Multi-objective Service Composition with Time- and Input-Dependent QoS

Florian Wagner, Adrian Klein
University of Tokyo
Tokyo, Japan
{*florian,adrian*}*@nii.ac.jp*

Benjamin Klöpper
Universität Paderborn
Paderborn, Germany
kloepper@hni.upb.de

Fuyuki Ishikawa, Shinichi Honiden
National Institute of Informatics
Tokyo, Japan
{*f-ishikawa, honiden*}*@nii.ac.jp*

***Abstract*—Optimizing the Quality-of-Service (QoS) levels of a service workflow is essential for the user satisfaction in Service-oriented Computing. For that purpose, QoS computation models are applied to reflect the actual QoS experienced by the user during service execution. Current QoS models ignore the possible dependencies of QoS attributes, such as the dependency on the time of the execution or on the input data supplied to the service. Apart from that, composition approaches consider only single workflows during service selection, narrowing the number of possible compositions. Thus, we introduce a novel QoS model that covers QoS dependencies and discuss how this model can be used to consider multiple workflows at the same time. Moreover, we adopt a multi-objective optimization approach to offer solutions varying in QoS such as finishing time and price, allowing the user to make fine-grained decisions.**

***Keywords*-QoS dependencies, QoS-aware Service composition, multi-objective optimization**

I. Introduction

Services are re-usable, interoperable components that encapsulate well-defined business functionality. Service-oriented Computing (SoC) facilitates the development of software by discovering and combining these services in a loosely coupled way [1], resulting in workflows like in Fig. 1.

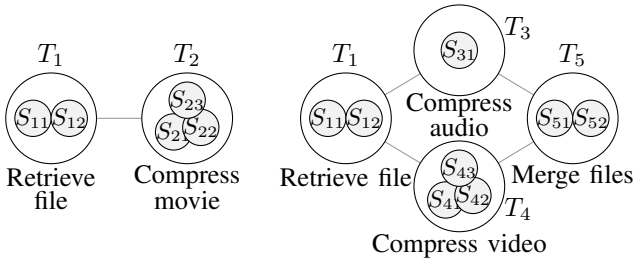


Figure 1: Two example workflows that both provide the retrieval and compression of movie files. S_{ij} refers to services and T_i to service tasks

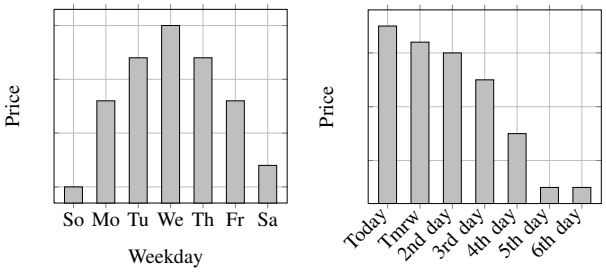
A. Quality-of-Service

In order to achieve user satisfaction, both the functional and non-functional requirements [2] have to be considered when selecting services for a workflow. Non-functional properties are expressed by Quality of Service (QoS) attributes, such as price, finishing time, compression rate, etc. The QoS of a composition are the aggregated QoS of the individual services according to the workflow patterns [3]. The user specifies constraints on the QoS in order to limit possible solutions. The appropriate matching of user requirements and available services is one of the major factors of user satisfaction [4].

Service providers declare the QoS attributes of their services as fixed values in a Service Level Agreement (SLA). These values for example define the price of the service or guarantees a maximal response time.

B. Time- and Input-Dependencies

Certain mandatory aspects cannot be described by a single value though. Consider for instance a time-dependent [5], [6] pricing model for the "compress movie" service of Fig. 1, illustrated in Fig. 2a. Invoking the service during business hours is more expensive then invoking it during weekends. Moreover, the price might dependent on the release date of the movie for service "retrieve file", as shown in Fig. 2b. If we want to watch the movie right after its release we have to pay more. As a consequence, the user has several choices to execute the service, resulting in varying costs and finishing times of the services. Table I illustrates resulting solutions of the scenario using the time-dependent pricing model of Fig. 2b.



(a) Repeated Pattern Scheme (b) Saturation Scheme

Figure 2: Time-dependent pricing models for the two example services S_{22} and S_{12}

Apart from that, more general dependencies between the QoS values of interacting services are possible. For instance the fee of "compress movie" might depend on the amount

of data generated by the service "retrieve file". Inter-service dependencies are considered as one central issue in service computing in [7].

C. Multi-Objective Approach

In reality, QoS such as the price and response time tend to be anti-correlated among the services. Therefore, in most cases no single selection of services exists that dominates all other solutions in terms of QoS. In order to select one solution automatically, usually single-objective optimization is applied, aggregating the QoS into a single utility value. However, in this case the user is forced to precisely define his QoS preferences beforehand, require time-consuming methods such as the Analytical Hierarchical Process [8]. Moreover, including *QoS dependencies*, the number of decision variables and possible solutions increases even further.

