Opt4J – A Modular Framework for Meta-heuristic Optimization

Martin Lukasiewycz
TU Munich, Germany
martin.lukasiewycz@rcs.ei.tum.de

Michael Glaß, Felix Reimann, Jürgen Teich
University of Erlangen-Nuremberg, Germany
{glass,felix.reimann,teich}@cs.fau.de

ABSTRACT

This paper presents a modular framework for meta-heuristic optimization of complex optimization tasks by decomposing them into subtasks that may be designed and developed separately. Since these subtasks are generally correlated, a separate optimization is prohibited and the framework has to be capable of optimizing the subtasks concurrently. For this purpose, a distinction of genetic representation (genotype) and representation of a solution of the optimization problem (phenotype) is imposed. A compositional genotype and appropriate operators enable the separate development and testing of the optimization of subtasks by a strict decoupling. The proposed concept is implemented as open source reference OPT4J [6]. The architecture of this implementation is outlined and design decisions are discussed that enable a maximal decoupling and flexibility. A case study of a complex real-world optimization problem from the automotive domain is introduced. This case study requires the concurrent optimization of several heterogeneous aspects. Exemplary, it is shown how the proposed framework allows to efficiently optimize this complex problem by decomposing it into subtasks that are optimized concurrently.

Categories and Subject Descriptors

G.3 [**Probability and Statistics**]: Probabilistic algorithms (including Monte Carlo); I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

General Terms

Design

Keywords

Optimization, Modular, Framework

1. INTRODUCTION

The domain of meta-heuristic optimization comprises methods such as Evolutionary Algorithms (EAs) [8], Simulated Annealing (SA) [11], Particle Swarm Optimization (PSO) [20], and Differential Evolution (DE) [27]. These methods are successfully applied to complex real-world problems from different domains where exact optimization techniques like Linear Programming (LP), Quadratic Programming (QP), or Geometric Programming (GP) are not applicable due to their restrictive expressiveness. The targeted problems of meta-heuristic optimization might comprise multiple nonlinear objectives or constraints. In contrast to common optimization benchmarks, the complexity of realistic problems might be also characterized by the problem structure where multiple heterogeneous subtasks have to be optimized concurrently. For instance, such a *complex optimization task* might be the optimization of automotive networks that requires the concurrent determination of the allocation of hardware resources, the binding of software tasks, the routing of messages, and the scheduling.

Numerous frameworks and libraries for meta-heuristic optimization have been provided that allow a separate development of optimization algorithms and optimization tasks. However, current frameworks do not support heterogeneous optimization tasks that require a concurrent optimization of correlated subtasks innately. Thus, the design and development of these complex optimization tasks is hampered. To close this gap, this paper presents a framework that is tailored to meta-heuristic optimization of complex optimization tasks by enabling the decomposition into subtasks. Since the subtasks cannot be optimized separately due to their correlation, the framework is capable of optimizing subtasks concurrently. As a proof of concept, a real-world automotive network optimization problem is introduced and implemented using the proposed framework.

Contributions of the Paper. This paper presents a modular framework for meta-heuristic optimization. It provides an efficient design and development approach for complex optimization tasks by decomposing these into correlated subtasks that are optimized concurrently. For this purpose, a strict distinction between the *genotype* and *phenotype* is imposed that separates genetic representation and solution representation of an optimization task. The optimization tasks are decomposed into subtasks that might be designed and developed separately. In order to enable this modular design of optimization tasks, compositional genotypes and appropriate operators are proposed.

A reference implementation of the proposed concept is presented. This implementation is written in Java and made publicly available, see OPT4J [6]. In order to support the modular development, the framework makes use of the dependency injection (DI) design pattern [16]. This design pattern separates the behavior from the dependency resolution and, thus, enables the implementation of the proposed concepts ideally by a reduced coupling. As a result, the configuration of the dependencies is done in separate modules. A Graphical User Interface (GUI) enables the selection and configuration of these modules and further enhances the swift development and testing.

The proposed framework is applied to a complex optimization task from the automotive domain. This Design Space Exploration (DSE) of an automotive network comprises the allocation of resources, binding of tasks, routing of messages, and the determination of a schedule. A separate optimization of these different

Supported in part by the German Ministry for Research and Education (BMBF Grant SEIS 01BV0910).

GECCO'11, July 12–16, 2011, Dublin, Ireland.

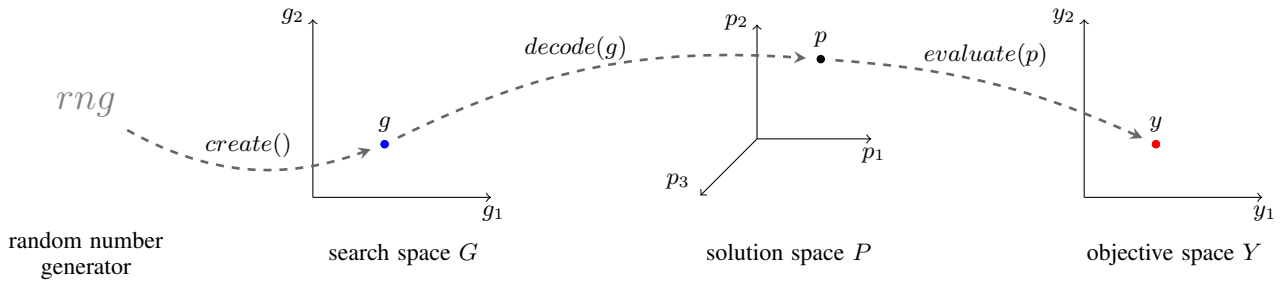

Figure 1: The $create()$ function creates a random genotype g in the search space G. For a genotype g, the $decode$ function determines a phenotype p in the solution space P. For a phenotype p, the $evaluate$ function determines the objectives y in the objective space Y.

aspects might lead to suboptimal solutions. It is shown how the entire optimization task is segregated into subtasks and a modular development is carried out using the proposed framework. Thus, several subtasks are designed separately such that multiple developers might work in parallel on subtasks or some subtasks might be replaced in the development process later without affecting any interdependencies.

