Implementation of the optimizer of SOA system deployment architecture

A. P. Woźniak
Military University of Technology
Institute of Computer and Information Systems
Ul. Gen. Sylwestra Kaliskiego 2, 00-908 Warsaw, Poland
E-mail: adrian.wozniak@wat.edu.pl

KEYWORDS
Service-Oriented Architecture, SOA, business process, optimization, simulation

ABSTRACT

Optimization of business processes in SOA systems has been done using three separate types of methods: Resource Allocation, Service Scheduling and Service Composition. All three may influence each other, so the new method has been proposed to find an optimal combination of those three. It is based on a genetic algorithm that uses a simulator of the SOA system to evaluate solutions. The article describes a model for the optimization criteria for such solutions. Subsequently, some basic concepts used to implement the simulator and optimizer have been presented. Finally, the performance results of the optimizer have been described, including the conclusions on how they might be improved.

INTRODUCTION

The optimization of the system performance has always been important. It is due to many reasons, but first and foremost due to the limitation of resources or drive to increase system performance. It is no different in the case of the Service-Oriented Architecture (SOA) systems. Nonetheless, there are a few differences in how the systems might be optimized. The differences are mainly caused by fragmentation of the SOA systems. To add value to such systems, many components must cooperate. Each component is a software module that may be implemented in a technology different than other components and deployed independently. The most important part of the SOA components is that they deliver services. A service is a function of a component that is usually provided through a www. Users get value out of the SOA system by invoking the so-called composite services or business processes which are sequences of services realized by components.

The literature includes 3 types of methods for optimizing business processes in the SOA systems: Service Composition, Service Scheduling and Resource Allocation. Each of them is focused on a different stage of the SOA system implementation or execution. The first type is the Resource Allocation. It consists in determining which components should be deployed on which servers. Each component may be deployed simultaneously on many servers. Therefore, during Resource Allocation, it is also decided how many component instances should be running. Example of such a method uses Quality of Service (QoS) constrains and resource usage cost as an input (Almeida et al. 2006; Huang et al. 2016; Mennes et al. 2016). Then it searches for optimal allocation using Fixed Point Iteration technique. The second way of optimizing business processes in SOA is to use the Service Composition method, which is the most popular in the literature (e.g. Ebrahim 2011; da Silva et al. 2015; Zhao et al. 2017; Wang et al. 2011; Xianwen et al. 2009). This type of optimization method is used when a given service is available on multiple servers. Usually, it is because the component is deployed on many servers. The Service Composition method is about deciding which server should execute a service instance. Usage of genetic algorithm is very common in solving this problem. Example of such approach is presented by Ebrahim (Ebrahim 2011). He suggests using a genetic algorithm where the chromosome has a number of genes equal to the number of services that must be called in the process. Each gene indicates an instance of the service that should be called in the process. The best chromosomes are those that provide the best QoS with minimal cost of service and minimal diversity of suppliers. The third type of methods is Service Scheduling (Dyachuk and Deters 2008). It is executed last and it is least popular in the literature. It may be used when multiple service invocations are organized in a queue of one component. Then it is possible to determine the order of their execution. For example, the Service Scheduling method presented in (Dyachuk and Deters 2007) finds services on a critical path of a business process and prioritizes them in the component queue.

All of these three methods are considered independently in the literature, even though they influence each other. Different service composition methods may give best results on different allocations and service scheduling algorithms. It means that we should strive to optimize all three aspects. Such optimization concept is proposed in (Woźniak and Nowicki 2019). It is based on a SOA system simulator that is used to evaluate solutions. The simulator takes, as an input, the SOA system model, which includes: services, components, execution environments, servers, business processes, etc. Each of the above-mentioned elements should be described with attributes, such as a random variable resources (CPU and RAM) used by each service invocation. In addition to the model, the simulator takes, as an input, the matrix of resource allocation and selected algorithms of the Service Composition and Service Scheduling methods. During simulation, output values that constitute criteria for selecting the best solutions (out of those that were simulated) may be obtained. To search through solutions,

