STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume L, Number 2, 2005

COMPONENTS EXECUTION ORDER USING GENETIC ALGORITHMS

ANDREEA FANEA AND LAURA DIOŞAN

Abstract. The current trend in software engineering is towards component-based development. A challenge in component-based software development is how to assemble components effectively and efficiently. In this paper we present a new approach for computing components execution order. Using Genetic Algorithms GA we compute the final system from specified components with conditions order. Some numerical experiments are performed.

1. Introduction

Component-based software engineering (CBSE) is the emerging discipline of the development of software components and the development of systems incorporating such components. In a component-based development process we distinguish the development of components from development of systems; components can be developed independently of systems. However, the processes may have many interaction points.

The main advantage of component-based system development is the reuse of components when building applications. Instead of developing a new system from scratch, already existing components are assembled to give the required result.

Unlike previous work [5] which use backtracking algorithm (BA) to determine the execution order for system components, this work looks for designing an execution order using evolutionary methods.

Our main purpose is to evolve the execution order for system components. This basically means that we want to find which component should be executed and which is the order in which these elements are "computed". In this respect we propose a new technique which is used for evolving the execution structure of a component-based system. We evolve arrays of integers which provide a meaning for executing the elements within a system.

Received by the editors: November 10, 2005.

2000 *Mathematics Subject Classification.* 68T20,68N30.

1998 *CR Categories and Descriptors.* I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search – *Heuristic methods*; I.6.5 [**Model Development**]: Modeling methodologies .

The number of evolved orders found with GA is compared to configurations number obtained with a backtracking algorithm from [5]. Numerical experiments show that the GA performs similarly and sometimes even better than standard backtracking approaches for several human-defined systems.

The paper is structured as follows: Section 2 discusses work related to components integration and composition. Section 3 describes, in detail, the proposed model. Several numerical experiments are performed in Section 4. Conclusions and further work are given in Section 5.

2. Related work

There are two issues which need to be addressed where a software system is to be constructed from a collection of components. First there has to be a way to connect the components together. Then we have to get them to do what we want. We need to ensure that the assembled system does what it is required [7], [9].

To successfully incorporate a component [4] in a system, certain steps must be followed: selection, composition and integration, and finally, test and verification must be followed. In the following we discuss integration and composition of components.

Component integration and composition are not synonymous. Component integration is the mechanical process of "wiring" components together, whereas composition takes one step further to ensure that assemblies can be used as components in larger assemblies. Component composition focuses on emergent assembly-level behavior, assuring that the assembly will perform as desired and that it could be used as a building block in a larger system. The constituent components must not only plug together, they must work well together.

The problem of making pieces of software fit together has been the subject of considerable effort. Systems and schemes exist which address these issues (COM, EJB, RMI, MSMQ) [10], [12], [1], [2], [8]. These arrangements work by managing and controlling the interfaces between components. By forcing components to conform to rules about how they interact with the outside world, systems ensure that components do not damage each others when they are connected. To a large extent, this problem may be addressed by managing component interfaces and ensuring that components are only connected through compatible interfaces.

In [5] a formal model for component composition is described. We consider that a compound component is formed from two or more simple components. There are two basic ways [11] in which these simple components can depend on each other, parallel and serial operation:

- **parallel composition**, $A||B$, in which the operations performed on data are independent and there is no dependency between outports(A) and outports(B);
- **serial composition**, $A + B$, in which the B component expects some results from component A.

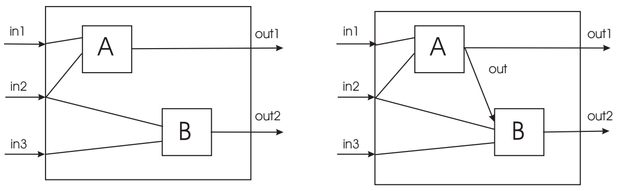

FIGURE 1. Parallel and serial compositions

3. PROPOSED MODEL

This section presents the proposed model for finding the components execution order. We will use a GA [3, 6] for evolving the execution order of components. Each GA individual is a fixed-length string of genes. Each gene is an integer number, from $[0...NumberOfComponents]$. These values represent indexes order of the components. They will indicate the time moments for execution.

