QoS-aware Complex Service Composition in SOA-Based Systems

Adam Grzech, Piotr Rygielski, and Paweł Świątek

Institute of Computer Science
Wrocław University of Technology,
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland
{Adam.Grzech,Piotr.Rygielski,Pawel.Swiatek}@pwr.wroc.pl

Abstract. In this work, a method for the QoS-aware complex service composition in SOA-based systems is presented. The considered complex service composition process consists of three stages: complex service structure, scenario and plan composition. Complex service structure contains a set of required functionalities and precedence relations between them, and is derived directly from a service level agreement. Service scenario is derived from service structure by choosing the optimal (in the sense of certain quality criterion) order of execution of required functionalities. Finally, a service plan emerges from a scenario by choosing the best versions of atomic services for the delivery of required functionalities. For such distinguished composition process methods for complex service scenario and execution plan optimization are proposed. Optimization of a complex service scenario is based on the analysis of possible parallel executions of required functionalities. In order to deliver required level of the quality of service, well known QoS assurance models (i.e.: best effort, integrated services and differentiated services) are applied in the process of complex service execution plan optimization.

Keywords: Service Oriented Architecture (SOA), Quality of Service (QoS), complex services composition

1 Introduction

Systems based on the SOA (*Service Oriented Architecture*) paradigm offer services (complex services) which are often delivered as a composition of atomic services [12, 13]. The main feature of such an attempt is that the required complex services may be efficiently and flexibly composed of available atomic services providing certain, well defined, required and personalized functionalities. Requested complex services are characterized by a set of parameters specifying both functional and nonfunctional requirements; the former define the exact data processing procedures, while the latter describe various aspects of required service quality. The set of parameters describing requested complex service form SLA (*Service Level Agreement*) [1, 14].

Functionalities of the requested complex service are available as a sum of atomic services functionalities. In order to deliver a complex service with requested functional and non-functional properties appropriate atomic services must be chosen in the process of complex service composition [10]. Required functionality, precisely defined in the SLA, determines a set of required atomic services as well as a plan according to which atomic services are performed in the distributed environment. Properties of a requested complex service, which are mainly related to QoS (*Quality of Service*) may be in most cases assured or obtained by proper resources (processing and communication) and tasks (atomic services) allocation [4, 5, 7, 17].

Discussed complex services delivery approach is available only in a distributed environment; possible parallel executions of distinguishable atomic services requires the allocation of a proper amount of processing and communication resources in a parallel manner. The distributed environment may be obtained both by the allocation of separated or virtualized resources.

In order to obtain various required quality of service levels in distributed environment well-known QoS strategies, i.e., best-effort, integrated services and differentiated services concepts may be applied. Usefulness of the mentioned concepts strongly depends on the formulation of the non-functional part of the entire SLA. The application of the best-effort concept, based on common resources sharing, leads to a solution where the same, high enough, average quality of service is delivered to all performed services. The next two previously mentioned concepts offer differentiated quality of service for requested services (also guarantees) and are mainly based on resources reservation for individual requests (integrated services concept) or for classes of requests (differentiated services concept) [4, 15].

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 execution such that non-functional requirements are met. The task of complex service composition can be decomposed into three subtasks, each of which providing an input for subsequent subtask:

1. Complex service structure composition — transformation of the SLA into a set of required functionalities and the precedence relations between them. The result of this task is a complex service structure represented as a directed graph (not necessarily connected) of required functionalities.
2. Complex service scenario composition — transformation of a complex service structure graph into a single and consistent graph of required functionalities

with precisely defined order of execution of all atomic functionalities. The determination of the complex service scenario is composed of two stages. The goal of the first is to propose for a given complex service structure graph a set of consistent graphs, each representing certain order of execution of atomic functionalities. The aim of the second stage is to select the best (for assumed criteria) connected graph (scenario). Among others this task consists in making the decision on whether to apply possible parallel executions to functionalities, which are not bound by precedence relations in the complex service structure. Since it is possible that a single functionality is delivered by more than one atomic service (different versions of atomic service), the scenario graph represents in fact a family of execution graphs where member graphs differ in the atomic service versions applied to deliver the required atomic functionality.

3. Complex service execution plan composition — the choice of particular atomic services in a complex service scenario graph such that non-functional requirements of complex service are met. This stage is composed of three sub-stages. In the first one, the nodes (functionalities) of the corresponding optimal scenario graph are replaced by subsets of atomic services providing respective functionalities. In the second substage particular atomic services from atomic services subsets are chosen. In the last substage particular versions of chosen atomic services are picked.

The main advantage of SOA-based systems is that atomic services which deliver certain functionalities may be provided by different service providers, what allows users to chose the required complex services from many, functionally and non-functionally equivalent, available alternatives. In order to facilitate such capabilities service providers have to describe delivered services semantically in a unified manner. In order to deliver to the user a complex service with a requested functionality, semantic matching of service request with the available complex services has to be performed. This very important step of complex service composition is performed as a part of the first stage of the complex service structure composition task which was described above.

Since service requests, besides required functionality, include required non-functional parameters, they must be taken into account during service composition procedure. These parameters however concern, in general, the quality of required service, which vary in time and depends mostly on current state of computer communication system used as a backbone for complex service delivery. The aim of this paper is to provide framework for QoS-aware complex service composition, which allows to make decisions based on the network state of the SOA-based system.

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).

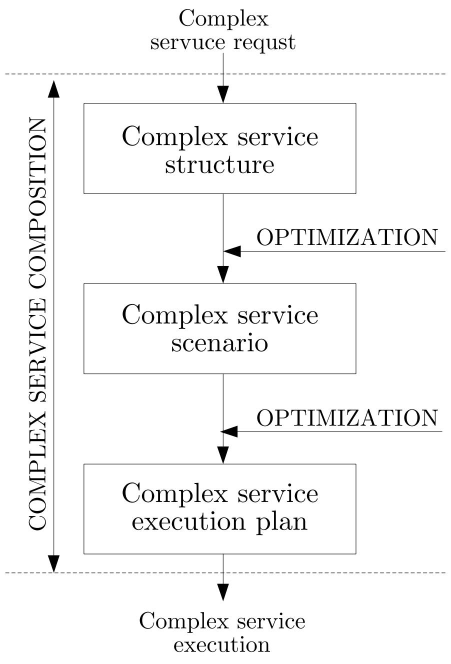

Fig. 1. Decomposition of the process of complex service composition

In this chapter a model of complex service, which facilitates decomposition of complex service composition task and allows to formulate and solve various complex service optimization tasks, is proposed. Decomposition of the complex service composition task is presented on Figure 1. This work is organized as follows. In Section 2 model of complex service and complex service structure composition task is presented. In section 3 the task of complex service scenario composition is presented. Section 4 describes three approaches to the task of complex service execution plan composition, which are based on well-known QoS strategies. Some numerical examples of proposed solutions are presented in section 5. Section 6 summarizes presented work and gives directions for future research.