Solution	Finished by	Costs	Reliability
Solution 1	Today	10$	99%
Solution 2	Today	5$	95%
Solution 3	Saturday	5$	99%
Solution 4	Saturday	1$	95%

Table I: Example solutions for the workflow schemes of Fig. 1, computed by a MOO algorithm

Thus, we employ a multi-objective optimization approach, which computes a set of relevant solutions. In [9] we have introduced a graphical tool that lets the user visualize and compare possible solutions. This way, the user can choose the solution with the best trade-off.

In contrast to related approaches, our approach also considers multiple workflow schemes. For that purpose, we encode planning decisions as input variables in our optimization approach. This way, QoS aspects can be integrated with composition planning.

D. Contributions

In this paper we present the following contributions:

1) We motivate and present a QoS model to cover time- and input-dependent QoS attributes.
2) We leverage a data-structure to consider alternative workflow templates in a service composition.
3) We solve the extended service composition problem by applying an evolutionary algorithm. For that purpose, we show which modification are necessary to encode the composition problem as a genome.
4) We determine suitable parameters for the evolutionary algorithm, and point out possibilities to further improve the algorithm.

II. Preliminaries

In this section we briefly present the preliminaries of this paper. First, we discuss services and service-level agreements. Next, we describe workflows and algorithms to compose workflows automatically. Finally, we discuss algorithms that are used to optimize the QoS of a workflow, in particular multi-objective optimization algorithms.

A. Services & Service Tasks

Service providers specify both the functional and non-functional properties of their *services* in a Service-Level Agreement (SLA). Functionally equivalent services are grouped in *service tasks* and non-functional properties, QoS, can be used to differentiate between those services. QoS are either discrete or continuous, numerical or textual. In order to deal with QoS in a unified manner, we treat QoS as continuous, numerical values, possibly by performing some transformation. In this paper we consider the price per service invocation, the time needed to return a response, and the reliability of a service.

In addition, there can be many other QoS, which can be domain specific, such as the compression rate or an approximation ratio. Service providers either define exact values, for e.g. the price, or an upper or lower bound, for e.g. the response time. In this paper, we further consider functions to compute more complex QoS, for instance a time-dependent pricing model.

B. Workflows & Workflow Templates

Service workflows consist of a set of services that are invoked to carry out a certain purpose. The workflow describes the dataflow between the services by defining *service links*. If a service link exists between services S and S' then S provides output to service S'. Obviously, service S has to be invoked and return a result before calling S'. In other words, service links define a partial-ordering regarding the execution time of the services.

Workflow templates contain service tasks instead of services, therefore only the dataflow is defined but not the concrete services. This way, QoS optimization algorithms can customize the workflow to meet the non-functional requirements of the users. The two workflows in Fig. 1 are workflow templates. In order to obtain an executable workflow, we choose one service S_{ij} for each task T_i.

Users usually define certain preferences towards the QoS values, and some QoS constraints, e.g. a maximum price for the workflow.

C. Service Composition

Service composition is applied to select services automatically on behalf of the user to carry out a certain purpose. Generally speaking, two general approaches have emerged in the past years to realize service composition: service planning and service selection.

1) Service Planning: In *service planning* an AI planning algorithm usually starts from scratch, combining existing services to a workflow. In [10] several service planning approaches applied to SoC are discussed.

In this paper we employ Hierarchical Task Network (HTN) planning where the planning problem is initially described as a compound task. Compound tasks are split up by decomposition rules into smaller sub-tasks or primitive tasks that correspond to services tasks. Since multiple decomposition rules might be applicable at the same time, the main task of the planner is to select one decomposition rule for each compound task until only primitive tasks remain.

2) Service Selection: In contrast to service planning, *service selection* refines workflow templates to executable workflows. QoS-aware selection algorithms choose for each service task one single service in a way that the QoS of the workflow are optimized and constraints are met.

Usually workflows are internally represented by a tree structure. Inner nodes are either sequences, AND/OR branches, or loop nodes. The leaf nodes of the tree are the service tasks. Using this representation the computation rules in [11, Table 1] can be applied to compute the QoS vector $F(x) = \big(f_1(x).f_2(x), \dots\big)$ of the entire service selection. Each of the $f_i(x)$ yields a certain QoS of the workflow. Most approaches scale the QoS between $[0, 1]$ [12] and then aggregate the vector to a single utility value μ by applying e.g. a weighted sum:

$$\mu\big(F(x)\big) = \sum_{1}^{n} w_i \cdot f_i(x)$$

where w_i are the preferences of the user towards the QoS.

D. Multi-objective Optimization

Instead of aggregating the QoS values into a single utility value, in *multi-objective optimization* a *dominates* relation is considered: a service selection x dominates another selection x' if all objective functions $f_i(x)$ yield at least the same value as $f_i(x')$ and is strictly better for at least one i. Therefore, the dominates relation defines a partial-ordering on the selections. The set of non-dominated solutions is called the Pareto-optimal set of the solution space. Multi-objective optimization algorithms are applied to compute approximations of this set.