Organization of the Paper. The remainder of the paper is outlined as follows: Section 2 discusses related work. Section 3 presents the concepts of the proposed framework. The reference implementation of the framework written in Java is outlined in Section 4. Exemplary as a case study, a real-world automotive optimization problem is implemented, using the proposed framework in Section 5. Finally, the paper is concluded in Section 6.

2. RELATED WORK

In the recent years, several interfaces, libraries, and frameworks for the development of meta-heuristic optimization algorithms and optimization tasks have been proposed. PISA [9] provides an interface concept to selection algorithms like the Strength Pareto Evolutionary Algorithm 2 (SPEA2) [31] or the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [12]. While interface concepts like PISA [9] are very flexible in terms of the target platform or programming language, each optimization algorithm or task has to be developed from the scratch. As a remedy, libraries and frameworks provide a set of predefined genotypes, operators, and algorithms to simplify the development. These libraries and frameworks are written in different programming languages such as Java [2, 4, 5, 7, 10, 14, 22, 25, 26, 29], C/C++ [23, 13, 19, 17, 1], C# [30], or Matlab [3].

Some of the referenced libraries and frameworks are restricted to a single meta-heuristic approach, optimize only single-objective problems, or have a restrictive or commercial license. Moreover, the existing libraries and frameworks are not tailored to complex optimization problems that require a concurrent optimization of multiple aspects. As a remedy, the proposed framework is capable of optimizing complex problems by providing compositional genotypes and operators. Thus, the implementation does not require boilerplate code as known frameworks and the development is significantly improved in terms of code complexity and development time. The provided reference implementation OPT4J is free and open source under LGPL [6]. Currently, the proposed framework is used by numerous researchers from different domains for applications in FPGA design, software development, similarity search, etc.

3. MODULAR FRAMEWORK

This section presents a modular framework for meta-heuristic optimization. Besides a clear separation between the *optimization algorithm* and *optimization task*, the goal is an eased development of the optimization of complex problems.

In the following, a distinction of genetic representation and one solution of the optimization is imposed. Thus, one individual consists of a genotype, a phenotype, and the evaluated objectives. A compositional concept is presented that allows the separate design and development of subtasks of the optimization task. This concept requires appropriate compositional operators that are capable of handling composite genotypes.

3.1 Genotype-Phenotype Distinction

In the proposed framework, a separation of the genetic representation, the *genotype*, and one solution of the optimization problem, the *phenotype*, is imposed. A decoder [21] is used to translate a genotype into a phenotype. The separation allows to compose the genotype from a set of predefined genotypes without reimplementing these or appropriate operators for each optimization task, respectively. Thus, only the problem-dependent phenotype as well as the decoder have to be specified.

One individual i is defined as 3-tuple consisting of the genotype g, the phenotype p, and the objectives y. Commonly, the objectives are defined in $\mathbb{R}^m$. However, the framework does not impose this restriction to allow also other types of objectives, e.g., to consider uncertainty [18]. The set of all genotypes is defined as *search space* G. The set of all phenotypes is defined as *solution space* P. The set of all objectives is defined as *objective space* Y. Here, function $create : \emptyset \rightarrow G$ generates a random genotype $g \in G$, function $decode : G \rightarrow P$ translates a genotype $g \in G$ into a phenotype $p \in P$, and function $evaluate : P \rightarrow Y$ determines the objectives of a phenotype. An overview of this concept is illustrated in Fig. 1.

A distinction of genotype and phenotype enables the optimization of complex problems where the genotype has to be processed to become a feasible solution. This holds also for repair algorithms that use a local search method to improve obtained solutions. A further advantage of the separation of genotype and phenotype is an improved maintainability and extensibility of the optimization problem. In particular, the modification of the genetic representation becomes possible without affecting the evaluation function $evaluate$.

3.2 Composite Genotypes

Many complex optimization tasks might be decomposed into subtasks. In general, these subtasks cannot be optimized separately due to their mutual correlation. To cope with such a high problem complexity, a compositional approach is provided. Here, the genotype, the $create$, and the $decode$ function can be composed in a treelike structure. This significantly reduces the coupling and improves the flexibility such that subtasks of the optimization problem can be developed and tested separately. In a final step, the separately developed parts are merged seamlessly to the entire optimization task.

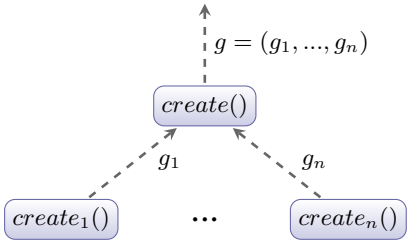

Figure 2: Composite $create$ function.

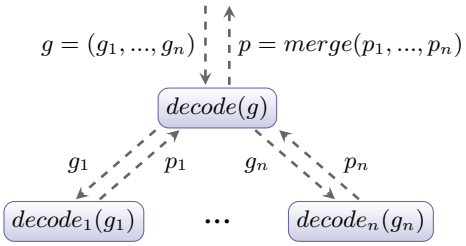

Figure 3: Composite $decode$ function.

A genotype might be a composite of multiple genotypes, i.e., $g = (g_1, ..., g_n)$. Thus, a compositional treelike structure is enabled. The search space for a composite genotype is defined as $G = G_1 \times ... \times G_n$. The basic genotypes might be vectors of real values, integers, binary values, permutations, or other data structures.

This also enables a compositional development of $create$ and $decode$ functions, i.e., a $create$ and $decode$ function for each subtask of a complex optimization tasks. A compositional $create$ function is illustrated in Fig. 2 and given as follows:

$$create() = (create_1(), ..., create_n()) \tag{1}$$

The returned genotypes of each subtask are put together to a composite genotype.

