the representations would be problem-independent. GUIDE is in fact an abstraction layer for the user or for the application that wants to use a meta-heuristic as means of finding a solution (this idea is represented on Fig. 2). It uses a description language [6] to manipulate the basic types and associated initialization and variation operators, and fully implement the different types of representation-independent operators.
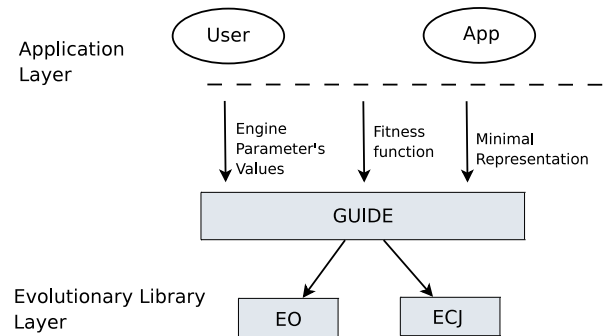
Figure 2: GUIDE using different Evolutionary Libraries (EO and ECJ, in this case)

GUIDE can, theoretically, generate code for any library or programming language; in practice we have followed the opinion handed by Gagné and Parizeau, who performed a survey of existing evolutionary frameworks in [13]. The authors conclude that only 3 of them fulfill their criteria for genericity: ECJ [19], EO (although they express some restriction regarding the configurable output aspect) and OpenBeagle (described in the last section of [13]). The current implementation of GUIDE allows for code generation in EO (C++) and ECJ (Java – `cs.gmu.edu/~eclab/projects/ecj/`). The addition of more libraries requires relatively little work, as it only involves the definition of templates in Velocity (`velocity.apache.org`) for the specific language.

The main features of GUIDE can be summarized as follows:

1. **Target public**: as stressed through the paper, our main objective is to bring the main ideas of the field of EC to the larger community of researchers. However, it is important to stress that the users of this technology have to be able to describe the problem they are trying to solve. In the domain of EC this requirement is visible on the fact that the fitness function has to be programmed by the user in order to use it in GUIDE. This requirement is very reasonable, as the definition of the fitness function is part of the description of the problem. Other than this, no programming is needed in order to use GUIDE.

2. **Management as a software product**: it is maintained by a group of developers, and publicly offered at `guide.inria.gforge.fr` under a CeCILL-C license[1]

---

[1] The CeCILL family of licenses was first released in July 2004 by the initiative of three French research organizations, **CEA**, **CNRS** and **INRIA**. CeCILL is the first license defining the principles of use and dissemination of Free Software in conformance with French law, following the principles of the GNU GPL. The license is backed by identified, French, law; this is a plus in case of litigation and thus incites confidence, especially concerning the use and development of free software in business. CeCILL is fully compatible with GNU GPL. For more details and discussion, please see (`www.cecill.info/index.en.html`) and `www.inria.fr/actualites/inedit/inedit47_actu.en.html`
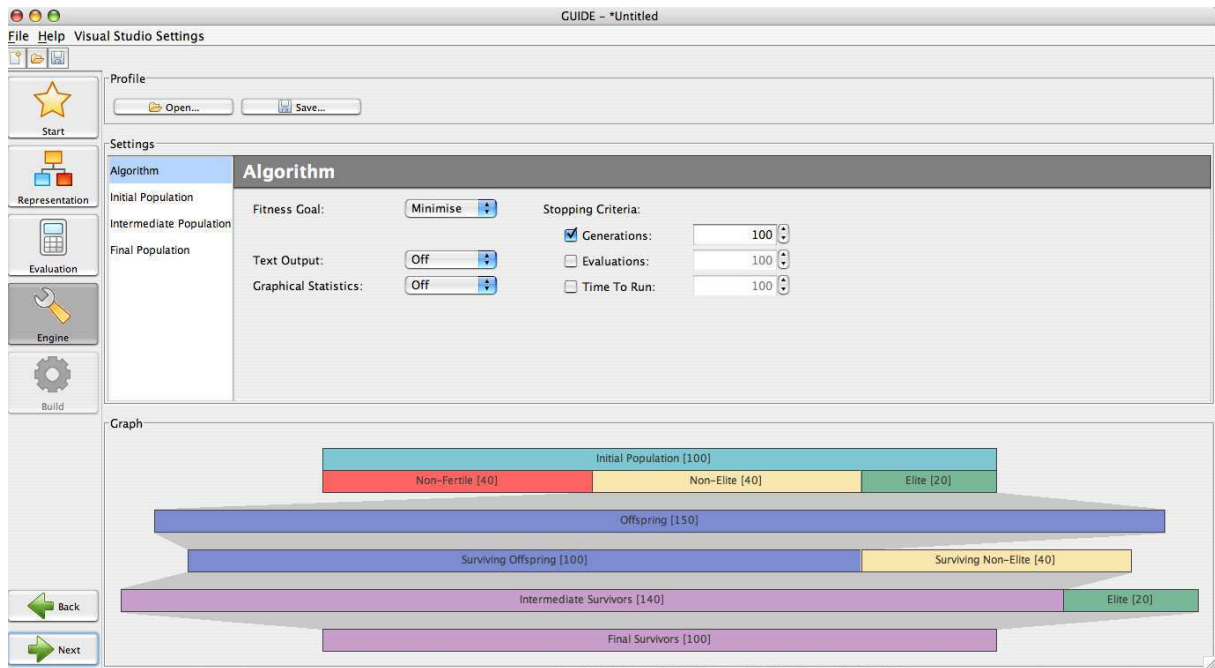
Figure 1: Evolution engine specification in GUIDE.
The engine is represented with its values, and in graphical form.