a genetic algorithm was used in combination with a brute force approach. It is a unique feature of the SOA system allowing to define the optimization criteria from the business process point of view, which will be subsequently translated into the infrastructure. The reason behind this is that in SOA, business processes may be mapped to services. The following optimization model is an extension of one presented in (Woźniak and Nowicki 2019). Its main difference from the three SOA optimization types of methods is that it takes into account that all of those three influences each other so it finds optimal three: resource allocation, service composition algorithm and service scheduling algorithm.

OPTIMIZATION MODEL

Several optimization criteria for business processes in the SOA system may be defined. They are focused on two aspects of the process: service quality for the user and costs for the company. The first two criteria are the following:

1. Average execution time of business processes weighted by the expected number of instances of business process.

$$\bar{k}_1(t,X) = E\left(k_1(t,X)\right)$$

where:

$$k_1(t,X) = \sum_{b=1}^{B}\left(\frac{H_b(t)}{\sum_{i=1}^{B} H_i(t)} \cdot \frac{\sum_{i=1}^{LR_b} CR_{i,b}(t,X)}{LR_b(t)}\right)$$

B – number of business processes,
t – simulation time,
X – analysed solution which consists: boolean matrix of allocation of components to servers (genotype), selected Service Scheduling and Service Composition algorithms,
$H_x(t)$ – expected value for the number of x-type business process instances running,
$CR_{i,b}(t,X)$ – random variable denoting the time of implementation of the i-th instance of the b-th business process during t,
$LR_b(t)$ – random variable denoting the number of completed instances of the bth business process during time t.

2. Average variance of execution time of business processes

$$k_2(t,X) = \sum_{b=1}^{B}\left(\frac{H_b(t)}{\sum_{i=1}^{B} H_i(t)} \cdot \frac{\sum_{i=1}^{LR_b}\left(\frac{\sum_{i=1}^{LR_b} CR_{i,b}(t,X)}{LR_b(t)} - CR_{i,b}(t,X)\right)^2}{LR_b(t)}\right)$$

The costs criteria are included in the form of resources that are used by the system to calculate how they should be minimized. They are defined in the following manner:

1. The expected amount of processor resources used to provide services over a given time t.

$$\bar{k}_3(t,X) = E\left(k_3(t,X)\right)$$

where:

$$k_3(t,X) = \frac{\sum_{s=1}^{S} m_s tu_s(t,X)}{\sum_{s=1}^{S}\left(m_s \cdot Y(s)\right)\cdot t}$$

S – number of servers,
m_s – computational power of the s-th server,
$tu_s(t,X)$ – random variable denoting the time spent by s-th server on processing services
Y(s) – function Y (s) takes the value 1 if any component is assigned to server s.

2. expected utilization rate of allocated memory resources for the provision of services during t

$$\bar{k}_4(t,X) = E\left(k_4(t,X)\right)$$

where:

$$k_4(t,X) = \sum_{s=1}^{S}\left(p_s \cdot Y(s)\right)\cdot t$$

p_s – amount of RAM on s-th server.

However, during the tests of optimizers, the $\bar{k}_3$ criterion had two effects. The first one led to maximizing the use of a processor, which was beneficial. The second effect promoted the solutions that had long queues leading to longer execution times of business processes, which was unintended. The effect was partly nullified by the first criterion. However, to increase the convergence of the method, it was decided to replace it with a simpler one:
The expected degree of utilization of allocated processor resources for the provision of services in a given time t

$$\bar{k}_5(t,X) = E\left(k_5(t,X)\right)$$

where:

$$k_5(t,X) = \sum_{s=1}^{S}\left(m_s \cdot Y(s)\right)\cdot t$$

Furthermore, during the experiments, it was noticeable that some solutions with high evaluation scores, according to the above criteria, had many unrealized business processes. The process may be abandoned if a server does not have enough resources to realize it or is damaged. To solve that, the another criterion was added:

