Proceedings of the 2014 Federated Conference on Computer Science and Information Systems pp. 501–509
DOI: 10.15439/2014F208
ACSIS, Vol. 2

An Application of Developmental Genetic Programming for Automatic Creation of Supervisors of Multi-task Real-Time Object-Oriented Systems

Krzysztof Sapiecha
Department of Computer Engineering
Cracow University of Technology
Cracow, Poland
Email: krzysztof.sapiecha@gmail.com

Leszek Ciopiński
Department of Computer Science
Kielce University of Technology
Kielce, Poland
Email: l.ciopinski@tu.kielce.pl

Stanisław Deniziak
Department of Computer Science
Kielce University of Technology
Kielce, Poland
Email: s.deniziak@tu.kielce.pl

***Abstract*—A concept of artificial supervisor of multi-task real-time object-oriented system is introduced. Next, a procedure for automatic creation of artificial supervisors is presented. The procedure is based on developmental genetic programming. As an input data, UML diagrams are used. A representative example of creation of a supervisor of building a house illustrates the procedure. The efficiency of the procedure from various points of view and comparison considerations are given.**

I. Introduction

A SYSTEM must meet user requirements and constraints. Besides specified functionalities, cost effectiveness and high performance are among the most important ones. Real-time (RT) systems are present in all areas of human life. Punctuality is an extra requirement for them. Sometimes punctuality requires very high performance. However, the higher performance the higher cost of the system. Usually the cost is limited.

Going into details, one can find RT systems in civil engineering, in traveling, in computer engineering, in banking, and so on. In the first case, a building enterprise is such a system. It owns resources, such as workers and building machinery, necessary to build a house according to the requirements of a client. These usually comprise functionalities of the house, its cost and a deadline. In the second case a human being is an RT system. He knows which means of transportation may use to meet his requirements. From among flights, trains, buses, rented cars, and even walking he selects a set, so that to reach a target on time and at affordable cost. An embedded computer system may be another example of RT system. A designer of such a system has to decide what of the tasks of the system is to allocate to what of its processing components, so that to get maximum of the performance and not to surpass the cost. In a bank an account may be operated in different ways. However, an owner of the account usually wants to get maximum profit with an acceptable risk in a specific time period. Summarizing, an RT system uses some hardware or software objects of its resources so that a specific goal is achieved, on time.

Multi-task system (MS) is a system where more than one task can be processed at the same time. A home computer is a familiar example of the MS. Common tasks are word processing, printing, communicating, and playing games. The system contains objects: hard drive, a monitor, a printer, a network adapter and an optical drive. Some of these objects are required for a subset of the tasks while others are required for all these tasks. The monitor and hard drive will always be in use whereas the printer is used only for printing, the network adapter is used for communicating, and the optical drive is used for reading stored materials. The enterprise is an example of RT MS, while a traveler is not.

Usually, RT systems should be optimized for cost vs. speed of operation (speed of reaching a goal or a target). Therefore, a building enterprise, and a traveler, and hardware/software system designer, and other RT systems have to be endowed with optimization engine. We will call these engines: artificial supervisors (AS) or artificial managers (AM) of resources. An AS should find an optimum use of supervised resources, taking into account the requirements and the constraints. This means that the AS decides what functionalities should be allocated to what resources and in which order these functionalities should be executed. Actually, it has to find a solution for a specific case of the well-known Resource-Constrained Project Scheduling Problem (RCPSP) which consists in rescheduling the project tasks (RT system tasks) efficiently using limited renewable resources (components/objects of the RT system) minimizing the maximal completion time of all activities [1].

The RCPSP is an NP-complete problem which is computationally very hard [2] [3]. Möhring [4] states that it is one of the hardest problems of Operational Research. Therefore, a skilled specialist with an assistance of the planner (Computer Aided System Engineering in case of the enterprise) might play a role of such an AS only for small systems containing a limited number of tasks and a moderate number of resources. No doubt, in case of real life systems, particularly RT MS, the AS must be a very powerful optimization engine.

The general RCPSP model cannot cover all situations that occur in practice. Therefore, many researchers have developed many variants and extensions of project scheduling problems, often using the standard RCPSP as a starting point [5].

978-83-60810-58-3/$25.00

Constructing an efficient AS for a given class of RCPSP problems is very difficult and time consuming. Moreover, a scheduling strategy that is optimal for one problem may not be efficient for others. Hence, instead of developing a general AS for all RT MS, in this paper, we propose a method that automatically generates a dedicated AS for the specific RT MS. The method is based on an idea derived from developmental genetic programming (DGP). It is universal and can be applied to optimization of RT MS of any kind. Our methodology is illustrated and evaluated with the help of a representative example.

Genetic programming (GP) is an extension of the genetic algorithm [6], in which the population consists of computer programs. In the DGP [7] [8], strategies that create solutions evolve, instead of computer programs. In this approach a genotype and a phenotype are distinguished. The genotype is a procedure that constructs a solution of the problem. It is composed of genes representing elementary functions, constructing the solution. The phenotype represents a target solution. During evolution, only genotypes are evolved, while genotype-to-phenotype mapping is used in the fitness computation, which is required for the genotype selection process. Next, all genotypes are rated according to an estimated quality of the corresponding phenotypes. The goal of the optimization is to find the procedure constructing the best solution. The idea is based on the theory from the molecular biology, concerning protein synthesis that produces proteins (phenotype) from the DNA (genotype). In our approach the AS corresponds to the genotype while the phenotype is the solution i.e. a makespan. First, the DGP is used to find the optimal solution, and the genotype constructing this solution is saved as the AS.