in order to be further developed by the community. It is a pure Java application, and thus it runs on any architecture able to run a Java virtual machine.

3. **Code generation**: starting from a high-level description of an EA it generates the actual code (in a specified programming language) that can be executed as a normal computer program. It liberates the user from the burden of investing time in understanding programming details that are probably superfluous in their everyday research life. It can generate code on several platforms and languages, and other libraries can be added on request.

4. **Allows easy user-understanding of EA ideas** and **incorporates results from the research community**, particularly in the sub-field of automatic parameter tuning.

3. CURRENT INDUSTRIAL APPLICATIONS

In this Section we present three examples of current industrial applications that are using GUIDE (embedded on a higher-level framework) in order to perform testing on their software systems.

Testing is at the moment the most used quality assurance technique for software systems. However, the development of cost effective and high-quality complex systems opens challenges that cannot be faced only with traditional testing approaches. Automation of difficult and time-consuming tasks like test case design seems to be the only way to master the complexity, and develop quality systems within a competitive amount of time.

Evotest is a multidisciplinary project that combines the power of evolutionary adaptive techniques with software engineering techniques like slicing, program transformation and reliability analysis in order to find solutions to the problems of testing software systems and dealing with its complexity.

Now, in order to provide results on how effective and how efficient the use of Evolutionary Testing is for functional testing and how applicable it is in industrial practice extensive case studies must be defined and carried out. A case study will also detect weaknesses in the Evolutionary Testing Concept and will contribute to further improvements in Evolutionary Testing technology.

GUIDE is the evolutionary engine generator for the software system produced by the **Evotest** European Project (`www.evotest.eu`). The goal of Evotest is to perform testing on software programs using the power of EC. Under Evotest we have developed, with the rest of the partners, a framework that allows the development of testing scenarios (the framework will be referred to as **ETF** for the rest of this paper). The software architecture for the ETF is presented on Fig. 3 (taken from [16]). GUIDE is the evolutionary engine generator for this framework; its role is to generate, following the specifications produced by the users through the ETF, the engine performing the evolutionary search. This (generated) engine is identified on the middle bottom of Fig. 3, labeled as **EvolutionaryEngine**.

The objective of work with the ETF is to perform testing on software programs using the power of EC, and thus to use the Evolutionary Testing Concept on real-world studies; Evotest case study partners will evaluate this concept by applying the resulting framework to problems of their respective application domain. Here we present three of the problems, with a short introduction of the application domain and the objective to be reached by each of them.