2 Complex Service Model

Let $CS = \{cs(1), \ldots, cs(i), \ldots, cs(I)\}$ denote the set of all complex services delivered in the considered system. It is assumed, that a functionality $\varphi(cs(i))$ of a complex service $cs(i)$ is delivered by the execution of a certain number of atomic services $as(j)$ from the set $AS = \{as(1), \ldots, as(j), \ldots, as(J)\}$ of atomic services available in the system. The functionality $\varphi(as(j))$ of complex service $as(j)$ is an aggregation of the functionalities $\varphi(as(j))$ of the atomic services $as(j)$, which compose the complex service $cs(i)$. Similarly non-functional properties $\psi(cs(i))$ of complex service $cs(i)$ are the aggregation of the non-functional properties $\psi(as(j))$ of the applied atomic services $as(j)$.

The i-th ($i = 1, 2, \ldots, I$) complex service $cs(i)$ is a response to a call for a service fully described by the appropriate service level agreement denoted by $SLA_l = (SLA_{fl}, SLA_{nfl})$, which contains information about the functionality (SLA_{fl}) required by the user, and the nonfunctional requirements (SLA_{nfl}) determining the required level of the quality of service. The functional part of $SLA_{fl} = (\Phi_l, R_l)$ is a subject of the structure composition process which delivers the set of atomic functionalities $\Phi = \{\varphi_{l1}, \ldots, \varphi_{ln_l}\}$ present in the system. It defines the allowed order of execution of the required functionalities with use of the precedence relations $\prec$, given in matrix R_l.

The discussed precedence matrix, describing the set of precedence relations may be — in the simplest case — described by a square binary order constraints matrix R_l of size $(n_l + 2) \times (n_l + 2)$. The R_l matrix defines which functionalities are bound with the relation of precedence.

Exemplary order constraints (precedence) matrix is presented below:

$$R_l = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{1}$$

The matrix R_l of dimension 5×5 corresponds to three atomic functionalities with the addition of abstract starting and ending functionalities $\Phi_l = \{\varphi_{ls}, \varphi_{l1}, \varphi_{l2}, \varphi_{l3}, \varphi_{le}\}$. The matrix should be understood as follows:

- column 1 — Functionality φ_{ls} (abstract start functionality) cannot be preceded by any other functionality;
- column 2 — functionality φ_{l1} can be preceded by functionality φ_{ls} or φ_{l3};
- column 3 — functionality φ_{l2} can be preceded by functionalities φ_{l1} or φ_{l3};
- column 4 — functionality φ_{l3} can be preceded by functionalities φ_{ls}, φ_{l1} or φ_{l2};
- column 5 — functionality φ_{le} (abstract end functionality) can be preceded by functionalities φ_{l2} or φ_{l3}.

The ordering constraints given with by matrix R_l can be transformed into description using the precedence relation $\prec$ as follows:

$$\begin{aligned} &\varphi_{ls} \prec \{\varphi_{l1}, \varphi_{l3}\}; \\ &\varphi_{l1} \prec \{\varphi_{l2}, \varphi_{l3}\}; \\ &\varphi_{l2} \prec \{\varphi_{l3}, \varphi_{le}\}; \\ &\varphi_{l3} \prec \{\varphi_{l1}, \varphi_{l2}, \varphi_{le}\}; \\ &\varphi_{le} \prec \emptyset \end{aligned} \tag{2}$$

In general, the binary 1 in i-th row and in j-th column ($r_{lij} = 1$) means that the functionality φ_j can be preceded by the functionality φ_{li}. The zero value in i-th row and in j-th column ($r_{lij} = 0$) means that the functionality φ_{lj} cannot be preceded by the functionality φ_{li}. Abstract start and abstract end is guaranteed by zeros in column $j = 0$ and row $i = n_l + 1$.

$$r_{lij} = \begin{cases} 1 & \text{if } \varphi_{li} \prec \varphi_{lj} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Moreover each row (except for row $i = n_l + 1$) has to have at least one 1 value which guarantees the presence of exactly one end functionality. Additionally guarantee of having exactly one start functionality is determined by having at least one 1 in each column (except for $j = 0$ column). Above assumptions can be summarized with the following formulas:

$$\forall i \in \{0, \ldots, n_l\} \ \sum_{j=1}^{n_l+1} r_{lij} \geq 1 \tag{4}$$

$$\forall j \in \{0, \ldots, n_l + 1\} \ \sum_{i=0}^{n_l} r_{lij} \geq 1 \tag{5}$$

Having the functionalities set $\Phi_l = \{\varphi_{l1}, \ldots, \varphi_{ln_l}\}$ and the order constraints matrix R_l gives the ability to build a base graph denoted by GB_l. $GB_l = GB(SLA_{fl}) = GB(\{\Phi_l, R_l\}) = GB(VB_l, EB_l)$ is a graph defining the structure of a complex service, where $VB_l = \{vb_{l1}, vb_{l2}, \ldots, vb_{lk}, \ldots, vb_{ln}\}$ is the set of vertex of a base graph (each vertex vb_{lk} corresponds to a proper functionality φ_{lk}) and EB_l is set of edges corresponding to the precedence relations defined by matrix R_l.

The exemple graph for R_l matrix (eq. 1) and the functionalities set $\Phi_l = \{\varphi_{ls}$, φ_{l1}, φ_{l2}, φ_{l3}, $\varphi_{le}\}$, is presented on Figure 2. Each binary value “1” represents an edge between the functionalities in graph GB_l.

3 Complex Service Scenario Composition

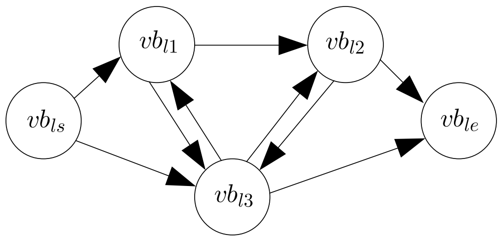

Fig. 2. Graph GB_l representation for matrix R_l (eq. 1). Each binary "1" value in the matrix corresponds to an edge in the structure graph GB_l. Redundant edges make the graph cyclic

The structure of a complex service determines which atomic functionalities are delivered within it and what are the order bounds. Such a service can be an entry of the optimization process concerning the determination of the exact order of functionalities delivery and parallel execution. The result of processing a complex service structure is called a l-th complex service execution scenario.

