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

Piotr RYGIELSKI*, Paweł ŚWIĄTEK*

OPTIMAL COMPLEX SERVICES COMPOSITION IN SOA SYSTEMS

One of the most important tasks in service oriented architecture paradigm based systems is the task of composition of the complex service. Aim of this paper is to focus on the very last phase of the composition process, where proper atomic service versions are picked to execute the whole complex service. An exact algorithm is proposed in this paper solving two tasks based on popular quality of service delivery approaches: best-effort and differentiated services. The procedure of graph reduction has been introduced to generate space of possible solutions in such way that decision is made in more efficient way. The graph-fold algorithm was tested in simulation environment where its performance was compared to five reference algorithms. Moreover ability to deliver quality at desired level has been tested.

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 a request that is specifying functionality demanded. To deliver requested functionality system uses service composition procedure which consists of choosing proper atomic services with an execution order to satisfy user requirement. 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 them. The result of this task is complex service structure represented as a directed graph (not connected in general) of required functionalities.

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

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].

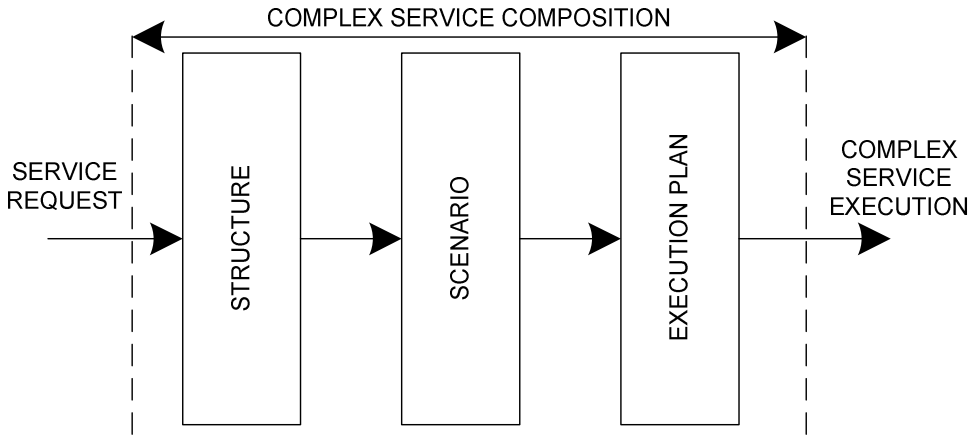

Fig. 1. Decomposition of the process of complex service composition

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 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.

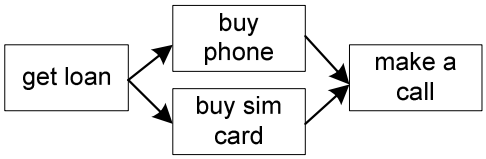

Fig. 2. Exemplary scenario of complex serviced which goal is to make a cell phone call. Each of distinguished functionalities can be delivered by many versions of atomic services.

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 money to execute this service, first user has to get the money, 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 – user can buy same phone in many stores – and with sim card – many cellular operators can offer user a sim card. Finally user can make a call using a newly bought cell phone and sim card using various versions of calling service e.g. VoIP call, gsm call, video call etc.

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

3. PROBLEM FORMULATION

A 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 as follows. The systems task is to pick one version of each atomic service defined uniquely by the scenario in such 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 once. The task is similar to the task of finding an optimal path in graph considered in earlier work [4] but the difference is that in this approach the solution is represented as a graph - not as a single path - and the former algorithms can not be used. Formally the task of finding optimal a complex service execution plan can be 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

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

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 choosen quality criterion Q.

The solution of the formulated problem is known and some algorithms solving such problem has been presented i.e. in [7]. But the problem formulated at the beginning of this chapter can not be directly transformated into multichoice knapsack problem because of scenario of execution of complex service is given as a graph in general. In order to transformate problem of determination of 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 order to solve stated problems the procedure of graph reduction is proposed in next chapter of this paper.

4. SCENARIO GRAPH REDUCTION PROCEDURE

The goal of scenario graph reduction procedure is to transform the scenario graph into connected graph where all vertices have input and output degree no higher than one. Transformation of input graph scenario consists of determination of sub graph - called a pattern - and defining a result of reduction for this defined pattern. Moreover all versions of atomic services present within this pattern should be aggregated in the new resulting vertex in such way that latter distinguishing of the original versions in possible. The most important part of pattern reduction is to find an aggregating function that aggregates values of nonfunctional parameters into new ones in newly created vertex of the reduction process. The whole procedure of pattern reduction has been described below.