III. Related work

In this section we review related service composition algorithms and related works in multi-objective optimization.

A. Service Composition

As pointed out in Section II, we categorize service composition approaches into service planning and service selection algorithms. We will compare our approach with algorithms from both categories in the following.

1) Planning-Based Service Composition: Sirin et al. successfully demonstrate the application of HTN-planning to compute workflows automatically in [13], using the HTN planner SHOP2. However, they do not consider QoS aspects. For that purpose, Chen et al. [14] extend this approach to optimize QoS during the planning phase. Similar to our approach, a number of feasible plans are generated, and an optimal plan regarding the expected utility is generated by solving a Markov decision process. However, since possible decompositions are pruned depending on static threshold values, a Pareto set cannot be obtained or approximated.

Kalasapur et al. [15] introduce a framework for service composition in pervasive computing. Their service model was a partial archetype of our service model. The authors put more emphasize on the semantic integration of services, and their service composition relies on a shortest path algorithm. Thus, they only find a single solution.

In [16] we apply a cluster algorithm to determine services with the same purpose to detect backup services and prune the solution space of the planning tool. This way, QoS optimization algorithms can be employed after a workflow template is fixed. Therefore, alternative templates cannot be explored, like in approach presented in this paper.

2) Selection-Based Service Composition: The optimal service selection cannot be computed in feasible time for complex service workflows [11]. For that reason, heuristic algorithms such as [17]–[19] are used to solve the selection problem. The approach described in [20] extends the service composition problem by introducing functional inter-service dependencies; certain service combinations are declared as not feasible. In our setting, inter-service dependencies only concern the QoS attributes.

All presented selection approaches only optimize an aggregated QoS utility value. Therefore, the user cannot compare candidate solutions with different QoS trade-offs against each other.

In summary, service selection algorithms narrow the number of possible solutions as they can only consider one workflow template. Our approach provides more flexibility as it picks an appropriate workflow template depending on the QoS preferences of the user. Compared with planning approaches, our approach computes a set of non-dominated solutions. Apart from that, non of these approaches considered service-dependent QoS.

B. Multi-objective QoS-aware Service Composition

In [21], [22] genetic algorithms are employed for multi-objective optimization. Wiesemann et al. [23] introduce a multi-objective stochastic programm to consider risk during the service composition process, which is not a trade-off between QoS attributes, but the worst-case risk functionals for execution time and total costs. In [9] we apply an extension of the NSGA-II algorithm that considers functionally diverse services.

None of these approaches consider alternative workflow schemes within complex services or time- or input-dependent QoS.

IV. Approach

Our approach consists of three steps: first, we model the problem by using a Hierarchical Workflow Graph (HWG). This way, we can treat alternative workflow schemes in an unified manner. Next, we use the HWG to determine time- and input-dependent QoS. Finally, we compute a set of feasible solutions of the HWG by employing an evolutionary multi-objective optimization algorithm.

A. Hierarchical Workflow Graph

In [5] we extended the service workflow model by Kalasupur et al. [15] by introducing *complex service nodes* in order to consider alternative workflow schemes. A workflow scheme is defined by a directed acyclic graph (DAG) $w = (S, D)$ where the vertex set S represents the services and the edge set D describes the input/output relations between services. If an edge $d = (S_A, S_B) \in D$ exists, then service S_B requires data from S_A. Thus, service S_A has to finish, before invoking service S_B. In the following, we will refer to service S_A as being *parent* of S_B.

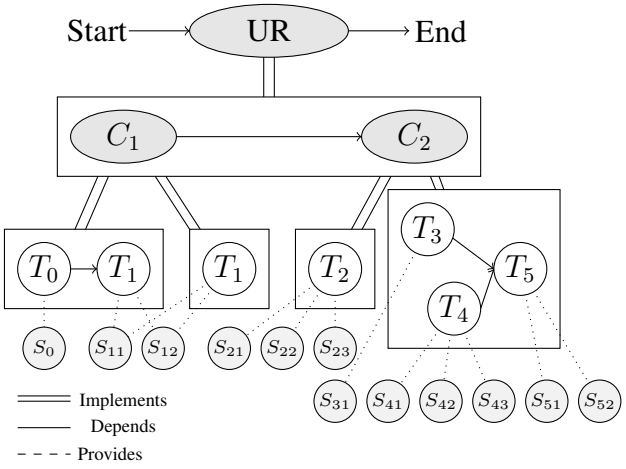


Figure 3: HWG of the scenario of Sect. I, including an optional Task T_0 used for encrypting the content. Ellipses indicate complex services and user requests, circles are atomic tasks and services. Rectangles are workflow schemes.

Similar to HTN planning, a workflow scheme can contain three different types of services:

1) A *user request* (UR) that describes the functionality of the requested workflow by the user.
2) A *complex service* c that can be realized by one or more partial workflow schemes.
3) An *atomic task* t that can be realized by one or more concrete services. All services provide the same functionality.