A complex service execution scenario is a graph GC_1 defined by the scenario determination process. The scenario $GC_1 = GC(SLA_{fl}) = GC(\{\Phi_l, R_l\}) = GC(VC_l, EC_l)$ is a graph containing the vertices set $VC_l = VB_l = \{vb_{l1}, vb_{l2}, \ldots, vb_{lk}, \ldots, vb_{ln}\}$ the same as in service structure, and the edge set $EC_l = EB_l \setminus EA_l$ which is a subset of the structure edge set EB_l:

$$\begin{aligned} EC_l &\subseteq EB_l \\ EC_l \cup EA_l &= EB_l \end{aligned}$$

The problem of finding an optimal scenario can be formulated as follows:
Given:

— the l-th complex service request given with SLA_l;
— the graph GB_l with set of vertices VB_l and set of edges EB_l;
— the order constraints matrix R_l;
— the other parameters vector a_l.

Find:

An adjacency matrix R_{GC_l} (as a representation of the graph GC_l) from the set of binary matrices of size $(n_l + 2) \times (n_l + 2)$ which minimizes the function f:

$$R^*_{GC_l} = \underset{R_{GC_l}}{\arg\min}\, f(R_{GC_l}; a_l) \tag{6}$$

with respect to constraints:

— the graph GC_l represented by the matrix R_{GC_l} should be acyclic;
— the graph GC_l represented by the matrix R_{GC_l} should have exactly one abstract start and an abstract end functionality:

$$\forall i \in \{0, \ldots, n_l\} \ \sum_{j=1}^{n_l+1} r_{CG_{ijl}} \geq 1$$
$$\forall j \in \{1, \ldots, n_l+1\} \ \sum_{i=0}^{n_l} r_{CG_{ijl}} \geq 1$$

— the matrix R_{GC_l} should comply with the order constraints matrix R_l: $R_{GC_l} \otimes R_l = R_{GC_l}$.

The satisfaction of ordering constraints can be determined via the logical multiplication of each binary element of matrix. The operation $\otimes$ is defined as follows:

$$A \otimes B = C$$

$$\forall i \in \{0, \ldots, n_l+1\} \ \ \forall j \in \{0, \ldots, n_l+1\}: \ c_{ij} = \begin{cases} 1 & \text{if } a_{ij} = 1 \text{ and } b_{ij} = 1 \\ 0 & \text{otherwise} \end{cases}$$

The equality $R_{GC_l} \otimes R_l = R_G C$ determines the satisfaction of the ordering constraints. The function $f(R_G C; a_l)$ can be any function determining the quality of service level e.g. execution time, cost, security level, reliability, etc. Depending on the function relationship with quality (a function value growth can mean quality level increase or decrease), the optimization task might has to be changed to maximization or minimization.

The determination of the optimal scenario consists in removing a subset of edges EA_l from the EB_l set in such a way, that each vertex in the result graph belongs to some path connecting the start vertex and the end vertex in such a way that the input and output degrees of vertices $\{\varphi_{l1}, \varphi_{l2}, \ldots, \varphi_{ln_l}\}$ are at least equal to 1. Moreover the input degree of start vertex and the output degree of end vertex must be equal to 0. Additionally, the result graph GC_l representing a scenario must be acyclic.

The set of edges that are subject to be removed is uniquely defined by the subtraction of the adjacency matrices $R^*_{A_l} = R_l - R^*_{GC_l}$. The remaining binary "1" values in the adjacency matrix $R^*_{A_l}$ define the edges that are subject to be removed from the graph GB_l to obtain an optimal complex service execution scenario GC^*_l.

For exemple with the matrix R_l (eq. 1), there are six possibilities of scenario graphs GC_l. All are presented in Figure 3.

The scenario presented in Figure 3a was obtained after the substraction $R_{GC_{ln}} = R_l - R_{GA_{ln}}$:

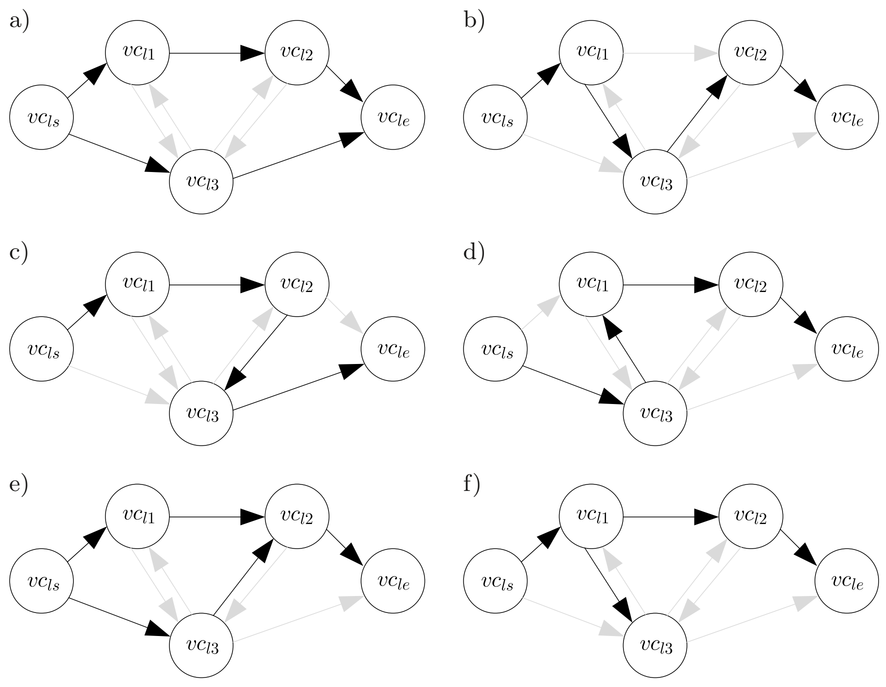

Fig. 3. Six possible scenario graphs GC_l obtained for the structure graph GB_l with the ordering constraints given in matrix R_l (eq. 1). Black edges are edges from EC_l, grey ones are from EA_l

$$R_{GC_{ln}} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

One can notice that the scenarios presented in figure 3 differ in the serial and parallel execution cases. All functionalities from scenarios b), c) and d) are executed serially (one-by-one), thus one can suspect that quality expressed e.g. the execution time, may be worse than in scenarios a), e) and f) wich execute in parallel. But if a parallel execution of functionalities shortens a complex service execution time, it uses more system resources than a serial execution.

In the case of a serial scenario, we can estimate the time of execution of a complex service using the following equation:

$$t(GC_{lserial}) = \sum_{k=1}^{n_l+2} t(vb_{lk}) + t(EC_l)$$

where $t(EC_l)$ is the overall time of transferring the requests between functionalities in a complex service scenario.

If a transfer delay request between the i-th and the $(i+1)$-th functionality is denoted by $t(ec_{li,i+1})$, the calculation of the transfer times in a whole complex service within a serial scenario is as follows:

$$EC_l = \{ec_{lij} : r_{CG_{ijl}} = 1\}$$
$$t(EC_l) = \sum_{i=1}^{n_l+1} t(ec_{li,i+1}).$$

In the extreme parallel scenario, above calculations are slightly different. The time of execution of a complex service is calculated as follows:

$$t(GC_{lparallel}) = \max_{k \in \{1,2,\ldots,n_l+2\}} \{t(vb_{lk}) + t(ec_{l1k}) + t(ec_{lkn+2})\}$$

Two extreme scenarios — by mean of best and worst execution time — are presented on Figure 4.