A compositional $decode$ function is illustrated in Fig. 3 and given as follows:

$$decode(g) = merge(decode_1(g_1), ..., decode_n(g_n)) \tag{2}$$

The genotypes are delegated to each correspondent $decode$ function to determine the phenotypes. The $merge$ function is problem-specific and combines the sub-phenotypes $p_1, ..., p_n$ appropriately to p.

Here, each $create$ and $decode$ function for a subtask might be developed separately. This improves the maintainability of complex optimization tasks since each sub-genotype is fully isolated in a branch of the composite genotype.

3.3 Composite Operators

In order to leverage the composite genotype, operators become necessary that are capable of handling these composite genotypes. In case g is not composite, a dispatcher applies the appropriate base operator. The dispatcher function is applied to each leaf of the composite genotype tree to determine the specific operator.

In the following, several standard operators for composite genotypes are outlined:

$$size(g) = \sum_{i=1}^{n} size(g_i) \tag{3}$$

$$mutate(g, p) = (mutate(g_1, p), .., mutate(g_n, p)) \tag{4}$$

$$crossover(g, \tilde{g}) = (crossover(g_1, \tilde{g_1}), .., crossover(g_n, \tilde{g_n})) \tag{5}$$

$$neighbor(g) = (g_1, .., neighbor(g_i), .., g_n) \tag{6}$$
$$\text{with probability } p_i = \frac{size(g_i)}{size(g)}$$

$$algebra(g[], f) = (algebra(g[]_1, f), .., algebra(g[]_n, f)) \tag{7}$$

$$copy(g) = (copy(g_1), .., copy(g_n)) \tag{8}$$

$$diversity(g) = \sum_{i=1}^{n} \frac{diversity(g_i) \cdot size(g_i)}{size(g)} \tag{9}$$

The $size$ operator in Eq. (3) is required by many other operators. Here, the size of the genotype is determined by the sum of the genotype sizes of the leafs. The $mutate$ and $crossover$ operators in Eq. (4) and Eq. (5) are necessary for Evolutionary Algorithms (EAs). Here, the $mutate$ operator changes an existing genotype with a mutation rate p. The $crossover$ operator creates a new genotype from two existing genotypes and might be generalized to handle an arbitrary number of genotypes. The $neighbor$ operator in Eq. (6) is required by optimization algorithms such as Simulated Annealing (SA) or Tabu Search (TS). Here, the operator is applied to one child of the composite genotype with a probability that is determined by the ratio of the size of the child to the size of the composite genotype. The $algebra$ operator in Eq. (7) is used by optimization algorithms such as Particle Swarm Optimization (PSO) or Differential Evolution (DE). Here, $g[]$ indicates a list of genotypes and f is a linear function that is applied to this list to determine a new $\tilde{g}$. The $copy$ and $diversity$ operators in Eq. (8) and Eq. (9), respectively, might be used for local search algorithms.

4. REFERENCE IMPLEMENTATION

The proposed concepts have been implemented in the reference implementation OPT4J [6]. This reference implementation is written in Java and open source licensed under LGPL. The goal of the framework is to enable a development of optimization algorithms, operators, and optimization tasks, avoiding boilerplate code. In the following, the basic architecture of this framework and the module-based configuration approach based on the dependency injection (DI) design pattern is presented.

4.1 Basic Architecture

The basic architecture is outlined in Fig. 4. An optimization task is defined by implementing the interfaces Creator, Decoder, and Evaluator. Thus, the *create*, *decode*, and *evaluate* functions are defined implicitly.

An optimization algorithm is implemented by the interface Optimizer. The Optimizer manipulates the Population and Archive. The Population contains the Individuals of the current iteration, while the Archive contains the non-dominated Individuals that were obtained throughout the optimization process. One Individual contains the Genotype, the Phenotype, and the Objectives. The Optimizer might use any Operator to vary the Genotype and use the IndividualFactory to build new Individuals. Note that instead using the Decoder and Evaluator directly, an IndividualCompleter is used to determine the Phenotype and Objectives of a set of Individuals. In most real-world problems, the decoding and the objectives evaluation are the most