Fig. 3 illustrates the HWG for the example of Section I. By using complex services, alternative workflow schemes are selectable. For instance, in the c_1 node an additional task T_0 can be invoked that provides encryption of the input data. This decision depends on the QoS preferences of the user since invoking this task will increase the response time and probably the price but also increase the security QoS.

The structure was originally defined for modelling manufacturing problems and is called Hierarchical Precedence Graph [24]. However, for the composition problem the term Hierarchical Workflow Graph (HWG) is more appropriate.

B. QoS Dependencies

In general, we distinguish between two impact factors on the QoS of a service:

- the *time of the invocation*, e.g. time-dependent pricing models (cf. Fig. 2) or response times that depend on the server load.
- the *QoS properties* describing one or more input parameters, e.g. encryption strength or the amount of data sent to a storage service.

As mentioned in Sect. I, the dependent QoS of a service cannot be modeled by a static value in the SLA. Therefore, we model QoS_S^i for each attribute i of a service instance S by a set of functions $F_{QoS_S^i}$, shown in Equation 1.

$$QoS_S^i = F_{QoS_S^i}\left(t_S, \overrightarrow{QoS}_{P_1}, \ldots, \overrightarrow{QoS}_{P_n}\right) \quad (1)$$

$$t_S^{end} = t_S + F_{t_s}\left(t_S, \overrightarrow{QoS}_{P_1}, \ldots, \overrightarrow{QoS}_{P_n}\right) \quad (2)$$

The value t_S refers to the start time of service S and $P_1, \ldots, P_n$ are the direct predecessors of S in the workflow, i.e. all parent service nodes in the HWG. Since the parent nodes depend on the actual workflow scheme, the functions depend on the selected scheme as well. As a shortcut, we define $\overrightarrow{QoS_S} = (QoS_S^1, ..., QoS_S^n)$ as the tuple containing all QoS of S. A special case is the end time t_S^{end} of the service (cf. Eq. 2), which is computed with the function F_{t_S}. In Fig. 4, the QoS dependencies are visualized for the example scenario.

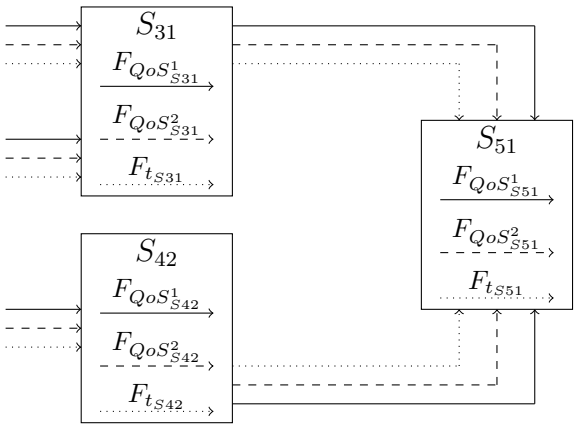


Figure 4: Considering the QoS interdependencies on S_{51} of the scenario, taking the price QoS^1 and the response time QoS^2 into consideration

Examples: Apart from the price and response time, the functions $F_{QoS_S^i}$ are used to cover any dependent QoS attribute. In the following, we show three examples of the $F_{QoS_S^i}$ function:

1) A data storage service: the price depends on the size of the received file (cf. Eq. 3)
2) A sorting service: the execution time is estimated by the average run time in dependency on the number of received records (cf. Eq. 4)
3) A hotel inquiry service: the expected number of available rooms depends on the destination, check-in-date (provided by the user), and start time t_S (cf. Eq. 5)

The corresponding $F_{QoS_S^i}$ are as follows:

$$QoS_S^{cost} = \text{price per byte} \cdot \sum_{i \ldots n} QoS_i^{filesize} \quad (3)$$

$$QoS_S^{time} = \sum_{i \ldots n} QoS_i^{\#recs} \cdot \log\Big(\sum_{i \ldots n} QoS_i^{\#recs}\Big) \quad (4)$$

$$QoS_S^{\#rooms} = E(\#\text{rooms} \mid \text{check-in-date}, t_S, \text{dest.}) \quad (5)$$

For a complex service c the functions $F_{QoS_c^i}$ are not explicitly defined, but derived recursively from the services within the selected workflow scheme. This is done in three steps:

1) Each child service of the workflow source *inherits* the parent values QoS_s^i of the complex service
2) The workflow is instantiated during the composition and the QoS of the services are determined
3) Aggregation functions defined in [11, Table 1] are used to determine the QoS values of the complex services

The start time parameter t_S in $F_{QoS_s}^i$ of a service S is a relevant decision variable, since by delaying the execution of a service, certain QoS attributes can be manipulated. Obviously, even on a limited discrete time-horizon the number of possible start times can be very high and drastically increase the number of feasible composition plans. However, the relationship between start time and service quality is expected to follow certain schemes. Figure 2 shows the two expected schemes of time-dependent QoS attributes.

In the *repeated pattern scheme* (cf. Fig. 2a) we assume that the same sequence of QoS values is repeated after a certain number of periods. For instance, this pattern can describe service execution prices or the availability depending on the daytime or weekday.