The method is universal, but an AS must be well-fitted to a particular RT MS. The RT MS is a micro-world with its own functionalities and resources. Therefore, a formal specification of the RT MS is an input data to the method. RT MS, where a number of resources have punctually to execute a number of tasks are good micro-worlds for object-oriented modeling. Objects may play a role of resources that execute tasks in real time for some costs. A widely accepted standard for modeling object-oriented systems (OOSs) is the Unified Modeling Language (UML) [9]. It shows how to write a system's blueprints, including conceptual things such as business processes and system functions. It encompasses OOSs of any kind, particularly real-time multi-task OOSs (RT MOOSs). Using the UML for modeling RT OOS has been a subject of many publications [10] [11] [12]. Hence, this will be applied here.

Related work is briefly described in section II. In section III the problem is stated. Section IV briefly shows how early UML models should be used as input data for the method, and section V explains how DGP can create the supervisors and the initial solutions. In section VI a computational experiment evaluating our approach is described. The experiment explains of how a supervisor of a simple RT MOOS (of building a house) is created. Finally, section VII contains conclusions.

II. Related Work

Genetic approach was proved as very efficient for solving RCPSP problems. Ones of the most efficient genetic algorithms for RCPSP are presented in [13] [14]. In [15] the method of improving the genetic algorithm for optimization of multi-task project scheduling was proposed. It was showed that the method is competitive in comparison with 11 other heuristic approaches. A method of solving a large scale RCPSP is presented in [16]. In this solution, a genetic algorithm is used and a method of encoding classical RCPSP problem in the chromosome is described. Results achieved by authors of [16] give a slight improvement, in comparison with other existing heuristics.

For the first time Developmental Genetic Programming was proposed by Koza, Bennet, Andre and Keane [17], to create electrical circuits. This methodology evolves circuit-construction tree, in which nodes correspond to functions defining the developmental process. The initial circuit consists of an embryo and a test fixture. The sample embryo is at least one modifiable wire while fixture is one or more unmodifiable wires or electrical components. The circuit is developed by progressively applying functions in the circuit-construction tree to the modifiable parts (wires and electronic components) of the embryonic circuit.

A similar methodology was used by Deniziak and Górski in the co-synthesis of embedded systems described by task graphs [18]. The system-construction tree is based on a task graph. Each node of the tree specifies an implementation of the corresponding task. The embryo is an allocation of the first task. First (initial) population is created randomly. Then after evolution, using crossover, mutation and reproduction, an optimal (or suboptimal) solution is found.

In [19] a list of 36 instances of human-competitive results produced by the GP is presented. A lot of them concern of synthesis of an analog electrical circuits, developing quantum algorithms, designing controllers. According to our best knowledge, there is no approach concerning optimization of object-oriented real-time multi-task systems using DGP methods.

III. Problem Statement

Let us assume that information about the functionalities of RT MOOS and resources available for implementation of these functionalities are specified with the help of UML early diagrams: use case, activity, and sequence. This is typical while designing OOS of any kind. To start the system a supervisor is needed, which allocates the functionalities into the resources in such a way that the cost will be minimal while all real-time constraints will be satisfied. Both, number of the functionalities and the number of resources, are large enough to exclude a human being as the supervisor. Therefore, an engine which optimizes supervising the system should be worked out. The engine will be named an artificial supervisor (AS), since it does what the supervisor should do.

An AS should work as follows:

1) it should work out a schedule for RT MOOS which would be optimal under current operational conditions, and
2) adjust the schedule, to keep its optimality, when the conditions have changed (some of the resources had failed, for example).

The goal of the research is to introduce a method of automatic generation of ASs from the diagrams. An approach based on an idea derived from developmental genetic programming is used.

The procedure consists of two steps. In the first one information included in the UML diagrams are transformed into a task graph and a library of objects working for the system. These are input data to the second step. In this step (Section V) the AS is created. To this end a universal method of evolution of a genotype of the AS is applied. Decision options, which may be contained in the genes of the AS, are defined and then the genotype is created developmentally, using DGP-like approach.

An example of the generation of a supervisor in a building enterprise is used to illustrate the method. The enterprise is an RT MOOS because its resources may be dealt with as objects of different kinds, human or technical, which work in real time and in multi-task mode of operation. A user of the RT MOOS specifies the functionalities of the house, a deadline of the implementation and cost constraints. In the case of a small building enterprise, a contractor assisted by a CASE tool (Computer Aided Software Engineering) can elaborate optimal or semi-optimal schedule of building the house. However, big consortia own a large number of resources and implement many different constructions. Hence, this duty must be waived from the contractor and placed onto an AS. Not the contractor, but the AS, which is engaged in building the house, should generate an optimal schedule for management of enterprise resources.

Summarizing, for each of the implementations an AS should be generated. The AS elaborates optimal schedule of the implementation. A procedure of generation of ASs is universal. However, to generate a specific AS (for building a housing estate or LNG terminal, or a bridge, or managing a bank account, and so on) it should be supplemented with data describing what should be supervised. This is done with the help of UML diagrams.