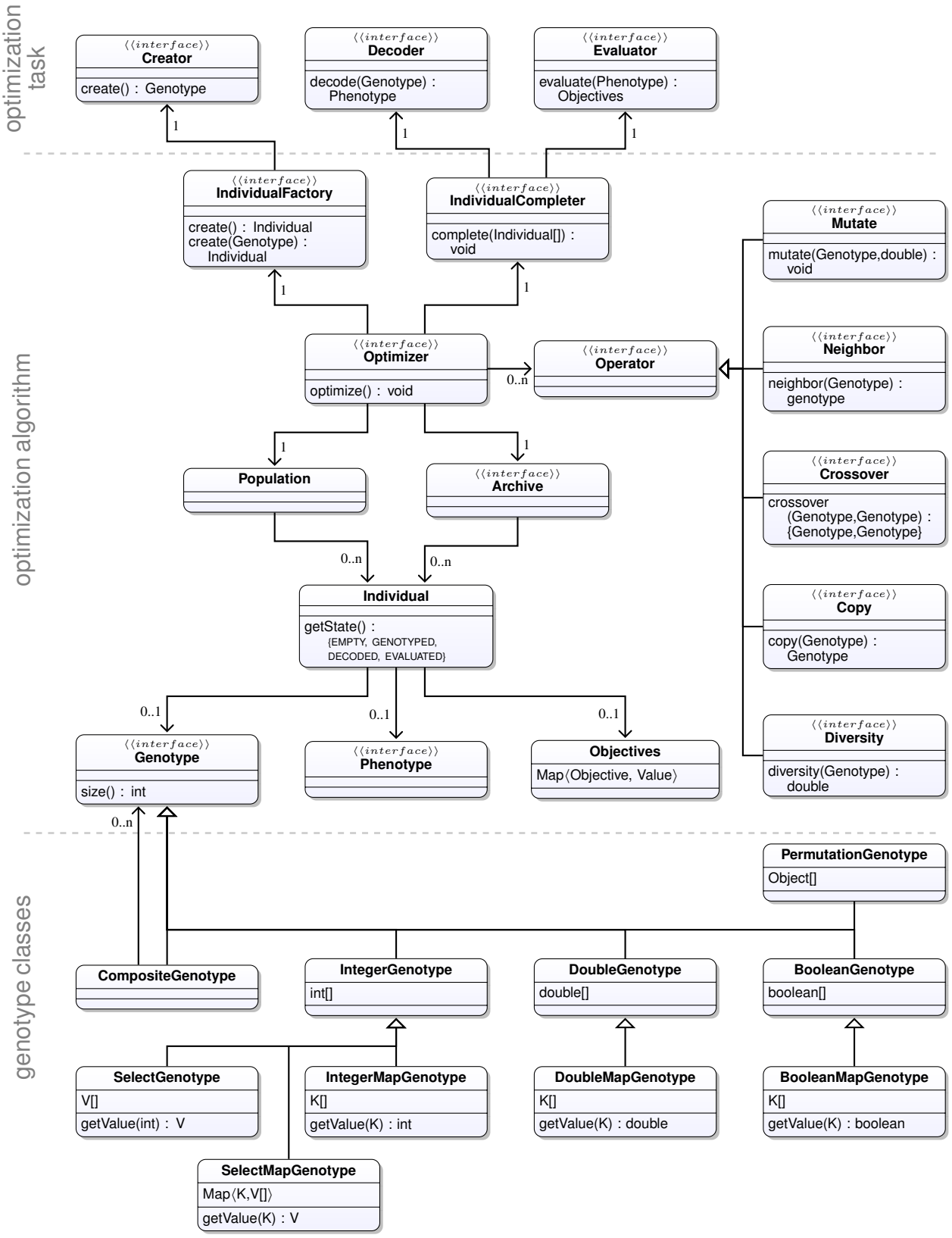

Figure 4: Overview of the architecture of the reference implementation OPT4J [6].

time-consuming tasks. In order to do this efficiently in a concurrent manner, the IndividualCompleterParallel is an implementation of the IndividualCompleter and capable of performing this task in parallel to leverage modern multi-core systems. Alternatively, this task could be even carried on a cluster of multiple machines. Additionally, the Optimizer can use algorithm-specific operators such as a Selector, e.g., NSGA-II or SPEA2, to perform the selection after each iteration.

The composite genotype concept is implemented by the CompositeGenotype that can contain multiple genotypes. Here, a set of several useful base genotypes is provided such as vectors of integer values (IntegerGenotype), real values (DoubleGenotype), binary values (BooleanGenotype), or permutations (PermutationGenotype). Additionally, for each base genotype, a Map functionality is implemented to simplify its usage by extending the index-based access to a key-based access.

4.2 Module-based configuration

To achieve a high degree of flexibility and reduced coupling, the implementation is using the dependency injection (DI) [16] framework GUICE [28]. DI is a design pattern that is responsible for wiring the elements of the software architecture together based on their dependencies. Thus, for each interface of the architecture in Fig. 4, an appropriate implementation has to be defined by a so-called *binding*. For instance the EvolutionaryAlgorithmModule internally binds the Optimizer interface to the EvolutionaryAlgorithm implementation. Correspondingly, the optimization task is configured by binding the Creator, Decoder, and Evaluator to appropriate implementations. Note that for each other interface there exists a default implementation, for instance the Archive is by default implemented by a CrowdingArchive with a maximal size of 100 using the crowding distance [12].

Since the modules configure the binding of the implementations to the interfaces, the current configuration is done by a selection of modules. Additionally, the modules allow to set parameters. To further improve the testing and configuration, the implemented framework comprises a Graphical User Interface (GUI) for the selection and configuration of these modules as illustrated in Fig. 5. This GUI automatically lists all available modules and their methods such that no user code is necessary to obtain the visualization. Moreover, each configuration might be saved in or loaded from *XML* files, allowing also an execution of the optimization process from the command line without the GUI.

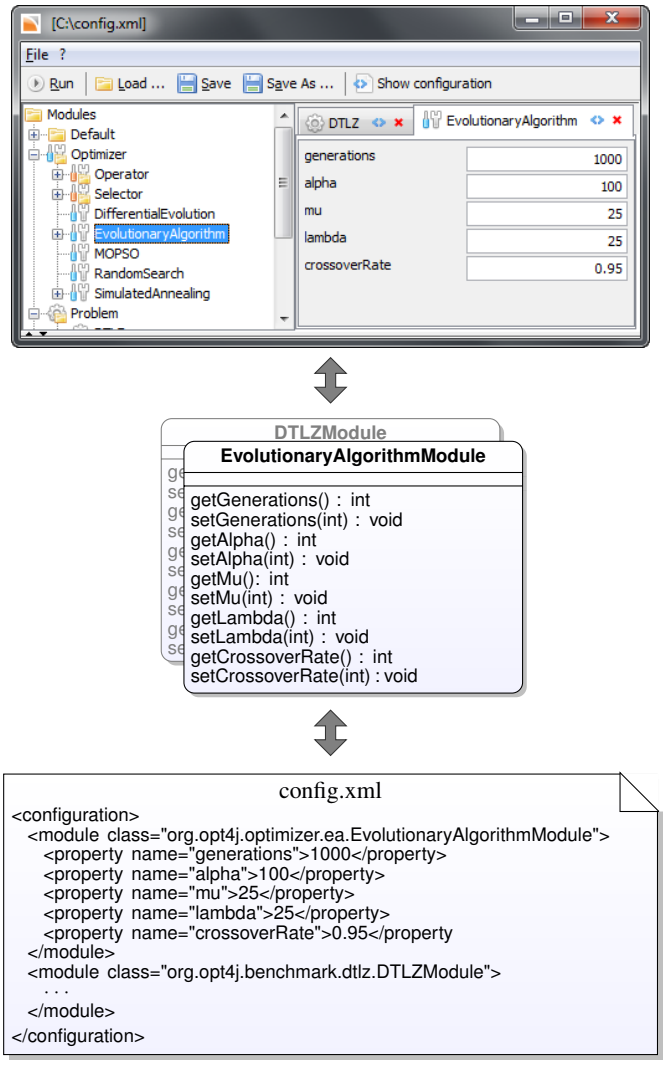

Figure 5: Configuration GUI for the modules of the reference implementation.