Applying the *saturation scheme* (cf. Fig. 2b) we expect the QoS values to constantly move towards a worst value and finally converge to that value. This scheme is appropriate to describe the availability of physical services and goods, e.g. such as hotel rooms depending on the time until check-in.

Both schemes can be used to reduce the number of start times we need to consider. Given the *repeated pattern scheme*, delaying the starting time within the length of a sequence will not result in a new QoS value. As well, given the *saturation scheme* a delay beyond the saturation period does not result in a new QoS value. Given several time-dependent QoS values of a service S, the lowest common multiple of the period length or the distance to the saturation point is sufficient to cover all relevant start times of S.

The considered QoS attributes define an n-dimensional evaluation vector ev that is used to compare composition plans. The dominance criterion is applied to identify a relevant set of composition plans which should be presented to the user. If CP is the set of feasible composition plans given the HWG and the set of service providers, the set of Pareto-optimal evaluation vectors $ev \in EV_p$ meets two conditions: there is not ev_d that dominates ev and there is at least one feasible composition cp in CP $ev = ev_{cp}$.

C. Multi-Objective Optimization

In the optimization phase, three types of decision variables are taken into account:

- select a workflow scheme for each complex service c
- select a concrete service for each atomic task T
- determine a valid starting time for each service S

The objective vector contains the QoS of the resulting workflow and the expected end time. In most cases, the objectives are anti-correlated, i.e. are optimized in opposite directions. Consider for instance Table. I. If we insist on a prompt execution or want to improve the reliability of the workflow, the price will increase. Therefore, no single solution exists that dominates all other solutions. In these cases, the user picks the solution that fits his needs best.

For that purpose, MOO algorithms are used to compute a set of feasible solutions that are not dominating each other. The decision variables of the workflow are encoded as a genome and then a number of evolution steps are performed. In each generation, the genomes are mutated, joined by a crossover operator, and selected. In the end, a set of non-dominated solutions is retrieved and presented to the user.

1) Encoding a workflow as a genome: In order to reflect the three types of decision variables, the genomes are split into three sub sequences, each representing one type of variable. All decision variables are from a discrete and finite set. Figure 5 illustrates the structure of the genome. Basically, each gene in the genomes encodes one selection.

Due to the form of the saturation scheme, the number of relevant start times depends on the earliest possible start time of a service, and thus, on previous composition decisions. Therefore, a static set of integers (referring to the start times) is not sufficient. Instead, real valued variables are used to select the workflow scheme, service provider, and the start times from the actual available sets. This way, standard algorithms and operators for mutation, crossover, and selection are applicable.

2) Determining the start times of the services: Determining the start times of the services is a crucial task during the optimization phase. Each start time is constricted by the preceding services. In order to determine the earliest possible

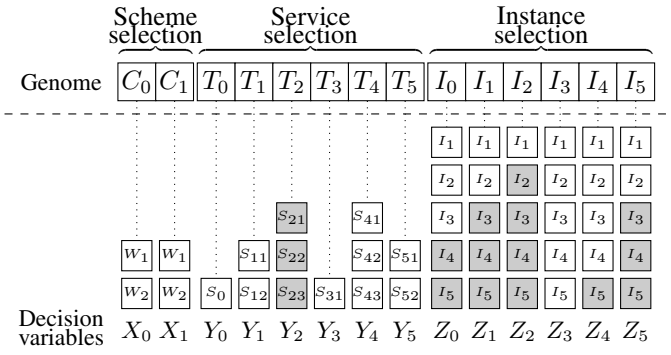


Figure 5: Encoding the HWG of the motivating scenario as a genome. Variables Y_0 and Y_3 are unnecessary in this example and just shown for the sake of completeness. Gray nodes indicate invalid/disregarded values.

start time of each service, we maintain the following four lists of services:

1) **Composition**: Composition plan with the selected services, service providers, and start times.
2) **Ready**: Operations that can be included into the composition plan. All preceding services of these operations are contained in *Composition*.
3) **Open**: Services that cannot be included yet into the composition, since at least one preceding service is not contained in *Composition*.
4) **Wait**: Complex services that have been selected, but are not entirely included in the composition plan.

In each iteration, one service S_i from the *Ready* list is moved to *Composition*. In case S_i is an atomic service, a service provider and a start time is determined. All directly succeeding services from *Open* are updated. If a child service S_c of S_i is ready (all parents are included in *Composition*) the service is moved to *Ready*. In the update step, the QoS values of S_i are forwarded to the child services S_c in order to determine the $\overrightarrow{QoS}_i^x$. If S_i is a complex service, it is moved to *Wait* instead and a workflow implementing S_i is selected. The source of the workflow is moved to *Ready*, and all its child services are moved to *Open*.

3) Computing the solution set: We apply a genetic algorithm (GA) to compute a set of feasible solutions. GAs initially generate a set of candidate solutions, encoded as genomes. In each iteration, an offspring population is created by applying a select, crossover, and mutate operator. The GA terminates either after a fixed number of generations or if the fitness of the top genomes converges.