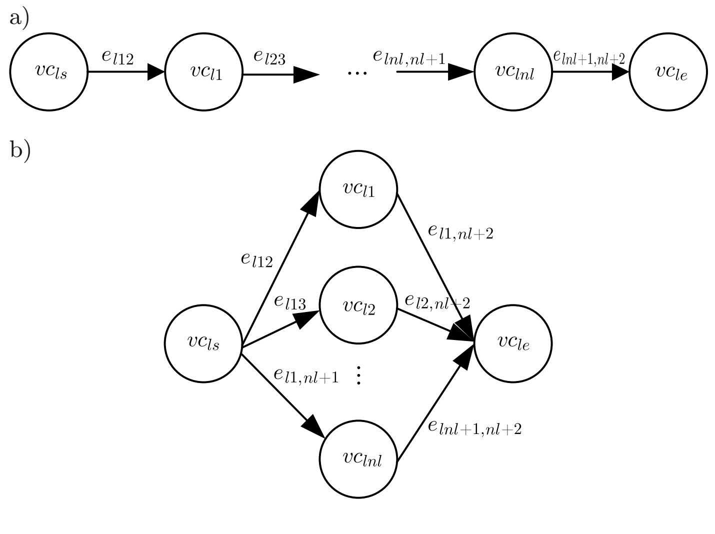

Fig. 4. Two extreme scenario graphs. Serial scenario: a — gives the worst execution time, extreme parallel scenario b — gives the best execution time

Situation where the maximum number of parallel functionalities delivery is limited must be considered. A need for the introduction of such a limitation may arise in the case where the utilization of a larger number of parallel functionalities delivery can decrease the end-user quality e.g. increases the cost of the service to such a level which violates user requirement.

In order to determine the parallelism in an execution scenario, it is needed to introduce a measure to determine parallelism level. In the literature [16] the most popular parallelism level is the ILP — Instruction Level Parallelism — which measures the number of operations that can be done simultaneously divided by the number of operations (eq. 7):

$$l_{p_{ILP}} = \frac{n_{l_{par}}}{n_l} \tag{7}$$

where $l_{p_{ILP}}$ is the ILP measure and $n_{l_{par}}$ is the number of operation that can be executed in parallel manner.

For scenarios given by graphs, other parallelism measures can be used, e.g. a measure that uses weight path length in a graph, where weights correspond to time needed to deliver a proper functionality φ_{lk}. The proposed measure l_{pMP} can be defined as follows:

$$l_{pMP}(GC_l) = \left(\frac{1}{n_p} \sum_{i=1}^{n_p} p_l(i) \right)^{-1},$$

where n_p is the number of paths in the graph GC_l and $p_l(i)$ is a weight of i-th path length. Other parallelism measures can be found e.g. in [11].

In order to optimize end-user quality there is need to introduce constraints concerning the parallelism level in an execution scenario graph GC_l. When there are no constraints in SLA_{nfl} concerning QoS parameters like e.g. cost, the formerly formulated problem can be used without change; otherwise one should add a constraint concerning the parallelism level of a scenario:

$$l_p(GC_l) = l_p(R_{GC_l}) \leq l_{p\max},$$

where the l_p function returns a parallelism level (with respect to the chosen measure) for a graph or for an adjacency matrix which is a representation of the graph. This function should be designed in such a way that a higher value means the delivery of a larger number of functionalities simultaneously, when a smaller value means that the scenario graph is mainly executed like the extreme serial scenario.

4 Complex Service Execution Plan

The result of the scenario composition stage is an optimal scenario graph $GC_l^* = (VC_l, EC_l^*)$, where $VC_l \subset AS$ is a set of atomic services, which provide the functionality requested in SLA_{fl}, and EC_l^* is a set of edges describing the optimal order of atomic services execution. The main feature of systems based on the SOA paradigm is that each atomic service may be available in many versions which differ in non-functional properties. Therefore, a scenario graph GC_l^* must be treated as a set of graphs, which differ in the versions of atomic services.

$AS(\varphi_{lk}) = \{as_{lk1}, \ldots, as_{lkn_{lk}}\} \subset AS$ denotes a set of versions of atomic services, which provide functionality φ_{lk}, where n_{lk} is the number of versions:

$$AS(\varphi_{lk}) = \{as_{lkm_{lk}} \in AS : \varphi(as_{lkm_{lk}}) = \varphi_{lk}, m_{lk} = 1, \ldots, n_{lk}\}.$$

As an example, consider graph GC_l^* presented in Figure 3a, which consists of three functionalities. Functionality φ_{l1} is provided by three different versions of atomic services, and functionalities φ_{l2} and φ_{l3} are provided by two versions. The number m_l of all possible complex service execution plans is equal to the product of the numbers n_{lk} of the atomic service versions providing the respective

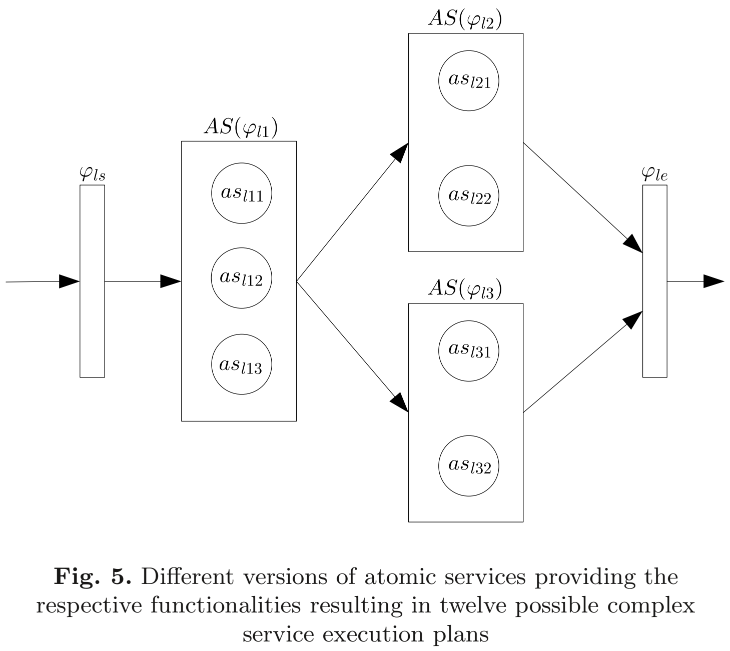

Fig. 5. Different versions of atomic services providing the respective functionalities resulting in twelve possible complex service execution plans

functionalities φ_{lk}; $m_l = \prod_{k=1}^{n_l} n_{lk}$. The twelve possible execution plans of complex service represented by graph GC_l^* are presented on Figure 5.