An AS should react to events that make the schedule non-optimal, such as failures of the resources, unexpected delays of task executions and so on. In such situations, any break in work could generate huge costs. Thus, changes in the schedule should be done in real-time, and as soon as possible.

IV. From UML Early Models to Library of Resources

The first step of the method consists in the generation of input data for DGP, which in turn will create an adequate supervisor. To this end UML early models of RT MOOS, which will be under optimization, are used. In case of building a house the models describe all activities of the supervisor that controls the whole process of the construction.

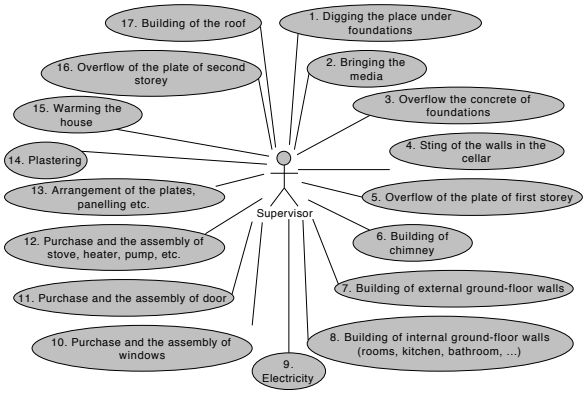

Fig. 1. Use case diagram for building a house

A. Functionalities and sequential constraints

Functionalities are described with the help of a UML use case diagram where a Supervisor is the main actor. It owns resources (objects) performing tasks in real-time and may face orders of task executions. Use case diagram describing building a house is given on Fig. 1. Actually, it maintains the enterprise which works as a real RT MOOS.

A task is an activity performed by a specific user of an RT MOOS. In the diagram, each of the use cases corresponds to one of such tasks, since a use case is an action performed by an object (objects) which aims to yield an observable goal for the user. Thus, each of the tasks has a use case that explains what the task is, and how it should function. Moreover, a use case may include statements about pre-conditions (required before the task began), post-conditions (valid when the task was successfully completed) and, if needed, exceptions.

The diagram on Fig. 1 contains 17 use cases (stages of a house building; numbered from 1 to 17 on Fig. 2) which should be scheduled for enterprise resources. Therefore, assignment of the use cases, to the resources, is a subject of optimization. However, the diagram may not say anything, that one of the cases must be used before another one. Digging foundations must precede their laying, and plastering must be done before warming a house, for example[1]. In general, an RT MOOS as an example of a multi-task system may have sequential constraints. Tasks should not be executed in arbitrary orders because some of the tasks need to be executed before others.

The Supervisor knows use case sequential constraints. This can be specified with the help of an extension and of an inclusion associations («extend» or «include» stereotypes [9]) and on pre- and post-conditions defined for the use cases (sequential dependencies [20]). As the summary, a UML

[1]Numerical prefixes are introduced to identify the use cases and will be used later on.

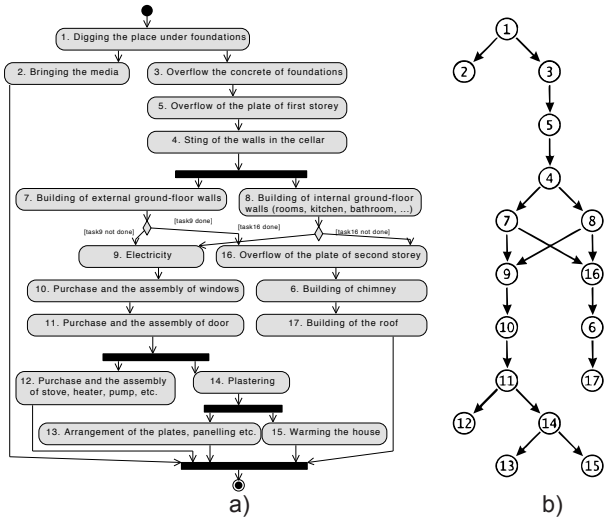

Fig. 2. Activity Diagram (a) and Task Graph (b) of the system.

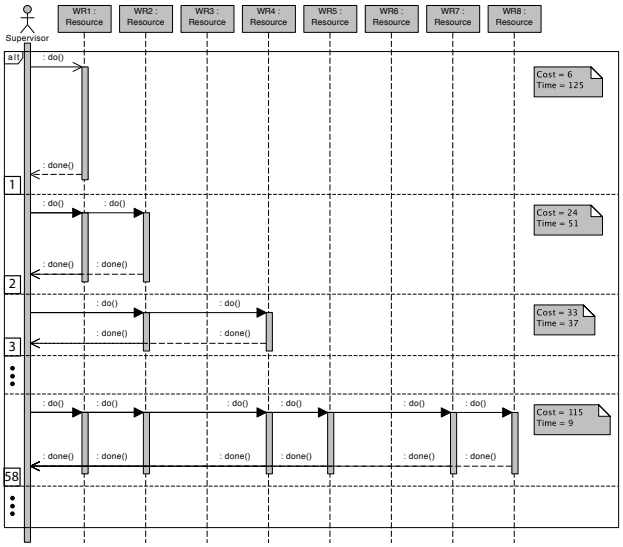

Fig. 3. Sequence diagram of Bringing the media use case.

activity diagram [9] for an RT MOOS can be defined, and then transferred into a task graph (TG) showing an execution of the tasks in a real-time. The constraints for the example are shown on Fig. 2a whereas their corresponding TG is shown on Fig. 2b[2].

