Key words – SOA, complex service, optimal composition, Quality of Service

Piotr RYGIELSKI*, Paweł ŚWIĄTEK*

GRAPH-FOLD: AN EFFICIENT METHOD FOR COMPLEX SERVICE EXECUTION PLAN OPTIMIZATION

The task of optimization of the complex service execution plan is the last stage of the process of complex services composition in systems based on the SOA paradigm. In general this task consists of choosing proper versions of atomic services, delivering requested functionalities, such that non-functional requirements for complex service are satisfied. In most real-life cases this task is reduced to the problem of multidimensional knapsack problem (MKP), which is known to be NP-hard. Therefore, in order to facilitate effective complex service composition, it is crucial to provide an efficient MKP solution algorithm. In this paper a novel approach to the problem of optimization of complex service execution plan is introduced. In the proposed approach the MKP feasible solution space is constructed in such a way, that the optimal solution can be found in constant time. Of course, since MKP is NP-hard, the process of solution space construction takes an exponential time, This time, however, is amortized by multiple usages of single solution space for determination of optimal execution plans for multiple complex service requests.

1. INTRODUCTION

In systems based on SOA (Service-Oriented Architecture) paradigm services delivered to end-users (complex services) are composed with use of atomic services (services that have atomic functionality). The functionality of a complex service is an aggregation of functionalities of atomic services [5]. In general system is distributed, what means that applications acting as atomic services can be installed on any machine with communication interface available. An user requesting a service from the system formulates the request that precisely specifies demanded functionality. To deliver the requested functionality system uses the service composition procedure which consists of choosing the proper atomic services with an execution order. Moreover, user is able to formulate an additional terms of service delivery which involves non-functional aspect of delivering the service – Quality of Service requirements. Such complex request for service is called SLA – Service Level Agreement – and uniquely defines functional and non-functional user needs.

1.1. SERVICE COMPOSITION TASK

In general, the task of complex service composition consists of finding, for given ordered set of required functionalities (stated in the SLA), an order of atomic services versions execution such that non-functional requirements are met. The task of complex service composition can be decomposed into three sequential subtasks (see figure 1):

1. Complex service structure composition – transformation of the SLA into set of required functionalities and the precedence relations between functionalities. The result of this task is

* Institute of Computer Science, Wroclaw University of Technology, Wybrzeze Wyspianskiego 27, 50-370 Wroclaw. piotr.rygielski@pwr.wroc.pl, pawel.swiatek@pwr.wroc.pl

complex service structure represented as a directed graph (not connected in general) of required functionalities.

2. Complex service scenario composition – transformation of complex service structure graph into single and consistent graph of required functionalities with precisely defined order of execution of all atomic functionalities. Since it is possible, that single functionality is delivered by more than one atomic service version, the scenario graph represents in fact a family of execution graphs where member graphs differ in atomic service versions applied to deliver required atomic functionality.

3. Complex service execution plan composition – choice of particular atomic services in complex service scenario graph such that non-functional requirements of complex service are met.

In order to deliver complex service with requested functionality and non-functional properties various optimization tasks need to be solved on consecutive stages of complex service composition task (i.e.: stages of complex service structure, scenario and execution plan composition) [9].

