Method of Adaptive Quality Control in Service Oriented Architectures

Tomasz Szydło and Krzysztof Zielinski

Department of Computer Science,
AGH University of Science and Technology
tomasz.szydlo@agh.edu.pl, kz@ics.agh.edu.pl

Abstract. Internet era is very attractive for growing up small companies and start-ups because of simplicity of selecting services that need to be used for composing complex applications. This model requires a novel approach to accounting and providing Quality of Service defined by Service Level Agreement. By characterizing these needs we make a proposal for service adaptation that can dynamically change offered quality with respect to customer's preferences and budget.

1 Introduction

Software systems becoming larger and far more complex than ever. At the same time, they have to provide assured quality and performance. What is more, time to market is shortened significantly. Most of the basis functionalities of modern software are reusable components that might be shared between projects. Analyzing many projects from 2003 to 2006, IBM research has found that most of them express the same architectural template. The result of this findings is the Service-Oriented Solution Stack (S3) [5], which provides a detailed architectural definition of an *SOA* across nine layers from business process to operational systems layer. These layers are crossed by integration, quality of service, information architecture and governance layers. Each of them has a logical and physical aspect. Logical aspect includes architectural elements, design decisions, where physical aspect is related to the technology of implementation.

Service Level Agreement is the contract between provider and customer. *SLA* defines the terms and conditions of service quality that provider delivers to the customers. Most important in *SLA* is the Quality of Service information. *QoS* information consists of several criteria like execution duration, availability, execution time, and many more. *SLA* constitutes also financial information as a price for using service, and the way in which penalties are compensated.

With such an complex systems, it is not possible or it is very difficult to analyze and tune the application manually to fulfil *SLA* requirements. Any change can have influence on financial condition of the company because any deviations on business agreements are needed to be compensated. Additionally, growing up companies insist on incorporating more flexible way of accounting and payment methods for service

usage. It is very convenient, from customer point of view, to point out what *QoS* metrics are more important than the others, and what is the maximum budget which might be spent for using service. Aim of service provider is not only to assure *QoS*, but dynamically change it not to overdraft budget. We think that service oriented architecture allows for building applications which might adapt to the changes in execution environment.

The structure of this paper is as follows. Section 2 discusses related work. In Section 3, concept of adaptive quality control that we propose is presented. Motivating scenario is presented and evaluated in Section 4. Finally, conclusions and future work are sketched in Section 6.

2 Related Work

In this section, we cover related work on *QoS* driven adaptable architectures and approaches.

2.1 Autonomic Computing

System that is called adaptive is a one that is able to modify itself to adapt to the changes in the environment. System must be aware of context information. This is done mostly by monitoring modules and the implemented sensors. As a reaction to the changes, system modifies itself through the executing modules by plenty of effectors. Planning and evaluating what to do when changes are noticed is described by adaptation logic. This approach is investigated further, but it derived from Autonomic Computing that has been started by IBM in 2001. Main goal is to create self-management systems to overcome growing complexity and to reduce effort of maintenance.

Fig. 1. Monitor-Analyze-Plan-Execute loop

In Autonomic System, operator does not influence system directly, but defines general policies, which defines system behaviours. IBM has defined the following four areas of usage: *Self-configuration* is an autonomic configuration of components; *Self-healing* automatically discovers and corrects faults; *Self-optimization* automatically monitors and controls of resources to ensure good functioning with respect to defined requirements; *Self-protection* is an ability to identify and protect system from attacks.

Adaptation process might be divided into two orthogonal aspects:

- *Behavioural versus architectural adaptability.* Adaptation is behavioural when the behaviour of the service can be modified without modifying its structure. This is done by tuning up some parameters. In contrast, architectural adaptability takes place when the structure of the system is modified by means e.g. of switching between instances of the service.
- *Run-time versus design-time adaptation.* Adaptation actions may be executed at run-time or at design-time. The adaptation is run-time when it can be performed during execution, and design-time is when selection of services is done on the designing stage.

2.2 QoS Frameworks

Several approaches have been proposed for *QoS* driven service selection. Zeng [4] considers service selection as a global optimization problem using linear programming. Target function that is optimised is a linear combination of *QoS* metrics. Similar approaches incorporate modified Dijkstra's algorithm [7], constraint programming logic or a knapsack problem. That approach is only applicable in design time adaptation because it does not recalculate solution during composite service execution. We share Kokash [6] opinion that quality of these solutions depends strongly on the user weights for each *QoS* metric that is not trivial to establish them in the right way.

Abdelzaher et al. [8] shows how control-theoretic approach can be used to achieve quality of service guarantees. They demonstrate that a software system can be approximated by a linearized model and controlled by actuators and sensors.

3 Concept of Adaptive Quality Control

Analysis of existing frameworks in terms of *QoS* unveils that this term is used interchangeably for describing quality from provider point of view as well as from client point of view. We have decided to distinguish *Quality of Experience* from *Quality of Service*.