$$\bar{k}_6(t,X) = E\left(k_6(t,X)\right)$$

where

$$k_6(t,X) = \sum_{b=1}^{B} r_b(t,X)$$

$r_b(t,X)$ – random variable denoting the number of unrealized instances of a b-th type process.
Therefore, the target function of such optimization may be defined as:

$$\bar{k}(t,X)=\left(\bar{k}_1(t,X),\bar{k}_2(t,X),\bar{k}_4(t,X),\bar{k}_5(t,X),\bar{k}_6(t,X)\right)$$

Not all component allocations are acceptable solutions. At least three restrictions on the solution should be defined:

Restriction 1. Processor. The server processor power should exceed its consumption as resulting from the operations of the components and execution environments.

$$\forall_{s\in(1,S)} \sum_{k=1}^{K} K_{k,s}M_k + \sum_{su=1}^{SU} S_{su,s}MU_{su} < m_s$$

where:

K – number of components

$K_{k,s}$ – binary value whether the k component has been assigned to the server s

M_k – amount of processing power consumed by k-th component

SU – number of execution environments,

$S_{su,s}$ – binary value whether the su-th execution environment has been assigned to the server s,

MU_k – amount of processing power consumed by su-th execution environment.

Restriction 2. RAM. The server RAM resources should exceed its consumption as resulting from the operations of the components and execution environments.

$$\forall_{s\in(1,S)} \sum_{k=1}^{K} K_{k,s}P_k + \sum_{su=1}^{SU} S_{su,s}PU_{su} < p_s$$

where:

P_k – amount of RAM consumed by k-th component

PU_{su} – amount of RAM consumed by su-th execution environment

Restriction 3. At least one instance of each component should be deployed.

$$\forall_{k\in(1,K)} \exists_{s\in(1,S)} K_{k,s} = 1$$

SIMULATOR IMPLEMENTATION

To evaluate the solutions using the above criteria, the simulator of the SOA system was implemented in Java language with the DISSim library. DISSim is a toolset for developers that allows to perform discrete event simulation. It uses BasicSimObject class for every element that is simulated and generates events in simulation time. The second class is BasicSimEvent that is put on simulation calendar. A simulation engine searches through the calendar and picks up the nearest event. Every event has a code attached thereto that is executed when the simulation time comes.

Class Model of the Simulator

The BasicSimEvent and BasicSimObject classes are abstract. The latter is a parent object to the Organization and Server classes (Figure 1). The organization starts and contains instances of business processes that are specified by the Business Process Definition. The business process is defined as a graph, in which each step is a service. All services in the process are interconnected by arches described with the probability of choosing each path (which reflects the operation of the gates in BPMN). The services are described by the processor power, the RAM they need for their operations and the volume of data that should be sent through the network to provide the service. The services are associated with the components that execute them. The components contain a list of execution environments on which they can run. The components and execution environments are run on a server, which is a simulation object. They are described by the processor power and RAM memory necessary for their operations. The servers have specified amount of the processing power and RAM needed to run the components and execution environments as well as to provide the services. In addition, the servers are described with a matrix of network bandwidth between them. What is more, each server class object contains random variables that indicate the time of its damage and repair. Server damage events are created after starting the simulator in random time according to the random variable assigned to the server. When a server damage event occurs, the server's status is changed to inoperative and a server repair event is generated at a random time from the time of failure. The repair event generates the damage event, etc.