B. Resources

The resource is an object required to execute a task. In general, it could be a human, a tool or any other object, which is reusable or renewable. If there are many resources of the same type, each of them should be presented as a separate resource.

In the example, two types of the resources are required for building a house. First are Workers. These are laborers like electricians, plasterers, and so on, that are able to execute some specific tasks. Also, a company, which could be used as an outsourcing, should be given as a resource. A worker could use the second type of the resources which are Tools. These are machines which could be used by workers during execution of tasks. Hardly ever one resource is able to complete a task itself. Thus resources are grouped into work teams, which will be described in the next subsection.

Not all resources are necessary for the execution of some tasks. This could be determined by the Supervisor with the help of the third kind of UML diagrams, namely sequence diagrams.

C. Scenarios of the resource cooperation

Inspired by real world, where most tasks are executed by a group of resources, a concept of a team is introduced. The team is a set of resources that are able to execute a task. Any task may be executed by more than one team. We assume that the execution cost and time of the task are known. One resource may belong to different teams, but teams having the same resources cannot be scheduled at the same time period.

A use case is refined into one or more sequence diagrams to show how the case might be implemented with the help of detailed actions. Therefore, each sequence of the actions defines an actual work-flow and reflects a sequence of decisions the supervisor should take to perform a single task. Every task is executed by teams (single worker team is also possible, but rarely). Different teams are able to finish their work faster or cheaper, using more or less resources. The sequence diagrams have to define these scenarios.

Fig. 3 shows an example how the supervisor might interact with objects participating in the construction of the building[3]. Moreover, the sequence diagrams show options (with time and cost of their implementations) available to the supervisor of the enterprise.

It is characteristic for a Supervisor that while traversing a task graph it determines step by step what should be done at that point and that its selections are usually optional. This means that it actually decides what functionalities of the RT MOOS should be assigned to what objects and in which order these functionalities should be executed. Its decisions should be optimal, taking into account costs and the time of execution. Therefore, the quality of the supervisor should be as high as possible.

D. Library of resources

Sequence diagrams specify how cooperating objects are organized in teams and how do they work. For example, Fig. 3 shows 4 teams. Each of the teams could bring media to a house under construction, but with different workload and costs. From sequence diagrams a table is derived which determines

[2]Automatic generation of TG from UML diagrams is possible but will not be discussed in the paper.

[3]Remaining 16 sequences are very similar.

TABLE I
A LIBRARY OF RESOURCES.

Task #	Team # (for the task #)	Time	Cost	Time * Cost	Members
1:	0	125	6	750	WR1
1:	1	51	24	1224	WR1, WR2
1:	2	37	33	1221	WR1, WR2
...	...	...	...	...	...
1:	58	9	115	1035	WR1, WR2, WR4, WR5, WR7, WR8
...	...	...	...	...	...
1:	62	82	183	15006	WR1, WR2, WR3, WR4, WR5, WR7, WR8
2:	0	57	62	3534	WR3
...	...	...	...	...	...
2:	3	41	82	3362	WR1, WR2, WR5
...	...	...	...	...	...
2:	62	10	189	1890	WR3, WR4, WR5, WR7, WR8
...	...	...	...	...	...

a binding of tasks with teams. For the example, this is given in Table I.

Columns "Time", "Cost" and "Members" of the table are filled in with data from sequence diagrams. Units of "Time" or "Cost" are inessential. It could be a day or an hour, dollar or euro. It is only important that all costs and all times are defined using the same units.

Column "Time * Cost" does not give any new information, but it is helpful to accelerate the time of the computations.

V. CREATION OF SUPERVISORS

The second step of the method consists of initiating and evolving genotypes, corresponding to the supervisors, with the help of DGP. It is assumed that the supervisor selects options defining the strategy of the allocation of resources. The way in which it does, it is a specific feature of its mind, and it is contained in its genotype. A supervisor with the best genotype (allocating the resources optimally) will be generated with the help of DGP. DGP evolves genotypes, while genotype-to-phenotype mapping is used in the fitness computation, which is required for the genotype selection process. It is possible, that one phenotype may be created from two different genotypes, because genotype-to-phenotype mapping always generates systems that meet the system requirements.

A genotype corresponding to the supervisor has a form of a tree engineering the system. A root of the tree specifies a construction of an embryonic system, while all other nodes correspond to functions which progressively build up the whole system. If the system is defined by a task graph, then an embryo is a system executing the first task from the task graph. Thus, the number of possible embryos equals the number of teams, in the library of resources, which are capable of executing the first task. Embryonic systems are selected randomly for each attempt to create an initial population of supervisors.

TABLE II
SUPERVISOR'S OPTIONS

Step	Option	P
1	a. The fastest team	0.16(6)
	b. The cheapest team	0.16(6)
	c. The lowest time * cost	0.16(6)
	d. Determination by second gene	0.16(6)
	e. The fastest starting team	0.16(6)
	f. The fastest ending team	0.16(6)
2	List scheduling	1

A. Supervisor's options

The supervisor undertakes the following two actions:

- resource allocation and task assignment, that send an appropriate team to execute a particular task and hence, allocate members of the team,
- task scheduling (only when more than one task is assigned to the same resource), that schedules the tasks assigned to the resources. When the resource is unavailable, the execution of the task is delayed as long as the resource is not released.