4.1. REDUCTION PROCEDURE

In order to reduce the scenario graph into form that stated optimization problems can be solved, the set of patterns with aggregation function has to be defined. On 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 new ones in the newly created vertex. The aggregation functions for the examples given on the figure 3 – considering execution time as a 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 has 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 execution time of first version of i-th atomic service. Respectively the execution times for split reduction procedure can by calculated by following the following pattern: $d(k_1) = d(i-1,1) + \max\{d(i,1),...,d(i+n,1)\}$; $d(k_2) = d(i-1,1) + \max\{d(i,1),...,d(i+n,2)\}$; and for the last m-th version of newly created service: $d(k_m) = d(i-1,2) + \max\{d(i,2),...,d(i+n,2)\}$;. 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 number of versions of *j*-th atomic service and *J* denotes count of atomic services in the pattern.

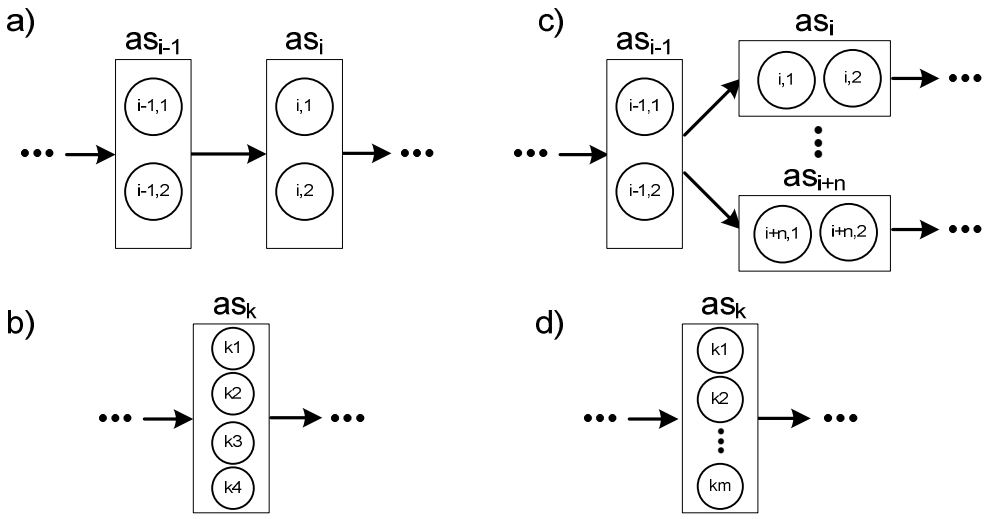

Fig. 3. Two subgraph patterns (serial - a and split - c) and corresponding results of reduction (b and d respectively). Atomic service versions before reduction (a,c) are transformed into new ones (b,d) which nonfunctional parameters values were also recalculated using proper aggregation functions. In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

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

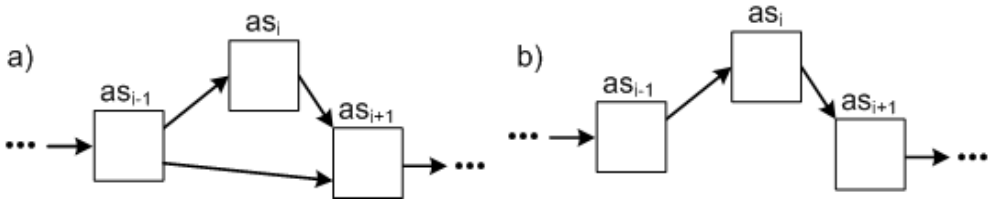

Fig. 4. Possible edge reduction non violating precedence relations between vertices of scenario graph; a) before edge removal, b) after removing redundant edge.

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

The reductions defined formerly are basic ones and one can define more of such reductions to cope with more complicated scenario graphs in more efficient way. The complete reduction operations set should contain at least two mentioned vertex reduction operations (serial and split) and redundant edge reduction. This gives guarantee to reduce any graph into desired form.

4.2. THE GRAPH-FOLD REDUCTION ALGORITHM

In order to reduce the scenario graph to the form that is appropriate for the multi-choice knapsack problem one has to perform steps concerning all vertex reduction patterns and all edge reduction patterns. Assuming that each reduction procedure reduces exactly one edge (in case of edge reduction) or exactly one pattern (in case of pattern reduction), the whole reduction process makes maximum $|VC|+|EC|$ reductions.