5. CASE STUDY

In the following, an optimization task from the automotive domain is presented. The proposed framework is used to decompose the complex problem into subtasks. It is shown how these subtasks might be combined to achieve an effective optimization.

5.1 Problem Definition

The Design Space Exploration (DSE) of automotive networks is a challenging optimization task. Commonly, the DSE is performed in a graph-based manner using the widely accepted Y-chart approach as illustrated in Fig. 6. Here, a specification consisting of the application, architecture, and mappings is used to obtain an implementation x (the phenotype) by performing four tasks: The allocation of resources, the mapping of processes, the routing of messages, and the scheduling.

The specification consists of an architecture graph $\mathcal{G}_R$, an application graph $\mathcal{G}_T$, and the mapping edges $\mathcal{E}_M$:

- The architecture is given by a directed graph $\mathcal{G}_R(R, \mathcal{E}_R)$. The vertices R represent resources such as Electronic Control Units (ECUs), buses, and gateways. The directed edges $\mathcal{E}_R$ indicate available communication connections between two resources.
- The application is given by a bipartite directed graph $\mathcal{G}_T(T, \mathcal{E}_T)$ with $T = P \cup C$. The vertices T are either processes $p \in P$ or messages $c \in C$. Each edge $e \in \mathcal{E}_T$ connects a vertex in P to one vertex in C, or vice versa. Each process might send one or more messages. To allow multi-cast communication, each message might have multiple successor process tasks.
- The set of mappings $\mathcal{E}_M$ contains the mapping information for the processes. Each mapping $m = (p, r) \in \mathcal{E}_M$ indicates a possible implementation of process $p \in P$ on resource $r \in R$.

One implementation consists of the allocation graph $\mathcal{G}_\alpha$ and the binding $\mathcal{E}_\beta$. Additionally, for each message $c \in C$ a sub-graph of the allocation $\mathcal{G}_{\gamma,c}$ is determined that fulfills the data dependencies such that the communication is established between each sender process and the corresponding receiver processes. Moreover, priorities for each message and process task have to be determined for the event-triggered scheduling:

- The allocation is a directed graph $\mathcal{G}_\alpha(\alpha, \mathcal{E}_\alpha)$ that is an induced sub-graph of the architecture graph $\mathcal{G}_R$.

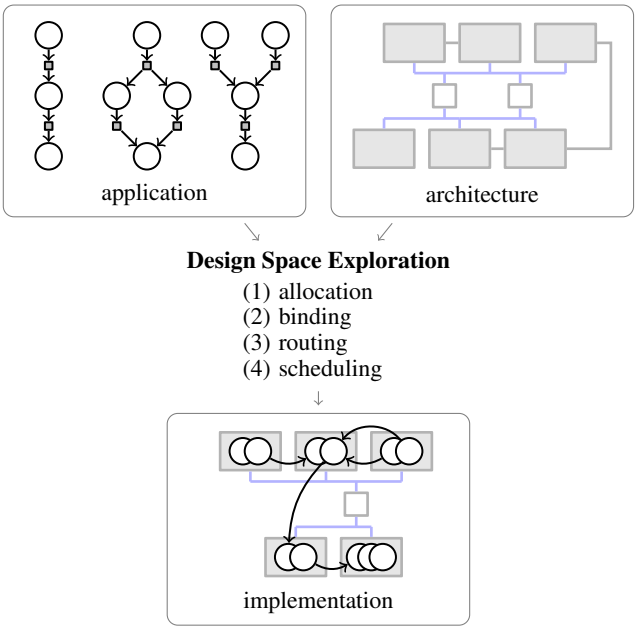

Figure 6: Illustration of the Y-chart approach for a DSE of embedded systems. The mapping of an application to an architecture results in an implementation.

- The binding $\mathcal{E}_\beta$ is deduced from $\mathcal{E}_M$. Each process $p \in P$ in the application is bound to exactly one allocated resource.
- Each message in $c \in C$ is routed on a tree $\mathcal{G}_{\gamma,c}$ that is a subgraph of the allocation $\mathcal{G}_\alpha$. For each message, the following two conditions have to be satisfied: (1) The root of the routing has to equal the target resource of the predecessor sender process. (2) The message has to be routed on target resources of the successive processes.
- The priorities are determined by a function $prio : P \cup C \rightarrow \mathbb{N}$. In general, default priorities are given.

An implementation is termed *feasible* if it satisfies all data dependencies of the application. Additionally, stringent real-time and processor load constraints have to be satisfied. The objectives in the work at hand are the average energy consumption in *Ampere* (for 12 *Volts*) and the hardware costs in *Euro*.

5.2 Implementation

The presented DSE is implemented using the proposed optimization framework. To this end, the design decision is made to separately develop the optimization problem for allocation, binding, routing (task a) and scheduling (task b). For task a, two different problem representations are developed separately.

Optimization Task $a1$. First, an approach $a1$ based on a binary encoding of the allocation and priority lists for the task mapping is used. The genotype additionally has a priority list for the allocation to allow a local repair approach to minimize the number of infeasible implementations. The routing is determined by shortest path routing. The genotype of this approach is defined as follows:

$$g_{a1} \in G_{a1} = \{0,1\}^{|R|} \times \mathbb{N}^R \times \mathbb{N}^{M_{p_1}} \times ... \times \mathbb{N}^{M_{p_n}} \quad (10)$$

Here, $M_{p_x} = \{m \mid m = (p_x, r) \in M\}$ is the set of mappings with the source process p_x.