Initial population of supervisors consists of randomly generated genotypes. It selects one of the options given in part 1 of its decision table. Table II contains the options which the supervisor may choose. The last column in Table II shows a probability of the selection.

The first option prefers a team, which requires the smallest period of time to execute a task. Second one prefers a team, which brings the lowest cost increase. Third option prefers a team with the best ratio of the costs to the time of the execution. Fourth option works in a different way. It allows us to use "a little pushed" teams, what cannot be obtained as a result of the remaining options. The next option prefers a team, which could start an execution of the task as soon as possible (other teams might be busy). The last option prefers a team whose members could be the first to finish a task (be freed). For the second action only one option is available, namely the list scheduling method.

B. Genotype

The genotypes have forms of binary trees corresponding to various procedures of synthesis of phenotypes (target solutions). Every node has the same structure presented on Fig. 4

The first field *isLeaf* determines a role of the node in a tree. When it is true (the node is a leaf), the strategy for tasks is described in the field named *"strategy"*, which stores an option from Table II. In this case information from the other fields is omitted. When the node is not a leaf, a content of the field *"strategy"* is not important. In this case, *cutPos* contains a number describing which group of tasks should be scheduled by the left node and which one by the right node. Thus, *nextLeft* and *nextRight* must not be null pointers.

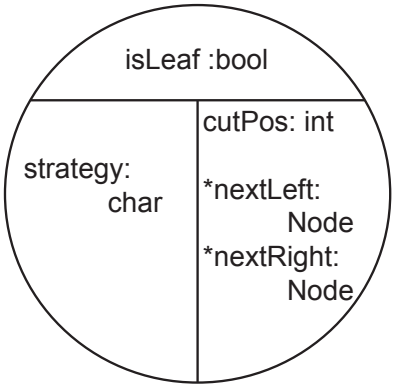

Fig. 4. A node of the genotype

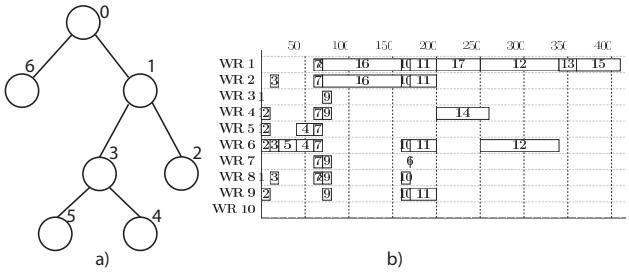

Fig. 5. A simple genotype (a) and the corresponding phenotype (b)

The simplest genotype consists of only one node, which is also a leaf and a root. A simple genotype and the corresponding phenotype are presented on Fig. 5.

During the evolution, a genotype grows but a size of the genotype tree is limited. If the tree exceeds the maximum size then too long branches are cut off. For example, if the maximum size is 6, every node on the sixth level which has a successor is changed into a leaf, and all its successors are destroyed.

An embryo of the tree could grow as an effect of genetic operators: mutation and crossover. An action associated with the mutation depends on the state of the node and is presented in the Table III.

The crossover is used to exchange information between two chromosomes. It is necessary to draw a point of cut a tree in both chromosomes. An example of the crossover is presented on Fig. 6

With every genotype an array is associated. Its size is equal to the number of tasks and contains indexes of teams. If for a task, strategy 'd' is chosen, the team with an index taken from the array is used. At the very beginning of the mutation, a place in the array is randomly chosen. Next a new index is randomly generated. During the crossover, parts of the arrays from both genotypes are swapped.

C. Genotype to phenotype mapping

The first step in a genotype-to-phenotype mapping is to assign strategies to tasks (that is teams from Table I to tasks

TABLE III
THE RULES OF MUTATIONS

Is a leaf?			
Yes		No	
Draw: switch leaf/node or not?			
Yes	No	Yes	No
Set *isLeaf* as FALSE. If *nextLeft* or *nextRight* is NULL - create a new leaf for it.	Draw new strategy	Set isLeaf as TRUE	Change value for a randomly chosen field: *cutPos*, *nextLeft* or *nextRight*

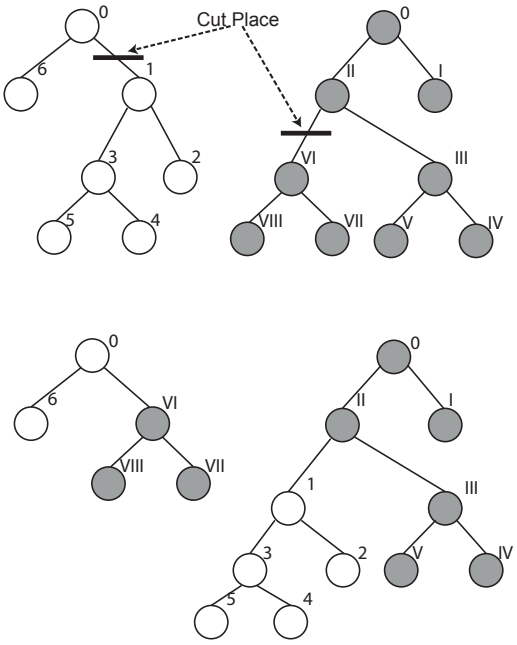

Fig. 6. An example of the crossover

from Fig. 2b, in the example). This step is illustrated on Fig. 7. Please note, that node 1 partitions tasks from 11 to 17 into two groups: from 11 to 18 and the rest. Because the first group is out of the range, in fact, there is only one group, which is taken by node 3.