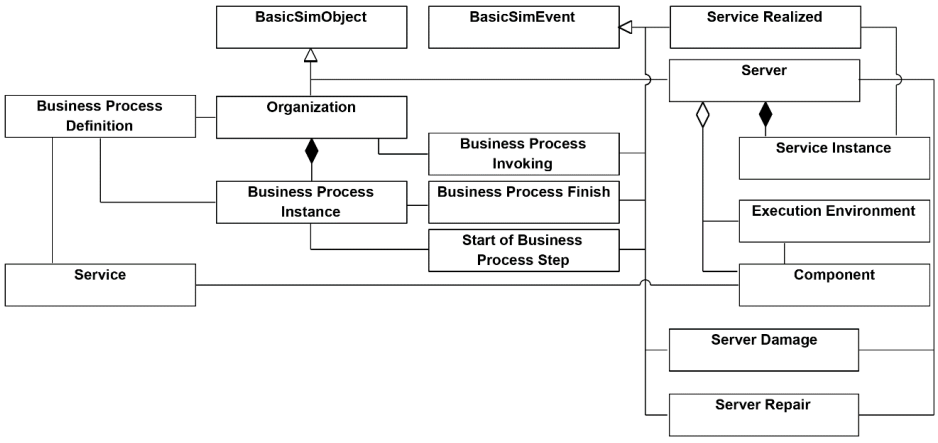


Figure 1 Class Model of the Simulator

Simulation Process

Creating the Business Process Instance

The events in the simulator are interdependent, and the logic of their occurrence is presented in Figure 2. After running the simulation, the "cyclic invocation of business process instance" events are generated. There is one event for every business process definition. It represents the creation of a new business process instance.

Once the simulation time reaches the event time, a new process instance and a new calendar event are created and will occur for the randomly generated simulation time. The time between the successive process start events is established according to the random variable specified in the business process definition.

Business Process Realization

When the business process instance is created, its first step with the current simulation time is generated. Each event representing a step in the business process aims at invoking the services to accomplish such step. First of all, a server is appointed to execute the service. It is the operation of the load balancer, which consists in the selection of the correct instance of the component, i.e. implementation of the Service composition algorithm. Two Service Selection strategies have been implemented in the simulator:

• select least loaded server,

• select server with the shortest expected response time.

It is possible to add further Service Composition algorithms to the simulator. If a server capable of performing the step in the process is not found, then the process is terminated and the information about the