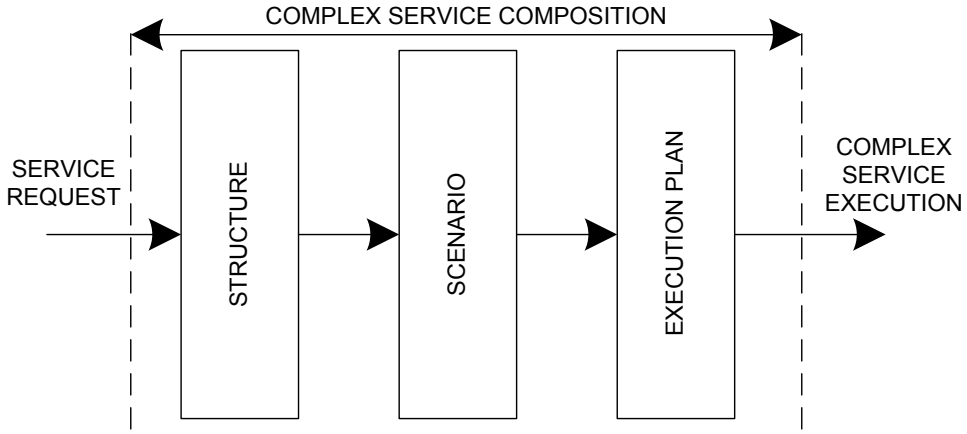

Fig. 1. Decomposition of the process of complex service composition

1.2. COMPLEX SERVICE EXECUTION PLAN

Aim of this paper is to focus on the complex service execution plan determination. Order constraints and atomic services selected under complex service scenario determination process should be processed further to select proper atomic service versions for execution. Atomic service versions can be available at the distributed execution system in many versions (each version differs from other in nonfunctional aspect) so choosing the proper execution plan can be treated as quality optimization task with popular QoS criteria i.e., execution time, cost, security, etc.

2. THE COMPLEX SERVICE SCENARIO

At the third stage of complex service composition process it is assumed that a graph called complex service execution scenario is given. The scenario defines unambiguously the order in which atomic functionalities should be delivered in order to fulfill the user functional requirement. The scenario is modeled as a graph $GC = (VC, EC)$ where VC is a vertex set where one vertex represents functionality to be delivered and EC is a set of edges which defines the order of delivering functionalities. Exemplary complex service scenario is presented on the figure 2.

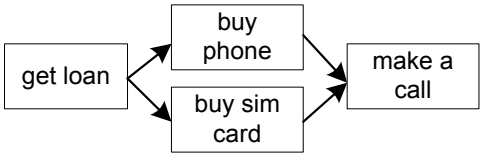

Fig. 2. Exemplary scenario of complex serviced which goal is to make a phone call. Each of distinguished functionalities can be delivered by many versions of atomic services.

An exemplary complex service - which goal is to make a cell phone call - consists of four operations that have to be executed. Assuming that user does not have enough funds to execute this service, first user has to get the funds, for example by getting a loan. A loan can be given by many institutions where each can be treated as another atomic service version. Similar situation is with buying a cell phone – the user can buy the same phone model in many stores – and with the sim card – many cellular operators can offer user a sim card activation service. Finally user can make a call using the newly bought cell phone and the sim card using various versions of calling service e.g. VoIP call, GSM call, video call etc.

An user formulating the request for service defines a set of quality parameters that should be delivered in order to satisfy the user. The requirements concern the whole process of delivering a service and should be taken into consideration during the composition process especially in the execution plan determination stage.

3. PROBLEM FORMULATION

The task of execution plan determination is the last task that needs to be solved in the complex service composition procedure. In general the task of determination of optimal execution plan of complex service can be described in the following way. The systems task is to pick one version of each atomic service defined uniquely by the scenario in such a way, that quality requirements are satisfied. Noteworthy is fact that picking two atomic service versions defines also a communication path that is going to be used to communicate between these services – assuming that there is always chosen the best possible path if there is more than one available in the system. The task is similar to the task of finding an optimal path in graph considered in earlier work [4] but the difference in this approach is the solution represented as a graph - not as a single path. Due to that the former algorithms cannot be used. Formally the task of finding optimal a complex service execution plan is formulated as follows:

For given:

- Scenario graph $GC = (VC, EC)$
- A l-th service request represented by the SLA_l
- Complex service nonfunctional requirement Ψ_l
- Quality criterion $Q(G, \Psi_l)$

Find:

- Such set of atomic services versions for scenario GC that quality criterion is minimized

$$as_{1j_k^*}, \ldots, as_{kj_k^*} = \arg\min_{as_{1j_k}, \ldots, as_{kj_k}} Q\left(G\left(\left\{as_{1j_k}, \ldots, as_{kj_k}\right\}, E\right), \Psi_l\right)$$

where:

- as_{1j_k} denotes j_k -th version of k-th atomic service

The problem stated above can be easily transformed into multi-choice knapsack problem in the following way. The task is to pick exactly one item (atomic service version) from each group (atomic service) in such way that quality of represented by picked items (complex service) is optimal in sense of chosen quality criterion Q.