In the second step all tasks without any predecessor, or with predecessors having already assigned teams, are being

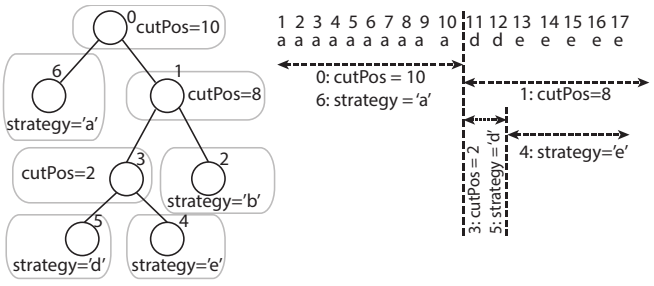

Fig. 7. The first step in genotype-to-phenotype mapping

searched for. For these tasks, teams are assigned according to their strategy. Then the step is repeated as long as there are tasks without assigned teams.

In the third step, the total cost of the solution could be calculated. For this purpose, the cost of each team from the resource library (Table I) is given.

D. Parameters of DGP

During the evolution, new populations of supervisors are created using genetic operations: reproduction, crossover (recombination) and mutation. After the genetic operations are performed on the current population, a new population replaces the current one. The number of individuals in each population is always equal:

$$\Pi = \alpha \prod_{i=1}^{n} s_i \tag{1}$$

where n is the number of tasks, s is the number of teams capable to solve specific problem and α is a constant between 0 and 1. If α is equal to 1, the population has as many individuals as many solutions of the problem exist. The evolution is controlled by parameters β, γ and δ, such that:

- $\Phi = \beta \cdot \Pi$ is the number of individuals created using the reproduction,
- $\Psi = \gamma \cdot \Pi$ is the number of individuals created using the crossover,
- $\Omega = \delta \cdot \Pi$ is the number of individuals created using the mutation and
- $\beta + \gamma + \delta = 1$

The last condition ensures that each of the created population will have the same number of individuals.

Finally, the selection of the best individuals by a tournament is chosen [21]. In this method, chromosomes (genotypes) are drawn with the same probability in quantity defined as a size of the tournament. From the drawn chromosomes the best one is taken. Hence, the tournament is repeated as many times as the number of chromosomes for a reproduction, crossover and mutation is required . A size of the tournament should not be too high, because the selection pressure is too strong and the evolution will be too greedy. It also could not be too low, because the time of finding any better result would be too long.

E. Fitness function

A fitness function determines the aim of DGP. In the presented approach, two options are possible. In the first one, the cheapest solution which has to be finished before a deadline is searched for. Such fitness function is applied when hard real time constraints have to be satisfied. In the second one, the DGP should find the fastest solution, which does not exceed a given budget. This case concerns systems with soft real-time requirements.

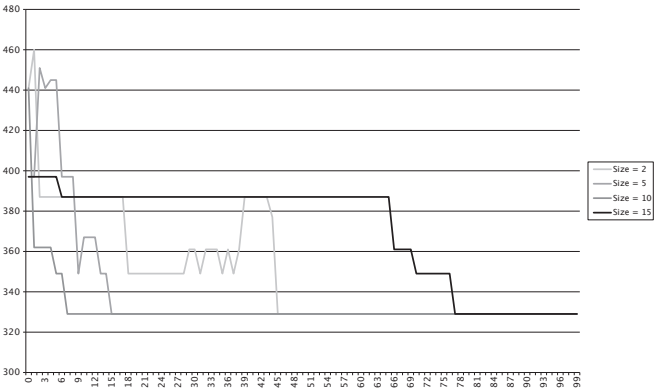

Fig. 8. Progress of the evolution for different tournament sizes

VI. Computational Experiments

High efficiency of the DGP may be achieved only when genetic parameters will be properly adjusted. The most important are the number of individuals created during reproduction, crossover, mutation and a size of the tournament. If their values are not chosen correctly, the DGP will find solutions far from the optimum or finding the best solution will take a lot of time.

The presented method was evaluated with the help of the example from Fig. 1. The deadline was equal 700 time units. During the experiments, the following values of genetic parameters were used:

- the evolution was stopped after 100 generations,
- each experiment was repeated 7 times, the mean of the best solutions received in each pass is given as the result,
- parameter α was equal $7 \cdot 10^{-30}$, thus the population size was equal 102.

First, the convergence of the DGP for different sizes of the tournament was explored. Tournament sizes equal 2, 5, 10 and 15 were examined (Fig. 8). We observed that the sizes 2 and 5 were too small. Very often the best solutions were skipped over, and not selected for further evolution. The best convergence was obtained for the size equal 10.