In this work we use the non-dominated sorting genetic algorithm II (NSGA-II) [25], employing binary-tournament selection, simulated binary crossover (SBX), and uniform distribute mutation operators. Binary-tournament selects solutions depending on (a) constraint violation, (b) dominance, and (c) distance to the population. SBX defines a spread factor β that is defined as the ratio of the distance between the real number representation of the offspring to the real number representation of the parents. This way, an one-point crossover for binary encoded genomes can be simulated since the offspring tends to be similar to their parents. Each objective has to be minimized.

V. Implementation and Validation

We have implemented our approach in the jMETAL 3.1 framework [26]. In the following section we want to determine by experiment the parameters that yield the best results in terms of utility and performance. Therefore, we compare various probabilities for applying the mutate and crossover operators since these values have a strong impact on the performance of an evolutionary algorithm.

For the experiment, HWGs are randomly generated based on five different workflow templates, containing three to five free slots. These templates are filled with complex services and 100 different atomic tasks, containing between 3 and 10 concrete services. The percentage of creating a complex services is set to 40% for services of depth 1, 25% for services of depth 2, and 10% for services of depth 3.

Each service is associated with four QoS values: reliability, execution time, price, and a general quality value. This quality value indicates the general quality of the service. Since we assume that services belonging to the same task will have similar QoS values we first generate a random mean value for a task for each QoS and then assign random values with a possible deviation between 30% below and 30% above to each service. The reliability is generated between 0.85 and 0.9 and the execution times between 8 possible periods. The other two attributes vary for each service from 0.0 to 1.0.

When adding an atomic service to the workflow, QoS dependencies are generated. As we regard the reliability and execution time of services as independent values, only the price and the general QoS value are considered as dependent values. These two values depend on one or more preceding services. All other parent services have an impact with probability 30%. The dependent QoS attributes of a service S are modelled according to Eq. 6, where $Base_S^i$ is a basic value for QoS_S^i, f is a factor between 0 (parent irrelevant) and 0.3. The direction of the influence (± 1) is chosen with 50% probability.

$$F_{QoS_S^i}\left(t_S, \overrightarrow{QoS}_{P_1}, \ldots, \overrightarrow{QoS}_{P_n}\right) = \\ = \underbrace{Base_S^i}_{\text{Base QoS of } S} + \underbrace{\sum_{j=1}^{n} \pm 1 \cdot QoS_{P_j}^i \cdot f}_{\text{Impact of the parent nodes}} \quad (6)$$

In total, a time horizon of $7 \cdot 24 = 168$ periods is considered. The pattern length for time-dependent QoS values varies between 5 and 20 periods. Finally, several QoS constraints are applied: the minimal reliability is set between 0.85 and

0.93, the maximal total execution time between 118 and 168 periods and random constraints on the two other QoS values.

In order to compare different settings, we test 30 HGW problem instances with different settings for the probabilities of the mutate and crossover operator (0.1, 0.4, 0.7, 0.95), resulting in 16 different configurations. Each setting is executed 30 times and repeated 5 times to falsify outliers. The initial population size was set to 80, evolving for 300 generations.

Due to the complexity of the problem, even for small problem sizes the optimal solution cannot be computed in feasible time. Therefore, we compute an approximation of the ideal solution in order to compare and evaluate the computed solution sets of the algorithm. For that purpose, in each run the best results of each QoS dimension are combined in a single vector. We compare two computed result sets N and M based on a modified epsilon indicator [27], determined by the following equations:

$$\epsilon_1 = \min_{\epsilon \in \mathbb{R}} \big(\forall n \in N \ . \ \forall m \in M \ . \ \exists i \in QoS \ . \ n_i < m_i(1+\epsilon) \big)$$

$$\epsilon_2 = \min_{\epsilon \in \mathbb{R}} \big(\exists n \in N \ . \ \forall m \in M \ . \ \forall i \in QoS \ . \ n_i < m_i(1+\epsilon) \big)$$

The basic idea is to multiply the objective vectors of all solutions in S_2 with a scalar $(1+\epsilon)$ until none of the solutions S_1 is dominated. The epsilon values were calculated in reference to the ideal solution and the best solution found.

In Fig. 6 the average epsilon values for varying crossover and mutation probabilities are shown. We conclude that a rather high crossover probability is important to achieve fast progress in the first generations and mutation has only a minor effect. However, with progressing generations, the difference between the crossover probabilities of 0.7 and 0.95 becomes smaller. At this step, a high mutation rate is mandatory to achieve further improvements. Considering that the crossover operation consumes up to 35% of the CPU time in dependency of the chromosome length, this indicates the necessity to adjust the probabilities once the improvements start to slow down.