Optimization Task $a2$. The second approach is using the *SAT decoding* [24] approach to avoid infeasible implementations. Here, the problem is encoded into a set of linear constraints and binary variables. In the following, a binary search problem is defined such that a solution $\mathbf{x} \in \{0,1\}^n$ corresponds to a *feasible* implementation x. The symbolic encoding consists of the following binary variables:

- $\mathbf{r}$ binary variable for each resource $r \in R$ indicating whether this resource is in the allocation α (1) or not (0)
- $\mathbf{m}$ binary variable for each mapping $m \in E_M$ indicating whether the mapping edge is in E_β (1) or not (0)
- $\mathbf{c_r}$ binary variable for each message $c \in C$ and the available resources $r \in R$ indicating whether the message is routed on the resource (1) or not (0)
- $\mathbf{c_{r,t}}$ binary variable for each message $c \in C$ and resource $r \in R$ indicating on which communication step $t \in \mathcal{T} = \{1, .., |\mathcal{T}|\}$ (message are propagated in steps or hops, respectively) it is routed on the resource

The linear constraints are formulated as follows:

$\forall p \in P$:
$$\sum_{m=(p,r)\in E_M} \mathbf{m} = 1 \quad (11)$$

$\forall m = (p,r) \in \mathcal{E}_M$:
$$\mathbf{r} - \mathbf{m} \geq 0 \quad (12)$$

$\forall c \in C, r \in R, (c,p) \in \mathcal{E}_T, m = (p,r) \in \mathcal{E}_M$:
$$\mathbf{c_r} - \mathbf{m} \geq 0 \quad (13)$$

$\forall c \in C$:
$$\sum_{r\in R} \mathbf{c_{r,1}} = 1 \quad (14)$$

$\forall c \in C, r \in R, (p,c) \in \mathcal{E}_T, m = (p,r) \in \mathcal{E}_M$:
$$\mathbf{m} - \mathbf{c_{r,1}} = 0 \quad (15)$$

$\forall c \in C, r \in R$:
$$\sum_{t\in\mathcal{T}} \mathbf{c_{r,t}} \leq 1 \quad (16)$$

$\forall c \in C, r \in R$:
$$\left(\sum_{t\in\mathcal{T}} \mathbf{c_{r,t}}\right) - \mathbf{c_r} \geq 0 \quad (17)$$

$\forall c \in C, r \in R, t \in \mathcal{T}$:
$$\mathbf{c_r} - \mathbf{c_{r,t}} \geq 0 \quad (18)$$

$\forall c \in C, r \in R, t = \{2, .., |\mathcal{T}|\}$:
$$\left(\sum_{\tilde{r}\in R, e=(\tilde{r},r)\in\mathcal{E}_R} \mathbf{c_{\tilde{r},t}}\right) - \mathbf{c_{r,t+1}} \geq 0 \quad (19)$$

$\forall c \in C, r \in R$:
$$\mathbf{r} - \mathbf{c_r} \geq 0 \quad (20)$$

$\forall r \in R$:
$$\left(\sum_{c\in C\wedge r\in R} \mathbf{c_r}\right) + \left(\sum_{m=(p,r)\in\mathcal{E}_M} \mathbf{m}\right) - \mathbf{r} \geq 0 \quad (21)$$

The constraints in Equation (11) and (12) fulfill the binding of each process to exactly one resource and the requirement that processes are only bound to allocated resources, respectively. The requirement that a message has to be routed on each target resource of the successive process is fulfilled by the constraints in Equation (13). The constraints in Equation (14) and (15) imply that each message has exactly one root that equals the target resource of the predecessor mapping. The constraints in Equation (16) ensure

that a message can pass a resource at most once such that cycles are prohibited. A message has to be existent in one communication step on a resource in order to be correctly routed on this resource as implied by the constraint in Equation (17) and (18). The constraints in Equation (19) state that a message may be routed only between adjacent resources in one communication step. In order ensure that the routing of each message is a sub-graph of the allocation, each message can be only routed on allocated resources as stated in the constraints in Equation (20). Additionally, the constraints in Equation (21) ensure that a resource is only allocated if it is used by at least one process or message such that suboptimal implementations are removed effectively from the search space. This minimizes the resulting allocation by redundant resources such that additional unnecessary costs are prohibited.

A backtracking *PB solver* [15] implements the *decode* function to obtain a feasible implementation by using the genotype as branching strategy, see [24]. Thus, the genotype defines a binary phase and priority as real value for each variable and looks as follows:

$$g_{a2} \in G_{a2} = \{0,1\}^{|R|+|M|+|R|\cdot|C|} \times \mathbb{R}^{|R|+|M|+|R|\cdot|C|} \quad (22)$$

Given a single solution $\mathbf{x}$ of this linear search problem, the corresponding implementation x is deduced. The allocation $\mathcal{G}_\alpha$ is deduced from the variables $\mathbf{r}$ and the binding $\mathcal{E}_\beta$ from the variables $\mathbf{m}$. For each message $c \in C$, the routing $\mathcal{G}_{\gamma,c}$ is deduced from the variables $\mathbf{c_r}$ and $\mathbf{c_{r,t}}$.

Optimization Task b. For task b, the priorities of the tasks and messages are determined separately using permutations. Thus, the genotype is defined as follows:

$$g_b \in G_b = \mathbb{N}^P \times \mathbb{N}^C \quad (23)$$

Concurrent Optimization. For each task $a1$, $a2$, b, the appropriate *create* and *decode* functions have to be determined. The *create* functions are defined implicitly by the required genotypes. The *decode* functions for $a1$ and $a2$, respectively, return an implementation $x \in X = P$. The *decode* function b returns a function $prio$ that determines the priorities of tasks and messages.