1. **Daimler's automotive case studies:**

    Daimler uses ETF to perform automotive case studies taken from serial production developments of the
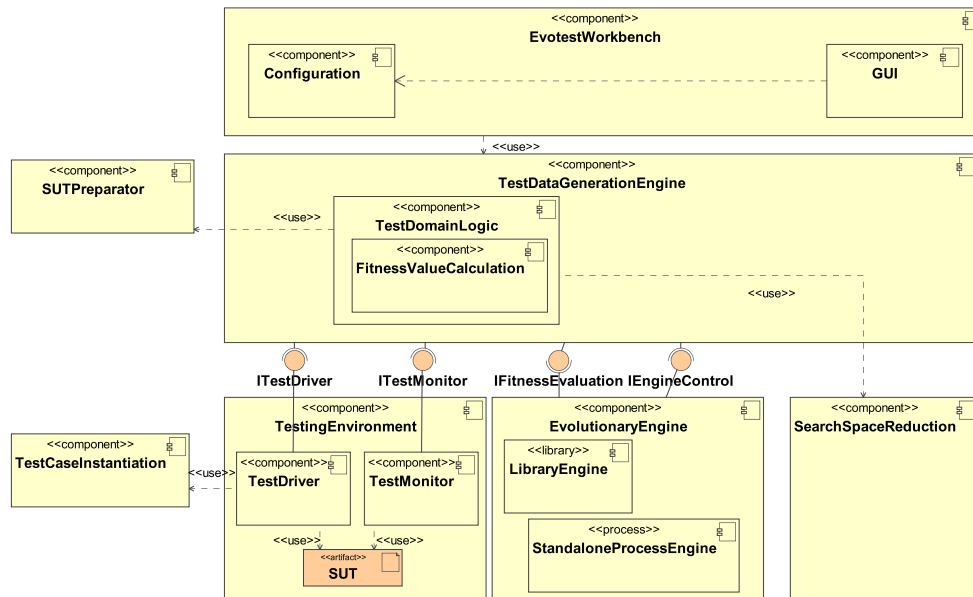
Figure 3: Evotest's ETF. GUIDE generates the engine labeled as EvolutionaryEngine.

power train development divisions, the vehicle dynamics development divisions or from driver assistance system developments.

Currently Daimler is working with their Adaptive Cruise Control (ACC) systems. The 1st generation ACC system was introduced in the Actros (Mercedes truck) in June 2000 while the 2nd generation ACC system was introduced in January 2005. MB Passenger Cars is currently working on the 3rd generation ACC. The proximity control function (ACC) is mainly used in the long-distance haulage sector, on freeways, highways or expressways as well as routes similar to freeways with curvature radiuses larger than 250m. The ACC's sensor system must therefore be capable of reliably detecting the traffic situations relevant to this area. These also include construction site areas or areas with changed traffic routing. Many accidents are rear-end collisions on highways. When activated, the purpose of ACC is not only to maintain a given speed but also to control the distance to preceding vehicles and thus prevent accidents.

Safety-critical requirements are tested. For this purpose input parameters (continuous signals) representing a specific real world scenario must be determined that yield a violation of the requirement under investigation. Safety requirement violations to consider are:

(a) The system does not intervene in a critical situation, possibly leading to a collision.

(b) The system still intervenes even though the situation is not critical anymore.

(c) The system intervenes even though the situation is not critical at all.

(d) The system neglects the driver's intent to deactivate the system by using the turn signals or by accelerating.

(e) Safety requirements on handling merging vehicles.

The most evident safety requirement to be tested is the compliance with the safety distance to the preceding vehicle.

2. **Berner & Mattner Systemtechnik GmbH (BMS) automotive case studies:** BMS uses the ETF in conjunction with some proprietary tools and simulators in order to test the performance of an anti-lock braking system (ABS). An ABS is a system which prevents the wheels of a vehicle from locking while braking, in order to allow the driver to maintain steering control under heavy braking and, in most situations, to shorten braking distances. ABS is very effective at braking in adverse weather conditions like ice, snow or rain. When ABS equipped brakes are depressed hard - like in an emergency braking situation - the ABS system pumps the brakes several times per second. Sensors measure the speed at which the wheels are turning. If the speed decreases rapidly, the electronic control system reports blocking danger. The pressure of the brake hydraulics is reduced immediately and then raised to just under the blocking threshold. This process can be repeated several times per second. The goal of the anti-locking control system is to maintain the slip of the wheels at a level which guarantees high-

est braking power and highest maneuverability of the vehicle.

For the testing of the anti-lock braking system input values for the input signals of the system have to be provided by the individuals generated. Typical input signals for the anti-lock braking system are the wheel speeds for the wheels of the vehicle and the braking torque requirement resulting from the driver pressing the brake pedal (probably supported by driver assistance systems controlling the overall vehicle performance). Additionally, individuals have to control parameters of the simulation environment, *e.g.*, the grip of the roadbed and the road conditions (icy road, wet road, gravel road), temperature of the brake discs. The individuals have to define input signals and simulation parameters for several seconds of realtime simulation of a braking maneuver.

3. **RILA's software case study:** The system under test is called ChatPC. ChatPC is a dynamic display augmentative communication device using a ruggedized PocketPC. It offers symbol-based, dynamic-screen communication. ChatPC is intended for individuals with existing or emerging literacy skills or using symbols for communication. It has all the features necessary to use and create vocabulary sets using symbols and photographs as well as spelling enhancement tools. ChatPC provides Word Prediction, Letter Prediction and Abbreviation Expansion. The main ChatPC application is built upon a PocketPC.