	ϵ_i	ϵ_b	FF_{avg}	FF_{max}	No solution
NSGAII	0.27	0.40	3	70	4%
Random	0.57	0.63	31	300	13%

Table II: Comparison of Random Search and NSGAII

Table II shows a comparison of the results of the best parameters for the NSGA-II algorithm and the baseline random search. The NSGA-II dominates the baseline in all relevant criteria. The epsilon values refer to the comparison with the ideal point (ϵ_i) and the best solution (ϵ_b) in the last generation. Due to the difficulty of comparing to an empty set and the higher number of failed optimization runs of the random search, the epsilon values favor the random search. FF_{avg} and FF_{max} give the average and maximum number of generations in which the first feasible solution was found. The average CPU time per generation[1] was 40ms for the genetic algorithm and 15ms for the random search.

VI. Conclusion

In this paper we have motivated the problem of service composition in the context of time- and input-dependent QoS values. By considering these values, complex interactions between services can be modeled and the QoS values of the resulting workflow can be determined. However, such dependencies increase the complexity of the service composition problem and hinder the users' ability to define a single objective function describing his QoS preferences a priori. Thus, we have applied multi-objective optimization (MOO) to present the user a set of feasible solutions. In order to solve the MOO problem we have employed a genetic algorithm (GA) and discussed how to encode the problem as a genome. Moreover, we have determined by experiment suitable parameters and also pointed out how to further optimize the performance of the GA.

Future work concerns the completion of the approach towards a holistic system for service composition. This includes a framework that derives QoS dependencies from past execution logs, and also provides means to define these dependencies. Apart from that, users should be able to define their time preferences in more detail. For instance, a user might consider all possible finishing times of a workflow as equally good until a certain point in time. Furthermore, a suitable selection interface has to be developed, similar to the interface presented in [9].

Acknowledgements

The work of Florian Wagner is partially supported by the KDDI Corporation. Adrian Klein is supported by a Research Fellowship for Young Scientists from the Japan Society for the Promotion of Science.

References

[1] M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, and B. J. Krämer, "Service-Oriented Computing: A Research Roadmap," in *Service Oriented Computing (SOC)*, ser. Dagstuhl Seminar Proceedings, 2006.

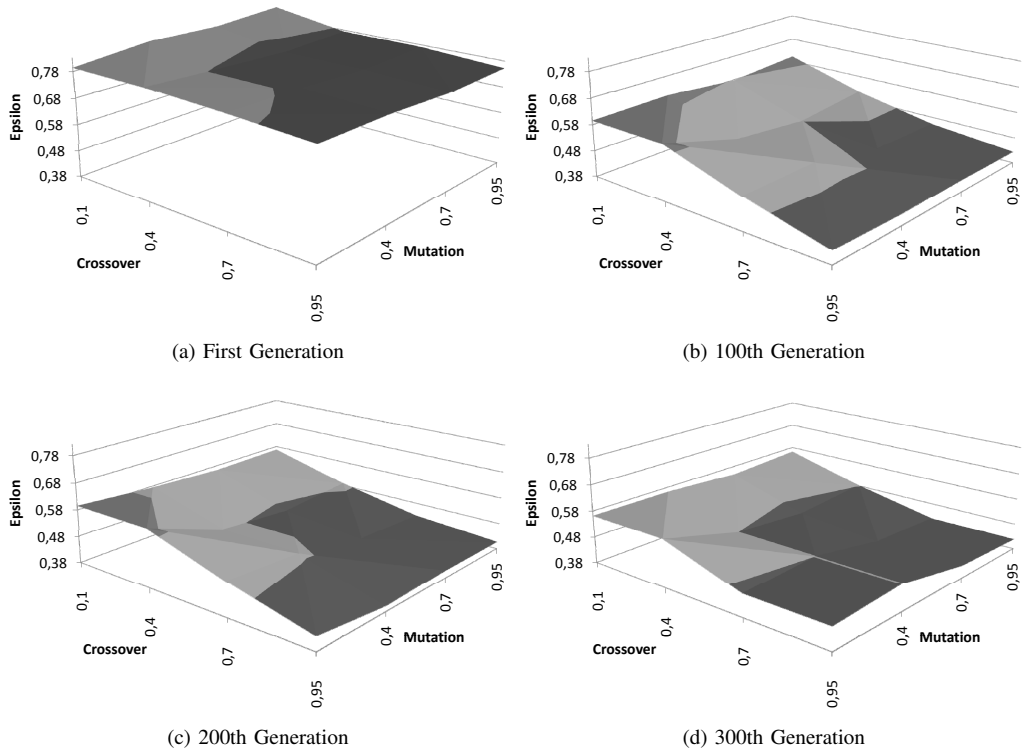
[2] J. O'Sullivan, D. Edmond, and A. Ter Hofstede, "What's in a Service?" *Distributed and Parallel Databases*, vol. 12, no. 2–3, pp. 117–133, 2002.

[3] M. Jaeger, G. Rojec-Goldmann, and G. Mühl, "QoS Aggregation for Web Service Composition using Workflow Patterns," in *Proceedings of the Eighth IEEE International Enterprise Distributed Object Computing Conference*, 2004.