Some components could be executed earlier and some of them are executed later. Therefore, a GA chromosome must be transformed to contain only the values from 0 to Max, where Max represents the number of different time moments (at one moment it is possible that one or more components will be executed).

Example 1. Suppose that we want to evolve the structure of a system that contains 9 components. This means that the algorithm will have chromosomes with 9 genes whose values are in the [1...9] range. The dependencies between components are:

TABLE 1. Conditions for the first example

CondNr	Condition	CondNr	Condition
1	$C1 + C3$	6	$C6 + C7$
2	$C2 + C4$,	7	$C7 + C8$,
3	$C3 + C5$,	8	$C5 + C9$,
4	$C4 + C5$,	9	$C8 + C9$,

A GA chromosome with 9 genes can be:

Genes	1	2	3	4	5	6	7	8	9
Values	2	1	4	3	8	5	6	9	7

For computing the fitness of this chromosome, during one generation, we will execute the components following this order: first, the second component (because the gene with minimum value is the second gene), then the first component and so on (it is needed to order the ascending the genes values): execute(C2), execute (C1), execute (C4), execute (C3), execute (C6), execute (C7), execute (C9), execute (C5), execute (C8).

In this example at each moment, we execute only one component.

Example 2. Let us consider another example which consists of a chromosome with 9 genes containing only 5 different values.

Genes	1	2	3	4	5	6	7	8	9
Values	6	2	1	4	7	1	6	2	6

In this case components 3 and 6 are executed in same time (also components 2 and 8, but after the previous execution, also the components 1, 7 and 9). Because of this synchronism we need to scale the genes of the GA chromosome to the interval [0 ... 5] (because we need 5 time moments for execution). The obtained chromosome is:

Genes	1	2	3	4	5	6	7	8	9
Values	4	2	1	3	5	1	4	2	4

To establish the correct order of execution for this chromosome the genes values must be ascending sorted. Then we obtain:

Genes	3	6	2	8	4	1	7	9	5
Values	1	1	2	2	3	4	4	4	5

3.1. **Fitness assignment.** The model proposed in this paper is a GA that evolves the order of execution for the components of a system.

The array of integers encoded into a GA chromosome represents the order of execution for system components. The fitness of a chromosome represents the sum of two values: the number of breached conditions and the number of components that are not executed.

Some components depend on the results provided by the other components (one component must wait until other component (s) finished its execution). For instance, we have two number a and b and we want to verify if the greatest common divisor of these two numbers is a prim number, that's equivalent to have two components: $C1$ that computes the greatest common divisor of a and b and $C2$ that verifies if a number is prime. In our problem $C1$ must be executed before $C2$. A system with more components will have more conditions. If one of these conditions is not respected (by the order given by the GA individual), than we increase the fitness.

Even if the number of genes from a GA individual is equal to the components number, it is possible that not all components can be executed. For instance, if the order provided by the GA chromosome supposes to execute the component $C2$ before $C1$, than $C2$ will not be executed and we increase the fitness value.

In the second example, the genes with same execution moments proceed in parallel and the genes with different values are executed serial. But for a parallel execution of two or more components these elements must be independent (are not the subject of one of the system conditions).

The system executes the components $C3$ and $C6$ if and only if there isn't any constrict for these components. Components $C3$ and $C6$ are independent, so the system can execute them in parallel (but also it is possible to execute them in serie). But really, the system executes only $C6$ because $C3$ can't b executed ($C1$ isn't yet executed). Then, component $C2$ is executed in series with $C6$ because the associated genes have different values. Then component $C8$ has the same gene value with $C2$ and there isn't any condition regarding these components, therefore they can be executed in parallel (but $C8$ will not be executed because, $C7$, must be executed first and so on. Finally, this chromosome will have a fitness equal to 8 (four components are not executed and four elements breached the dependencies).