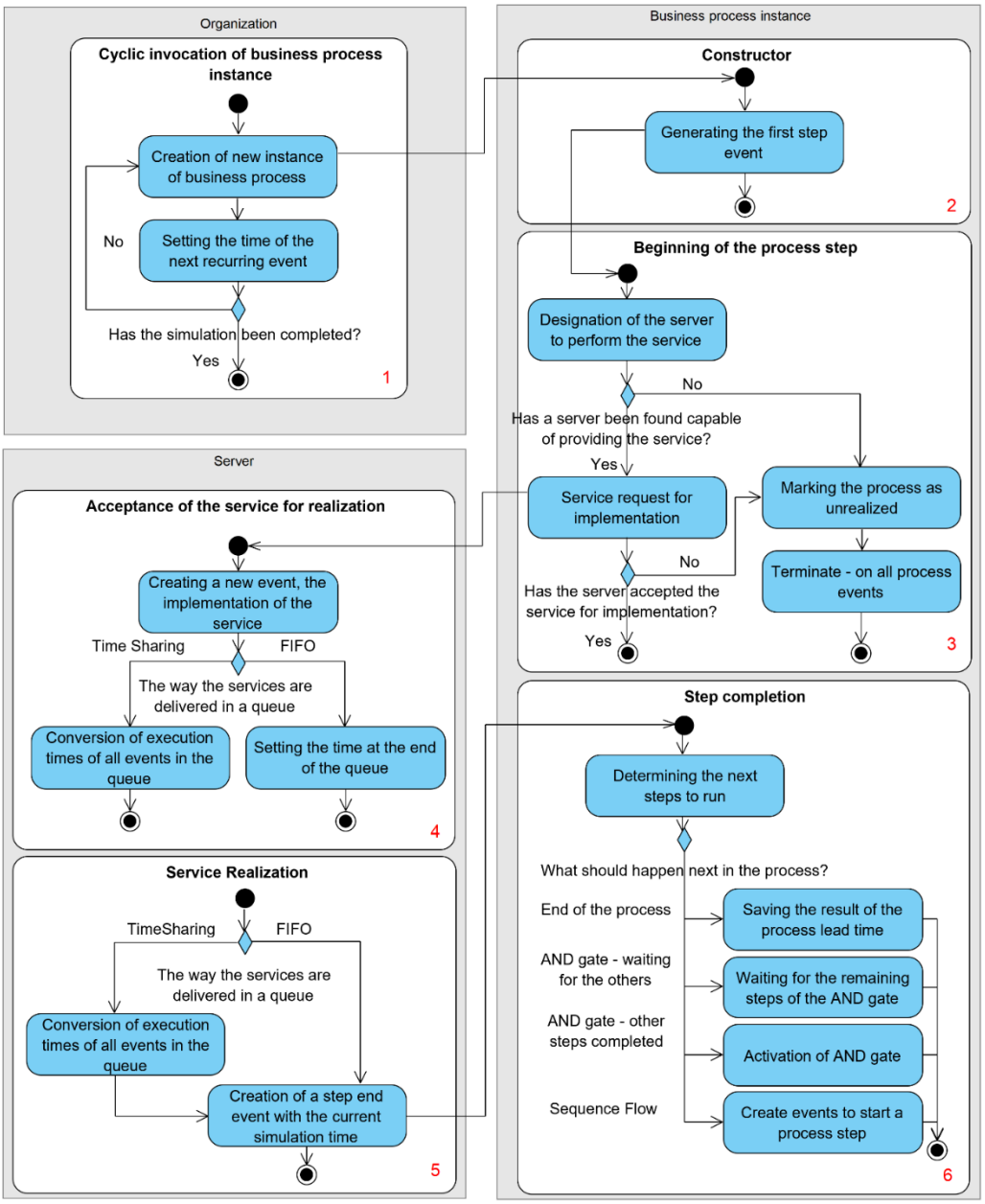


Figure 2 Simulation Process of the SOA System

system's inability to implement the process is included in the simulation results.

Service Execution
If the load balancer has found the server capable of providing the service, i.e. the server that:
• has enough free RAM memory,
• has a component capable of providing the service,
a service order is created. The time necessary for the service to be completed is the sum of:
• data transfer through the network,
• service execution time on the server.
The transfer time depends on the volume of data to be transferred, as defined in the service, and the network bandwidth. The execution time depends on:
• the processor power allocated by the server to execute the services,
• the number of services invoking the orders,
• the power needed to perform the services,
• the component operation model (FIFO or Time Sharing).
If the component operates in the time-sharing mode, then each appearance of a new service to be executed and each termination of the service require recalculation of the expected service realization times.

Business Process Step Execution
The completion of the service creates a step completion event in the business process. Its aim is to determine the next steps. It may cause the termination of the process or creation of start events for one or more steps. If one of the XOR, OR or AND gateways was used after the step in the business process, then such step is interrelated with many other subsequent steps. Each relationship is described by the probability of path selection for XOR and OR gates. In case of the XOR gate, exactly one next step in the process is selected within the probability of the relations that add up to 1. In case of the OR gate, the probability of each path is calculated independently. Their sum may be greater than 1, hence, one or more of the following steps may be chosen. For the AND gate, all of the following steps are always run. The same is true for connecting gates (before the step) - in this case, it is essential to first complete one or more of the previous steps before starting the next step.
The above-described process is executed for every instance of the business process that can occur in large numbers. Simulation is performed for a fixed simulation time defined as a parameter before its beginning.