[4] J. Lindenberg, P. Wouter, K. Kranenborg, J. Stegeman, and M. Neerincx, "Improving service matching and selection in ubiquitous computing environments: a user study," *Personal and Ubiquitous Computing*, vol. 11, pp. 59–68, 2007.

[5] B. Klöpper, F. Ishiakawa, and S. Honiden, "Service composition with pareto-optimality of time-dependent qos attributes," in *Proceedings of the 8th International Conference on Service-Oriented Computing*, 2010.

[1] Used system: 2.53 GHz Intel Core Duo, standard memory allocation



(a) First Generation

(b) 100th Generation

(c) 200th Generation

(d) 300th Generation

Figure 6: Average epsilon values with progressing evolution

[6] A. Klein, F. Ishikawa, and B. Bauer, "A Probabilistic Approach to Service Selection with Conditional Contracts and Usage Patterns," in *Proceedings of the 7th International Joint Conference on Service-Oriented Computing*, ser. ICSOC-ServiceWave '09. Springer-Verlag, 2009, pp. 253–268.

[7] A. Strunk, "QoS-Aware Service Composition: A Survey," *Web Services, European Conference on*, pp. 67–74, 2010.

[8] X. Vuong, T. Hidekazu, and M. Rzosuke, "Multi-QoS Criteria Based Selection of Web Services by Using AHP Approach," *IEEE Transactions on Electronics, Information and Systems,*, vol. 129, pp. pp.575–586, 2009.

[9] F. Wagner, B. Klöpper, F. Ishikawa, and S. Honiden, "Towards Robust Service Compositions in the Context of Functionally Diverse Services (to appear)," in *WWW '12: 21st international conference on World Wide Web*, 2012.

[10] J. Peer, "Web service composition as AI planning - a survey," University of St. Gallen, Switzerland, Tech. Rep., 2005.

[11] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "An approach for QoS-aware service composition based on genetic algorithms," in *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO)*, 2005.

[12] L. Zeng, B. Benatallah, A. H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-Aware Middleware for Web Services Composition," *IEEE Trans. Softw. Eng.*, vol. 30, no. 5, pp. 311–327, May 2004.

[13] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau, "HTN planning for Web Service composition using SHOP2," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, pp. 377–396, 2004.

[14] K. Chen, J. Xu, and S. Reiff-Marganiec, "Markov-HTN Planning Approach to Enhance Flexibility of Automatic Web Service Composition," *IEEE ICWS*.

[15] S. Kalasupur, M. Kumar, and A. Behrooz, "Dynamic Service Composition in Pervasive Computing," *IEEE Transactions on Parallel and Distributed System*, vol. 18, pp. 907 –917, 2007.

[16] F. Wagner, F. Ishikawa, and S. Honiden, "Qos-aware automatic service composition by applying functional clustering," *IEEE International Conference on Web Services*, 2011.

[17] M. Alrifai and T. Risse, "Combining global optimization with local selection for efficient QoS-aware service composition," in *WWW '09: 18th international conference on World Wide Web*. New York, NY, USA: ACM, 2009, pp. 881–890.

[18] A. Klein, F. Ishikawa, and S. Honiden, "Efficient QoS-Aware Service Composition with a Probabilistic Service Selection Policy," in *Proceedings of the International Conference on Service Oriented Computing (ICSOC)*, 2010, pp. 182–196.

[19] A. Klein, Adrian, F. Ishikawa, and S. Honiden, "Efficient Heuristic Approach with Improved Time Complexity for QoS-aware Service Composition," in *IEEE ICWS 2011: The 9th International Conference on Web Services*, 2011.

[20] L. Ai and M. Tang, "A Penalty-Based Genetic Algorithm for QoS-Aware Web Service Composition with Inter-service Dependencies and Conflicts," in *Proceedings of the 2008 International Conference on Computational Intelligence for Modelling Control & Automation*, 2008.

[21] H. Wada, P. Champrasert, J. Suzuki, and K. Oba, "Multi-objective optimization of sla-aware service composition," in *Proceedings of the 2008 IEEE Congress on Services - Part I*, ser. SERVICES '08, Washington, DC, USA, 2008.

[22] J. Wang and Y. Hou, "Optimal web service selection based on multi-objective genetic algorithm," in *Proceedings of the 2008 International Symposium on Computational Intelligence and Design - Volume 01*, Washington DC, USA, 2008.

[23] W. Wiesemann, R. Hochreiter, and D. Kuhn, "A stochastic programming approach for qos-aware service composition," in *Proceedings of the Eighth IEEE International Symposium on Cluster Computing and the Grid*, 2008, pp. 226–233.

[24] B. Klöpper, "Scheduling for self-optimizing and adaptive manufacturing systems," in *Proceedings of the 21st International Conference on Industrial Research*, 2011.

[25] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, 2002.

[26] J. J. Durillo and A. J. Nebro, "jMetal: A java framework for multi-objective optimization," *Advances in Engineering Software*, vol. 42, pp. 760–771, 2011.

[27] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca, "Performance Assessment of Multiobjective Optimizers: An Analysis and Review," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, 2003.