3.2. **Algorithm.** The algorithm used for evolving the integration order of components system is described in this section. The algorithm is a standard GA [3, 6]. We use a generational model as underlying mechanism for our GA implementation.

```
Population Initialization $P(0)$.
Evaluate $P(0)$.
**For** $t = 1$ **to** $NumberOfGeneration$ **do**
    Add the best individual from $P(t-1)$ to $P(t)$
    **While** $P(t)$ is not complete (full, whole) **do**
        Select two parents $P_1$ and $P_2$ from $P(t-1)$
        Recombine the parents, obtaining two offspring $O_1$ and $O_2$.
        Mutate each offspring.
        Add the two offspring $O_1$ and $O_2$ to $P(t)$.
        Evaluate $P(t)$
    **End While**
**End For**
```

The GA starts by creating a random population of individuals. Each individual is a fixed-length (equal to the number of system components) array of integer numbers. The following steps are repeated until a given number of generations is reached: we put in the next generation the best chromosome from current generation. Two parents are selected using a standard selection procedure (binary tournament selection) until we obtain a new complete generation. The parents are recombined (using one-cutting point crossover) in order to obtain two offspring. The offspring are considered for mutation which is performed by replacing some genes with randomly generated values. Finally, we put the offspring in next generation.

4. Numerical experiments

In order to test our approach we performed three didactical experiments with different number of components involved in the system and with different types of dependences between components. We compared the result obtained with the algorithm from [5] with the presented GA.

Having specified the simple components that we need to compose, a model for component composition [5] is used to obtain all the possibilities of using parallel and serial composition. In the end we will obtain the system that we want. First we have to check if the data can pass between the simple components involved in the composition. That is if inports of all the simple components are inports of the black_box or if they occur as outports of the other components. If the condition above is satisfied then we can obtain the black_box component (final system), respecting dependencies between components.

The dependencies between components must be first determined as follows: we must check if an inport of a component appears as an outport of other component; if an inport in_i of a component B appears as an outport out_j of a component A, ($in_i = out_j$) then we must use component A before component B, and we can use B only with the + operator (for serial composition). These conditions are incorporated in the backtracking algorithm: if the component that we check for integration depends on the results provided by other components than those components must finish their executions before using this component. The dependent component must have a higher position number in the solution array then the components it depends. Also, in the solution array before a component with dependencies we always use serial composition +.

Experiment 1. In this experiment we deal with ten involved components having nine dependencies between them. The system that we want to obtain is presented in Figure 2 with the following computation semantics: *gcd* - greatest common divisor, *sd* - sum of digits, *pd* - prime divisors, *np* - number of prime numbers, *scd* - smallest common divisor, *inv* - inverse, *oed* - product of sum between odd and even digits, *sed* - sum of even digits, *npn* - next prime number and *aa* - arithmetic average.

We denote by “+” the serial composition and by “||” the parallel composition. The dependencies between the involved components are presented in Table 2.

Taking into account the above conditions we compute all the possibilities to obtain the desired system. Using BA from [5] one solution is:

$$(((((((((C2||C6) + C4)||C1) + C7) + C3) + C8)||C9) + C5) + C10).$$

This formula means that the result for computing in parallel $C2$ with $C6$ is then computed serial with $C4$, because $C4$ expects the result from $C2$. Then, the

TABLE 2. Conditions for the first experiment

CondNr	Condition	CondNr	Condition
1	$C1 + C3$,	6	$C7 + C8$,
2	$C2 + C4$,	7	$C5 + C10$,
3	$C6 + C7$,	8	$C8 + C10$,
4	$C3 + C5$,	9	$C9 + C10$.
5	$C4 + C5$,		