5 Problem Formulation

The task of complex service execution plan composition consists in choosing for each functionality φ_{lk} a version of atomic service, such that the non-functional properties $\varphi(cs(i))$ of the composed complex service $cs(i)$ meet the requirements Ψ_l stated in the non-functional part of the service level agreement SLA_{nfl}. Formally this task can be formulated as follows:
Given:

— non-functional requirements for complex service Ψ_l;
— optimal complex service execution scenario $GC_l^* = (VS_l, ES_l^*)$;
— complex service quality criterion Q.

Find:
A set of atomic services versions that such the following quality criterion is minimized:

$$\begin{aligned} & as_{l1m_{l1}}, \ldots, as_{ln_lm_{lnj}} = \\ & = \arg \min_{as_{l1m_{l1}} \in AS(\varphi_{l1}), \ldots, as_{ln_lm_{ln_l}}} Q\left(G(\{as_{l1m_{l1}}, \ldots, as_{ln_lm_{lnj}}\}, E_l)\right) \end{aligned} \tag{8}$$

where the graph $G(\{as_{l1m_{l1}}, \ldots, as_{ln_lm_{ln_l}}\}, E_l) = G(A_l, E_l)$ originates from the graph $GC_l^* = (VS_l, ES_l^*)$ in such a way, that the nodes A_l are particular atomic services delivering the same functionalities than nodes VS_l, and $E_l = ES_l^*$.

6 QoS Assurance Models

Depending on specific non-functional requirements and on the assumed model of the quality of service assurance various complex service optimization tasks can be formulated basing on the above general task of complex service execution plan composition. In the literature, there are three main concepts of QoS assurance i.e.: best effort (BE), integrated services (IntServ) and differentiated services (DiffServ). In the best effort approach no quality of service is guaranteed. Incoming request are serviced according to their arrival order and for each of them the best possible execution plan is chosen. In this model, only average values of QoS parameters may be guaranteed with the use of proper admission control mechanism.

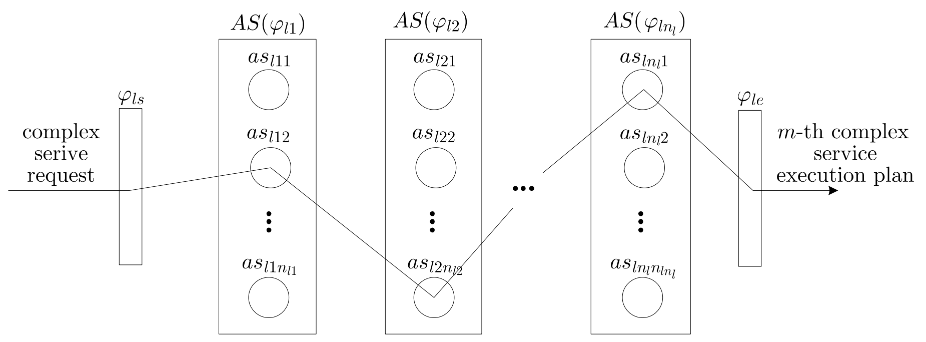

Fig. 6. Exemple scenario and plan of complex service execution with a serial execution order of atomic services

In the IntServ model necessary resources are reserved for each incoming request, what allows to deliver strict guarantees on the values of quality of service parameters. Besides resources reservation, this model requires the utilization of other QoS mechanisms such as: admission control, request scheduling, etc.

In the DiffServ model, incoming requests are separated into classes and for each class, required resources are reserved. Requests in different classes may be served according to the best effort or the IntServ model. Application of the best effort model to single DiffServ class, results in delivering guaranties on average values of QoS parameters for requests in this class. On the other hand, the application of the IntServ model to the single DiffServ class allows to provide strict QoS guarantees for each single request belonging to this class.

In order to present the application of aforementioned models of QoS assurance in the considered task of complex service execution plan optimization, let us assume that the optimal complex service scenario requires that the functionalities Φ_l are executed in serial order, and that the non-functional requirements Ψ_l of the requested complex service, apply to complex service response time. Above assumptions are made in order to clarify the presented approaches, and do not affect the generality of further considerations.

The optimal complex service execution scenario considered in this example is presented on Figure 6. This scenario requires n_l atomic services to be executed in serial order. Moreover, there are n_{lk} available versions of atomic services for each functionality $\varphi_{lk}(k = 1, \ldots, n_l)$ what results in $m_l = n_1 \cdot n_2 \cdot \ldots \cdot n_l$ possible complex service execution plans. For simplicity, we denote the m-th complex service execution plan as the sequence of indices of chosen versions of the consecutive atomic services $ep_{lm} = (m_{l1}, \ldots, m_{ln_1})$, where $n_{l1} = 1, \ldots, n_{l1}; \ldots; m_{ln_l} = 1, \ldots, n_{ln_l}$.

In the considered example, the non-functional property $\Psi(as_{lkm_{lk}})$ of the atomic service $as_{lkm_{lk}}$ is interpreted as an atomic service response time. Property

$\psi(as_{lkm_{lk}},\ as_{l(k+1)m_{l(k+1)}})$ denotes a communication delay on link between the atomic services $as_{lkm_{lk}}$ and $as_{l(k+1)m_{l(k+1)}}$. The response time $\psi(ep_{lm})$ of complex service executed according to the m-th execution plan $ep_{lm} = (m_{l1},\ \ldots,\ m_{ln_l})$ is equal to the sum of response times of the applied atomic services and communication delays between them:

$$\psi(ep_{lm}) = \sum_{m_{lk} \in ep_{lm}} \Big\lfloor \psi(as_{lkm_{lk}}) + \psi(as_{lkm_{lk}}, as_{l(k+1)m_{l(k+1)}}) \Big\rfloor \tag{9}$$

The average response time $\overline{d_l}$ experienced by the service requests in the whole system can be calculated as the weighted average over the response times of each execution plan ep_{lm} $(m = 1,\ \ldots,\ m_l)$):

$$\overline{d_l} = \sum_{m=1}^{m_l} p_m \cdot \psi(ep_{lm}) \tag{10}$$

where p_m is the probability, that certain service request will be served according to the m-th execution path ep_{lm}.