The solution of the formulated problem is known and some algorithms solving such problem has been presented i.e. in [7]. The problem formulated at the beginning of this chapter can not be directly transformated into multichoice knapsack problem because of execution scenario of complex service is given as a graph in general. In order to transformate the problem of determination the optimal execution plan into multi-choice knapsack problem, one has to reduce scenario graph into connected directed graph having all vertices with input and output degree no higher than one – in the special case just a graph with only one node. In order to solve the stated problems, the procedure of graph reduction is proposed in next chapter of this paper.

4. SCENARIO GRAPH REDUCTION PROCEDURE

The goal of the scenario graph reduction procedure is to transform the scenario graph into the graph having exactly one node. Transformation of the input scenario graph consists of determination of sub graph - called here a pattern - and defining a result of reduction for the considered pattern. Moreover, all versions of atomic services present within this pattern should be aggregated into new resulting vertex in such way that latter distinguishing the original versions in possible. The most important part of pattern reduction is to find an aggregation function that aggregates the values of the nonfunctional parameters into the new ones in the newly created vertex. The whole procedure of pattern reduction has been described in the section below.

4.1. PATTERN REDUCTION PROCEDURE

In order to reduce the scenario graph into the desired form, the set of patterns with aggregation function has to be defined. In the figure 3 there has been presented two scenario subgraph patterns (subfigures a and c) with corresponding resulting vertex (subfigures b and d respectively) obtained using aggregation function. The original versions of the atomic services are being merged into the new ones in the newly created vertex. The aggregation functions used in the examples depicted in the figure 3 – considering execution time as the nonfunctional parameter - are sum and sum-max - respectively for serial reduction (subfigures a,b) and split reduction (subfigures c,d). Therefore the resulting versions of k-th atomic service obtained in process of serial reduction have the following values of execution time: $d(k_1) = d(i-1,1) + d(i,1)$; $d(k_2) = d(i-1,1) + d(i,2)$; $d(k_3) = d(i-1,2) + d(i,1)$; $d(k_4) = d(i-1,2) + d(i,2)$; where $d(i,1)$ denotes the execution time of first version of i-th atomic service. Respectively the execution times for split reduction procedure can be calculated by following the following pattern: $d(k_1) = d(i-1,1) + \max\{d(i,1),\ldots,d(i+n,1)\}$; $d(k_2) = d(i-1,1) + \max\{d(i,1),\ldots,d(i+n,2)\}$; and for the last m-th version of the newly created service: $d(k_m) = d(i-1,2) + \max\{d(i,2),\ldots,d(i+n,2)\}$;. The number of versions in the newly created service can be calculated using the equation 1

$$m = \prod_{j=0}^{J} size(as_j), \tag{1}$$

where $size(as_j)$ denotes the number of versions of j-th atomic service and J denotes the number of atomic services present in the pattern.

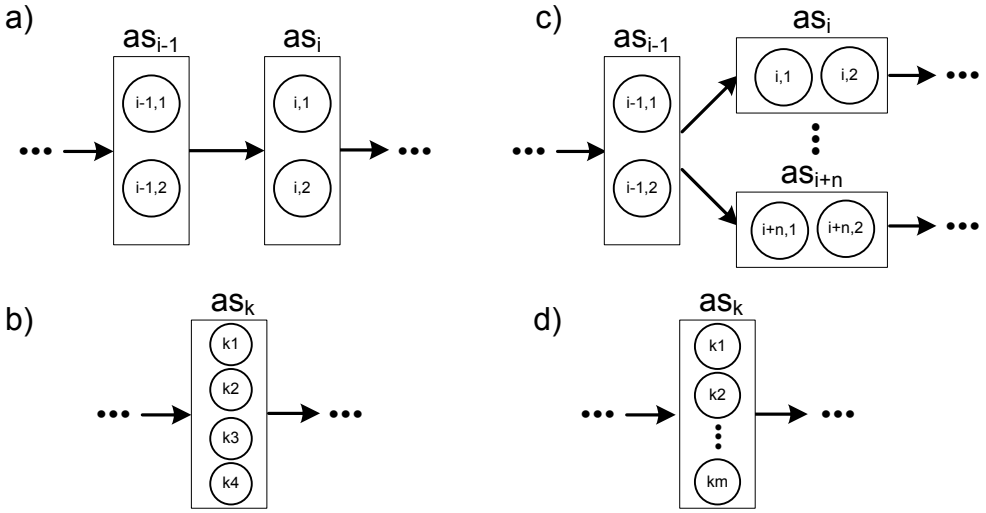

Fig. 3. Two subgraph patterns (serial - a and split - c) with the corresponding results of reduction (b and d respectively). Atomic service versions before reduction (a,c) are transformed into the new ones (b,d) which nonfunctional parameter values were also recalculated using proper aggregation functions. In this example the number of atomic service versions in serial reduction equals to 4 and in the split reduction can be calculated using equation (1).