ChatPC software package has two main applications - the client software which can operate on a mobile device or desktop computer and provides GUI to the user and a desktop-based editor for the client's content. The editor includes capability to import photos and other graphics that can be used as symbols in the content, loaded in the client software. Customization capabilities include the size, color, background, symbol/image, font and associated action of the rendered buttons as well as touch-screen button layout and linking between pages. ChatPC can generate speech in English, French, or Spanish and switch between them dynamically. ChatPC can provide access to its functionality either by a pointing device which can directly select buttons on the screen (stylus or fingertip on the touch screen, mouse, movement/gaze tracking system) or by using a specialized interactive mode called switch scanning where external switches can be used for navigation by mobility-impaired people.

4. FUTURE WORK

We consider that our efforts in further development will follow two main directions: first, improving and extending the feedback given to the user. Some advanced visualization capabilities are still missing in GUIDE; we believe that a clear vision of how the genome is changing over the generations is one of the key features allowing a user to make informed choices about the run of an EA.

We are also aware that, though GUIDE covers most industrial needs in term of the representation of solutions the main missing part is the representation of parse trees as used in Genetic Programming [17, 2]. It is true that few industrial applications seem to be concerned with GP; this can

actually be the result of the lack of a user-friendly programming environment for developing GP applications, and thus justify that tree structures are added to GUIDE.This addition represents a large amount of work, mainly due to the difficulties associated with a sensible graphic representation of solutions.

Another important area that is rapidly growing in our field and concerns many industrial application is multi-objective optimization [7, 4]. Most real-world problems are in fact multi-objective, even if they are often transformed in single-objective problems for the sake of simplicity. From the EC point of view, the 'only' part that needs to be modified in GUIDE so that it is able to handle multi-objective problems is the Evolution Engine (and the template for the fitness function, that would have to return a vector of values). Indeed, the largest part of GUIDE, the easy specification of representation and operators, is exactly the same for both single- and multi-objective EAs. This extension can hence be envisioned in a rather short term.

5. CONCLUSION

In this paper we presented GUIDE, a software framework that allows the easy and efficient manipulation of evolutionary algorithms. Based on a high-level description provided by the user, it generates a complete ready-to-use evolutionary algorithm, fully executable as a stand-alone application. Moreover, the ideas implemented in the application allow for a straightforward understanding of EC ideas, while at the same time incorporating results from the research community, like the automatic tuning of some of the sensitive parameters of the algorithm, both off-line (using Racing techniques) and on-line (using Adaptive Operator Selection methods).

We also described 3 industrial applications where GUIDE is actively used, all related to evolutionary software testing: Daimler's Adaptive Cruise Control testing, BMS's anti-lock braking system, and RILA's ChatPC software product. All three are actively using GUIDE as part of a set of tools provided under the umbrella of the Evotest European Project.

The need for GUIDE is motivated by our belief of the necessity of a common ground in our field, both in theory and communication, but also in practice; a main unifying idea from EC practitioners must reflect a single view for newcomers to the field. We argue in this paper that one big step toward such grouping ideas of the field can be the use of a unified environment, that will be useful not only in order to disseminate our work and research outside our community, but even to a better understanding and a faster fostering of ideas inside our field.

We believe that GUIDE is a good proposal for the beginning of a solution for the problem of making EAs widely-known, both in industry and in academia. As a further proof of concept, GUIDE is being used on a educational level for some courses on EC.

Acknowledgments

6. REFERENCES

[1] J. Antonisse. A new interpretation of schema notation that overturns the binary encoding constraint. In J. D. Schaffer, editor, *Proc. ICGA'89*, pages 86–91. Morgan Kaufmann, June 1989.

[2] W. Banzhaf, P. Nordin, R. Keller, and F. Francone. *Genetic Programming — An Introduction On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann, 1998.

[3] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proc. GECCO'02*. Morgan Kaufmann, 2002.

[4] C. A. Coello Coello, D. A. Van Veldhuizen, and G. B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, New York, 2002. ISBN 0-3064-6762-3.

[5] P. Collet and M. Schoenauer. GUIDE: Unifying evolutionary engines through a graphical user interface. In P. Liardet, P. Collet, C. Fonlupt, E. Lutton, and M. Schoenauer, editors, *Evolution Artificielle, 6th International Conference*, volume 2936 of *Lecture Notes in Computer Science*, pages 203–215, Marseilles, France, 27-30 Oct. 2003. Springer. Revised Selected Papers.

[6] L. Da Costa. Specification of language description. Technical report, INRIA Futurs, 2007. Produced as Evotest's Deliverable 3.2. EvoTest is identified as EU-IST STREP FP6-IST-2006-33472.