7 Average Complex Service Response Time Minimization (best effort)

In such a system, the task of minimization of the average complex service response time (best effort model) can be formulated as a task of finding such a vector $\mathbf{p} = [p_1,\ \ldots,\ p_{m_l}]$ of probabilities to chose a particular execution plan with minimized average response time:

$$\mathbf{p}^* = \arg\min_{\mathbf{p}} \sum_{m=1}^{m_l} p_m \cdot \psi(ep_{lm}) \tag{11}$$

with respect to constraints on probabilities $\mathbf{p}$:

$$\sum_{m=1}^{m_l} p_m = 1 \text{ and } p_m \geq 0 \text{ for } m = 1, \ldots, m_l \tag{12}$$

In general the average response time $\psi(ep_{lm})$ of each execution plan ep_{lm} depends on request arrival intensity and probabilities $\mathbf{p}$, which change over time. Therefore the optimization task (11) has to be solved iteratively in consecutive time steps. The execution of complex services consists in assigning to incoming request such execution plans that the number of requests executed according to each execution plan is proportional to calculated probabilities $\mathbf{p}^*$. For a large number

m_l of possible execution plans this approach may be inefficient due to the high computational complexity of the optimization task (11). In such a case, one can approximate optimal solution by the application of greedy approach, which for each new service request chooses the execution plan ep_{lm}, with the lowest average delay $\psi(ep_{lm^*})$:

$$m^* = \arg \min_{m=1,\ldots,m_l} \psi(ep_{lm}) \tag{13}$$

8 Service Response Time Guarantees (IntServ)

$S(t_l)$ denotes the state of the system at the moment t_l of arrival of the new request SLA_l. State $S(t_l)$ contains information concerning the moments of arrival, the assigned execution plans, and the location of all service requests present in the system at moment t_i. Given the system state $S(t_l)$, it is possible to calculate exact service response time $\psi(ep_{lm})$ for the request SLA_l for each execution plan ep_{lm} $(m = 1, \ldots, m_l)$:

$$\psi(ep_{lm}) = d(S(t_l), SLA_l, m) \tag{14}$$

where function $d(S(t_l), SLA_l, m)$ (presented in [6]) represents an iterative algorithm for the calculation of the response time of the service request SLA_l delivered according to the m-th execution plan.

In the the quality of service delivery task it is assumed that each incoming request SLA_l contains a set $SLA_{nfl} = \Psi_l$ of requirements concerning the values of various parameters describing quality of service such as: response time, security, cost, availability, etc. For the purpose of this example, it is assumed that the set $\Psi_l = \{\Psi_{l1}\}$ contains only one requirement concerning complex service response time.

The aim of the task of guarantying service response time is to find such a execution plan ep_{lm} for which the service response time requirements are satisfied:

$$m^* = \arg \min_{m=1,\ldots,m_l} \psi(ep_{lm}) = \arg \max_{m=1,\ldots,m_l} d(S(t_l), SLA_l, m) \tag{15}$$

with respect to:

$$\psi(ep_{lm}) \leq \psi_{l1}$$

It is possible that it does not exist such an execution plan for which the response time requirements are met. In this case requirements can be renegotiated, for example by suggesting a minimal possible service response time ψ^*_{l1}:

$$\psi^*_{l1} = \min_{m=1,\ldots,m_l} \psi(ep_{lm}) \tag{16}$$

When the required execution plan ep_{lm^*} is found (by solving either task (15) or (16) in order to be able to guarantee the requested service response time, resources in execution plan ep_{lm} have to be reserved.

9 Average Service Response Time Guaranties (DiffServ)

Assume that each incoming service requests SLA_l belongs to a certain class c_l ($c_l = 1, \ldots, C$). Each class c ($c = 1, \ldots, C$) is characterized by the probability q_c, that the response time requirements of requests from this class are met:

$$P\{\psi(ep_{lm}) \leq \psi_{l1}\} = q_{c_1} \tag{17}$$

where $\psi(ep_{lm})$ and ψ_{l1} denote respectively the response time of the request SLA_l executed according to the execution plan ep_{lm}, and the response time requirement of request SLA_l.

The aim of the task delivering the average service response time guaranties is to assign each incoming service request SLA_l to such an execution plan ep_{lm^*} for which equation (17) holds. Since the probability $P\{\psi(ep_{lm}) \leq \psi_{l1}\}$ for each service execution plan can be calculated by means of the cumulative distribution function $F_{lm}(\psi_{l1})$ [8], the task of delivering the average service response time guaranties can be formulated as follows:

$$m^* = \arg\min_{m=1,\ldots,m_l} \{F_{lm}(\psi_{l1})\} \tag{18}$$

with respect to:

$$F_{lm}(\psi_{l1}) \geq q_{c_l}$$

Similarly to the task of delivering strict guaranties, it is possible that none of the execution plans allows to obtain the required probability for the response time requirement. In such a case, the execution plan with the highest probability for response time requirement enforcement may be suggested:

$$m^* = \arg\max_{m=1,\ldots,m_l} \{F_{lm}(\psi_{l1})\} \tag{19}$$

10 Numerical Example

To illustrate the presented tasks of optimal execution plan determination a simulation study was carried out. Therefore a simulation environment has been designed and developed to allow the execution of various experiments concerning quality of service aspects in a system based on service oriented architecture paradigm [9].

The simulation environment has been configured for the needs of experiment in the following way. Simulated system consisted of three serially ordered functionalities each having atomic services in three versions: $AS(\varphi_{l1}) = \{as_{l11}, as_{l12}, as_{l13}\}$, $AS(\varphi_{l2}) = \{as_{l21}, as_{l22}, as_{l23}\}$ and $AS(\varphi_{l3}) = \{as_{l31}, as_{l32}, as_{l33}\}$. Moreover, in the simulation system, three classes of requests were considered; request belonging to the first class were served according to the best effort model in which the average complex service response time is minimized; requests from the second class were served according to IntServ model which allows to deliver strict guaranties for complex service maximal response time; requests from the third class were served according to the DiffServ model and has been divided into four subclasses, each class having different average response time guarantee. Subclasses from the DiffServ class have different average guarantee level set. Each request was required to be served in $\psi(ep_{lm}) = 0,5$ second although the first subclass with probability 0.8, the second with 0.7, the third with 0.6 and the fourth with a probability equal to 0.5.

A stream of requests following a Poisson law was connected to the input system, characterized with an average stream intensity $\lambda_0 = 50$. The share of each request class in the overall stream was as follows: best effort: 50%; IntServ: 10% and DiffServ: 40%. Each subclass of DiffServ request had 10% share in overall stream of requests. The ratio of the number of requests from different requests classes was chosen to be similar to the ratio of traffic volume in real computer communication networks.

The aim of the simulation was to evaluate the performance of the proposed approaches to a deliver quality, meant in this experiment as the response time guaranties delivered to the distinguished traffic classes for increasing the value of request arrival intensity. The results of the performed simulation are presented on Figures 7 and 8.