Except of patterns reducing vertices one can distinguish the edge reduction pattern which leads to decrease the graph complexity. The redundant edge reduction pattern has been presented in figure 4.

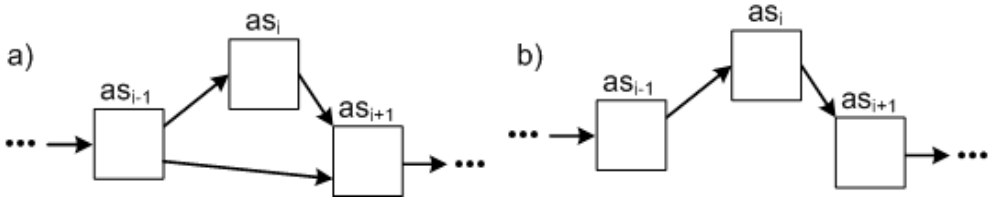

Fig. 4. Possible edge reduction non violating precedence relations between vertices of scenario graph; a) grah before edge removal, b) graph after removing the redundant edge.

Without loosing any information one can remove edge (as_{i-1}, as_{i+1}) because of transitivity of the precedence relation: $as_{i-1} \prec as_i \wedge as_i \prec as_{i+1} \Rightarrow as_{i-1} \prec as_{i+1}$. It is obvious that one can distinguish other patterns of edge removal if and only if one does not cause the loss of information in case of such a reduction.

The reductions defined formerly are the basic ones and one can define more of such reductions to cope with more specific scenario graphs in more efficient way. The complete reduction operations set should contain at least one vertex reduction operation and at least one redundant edge reduction. Such a set of reductions should give the guarantee to reduce any scenario graph into the form of single node.

4.2. THE REDUCTION ALGORITHM

In order to reduce the scenario graph to the form that is appropriate for the formulated problem one has to perform the steps concerning all vertex reduction patterns and all edge reduction patterns. Assuming that each edge reduction procedure reduces no less than one or no less than two nodes (in case of node pattern reduction), the whole reduction process makes maximum $\frac{1}{2}|VC| + |EC|$ reductions.

Each reduction has also its computational complexity which is influenced by the following operations: the complexity of procedure of finding a pattern, complexity of calculating the values of non-functional parameters of the newly created node, disconnecting removed vertices and connecting the newly created one. All of the mentioned procedures can have different implementations so complexities may vary. Trying to estimate the worst case complexities of former operations the results has been presented in table 1.

Operation	Complexity	Denotation
Finding an pattern	$O(kn)$	$n = \|VC\|$ k – largest output degree of vertex in graph M - number of vertices in the reduction pattern J_m – number of versions of m-th atomic service
Calculating new values of non-functional parameters	$O\left(\prod_{m=1}^{M} J_m\right)$	
Disconnection and connection of graph vertices	$O(n)$	
Finding and reducing redundant edge	$O(n^3 \log n)$	

Table. 1. Estimation of calculation complexities of operations used in reduction procedure.

Noteworthy is fact that vertices that are disconnected can be removed and not considered further, so value of n ($n = |VC|$) is decreasing during the process of reduction. Decreasing speed depends on number of vertices that are being reduced during the node reduction operation. The general procedure consists of looped execution of all reduction operations until the vertices count $n = 1$.