OPTIMIZER IMPLEMENTATION

Population Initiation

The optimizer is based on multiple simulations organized in a genetic algorithm and the brute force algorithm. The brute force algorithm is a loop that executes the genetic algorithm for each combination of the Service Composition and Service Scheduling methods. The genetic algorithm is used to find optimal resource allocation for the Service Composition and Service Scheduling methods. The genetic algorithm starts with the generation of the population of genotypes. A genotype is a Boolean matrix that shows allocation of components to servers, where 1 indicates that the component has been allocated to the server. To generate an initial population, two layers of randomness are applied. The first one is the generation of numbers between 0 and 1 for each genotype. The number represents the probability of allocation. The second layer is randomization of 0 or 1, which shows whether the component is allocated to the server. This randomization is done with the probability of allocation from the first layer. In this way, not only different allocations are analyzed, but also solutions with different density of allocation.
After the generation of the initial population, it is necessary to adjust the solutions so that all restrictions are met. To that end, the genotypes that do not meet the first two restrictions are selected. Subsequently, the loop is executed for each server that is overloaded. Within each iteration of the loop, the randomly selected components and their execution environments are removed from the server allocation. The process continues until the server is not overloaded anymore. To guarantee that the third restriction is met, all genotypes that have at least one component not allocated to any server are selected. Thereupon, in case of each such component, a randomly selected server having enough resources to handle it is allocated.

Solution Evaluation and Selection

To evaluate the solutions, each genotype is simulated. Each genotype is simulated multiple times to minimize the influence of randomness. During the simulation, the values of every decision criterium are subject to measurement. The criterion evaluation is averaged across multiple simulations of the same solution. After all solutions within a given population have been simulated, the ideal solution with the best values in each criterion in the population is formulated. The ideal solution is hypothetical. It is used as reference to evaluate other solutions. The values of all solutions in all criteria are normalized, where 0 constitutes the criterion value of the ideal solution and 1 - the worst value ever found. Subsequently, the distance from a given solution to the ideal one is calculated using the Euclidean metric. Best solutions are those that are closest to the ideal solution. The next step is the selection process. To do that, genotypes are sorted from best to worst. Survival chances are allocated to all solutions linearly, where the best solution has the probability of survival to the next population equal to 1. The worst solution has the probability of 0. The solutions are eliminated from the population according to the probability assigned thereto.

Crossover and Mutation

The final steps of the genetic algorithm are crossover and mutation. During the crossover, new solutions are generated. Each new genotype has two parents randomly

selected. The probability of being selected is higher in case of better solutions. The weight of being picked as a parent is the same as the probability of survival in the previous step. Each pair of parents has two children. The genes of the children are randomly picked from one of the parents. If the first child inherited a gene from one parent, then the second child inherits it from the other parent. Next, new genotypes are undergoing mutation. There is a small probability that each gene will be mutated. Mutation is changing the value of a gene from 0 to 1 or from 1 to 0. This is to broaden the spectrum of the solutions searched. Finally, to ensure that all restrictions on solutions are met, the same algorithm as in case of the population initiation is performed.

PERFORMANCE

The presented optimizer was used to find solutions to multiple problems. To test convergence, the same problem was optimized with multiple times using different seeds. Convergence depends on the following parameters:

- number of genetic algorithm iterations,
- size of population,
- variance of the simulation output data (which may be minimized by increasing the simulation time and number of repetitions),
- size of the problem (number of: servers, components, business processes, etc.).

The values of such parameters may be increased to achieve better convergence, but it would also make the optimization time longer. In the end it all comes down to the processing power and time. The more we have of those, the better convergence may be achieved.

The simulation time is not only dependent on how long it has to be processed, but also on how many events must be executed in the environment. The number of events depends on the number of business processes and their two attributes:

- expected value of a random variable of time between business process invocations,
- expected number of steps in a process to complete it.

Furthermore, there is one more value that has great impact on the simulation time. It is the ratio of load generated by the processes to available resources. The more load generated by the processes in comparison with the available resources, the longer service queues on the components. When the service instance is executed, the estimated execution times of all other service instances are updated. In case of a long queue, a lot of services have to be updated. According to the data gathered by a Java profiler, the service instance updating the process may consume up to 90% of the computation power provided to the optimizer. In case of a very short queue (shorter than 1 on average), it does not consume so many resources and the simulation process may be performed up to 10 times faster. Additionally, during experiments, it turned out that with short queues, the Service Scheduling algorithm of the optimal solution had a very low convergence. It is almost as if it was selected randomly. The reason for this is that when the queue is short, then the Service Scheduling algorithm has nothing to optimize.