[7] K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley, 2001.

[8] K. DeJong. *Evolutionary Computation. A unified Approach*. MIT Press, 2006.

[9] S. Droste and D. Wiesmann. On representation and genetic operators in evolutionary algorithms. Technical Report CI–41/98, Universität Dortmund, 1998.

[10] A. Eiben and J. Smith. *Introduction to Evolutionary Computing*. Springer Verlag, 2003.

[11] A. E. Eiben, Z. Michalewicz, M. Schoenauer, and J. E. Smith. Parameter Control in Evolutionary Algorithms. In F. Lobo, C. Lima, and Z. Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, chapter 2, pages 19–46. Springer Verlag, 2007.

[12] A. Fialho, L. Da Costa, M. Schoenauer, and M. Sebag. Extreme value based adaptive operator selection. In *Parallel Problem Solving from Nature– PPSN X*, volume 5199/2008, pages 175–184. Springer Berlin / Heidelberg, 2008.

[13] C. Gagné and M. Parizeau. Genericity in evolutionary computation software tools: Principles and case-study. *International Journal on Artificial Intelligence Tools*, 15(2):173–194, 2006.

[14] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[15] M. Keijzer, J. J. Merelo, G. Romero, and M. Schoenauer. Evolving Objects: a general purpose evolutionary computation library. In P. C. et al., editor, *Evolution Artificielle'01*, pages 229–241. LNCS 2310, Springer Verlag, 2002. URL: http://eodev.sourceforge.net/.

[16] G. Kock, J. Hänsel, and J. Gerlach. Automated evolutionary testing architecture. Technical report, Fraunhofer FIRST, November 2007. Produced as Evotest's Deliverable 5.1. EvoTest is identified as EU-IST STREP FP6-IST-2006-33472.

[17] J. R. Koza. *Genetic Programming: On the Programming of Computers by means of Natural Evolution*. MIT Press, Massachusetts, 1992.

[18] F. Lobo, C. Lima, and Z. Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*. Springer, 2007.

[19] S. Luke, L. Panait, G. Balan, S. Paus, Z. Skolicki, J. Bassett, and R. Hubley. ECJ: Evolutionary computation in java.

[20] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, New-York, 1992-1996. 1st-3rd edition.

[21] G. A. Moore. *Crossing The Chasm*. Collins Business, revised (aug 8 2002) edition, 2002. ISBN-10: 0060517123; ISBN-13: 978-0060517120.

[22] N. J. Radcliffe. Equivalence class analysis of genetic algorithms. *Complex Systems*, 5:183–20, 1991.

[23] N. J. Radcliffe. Forma analysis and random respectful recombination. In R. K. Belew and L. B. Booker, editors, *Proc. ICGA'91*, pages 222–229. Morgan Kaufmann, 1991.

[24] N. J. Radcliffe. Nonlinear genetic representations. In R. Manner and B. Manderick, editors, *Proc. PPSN'92*, pages 259–268. Morgan Kaufmann, 1992.

[25] N. J. Radcliffe. Set recombination and its application to neural network topology optimisation. *Neural Computing and Applications*, 1(1):67–90, 1993.

[26] N. J. Radcliffe and P. D. Surry. Formal memetic algorithms. In T. Fogarty, editor, *Evolutionary Computing: AISB Workshop*, pages 1–16. Springer Verlag LNCS 865, 1994.

[27] N. J. Radcliffe and P. D. Surry. Fitness variance of formae and performance prediction. In L. D. Whitley and M. D. Vose, editors, *Foundation Of Genetic Algorithms 3*, pages 51–72. Morgan Kaufmann, 1995.

[28] N. J. Radcliffe and P. D. Surry. Real representations. In L. D. Whitley and R. K. Belew, editors, *Foundation Of Genetic Algorithms 4*, pages 51–72. Morgan Kaufmann, 1997.

[29] B. Sendhoff, M. Kreutz, and W. von Seelen. A condition for the genotype-phenotype mapping: Causality. In T. Bäck, editor, *Proc. ICGA'97*. Morgan Kaufmann, 1997.

[30] P. Surry. *A Prescriptive Formalism for Contructing Domain-specific Evolutionary Algorithms*. PhD thesis, University of Edinburgh, 1998.

[31] P. D. Surry and N. J. Radcliffe. RPL2: A language and parallel framework for evolutionary computing. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *PPSN'94*, pages 628–637. Springer Verlag, 1994.

[32] D. Wolpert and W. Macready. No Free Lunch Theorems for Search. Technical report, Santa Fe Institute, 1995.

[33] D. Wolpert and W. Macready. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.