Example of the reduction process has been presented in the figure 5. Scenario graph used in this example contains six vertices (functionalities) with precedence relations given as on subfigure 1 of figure 5. Each functionality in the considered scenario graph has two versions of atomic service. In the first step there is a *split-reduction* pattern found and the result of reduction is presented on subfigure 2. In the second step situation repeats – split reduction pattern is found and reduced. Notice that number of versions in the resulting vertex grows. On subfigure 3 the serial reduction pattern is found and reduced. The algorithm stops in step four where only one vertex is left (subfigure 4). The resulting vertex has 64 versions

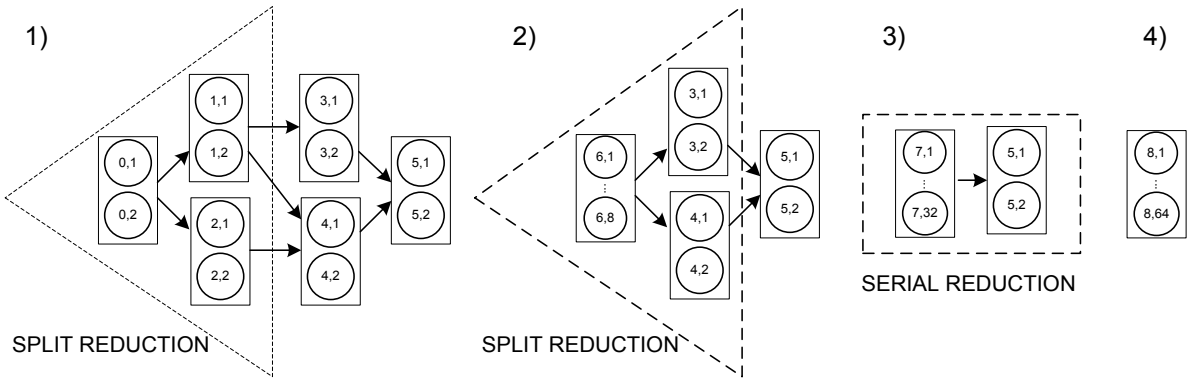

Fig. 5. Exemplary reduction process for scenario with six functionalities (atomic services), each having two versions. After two split reductions and one serial reduction the result is single atomic service with 64 versions. Every newly created functionality gets new number for ease of distinguishing.

4.3. THE GRAPH-FOLD PROCEDURE

The general procedure of folding the graph has been presented on the listing 1. Moreover, the spit reduction function has been presented. Other reductions has been omitted due to analogy to the split reduction function.

```
function GRAPH-FOLD(GC,qos)
   result <-- GC
   memory <-- empty
   do
      split <-- REDUCTION_SPLIT(result, qos, memory)
      serial <-- REDUCTION_SERIAL(result, qos, memory)
      edge <-- REDUCTION_EDGE(result)
   while(split + serial + edge > 0)
   RETURN qos
END


function REDUCTION_SPLIT(result, qos, memory)
   i <-- find_reduce_split (result)
   if i > -1 then
      create_new_node
      predecessors(new_node) <-- predecessors(i)
      for each j in successors(i) do
         successors(new_node) <-- successors(j)
      end
      qos_aggregation(new_node, i, qos, memory)
      remove(successors(i))
      remove(i)
      return 1
   end
   return 0
END
```

Listing. 1. A pseudo-code of the general procedure of folding the scenario graph GC with split reduction function. Other reductions has been omitted due to analogy to the spit reduction function.

The general graph-fold procedure consist of sequentially checking and reducing split and serial node reduction pattern and edge reduction pattern. The algorithm stops if there are no nodes and edges possible to reduce.

The split reduction procedure consists of finding a node that can be reduced with all its successors. If such a node is found, the new node is created. All successors of the successors of the found node are being attached to the newly created node and all predecessors of the found node become the predecessors of the new node. The old node along with all its successors are being removed after the quality parameters values are aggregated and saved into the new node. The serial reduction procedure is analogical to the split reduction procedure. The edge reduction procedure and quality values aggregation has been explained by example in the former sections of this paper.

5. PROBLEM SOLUTION

The problem of multi-choice knapsack problem formulated in chapter 3 can be solved using heuristic algorithms [2,7], but much more interesting solution can be obtained when scenario graph has been reduced to single vertex. In case of such a reduction one gets single atomic service with *m* versions. Moreover if there are used data structures like AVL trees [3] as a collection of QoS values for the resulting node's *m* versions, the optimal solution of the complex service execution plan determination problem can be found in $O(\log m)$ time.

Depending on the optimization goal given by quality criterion Q, one can achieve different graph-fold algorithm behavior. According to the popular quality of service delivery approaches - best-effort approach and differentiated services approach - one can formulate such a quality criterion that desired algorithm behavior can be achieved.