Next, we examined the influence of the crossover and the mutation on finding the best solution. For this purpose, an analysis of the best solutions achieved with different combination of parameters controlling the crossover and the mutation was performed. Fig. 9 presents the results obtained for different number of mutations. We may observe that for less than 10% of mutants in the population, the results are poor. The best result was obtained for 18% of the mutants. The number of mutants should not be too high. For more than 85% of the mutants the DGP usually produces also poor results. In this case, too much number of mutants probably disturbs the evolution. Fig. 10 presents results obtained for different numbers of individuals created using the crossover. The highest probability of obtaining the best results is when the crossover is applied for creation of at least 65% genotypes.

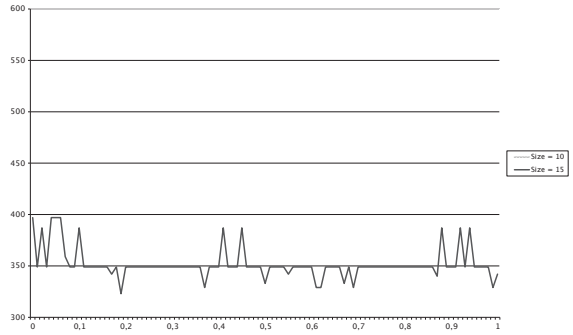

Fig. 9. The influence of δ

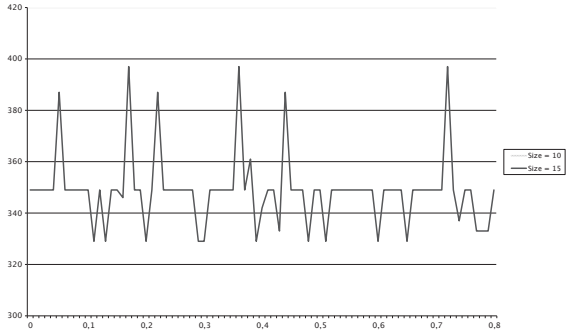

Fig. 10. The influence of γ

The results were also compared with greedy approach. The greedy algorithm assigns the cheapest teams to first tasks. However, if the deadline is to be exceeded, the algorithm assigns the fastest (usually the most expensive) teams to tasks at the end of the schedule as it is necessary to finish the work before the deadline. To the problem the greedy algorithm generates a solution which costs 1091 while the cost of the solution generated by the DGP is equal to 323. Thus, the result obtained with the help of our method is three times better.

To check whether our method led to global optimum or not, the entire space of the solutions was tested (Fig. 11). Out of $7.63 \cdot 10^{11}$ solutions, only 260 gave the best schedules, and a cost of the cheapest schedule was 323. So the answer is positive.

VII. Conclusions

A two-step procedure for automatic creation of supervisors of RT MOOS has been formulated. In the first step one has to specify functionalities of the system using early UML models (use case, activity, and sequence diagrams) and transform the models into a task graph and a library of resources of the system. In the second step one has to define decision options of a supervisor of the system and develop a genotype of the supervisor using DGP. An application of the procedure to RT MOOS, which was an enterprise of building houses, resulted in the creation of the best supervisor in acceptable time. It was evaluated from different points of view. Efficiency and precision were taken into account. Experiments showed

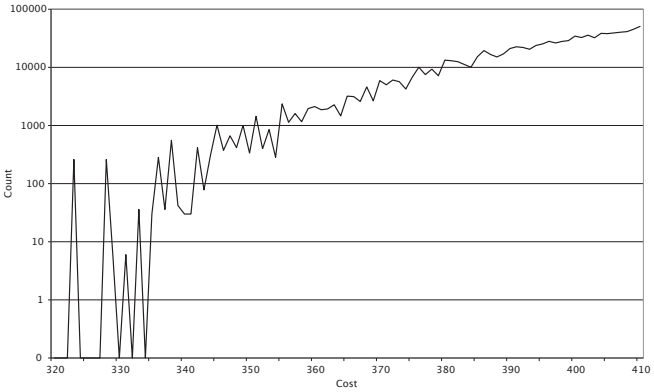

Fig. 11. The number of solutions to the problem for a given cost

that the supervisor can develop the best schedule, which corresponds to the global optimum.

Variant of the RCPSP presented in this paper is based on teams that are able to solve tasks. In general, a team is a group of workers and tools, but could also contain only one member. For different tasks, teams could contain different members, thus a team is able to start working when all of its members are idle. For this reason, it is possible, that choosing only the fastest teams do not yield the fastest solution. For the other hand, choosing only the cheapest teams could lead to the situation when the deadline of solution will be exceeded.

The influence of genetic parameters on the evolution of the genotype was also investigated. The most important parameter is δ, which defines the number of mutations. If it is too small or too large, DGP will have problems with escaping from local minima or it will behave too randomly. For the example presented in this paper, the best value for this parameter is about 20%.

The significant influence on the optimization has also a size of the tournament. If it is too small, DGP needs more time to find acceptable results. In the opposite case, if this value is too big, it leads DGP very quickly close to the global minimum, but has never achieved it. Actually, it stuck in local minima, because a variety of the population is decreasing.

The γ parameter corresponds to the number of crossovers. The crossover is responsible for a genotype tree development. Changing a genotype tree has also a bit similar effect of mutations, because after changing a position of the branch in the tree a new assignment of teams to tasks is achieved. Although this parameter has the smallest influence on results, it could be noticed that too high value of γ makes the solutions more random. If the value is too small, DGP need more time to achieve the optimal solution.