Each reduction has also its computational complexity which is influenced by: the complexity of procedure of finding a pattern, complexity of calculating the values of non-functional parameters of newly created node, disconnecting removed vertices and connecting newly created one. All of mentioned procedures can have different implementations so complexities may vary. Trying to estimate 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 disconnected during single reduction operation.

The general procedure consists of looped execution of all reduction operations until the vertices count $n=1$ (in case of reduction of all vertices of the graph) or input and output degrees of all vertices is no greater than 1.

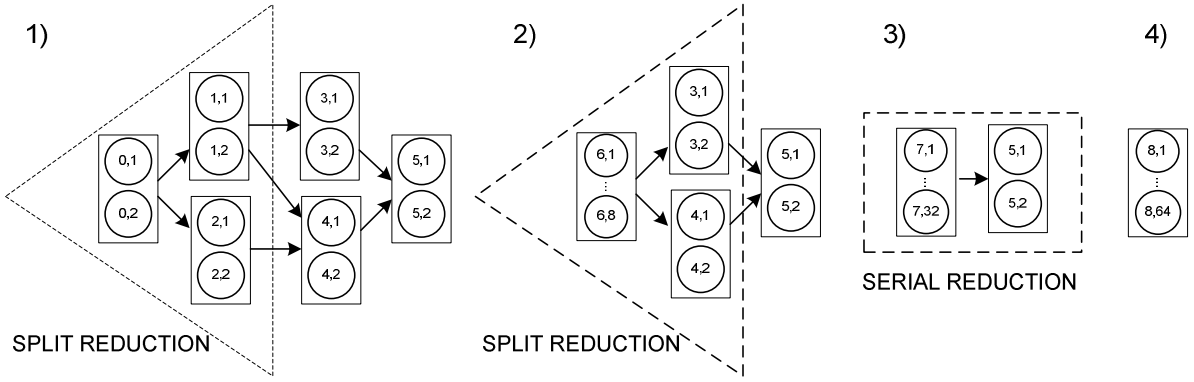

Fig. 5. Exemplary reduction process for scenario with six functionalities (atomic services), each having two versions. After two split reductions and one serial reduction the result is single atomic service with 64 versions. Every newly created scenario gets new number for ease of distinguishing.

Example of the reduction process has been presented on 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 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 increase. 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). Resulting vertex has 64 versions.

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 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 value for resulting *m* versions, the optimal solution of determination of complex service execution plan problem can be found in $O(\log m)$ time.

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

5.1. COMPLEX SERVICE EXECUTION TIME MINIMIZATION

This solution bases on the best-effort approach and finds minimum time of execution of complex service. A complex service execution time can be minimized using graph-fold algorithm when execution time is considered as a 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 *k*-th atomic service obtained after graph reduction procedure. According to 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 execution time requirement given by user. According to 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 but not higher than the user requirement. In case when there is no solution of this task, 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 quality delivered by presented 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 measured in kilobytes per second that can be processed. 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 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.

The results of the first experiment have been presented on figure 6. The experiment lasted for 1000 simulation seconds and consisted on determination of complex service execution plan at the moment of request was entering the system. The first observation shows that graph-fold algorithm obtained the lowest execution time from all tested algorithms. Moreover the execution time line on the chart is very close to the constant value of about 3 seconds while rest of algorithms resulted in 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 results of the second experiment have been presented on 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 to 5 seconds. On the chart presented on 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 case when system was not able to provide required execution time (in the 800-1000 seconds of simulation) lower value was assured. Analyzing the real complex service execution time curve on the figure 7, some deviations might be noticed. The real execution time approaches required value after 150 simulation seconds and was kept within 5% error range till 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 request was entering the system so the values describing execution time might have changed during the execution process.