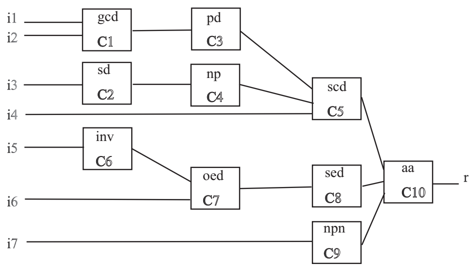

FIGURE 2. Component-based system for experiment 1

current result subsystem is composed in parallel with $C1$. Because component $C6$ was computed we can then serially execute $C7$. The result for computation of $C1$ is already obtained so can serially execute $C3$. Next, we can execute either serial $C5$, serial $C8$ or in parallel $C9$. We cannot compute $C10$ because we have to previous execute $C5$, $C8$ and $C9$. In previously experiment serial $C8$ is integrated. The following possibilities are parallel $C9$ and serial $C5$. Parallel $C9$ is executed and then serial $C5$. Having all the three components executed, we can now serial execute component $C10$.

We compare this approach with an evolutionary one. We use a standard GA for evolving the execution order. For GA we use a population with 200 individuals, each of them with 1 dimension. The gene number for a dimension will be equal to the components number (for this example it is 10). During 10000 generations we apply binary tournament selection, one cut point crossover with probability 0.8 and mutation (we mutate a random chosen gene) with probability 0.2.

But, since the GA uses pseudo-random numbers, it is very likely that successive runs of the same algorithm will generate completely different solutions. This problem can be handled in a standard manner: the genetic algorithm is run multiple times (100 runs in fact) and the number of the “good” chromosomes (with fitness 0) will be the average over all runs.

Applying GA one chromosome is:

We sort this chromosome after genes values and obtain:

Genes	1	2	3	4	5	6	7	8	9	10
Values	2	1	4	2	6	1	3	5	5	6

Genes	2	6	1	4	7	3	8	9	5	10
Values	1	1	2	2	3	4	5	5	6	6

Decoding it we obtain the following execution order: $C2$ in same time with $C6$, then $C1$ and $C4$, and then $C7$ and $C3$ (serial execution). At moment 5 we execute in parallel $C8$ and $C9$. Components $C5$ and $C10$ have the same execution moment, but they aren't executed in parallel because there exists a dependence between these two components. Therefore we execute first $C5$ and then, serial execution, $C10$. The experiment results are presented in Table 3 .

Table 3. Experiment 1: ten involved components with nine dependences

Algorithm	Solutions	Time
BA	3024	$2''$
GA	3883	$2''$

Experiment 2. In this experiment we deal with eleven involved components having ten dependencies between them. The computation semantics for each component involved in system integration are similar with those from the first experiment. We only present in Table 4 the dependencies between the involved components.

Table 4. Conditions for the second experiment

CondNr	Condition	CondNr	Condition
1	$C1 + C3$	6	$C6 + C7$
2	$C3 + C5,$	7	$C7 + C9,$
3	$C4 + C5,$	8	$C8 + C10,$
4	$C2 + C8,$	9	$C9 + C10,$
5	$C5 + C8,$	10	$C10 + C11.$

Applying algorithm from [5] one solution is:

$$((((((((((C3||C6) + C7)||C1) + C2)||C4) + C9) + C5) + C8) + C10) + C11).$$

This solution is represented in Figure 3. Each component has on the upper-left corner a number representing the order of execution.

Applying GA we obtain a chromosome of the following form:

We sort this chromosome on genes values and obtain:

The experimental results are shown in Table 5.