5.1. COMPLEX SERVICE EXECUTION TIME MINIMIZATION

This solution bases on the best-effort approach and finds lowest obtainable time of execution of the complex service. The complex service execution time can be minimized using graph-fold algorithm when execution time is considered as the quality parameter. In this case the following optimization task should be solved:

$$k_m^* = \arg\min_{k_1,\ldots,k_m} d(k_m), \tag{2}$$

where $d(k_m)$ denotes execution time of *m*-th version of the resulting *k*-th atomic service obtained after graph reduction procedure. According to the assumption that resulting collection of atomic service versions is ordered by the $d(k_m)$ value, the solution can be found in $O(1)$ and consist on picking the version with the lowest execution time value.

5.2. COMPLEX SERVICE EXECUTION TIME GUARANTIES

This solution bases on the differentiated services approach and finds optimal execution time value with respect to given execution time requirement. In this case the following optimization task should be solved:

$$k_m^* = \arg\min_{k_1,\ldots,k_m} \left(d(k_m) - d^*\right)^2, \tag{3}$$

with respect to the following constraint:

$$d(k_m^*) \le d^*, \tag{4}$$

where d^* denotes the execution time requirement given by user. According to the assumption that resulting collection of atomic service versions is ordered by the $d(k_m)$ value, the solution can be found in $O(\log m)$ and consist on picking the version with the highest execution time value fulfilling the user requirement. In the case when there is no valid solution, the request can be handled in one of the following ways: the SLA contract should be renegotiated, the request should be rejected and not executed in the system or can be served as in the best-effort approach.

6. SIMULATION STUDY

In order to evaluate the quality of service delivered by presented the graph-fold algorithm the simulation environment has been developed in OMNeT++ simulation framework [8]. The simulator consists of three distinguishable modules: generator module, abstract Enterprise Service Bus and set of web services representing atomic service versions. The simulation run consisted of generating stream of complex service requests with interarrival time given by exponential distribution with mean 0.1 second and random complex service execution scenarios. The generated request was processed in abstract ESB module where the task of finding the optimal execution plan was solved. There were six algorithms for finding optimal execution plan present in the system – five of those were reference algorithms and sixth one was the graph-fold algorithm. Each atomic service version present in the system had different performance index generated from exponential distribution with mean 10 expressed in kilobytes that can be processed per second. Moreover request data size that was going to be processed was also generated from exponential distribution with mean 10KB. The number of atomic services was set to 8 (also maximum 8 functionalities were present in generated scenario) each having 10 versions.

Using such a configured simulation environment there were two experiments performed. First consisted of solving the complex service execution time minimization task and second of solving the complex service execution time optimization task, where additionally the execution time requirement has been set by the user.

The results of the first experiment have been presented in the figure 6. The experiment lasted for 1000 seconds of simulation and consisted on determination of the complex service execution plan at the moment when the request was entering the system. The first observation shows that the graph-fold algorithm obtained the lowest execution time from all tested reference algorithms. Moreover, the execution time line on the chart is very close to the constant value of about 3 seconds while the rest of algorithms resulted much higher variation of execution time. Second best was best-avg algorithm which consisted of choosing such version of atomic service that average execution time was lowest. Other algorithms resulted in worse complex service execution time. The main disadvantage of the examined algorithms was the fact that the decision was made at the very beginning of the request processing. During the processing of the request the system's state could change what was impossible to take into consideration at the moment of decision making.