Since the *evaluate* function determines the objectives for an implementation x, either $a1$ or $a2$ can be used as optimization problem since both decode directly to the solution space P. Here, the priorities for the tasks and messages are predefined default values. To further improve the optimization, either $a1$ or $a2$ can be combined with b such that all four tasks allocation, binding, routing, and scheduling are determined concurrently. By optimizing the priorities, the end-to-end delays of the application are minimized such that the search space is increased by additional feasible implementations. Fig. 7 and Fig. 8 show the composite *create* and *decode* function, respectively. The *merge* method of the top-level *decode* function assigns the priorities of *prio* to the corresponding tasks and messages in the implementation x.

Results. For the case study, a specification that models a automotive sub-network is used. This specification consists of several safety, comfort, and driver assistance applications that are mapped to multiple ECUs, three buses, and one gateway. The architecture consists of 35 resources, the application comprises 104 processes and 61 messages, and the number of mappings is 192. This results in a large search space, containing 2^{227} solutions for allocation and binding as subtasks alone.

Fig. 9 shows the results that are obtained for the case study using an EA with 5000 objective function evaluations. Here, all four different possible combinations of $a1$, $a2$, and b are presented. Note that exchanging $a1$ by $a2$ as well as using b always improve the optimization. Thus, $a2$ together with b delivers the best results.

Fig. 10 shows the GUI that is provided by the framework to monitor and control (by pausing or stopping) the optimization process.

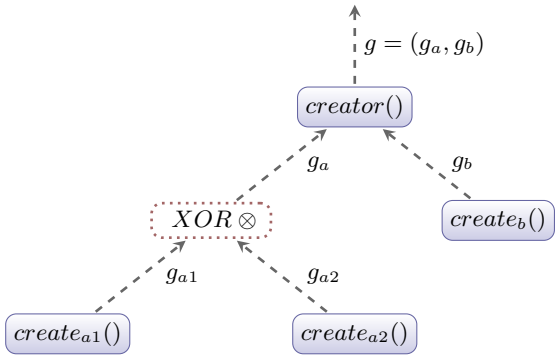

Figure 7: The entire $create$ function of the case study.

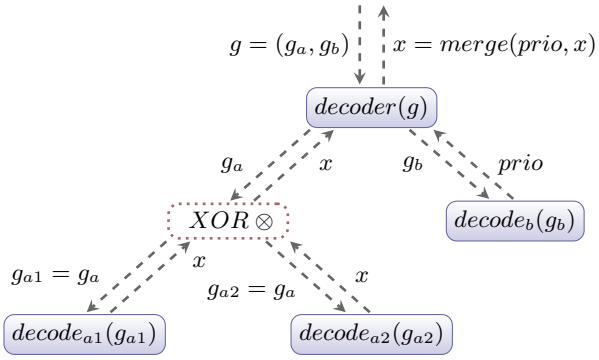

Figure 8: The entire $decode$ function of the case study.

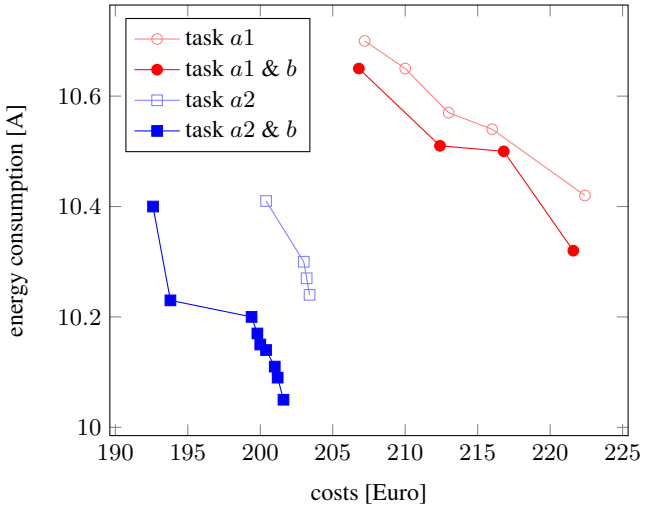

Figure 9: Results of the case study optimization task. Here, $a1$, $a2$, and b are subtasks that are appropriately combined.

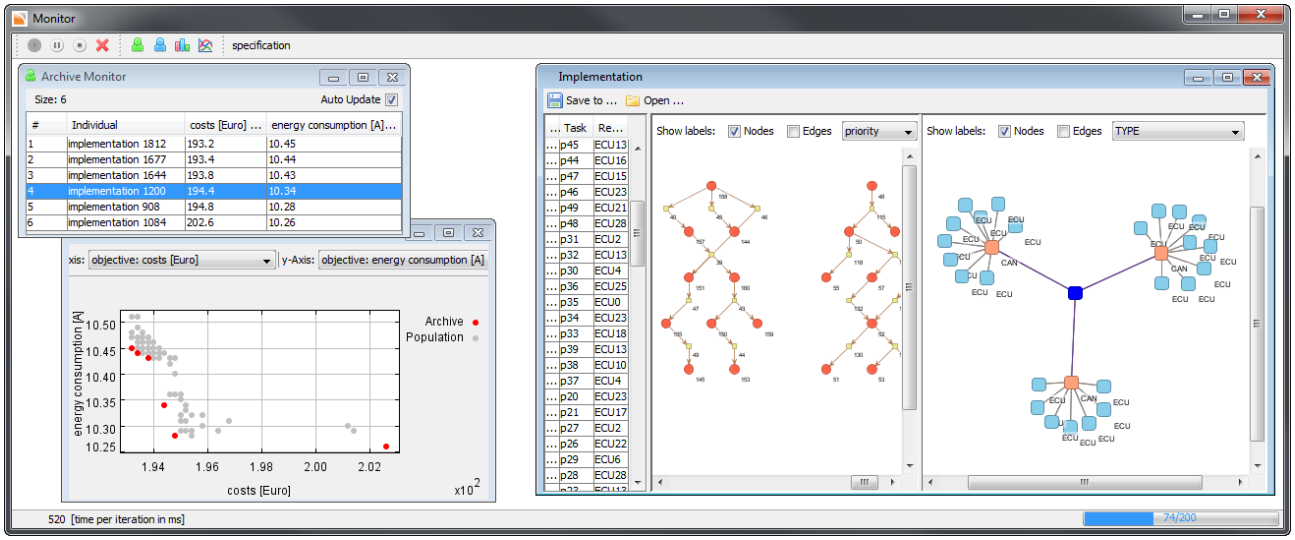

Figure 10: The monitor for the optimization process of the framework showing the archive and the two-dimensional projection of the archive and population of the current iteration. Additionally, each implementation from the archive can be viewed.