Figure 7 depicts the influence of an increasing request arrival rate on the average service response time for three main classes. Requests from both — best effort and IntServ — classes are served according to a plan that minimizes the average response time. There are two main differences between these classes: for IntServ, the computational and communication resources are reserved to provide strict guaranties. Moreover, the IntServ traffic is treated in prioritized way in comparison to the best effort traffic. Due to the lowest priority of the best effort class, all computational resources of atomic services are assigned to the traffic with higher

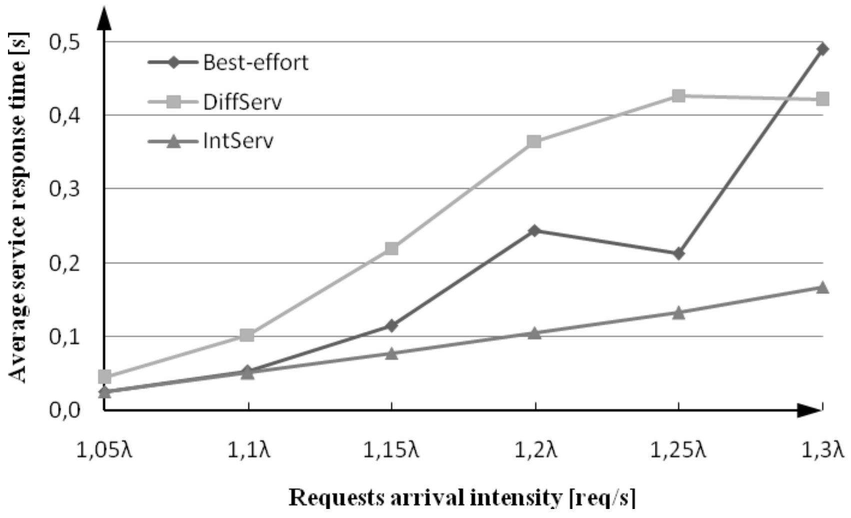

Fig. 7. Influence of increasing request arrival intensity λ on average service response time for three main requests classes: best effort, IntServ, DiffServ

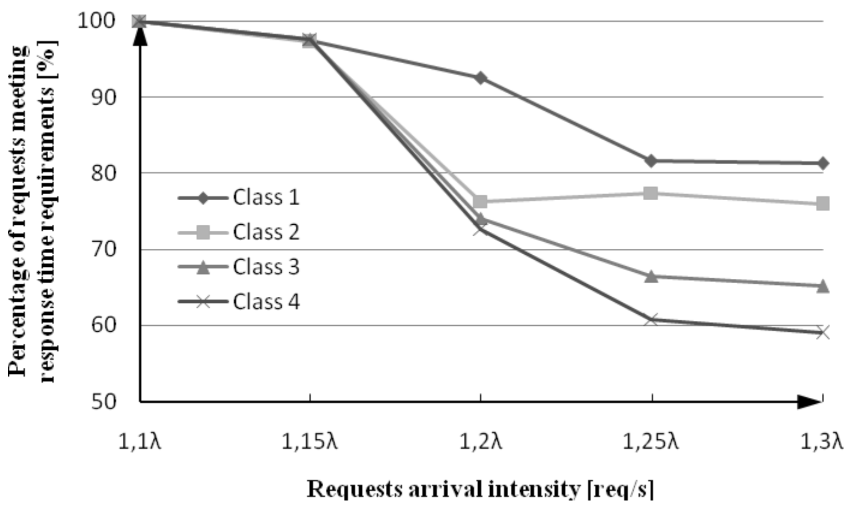

Fig. 8. Influence of the increasing request arrival intensity on the percentage of requests from each subclass of DiffServ meeting its response time requirements

priority and the best effort traffic is provided with only of the resources that are not consumed by other classes.

It is predictable that, for increasing the request arrival intensity, average service response time should grow for all traffic classes. An interesting situation takes place when intensity reaches $\lambda = 1.25\lambda_0$. Average response time of requests from the DiffServ class approaches its required response time at half a second delay, and stops increasing. At the same time, the response time of the best effort class slightly decreases and after a short moment begins to increase rapidly. This situation is caused by the fact, that when the DiffServ class reaches its requirement it does not need as much resources as earlier. Excess resources were assigned to the best effort class, what resulted in a decreased response time. When the request intensity increased, the DiffServ class needed more resources to provide the required response time guaranties. Necessary, resources were taken from the best effort class, what caused a rapid growth of the best effort average response time.

Each subclass of the DiffServ class have different requirements on the percentage of requests meeting response time requirements. The results of quantitative analysis of the influence of increasing the request arrival intensity on the percentage of requests from each subclass of DiffServ, meeting then response time requirements, is presented on Figure 8. One can notice, that as the request arrival rate grows, the percentage of requests not violating the response time guaranties approaches to the required values, which in the presented study were set to 0.8, 0.7, 0.6, and 0.5 for corresponding subclasses.

11 Related Work

The problem of QoS-aware complex service composition in the systems based on service-oriented architecture is being addresses as one of the main research field by both academy and industry [22]. Most of research analyzes the problem of service candidate selection as an optimization problem assuming that the business workflow is given [29]. In general it has been shown that the complex service composition problem is NP-hard when considered as a global optimization task [18, 19]. There are few factors which have an influence to the problem complexity. The main is the exponential growth of the solutions space with increasing number of service candidates, but the multidimensional QoS requirements along with the variety of connecting atomic services possibilities can also cause the problem to be more complicated [26, 27, 28]. The problem's complexity grows exponentialy so only heuristic algorithms can be proposed to obtain a feasible solution in runtime. The metaheuristics like simulated annealing, tabu search or genetic algorithms seems to be a logical idea to use due to intractable nature of the problem. Moreover a local optimization problem has been investigated in order to obtain a suboptimal solution [24, 25]. Most of researchers used the Integer or Linear programming in order to determine optimal QoS-aware service composition, but the proper problem formulation allows to use the multidimensional multichoice knapsack problem (MMKP) solutions in order to obtain desired composition satisfying all of the QoS constraints. However the MMKP problem has been also shown to be NP-hard [21]. [20] proposes a branch and bound algorithm (BBLP) to find the optimal solution for MMKP. In [30] author proposed an approach to optimize both service semantic functionality fit and the quality of service in single optimization stage. The aggregated task was complex, that is why author used metaheuristics to obtain only a suboptimal solution.

To cope with the complexity of the composition problem we propose to divide the composition process into three stages as shown in the paper. First stage, where the complex service structure is determined, allows to use semantic composition methods in order to satisfy end-user's functional requirements. In the second stage complex service execution scenario is determined, which addresses both aspects of composition process — functional and non-functional requirements satisfaction. The last stage — determination of complex service execution plan — is purely the quality of service optimization stage, which has been mostly investigated in the literature. The presented division of the composition process allows to perform pre-optimization tasks in order to reduce the possible solutions space. As shown in [23] the scenario graph optimization allows to estimate the obtainable complex service execution time before determining the execution plan which is the last optimization stage in our approach.

Besides the pre-optimization advantages, the composition process decomposition allows to investigate various optimization tasks concerning purely functionalities — with use of domain ontologies — in the first stage, and purely QoS-awareness in the last stage. The middle stage — where the complex service execution scenario is determined — allows to find a tradeoff between functional and non-functional requirements satisfaction. The presented three stage process can be also understood as a two layers of optimization (functional and non-functional) with the middle layer not considered in the literature before.