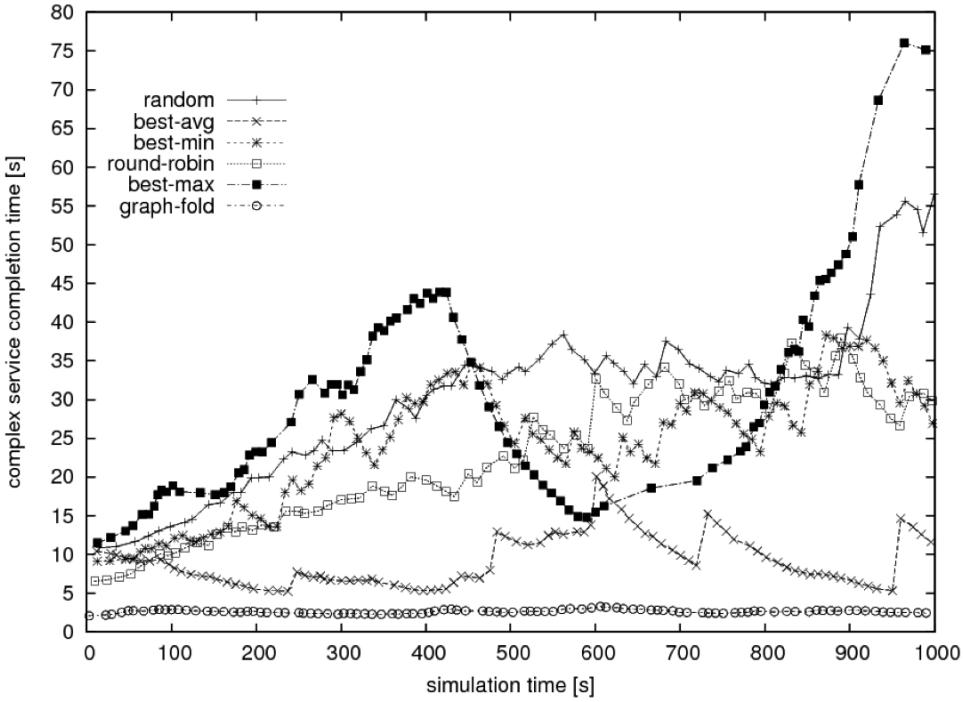

Fig. 6. Simulation results for the six algorithms solving the problem of complex service execution time minimization.

The results of the second experiment have been presented in figure 7. The task of the graph-fold algorithm was to keep complex service execution time as close as possible to the requirement which was set by user to 5 seconds. The complex service execution time was chosen again as a quality parameter. The graph-fold algorithm used the shifting window average in order to collect the data about atomic services versions execution time. On the chart presented in figure 7 one can notice, that the time of execution calculated by the graph-fold algorithm was always lower or equal to the requirement. In the case when system was not able to provide required execution time (in the period between 800 and 1000 seconds of simulation) lower value was assured. This was caused by the fact that no proper complex service version was found so the system delivered the closest to the optimum. Analyzing the real complex service execution time curve in the figure 7, some deviations might be noticed. The real execution time approaches required value after 150 simulation seconds and was kept within about 5% error range utill the end of simulation. The deviation of real complex service execution time with respect to the calculated time is caused by the no ideal method of prediction of single atomic service version execution time. The decision for each request was made at the moment when the request was entering the system so the values describing execution time might have changed during the execution process.

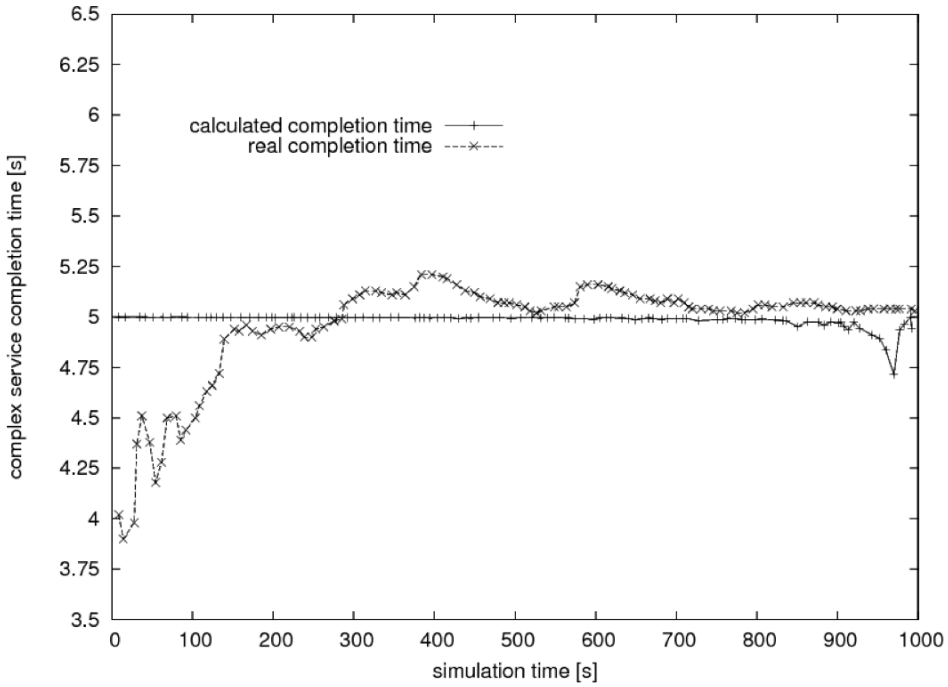

Fig. 7. Simulation results for the graph-fold algorithm solving the task of complex service execution time optimization. The calculated completion time curve is one predicted by the algorithm at the beginning of processing the request. The real completion time value was measured when the request has been completely processed.