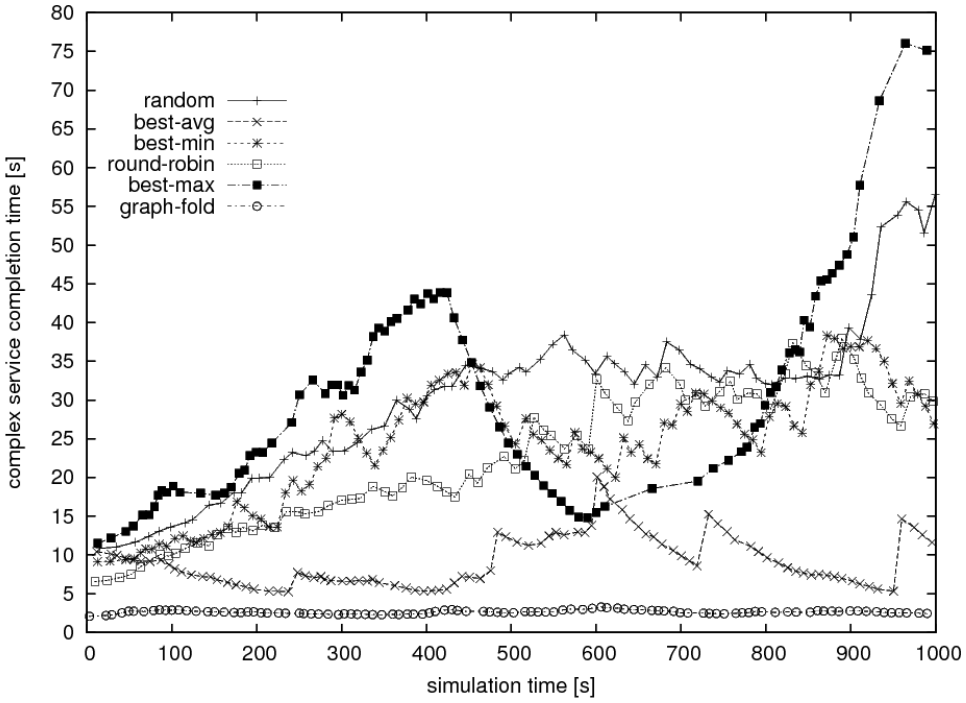

Fig. 6. Simulation results for the six algorithms solving the problem of complex service execution time minimization.

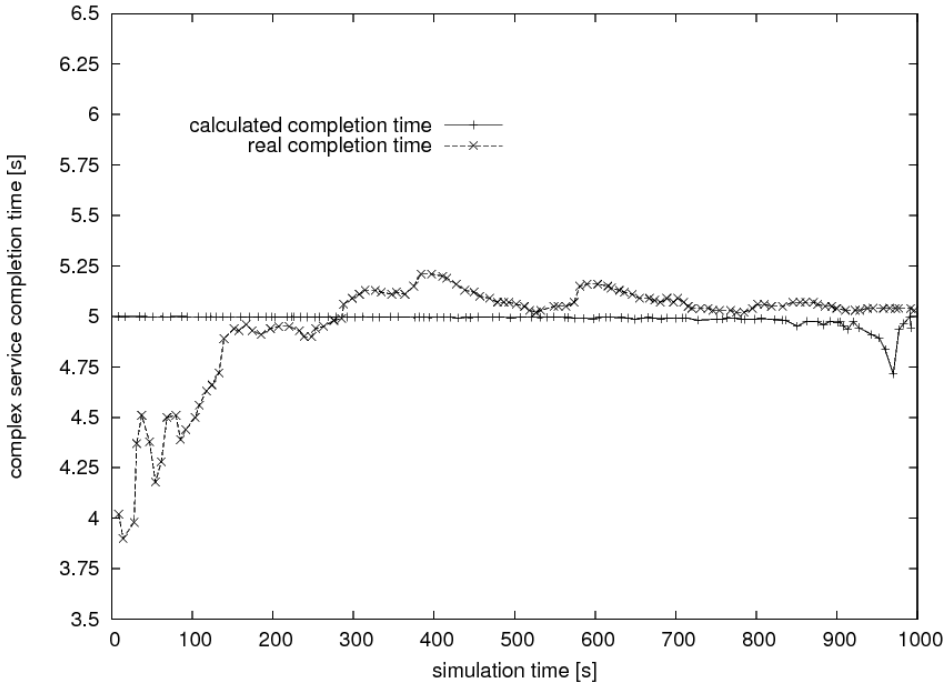

Fig. 7. Simulation results for the graph-fold algorithm solving the task of complex service execution time optimization. The calculated completion time curve is one predicted by the algorithm at the beginning of processing the request.

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. 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 stated problem. 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. Moreover in specific application of graph-fold algorithm there can be other set of reduction operations used which appears more often in input scenario graphs. Such modification can decrease the calculation complexity for 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.