Additionally, for this case study, a visualization that displays an implementation x was implemented.

6. CONCLUSION

This paper presents a framework for the optimization of tasks where multiple heterogeneous aspects have to be optimized concurrently. A concept based on the separation of genotype and phenotype as well as a corresponding compositional approach are presented. Thus, the optimization task might be determined in a compositional manner. A corresponding open source reference framework [6] is presented. Using the proposed concept and reference framework, an optimization task from the automotive domain is exemplary implemented. This complex optimization problem is decomposed into subtasks that are optimized concurrently. This enables a separate design, development, and testing as well as a smooth iterative improvement of genetic representations and the corresponding interpretation.

7. REFERENCES

[1] ECF – Evolutionary Computation Framework. `http://gp.zemris.fer.hr/ecf/`.

[2] ECJ. `www.cs.gmu.edu/~eclab/projects/ecj/`.

[3] GPLAB – A Genetic Programming Toolbox for MATLAB. `http://gplab.sourceforge.net`.

[4] JAGA–Java API for Genetic Algorithms. `http://www.jaga.org`.

[5] JGAP–Java Genetic Algorithms and Genetic Programming Package. `http://jgap.sourceforge.net`.

[6] Opt4J. `http://www.opt4j.org`.

[7] Watchmaker. `http://watchmaker.uncommons.org/`.

[8] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, USA, 1996.

[9] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA—A Platform and Programming Language Independent Interface for Search Algorithms. In *Conference on Evolutionary Multi-Criterion Optimization (EMO 2003)*, pages 494–508, 2003.

[10] J. Brownlee. OAT: The Optimization Algorithm Toolkit. Technical report, Faculty of Information and Communication Technologies, Swinburne University of Technology, 2007.

[11] V. Černỳ. Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, 1985.

[12] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II. In *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature (PPSN 2000)*, pages 849–858, 2000.

[13] L. Di Gaspero and A. Schaerf. EASYLOCAL++: An Object-oriented Framework for the Flexible Design of Local-search Algorithms. *Software: Practice and Experience*, 33(8):733–765, 2003.

[14] J. Durillo, A. Nebro, and E. Alba. The jMetal Framework for Multi-Objective Optimization: Design and Architecture. In *IEEE Congress on Evolutionary Computation 2010*, pages 4138–4325, Barcelona, Spain, 2010.

[15] N. Eén and N. Sörensson. Translating Pseudo-Boolean Constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

[16] M. Fowler. Inversion of Control Containers and the Dependency Injection Pattern, 2004.

[17] C. Gagné and M. Parizeau. Open BEAGLE: A C++ Framework for your Favorite Evolutionary Algorithm. *ACM SIGEVOlution*, 1(1):12–15, 2006.

[18] Y. Jin and J. Branke. Evolutionary Optimization in Uncertain Environments - A Survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–317, 2005.

[19] M. Keijzer, J. Merelo, G. Romero, and M. Schoenauer. Evolving objects: A General Purpose Evolutionary Computation Library. In *Artificial Evolution*, pages 829–888, 2002.

[20] J. Kennedy and R. C. Eberhart. Particle Swarm Optimization. In *Proceedings of the IEEE International Conference on Neural Networks (ICNN 1995)*, pages 1942–1948, 1995.

[21] S. Koziel and Z. Michalewicz. A Decoder-Based Evolutionary Algorithm for Constrained Parameter Optimization Problems. In *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature (PPSN 1998)*, pages 231–240, 1998.

[22] M. Kronfeld, H. Planatscher, and A. Zell. The EvA2 Optimization Framework. *Learning and Intelligent Optimization*, pages 247–250, 2010.

[23] A. Liefooghe, M. Basseur, L. Jourdan, and E. Talbi. ParadisEO-MOEO: A Framework for Evolutionary Multi-objective Optimization. In *IEEE Congress on Evolutionary Computation 2007*, pages 386–400, 2007.

[24] M. Lukasiewycz, M. Glaß, C. Haubelt, and J. Teich. SAT-Decoding in Evolutionary Algorithms for Discrete Constrained Optimization Problems. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2007)*, pages 935–942, 2007.

[25] J. Parejo, J. Racero, F. Guerrero, T. Kwok, and K. Smith. FOM: A Framework for Metaheuristic Optimization. *Computational Science - ICCS*, pages 721–721, 2003.

[26] A. Rummler and G. Scarbata. eaLib - A Java Frameword for Implementation of Evolutionary Algorithms. *Computational Intelligence. Theory and Applications*, pages 92–102, 2001.

[27] R. Storn and K. Price. Differential Evolution - A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces. *Technical Report TR-95-012*, 1995.

[28] R. Vanbrabrant. *Google Guice: Agile Lightweight Dependency Injection Framework*. Springer, 2008.

[29] S. Ventura, C. Romero, A. Zafra, J. Delgado, and C. Hervás. JCLEC: A Java Framework for Evolutionary Computation. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 12(4):381–392, 2008.

[30] S. Wagner and M. Affenzeller. HeuristicLab: A Generic and Extensible Optimization Environment. *Adaptive and Natural Computing Algorithms*, pages 538–541, 2005.

[31] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In *Evolutionary Methods for Design, Optimisation, and Control*, pages 19–26, 2002.