References

[1] C.Wei, P. Liu, Y. Tsai, "Resource-constrained project management using enhanced theory of constraint", International Journal of Project Management, Vo. 20, No.7, 2002, pp.561-567. http://dx.doi.org/10.1016/S0263-7863(01)00063-1

[2] J. Blazewicz, J.K. Lenstra, A.H.G. Rinnooy Kan, Scheduling subject to resource constraints: Classification and complexity, Discrete Applied Mathematics, No.5,1983, pp.11-24. http://dx.doi.org/10.1016/0166-218X(83)90012-4

[3] Pawiński G. and Sapiecha K., "Cost-efficient Project Management Based on Distributed Processing Model.", Proceedings of The 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, Belfast 2013 http://dx.doi.org/10.1109/PDP.2013.30

[4] R. H. Möhring, A. S. Schulz, F. Stork, M. Uetz, "Solving Project Scheduling Problems by Minimum Cut Computations", Management Science, v.49 n.3, pp.330-350, March 2003. http://dx.doi.org/10.1287/mnsc.49.3.330.12737

[5] Hartmann S., Briskorn D., A survey of variants and extensions of the resource-constrained project scheduling problem, European journal of operational research : EJOR. - Amsterdam : Elsevier, Vol. 207., 1 (16.11.), pp. 1-15 (2010). http://dx.doi.org/10.1016/j.ejor.2009.11.005

[6] J.H.Holland, "Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology", Control, and Artificial Intelligence, University of Michigan Press, Ann Arbor, MI (reprinted 1992, MIT Press, Cambridge, MA).

[7] R.E.Keller, W.Banzhaf, "The evolution of genetic code in genetic programming", Proc. of the Genetic and Evolutionary Computation Conference, 1999, pp.1077-1082.

[8] G. Wilson, M. Heywood "Probabilistic Adaptive Mapping Developmental Genetic Programming (PAM DGP): A New Developmental Approach", Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN IX), (Reykjavik 2007) http://dx.doi.org/10.1007/11844297_76

[9] L. C. Briand, Y. Labiche, A UML-Based Approach to System Testing, Software and Systems Modeling, vol. 1 (1), pp. 10-42, 2002. http://dx.doi.org/10.1007/s10270-002-0004-8

[10] J.L.M. Pasaje, M.G. Harbour, J.M. Drake, "MAST Real-Time View: a graphic UML tool for modeling object-oriented real-time systems", In proceeding of: IEEE 22nd Real-Time Systems Symposium, 2001. (RTSS 2001). http://dx.doi.org/10.1109/REAL.2001.990618

[11] H. Gomaa, Designing Concurrent, Distributed, and Real-Time Applications with UML. Addison-Wesley, 2000.

[12] R.Jigorea, S. Manolache, P.Eles, Zebo Peng, "Modelling of real-time embedded systems in an object-oriented design environment with UML," Proceedings. Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2000, pp.210-213. http://dx.doi.org/10.1109/ISORC.2000.839532

[13] Alcaraz, J., & Maroto, C. (2001). A robust genetic algorithm for resource allocation in project scheduling. Annals of Operations Research, 102, 83-109. http://dx.doi.org/10.1023/A:1010949931021

[14] Hartmann, S. (1998). An competitive genetic algorithm for resource-constrained project scheduling. Naval Research Logistics, 45(7), 733-750. http://dx.doi.org/10.1002/(SICI)1520-6750(199810)45:7%3C733::AID-NAV5%3E3.3.CO;2-7

[15] Xiang Li, Lishan Kang, Wei Tan, "Optimized Research of Resource Constrained Project Scheduling Problem Based on Genetic Algorithms", Lecture Notes in Computer Science, Vol. 4683, 2007, pp 177-186. http://dx.doi.org/10.1007/978-3-540-74581-5_19

[16] Hossein Zoulfaghari, Javad Nematian, Nader Mahmoudi, and Mehdi Khodabandeh. 2013. A New Genetic Algorithm for the RCPSP in Large Scale. Int. J. Appl. Evol. Comput. 4, 2 (April 2013), 29-40. http://dx.doi.org/10.4018/jaec.2013040103

[17] Koza, J., Bennett III , F. H., Andre, D., Keane, M. A., 1998. Evolutionary Design of Analog Electrical Circuits Using Genetic Programming. In: I. C. Parmee (ed.), Adaptive Computing in Design and Manufacture. http://dx.doi.org/10.1007/978-3-540-85857-7_8

[18] S.Deniziak, A.Górski, "Hardware/Software Co-Synthesis of Distributed Embedded Systems Using Genetic Programming", Lecture Notes in Computer Science, Springer-Verlag, 2008, pp.83-93. http://dx.doi.org/10.1007/978-3-540-85857-7_8

[19] J.R.Koza, R.Poli, "Genetic Programming", In Edmund Burke and Graham Kendal, editors. "Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques", Chapter 5. Springer, 2005. http://dx.doi.org/10.1007/0-387-28356-0_5

[20] Binder R. V., Testing Object-Oriented Systems - Models, Patterns, and Tools, Addison-Wesley (1999)

[21] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag Berlin Heidelberg, 1996.