Table 1 shows the execution times on different parameters for the problem that comprises: 100 business processes and an average of 17.5 steps needed for their execution. Each business process definition had an average time between launches in a range of 1 to 500. This average time was a parameter for calling subsequent instances of business processes with exponential distribution. However this average time between business process instances was subject to changes in different simulation variants. The services were assigned an average execution time on a standard processor from 5 to 120 (note that one server has multiple processors so real execution time can be much shorter). This value was an input parameter for the execution time of individual service instances that were randomized according to the normal distribution (both the average time and standard deviation). The simulation length was 2000 time units. Every solution was simulated 10 times to evaluate the values of its decision criteria. There was 3·10^6 solutions searched. Each solution had resource allocation problem size of 50 components allocated to 30 servers. The ratio between the resources required by the processes and the resources available on servers were from 2 to 1 (note that not all servers are used to minimize the resources consumed in the optimal solution). Optimization has been done on virtual machine with 3 Intel Xeon E5-2640 2.60 GHz virtual cores.

Table 1. Optimization times of problems with different amount of business process instances

Average time between invocations	Average number of events in simulation	Total optimization time (days)	Average time of evaluation of 1 solution (seconds)
10	350000	15,202	0,438
20	175000	8,445	0,243
40	87500	6,110	0,176

The differences in values of decision criteria between the variants varied between 1% and 15%, except for the variance of the business process execution time, which differed between 18% and 47%. Best convergence (1% defference between best solutions of different seeds) had been achieved when optimization was performing for 60 days for a problem of same size as presented above.

Another experiment was also carried out on the same problem, but with different population sizes and the number of iterations as shown in table 2. Each experiment was repeated five times, and the results of the time of each repetition are shown in Figure 3. The results show that the number of iterations in the proposed solution has a much greater impact on simulation time than the size of the population. Experiment 4 was carried out much longer than experiment 3 despite the need to review the same number of solutions. The reason for this may be greater optimization convergence with the

parameters of experiment 3 and thus a greater proportion of simulations closer to optimal, which require less computing power. Concept to combine all three types of optimization methods of business processes in SOA is new so it is impossible to compare those results to others.

Table 2. Parameters of experiments

	Exp. 1	Exp. 2	Exp. 3	Exp. 4
Iterations	5 000	5 000	5 000	10 000
Population size	100	150	200	100

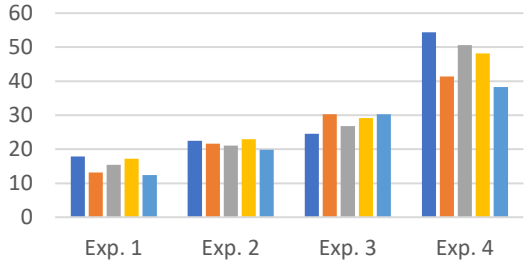


Figure 3 Execution time (days) of each experiment repetition.

SUMMARY

The results of the proposed method may be interpreted dually. On one hand, it may give optimal solutions with high convergence, but on the other, it greatly depends on the resources and the problem size. To optimize it, greater utilization of multi-threading should be implemented in the simulator. Each simulation should be run as a separate thread. If further increase of its efficiency is needed, additional changes could be made. For example, each solution could be simulated on a different virtual machine. It is up to the user to decide if such efficiency of the optimizer is satisfactory. Still, the core of the optimizer would be same as described in the paper. What is more, with proper amount of time, it may have high convergence. As this method is new, it is impossible to compare it to other optimizers.