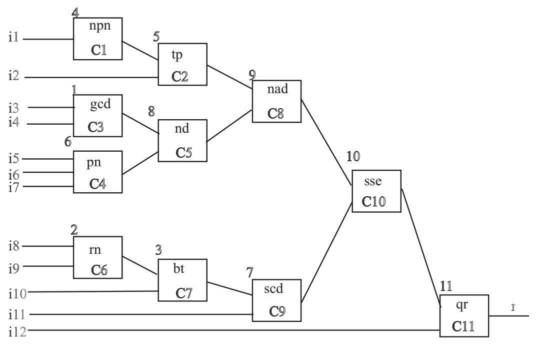

Figure 3. Component-based system for experiment 2

Genes	1	2	3	4	5	6	7	8	9	10	11
Values	2	3	1	3	5	1	2	6	4	6	7

Genes	3	6	1	7	2	4	9	5	8	10	11
Values	1	1	2	2	3	3	4	5	6	6	7

Table 5. Experiment 2: eleven involved components with ten dependences

Algorithm	Solutions	Time
BA	1680	$1''$
GA	2957	$1''$

5. Conclusions and Further work

In this paper a new method for component integration is proposed. The method is based on GA computing the order possibilities of components execution from a component-based system. A comparison between a previous developed backtracking-based algorithm and a GA algorithm is presented. The experiments on different data sets prove that we obtain much more solutions applying GA than applying other algorithms.

However, taking into account the No Free Lunch theorems for Search [13] and Optimization [14] we cannot make any assumption about the generalization ability of the evolved execution order. Further numerical experiments are required in order to assess the power of the evolved order.

Further works can be done in the following directions:

- how can we use GA to compute all the possibilities of obtaining a correct syntactical component-based system using only the information on the component interface;
- how can we use IA methods to analyze the behavior of a component-based system or to predict the behavior.

References

[1] R. Allen and D. Garlan, *A formal basis for architectural connection*, ACM Trans. on Software Eng. and Methodology, 6(3):213–249, July 1997.

[2] D. Box, *Essential COM*, Addison Wesley, 1998.

[3] H. J. Bremermann, *Optimization through evolution and recombination*, M.C. Yovits, G.T. Jacobi, and G.D. Goldstein, editors, Self-Organizing Systems 1962, Proceedings of the Conference on Self-Organizing Systems, Chicago, Illinois, 22.- 24.5.1962, pp. 93-106, 1962.

[4] I. Crnkovic, M. Larsson, *Building reliable component-based software systems*, Artech House, 2002

[5] A. Fanea, S. Motogna, *A Formal Model for Component Composition*, Proceedings of the Symposium "Zilele Academice Clujene", 2004, pp. 160-167

[6] D. Goldberg, *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley, Boston, USA, 1989.

[7] A. M. Gravell, and P. Henderson, *Executing formal specifications need not be harmful*, Software Engineering Journal, 11(2):104-110, IEE/BCS, March 1996

[8] D. N. Gray, J. Hotchkiss, S. LaForge, A. Shalit and T. Weinberg, *Modern Languages and Microsoft's Component Object Model*, Communications of the ACM 41(5): 55-65 1998.

[9] C. A. R. Hoare, *The role of formal techniques: past, current and future or how did software get so reliable without proof?*, 18th International Conference on Software Engineering (ICSE-18), Berlin, IEEE Computer Society Press, 1996, pp. 233-234.

[10] Object Management Group, http://www.omg.org/

[11] B. Parv, S. Motogna, *A formal model for components*, Bul. Stiint., Univ. Baia Mare, Ser. B, Matematica-Informatica, XVIII(2002), No.2, pag. 269-274

[12] Sun Microsystems, http://www.sun.com/

[13] D. H. Wolpert and W. G. McReady, *No Free Lunch Theorems for Search*, Technical Report SFI-TR-05-010, Santa Fe Institute, USA, 1995.

[14] D. H. Wolpert and W. G. McReady, *No Free Lunch Theorems for Optimization*, *IEEE Transaction on Evolutionary Computation*, Nr. 1, pp. 67-82, IEEE Press, NY, USA, 1997

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania

E-mail address: `afanea, lauras@cs.ubbcluj.ro`