7. CONCLUSION

The graph-fold algorithm presented in this paper is a good and precise approach for delivering optimal complex service execution plan in the process of complex service composition. However reader should remember that the graph-fold was designed in order to transform the complex service scenario graph to the single node with *m* versions. The abilities to develop the simple decision making algorithms comes straight from the fact, that the resulting versions are ordered and one can pick the worst and the best version very quick. The graph reduction procedure proposed in this paper completes the other approaches proposed in the literature [1,2,6,7,10]. The most important weak side of proposed approach is the complexity of the procedure which strongly depends on the number of the atomic services versions. The graph-fold algorithm was significantly slower than the reference algorithms what can cause growth of calculation time in case of complex service execution scenarios with large amount of functionalities or large amount of atomic service versions available.

The fact that presented method gives always the precise and optimal solution can be used as a reference point for developed heuristic algorithms. However, the algorithm uses the data about certain atomic services versions quality parameters values, so choosing the best data for the algorithm is crucial. Taking the values that are fast-changing will result in larger difference between the calculated and real complex service execution time. When the considered quality of service parameter values are changing slowly, the algorithm produces the results that are closer to the real ones.

We would like to emphasis that in the specific applications of the graph-fold algorithm other set of reduction operations can be used in order to reduce the patterns which appears much more

often in shorter time. Such a modification can decrease the calculation complexity for the majority of considered complex service scenarios.

8. REFERENCES

[1] M. ALRIFAI AND T. RISSE, *Combining global optimization with local selection for efficient QoS-aware service composition*, WWW '09: Proceedings of the 18th international conference on World wide web, pp.881-890, ACM 2009.

[2] BERBNER, R., SPAHN, M., REPP, N., HECKMANN, O., AND STEINMETZ, R. *Heuristics for QoS-aware Web Service Composition*. In Proceedings of the IEEE international Conference on Web Services (September 18 - 22, 2006). ICWS. IEEE Computer Society, Washington, DC, 72-82

[3] CORMEN T. ET.AL. *Introduction to Algorithms*, MIT Press, Cambridge, MA, Second edition, (2001)

[4] GRZECH A. RYGIELSKI P. ŚWIĄTEK P., *QoS-aware infrastructure resources allocation in systems based on service-oriented architecture paradigm.* Performance modelling and evaluation of heterogeneous networks, 6th Working International Conference HET - NETs 2010, Zakopane, January 14th-16th, 2010 s. 35-47.

[5] GRZECH A. AND ŚWIĄTEK P., *Parallel processing of connection streams in nodes of packet-switched computer communication systems*. Cybernetics and Systems. 2008, vol. 39, nr 2, pp. 155-170

[6] MICHAEL C. JAEGER, GERO MÜHL, SEBASTIAN GOLZE, *QoS-Aware Composition of Web Services: An Evaluation of Selection Algorithms* in On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE (2005), pp. 646-661.

[7] SHAHADAT KHAN AND KIN F. LI AND ERIC G. MANNING, *The Utility Model For Adaptive Multimedia Systems* In International Conference on Multimedia Modeling, 1997 pp.111-126.

[8] OMNeT++ Web page: *http://www.omnetpp.org/*

[9] RYGIELSKI P., ŚWIĄTEK P., *QoS-aware Complex Service Composition in SOA-based Systems*, in "SOA Infrastructure Tools: Concepts and Methods", Springer-Verlang, Berlin 2010.

[10] YU TAO, ZHANG YUE AND KWEI-JAY LIN, *Efficient algorithms for Web services selection with end-to-end QoS constraints*, ACM Trans. Web 2007, Vol. 1, No. 1, Article. 6.