12 Final Remarks

In this work we presented the general method for QoS-aware complex service composition which consists of three stages: complex service structure, execution scenario and execution plan composition. On illustrative examples, we showed how to enhance the quality of complex service by solving certain optimization task in each stage of complex service composition. In particular we showed that complex service scenario optimization, analysing of complex service parallelism degree, may prove useful in the task of quality of service assurance.

Moreover, we showed that it is possible to deliver a required level of quality of service and differentiate it between distinguished request classes by the application of commonly known quality of service assurance approaches in the process of composition of complex service execution plan. It is worth noting, that the presented method uses only few mechanisms (resource reservation, request scheduling) from classical QoS assurance models. The application of all QoS mechanisms (e.g.: traffic shaping and conditioning, request classification, contract renegotiation, congestion control, etc.) as well as knowledge engineering methods [5] (e.g.: prediction of client behavior, adaptive scheduling, atomic services load prediction, etc.) to the management of systems resources may allow to significantly improve delivered quality of service.

The main advantage of our proposed method for complex service composition is that it is not necessary to go through all stages during service composition. In the case, when a new service request arrives in the system, it is possible to make use of partial solutions derived for similar requests which were served earlier. For example a single execution scenario may be used for functionally similar requests. Quality differentiation between these requests is achieved by the application of different execution plans for each request. The determination of semantic similarity between different complex service requests is one of the tasks for our future research.

References

1. Anderson S., Grau A., Hughes C. Specification and satisfaction of SLAs in service oriented architectures. 5th Annual DIRC Research Conference, 141–150, 2005.
2. Garey M., Johnson D., Sethi R. The complexity of flowshop and jobshop scheduling. Math. Oper. Res. 1, 117–129, 1976.
3. Graham R.L., Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G. Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann. Discr. Math. 3, 287–326, 1979.
4. Grzech A. Teletraffic control in the computer communication networks. Wrocław University of Technology, 2002.
5. Grzech A., Świątek P. Parallel processing of connection streams in nodes of packet-switched computer communication networks. Cybernet. Syst. 39, 2, 155–170, 2008.
6. Grzech A., Świątek P. Modeling and optimization of complex services in service-based systems. Cybernet. Syst. 40, 08, 706–723, 2009.
7. Grzech A., Świątek P. The influence of load prediction methods on the quality of service of connections in the multiprocessor environment. Syst. Sci. 35, 3, 7–14, 2009.

8. Grzech A., Rygielski P., Świątek P. QoS-aware infrastructure resources allocation in systems based on service-oriented architecture paradigm. Proc. 6th Working Conf. on Performance Modeling and Evaluation of Heterogeneous Networks, Zakopane, Poland, 35–48, 2010.
9. Grzech A., Rygielski P., Świątek P. Simulation environment for delivering quality of service in systems based on service-oriented architecture paradigm. Proc. 6th Working Conf. on Performance Modeling and Evaluation of Heterogeneous Networks, Zakopane, Poland, 89–98, 2010.
10. Jaeger M.C., Rojec-Goldmann G., Muhl G. QoS aggregation in web service compositions. IEEE Int. Conf. on e-Technology, e-Commerce and e-Service, 181–185, 2005.
11. Jain K.K., Rajaraman V. Parallelism measures of task graphs for multiprocessors. Microproc. Microprogr. 40, 4, 249–259.
12. Johnson R., Gamma E., Helm R., Vlisides J. Design patterns; elements of reusable object-oriented software. Addison-Wesley, 1995.
13. Milanovic N., Malek M. Current solutions for web service composition. IEEE Internet Comp. 8, 6, 51–59, 2004.
14. Narayanan S., McIlraith S. Analysis and simulation of web services. Comp. Networks 42, 5, 675–693, 2003.
15. Rajan R., Verma D., Kamat S., Felstaine E., Herzog S. A policy framework for integrated and differentiated services in the Internet. IEEE Network, 34–41, Sept. 1999.
16. Rau B.R., Fisher J.A. Instruction-level parallel processing: history, overview, and perspective. Readings in computer architecture. M.D. Hill, N.P. Jouppi, G.S. Sohi (eds.), Morgan Kaufmann, San Francisco, 288–308.
17. Wang Z. Internet QoS: architecture and mechanisms for Quality of Service. Academic Press, London, 2001.
18. D. Ardagna and B. Pernici Global and local qos guarantee in web service selection. Business Process Management Workshops, 32–46, 2005.
19. P. A. Bonatti and P. Festa On optimal service selection. In WWW ’05: Proceedings of the 14th int. conf. on World Wide Web, 530–538, New York, NY, USA, 2005. ACM.
20. Khan, S. Quality Adaptation in a Multisession Multimedia System: Model, Algorithms and Architecture. Ph.D. Dissertation, Department of ECE, University of Victoria, Canada, May 1998.
21. Martello, S. and Toth, P. Algorithms for Knapsack Problems. Annals of Discrete Mathematics, 31, 70–79, April 1987.
22. M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann Service-oriented computing: State of the art and research challenges. IEEE Computer, 40(11), 38–45, November 2007.
23. Grzech A., Rygielski P. Translations of Service Level Agreement in Systems Based on Service Oriented Architecture. Knowledge-Based and Intelligent Information and Engineering Systems, 14th Int. Conf. KES2010, Part II, LNAI 6277, 523–532, 2010.

24. Kwiatkowski J., Pawlik M. Budowa struktur gridowych współpracujących komputerów i agregacji zasobów w strukturach organizacyjnych. Przegląd Elektrotechniczny, 221–225, 2009.
25. Kwiatkowski J., Pawlik M., Fraś M., Konieczny D. Request distribution in hybrid processing environments. Int. Conf. on Parallel Processing and Applied Mathematics, Wroclaw, Poland, 2009 (abstract).
26. Kołaczek G., Juszczyszyn K. Smart Security Assessment of Composed Web Services. Cybernetics and Systems, 41, 1, 46–61, 2010.
27. Kołaczek G. Multiagent Security Evaluation Framework for Service Oriented Architecture Systems. Int. Conf. on Knowledge Engineering Systems KES2009, LNAI, Santiago, Chile, 30–37, 2009.
28. Kołaczek G. A logical-based security level evaluation method for service oriented systems. Int. Journal on Information Technologies & Security, 2, 29–42, 2009.
29. Yu, T., Zhang, Y., Lin, K.-J Efficient algorithms for Web services selection with end-to-end QoS constraints. ACM Trans. Web 1, 1, 6, 2007.
30. Lecue F. Optimizing QoS-Aware Semantic Web Service Composition. The Semantic Web - ISWC 2009, LNCS 5823, 375–391, 2009.