REFERENCES

Almeida J.; V. Almeida; D. Ardagna; C. Francalanci; and M. Trubian. 2006. "Resource Management in the Autonomic Service-Oriented Architecture". *IEEE International Conference on Autonomic Computing*. Dublin, Ireland.

BPMN Specification documents. Accessed 31.01.2020. https://www.omg.org/spec/BPMN/2.0/About-BPMN/.

da Silva A. S.; H. Ma; and M. Zhang. 2015. "A GP approach to QoS-aware web service composition including conditional constraints". *IEEE Congress on Evolutionary Computation (CEC)*. Sendai, Japan.

Dyachuk D.; and R. Deters. 2008. "Ensuring Service Level Agreements for Service Workflows". *IEEE International Conference on Services Computing*. Honolulu, USA.

Dyachuk D.; and R. Deters. 2007. "Service Level Agreement Aware Workflow Scheduling". *IEEE International Conference on Services Computing*. Salt Lake City, USA.

Ebrahim G. A. 2011. "Intelligent Composition of Dynamic-Cost Services in Service-Oriented Architectures". *Fifth UKSim European Symposium*. Madrid, Spain.

Huang K. C.; Y. C. Lu; M. H. Tsai, Y. J. Wu; and H. Y. Chang. 2016. "Performance-Efficient Service Deployment and Scheduling Methods for Composite Cloud Services". *IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC)*. Shanghai, China.

Mennes R.; B. Spinnewyn; S. Latre; and J. F. Botero. 2016. "GRECO: A Distributed Genetic Algorithm for Reliable Application Placement in Hybrid Clouds". *5th IEEE International Conference on Cloud Networking (Cloudnet)*. Pisa, Italy.

Schmid M. 2011. "An approach for autonomic performance management in SOA workflows". *12th IFIP/IEEE International Symposium on Integrated Network Management and Workshops*. Dublin, Ireland.

Wang Z. J.; Z. Z. Liu; X. F. Zhou; and Y. S. Lou. 2011. "An approach for composite web service selection based on DGQoS". *The International Journal of Advanced Manufacturing Technology*, 56 (9). London, Great Britain 1167-1179.

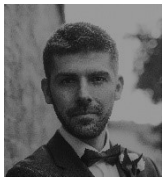
Woźniak A.; and T. Nowicki. 2019. "The Problem of Effective Deployment Architecture in SOA". *Computer Science and Mathematical Modelling* (9). Warsaw, Poland. 33-44.

Xianwen F.; F. Xiaoqin; and C. Jiang. 2009. "An Efficient Approach to Web Service Selection". *Web Information Systems and Mining: International Conference*. Shanghai, China. 271-280.

Xie L.; J. Luo; J. Qiu; J. A. Pershing; Y. Li; and Y. Chen. 2008. "Availability "weak point" analysis over an SOA deployment framework". *IEEE Network Operations and Management Symposium*. Salvador, Bahia, Brazil.

Zhang C.; R. N. Chang; C. S. Perng; E. So; C. Tang; and T. Tao. 2009. "An Optimal Capacity Planning Algorithm for Provisioning Cluster-Based Failure-Resilient Composite Services". *IEEE International Conference on Services Computing*. Bangalore, India.

Zhao Y.; W. Tan; and T. Jin. 2017. "QoS-aware Web Service Composition Considering the Constraints between Services". *12th Chinese Conference on Computer Supported Cooperative Work and Social Computing*. Chongqing, China.



ADRIAN P. WOŹNIAK studied at the Military University of Technology in Warsaw, where he obtained a degree in Information Technology in 2012. Subsequently, he worked as a system and business analyst. In this role, he designed many SOA systems. After that, his position changed to Architect and then Main IT Architect in the largest retail company in Poland, which allowed him to gain extensive experience in the field of the SOA system optimization problem. At the Military University of Technology, he is a lecturer in software engineering, system design and system integration. The subjects he teaches are also his main areas of research. His e-mail address is: `adrian.wozniak@wat.edu.pl`