Fig. 2. Different ways of perceiving quality

QoS is the quality of provided services; *QoE* is the observed quality by customer. The idea is presented in Fig. 2. It is very common that these values are completely different. Let us think of calculating balance account on the end of month. Accounting service which we are using is 99,9% available but, once a month is disabled for maintenance for 1 hour. If we are accessing this service during this hour, from our point of view, its availability would be 0,0%. When we assume that *QoS* is a vector of values for a given metrics, *QoE* is defined as:

$$QoE = \sum_{i=1}^n QoS_{t_i} \times \frac{1}{n} \quad \text{for } n \rightarrow \infty$$

where n is the number of invokes of composite service and t_i is the time when invocation takes place. It makes us to think of monitoring QoE and on that basis adapt composite service to fulfil requirements of an agreement.

Further research led us to the idea of QoS controller, depicted in Fig. 3 that would be responsible for providing desired QoE described by SLA . Service oriented architecture decomposes composite service into set of inter-working simple base services. Set of services with the same functionality will be called abstract service. During invocation of composite service, we can exchange simple service by any that belongs to the same abstract service set. We can think of this idea as of an interface and the class that implements it.

Fig. 3. Control loop

In our work we distinguish four kind of services:

- *Abstract Service* (S_i). Service, which is described by its functionality but instance is not pointed out;
- *Service* (I_i^k). Service is a functionality provided by provider;
- *Abstract composite service* (S). Composite service that contains at least one abstract service;
- *Composite service* (C). Composite service that might be executed because all of its base services are instances.

Composite abstract service consists of several abstract services $S = \{S_1, S_2, \dots, S_n\}$, each having several instances $S_i = \{I_i^1, I_i^2, \dots, I_i^{k_i}\}$. In this work, we are assuming that services S_i are executed sequentially. This is very strong assumption and applicable only for simple services, but problem of evaluating concurrent services [3] with structured activities is out of scope of this paper. However, idea of QoS controller elaborated later is applicable for this type of composite services as well.

To make a good use of service adaptation, we have to collect history of previous invocations of base services. Taking into account these data, we can envisage more or less accurately the quality of invocation. Analyzing it deeper we have figured out, that quality of provided service is strictly related to context of execution environment. Due to this fact, one can use services located on the globe where currently night is hence, servers are not overloaded.

Assuming that we have a history of I_i^k invocations, expected quality of further invocations might be calculated using regression analysis. In the simplest case, it can be simple moving average. Before any assumptions about expected quality, we can ask base service provider of execution environment context and on that basis we can analyse only subset of the invocation history where the context information was the same.

3.1 Quality of Service Metrics

Qualities of Service attributes include metrics like throughput, response time, availability but exact definition and measurement process must be well defined to give consumer and provider common understanding. As for describing Web service's functional aspects WSDL language has been defined, there is no universal language for describing metrics and the way of measuring it. Moreover, many of the well-known metrics like availability do not have formal definition. For example availability might be described as percentile value, or the number of positive invocations during last 50 invocations, or the availability during last hour. Metrics are not only applicable for simple services, but also for composite ones so for each metric must be provided algorithm for calculating values for a service composition [1]. In this paper we are considering only services invoked in a sequence.

We can distinguish metrics to the quantitative ones that have numerical values or might be described by values and the qualitative ones that are described in words. Nevertheless, for any metric must be provided an equation or algorithm to recalculate it to $<0;1>$ range. QoS is a vector of values for a given metrics.

If service a is better then service b from a given metric point of view, means that metric value of service a is greater than value for service b . In the other case, for evaluating we have to use inversions of the metric values.

3.2 QoS Controller

We have designed a QoS controller, which continuously monitors deviation of current user QoE from agreed SLA , and on that basis, service QoS is tuned by selecting service instances that need to be invoked. Deviation of provided quality of service is described as follows:

$$\Delta QoS = QoE_{SLA} - QoE$$

Before execution of abstract service, it is decided which instance to invoke. Decision process takes into account services already invoked, correction from feedback loop, and influence of each abstract service instance on possible overall quality. Number of total invocation possibilities is $\prod_{i=1..n} \|S_i\|$, but only when none of

abstract services was invoked until now. Assuming that services $S_1, \dots, S_k$ were invoked, total number p of possible invocations is $\prod_{i=k+1..n} \|S_i\|$. Before invocation of any base

service, all possible invocations are evaluated as presented in Fig. 4.

For the services that have not been invoked yet, mean value $\overline{I_i^k}$ from historical invocations is used for calculating QoS of this composition. From the set $\{C1, \dots, Cp\}$ of possible invocations, we are selecting one that fulfils:

Fig. 4. Execution model

$$\forall_i \text{sign}(QoS(C_x)^i - QoE^i) = \text{sign}(\Delta QoS^i)$$

It guaranties that if any of the metrics value in *SLA* is greater than in *QoE*, composite service will be selected with *QoS* containing that metric large enough to balance difference, and analogically small enough when metrics value in *SLA* is less then in *QoE*.

3.3 Variable QoS

Providing contracted quality of service is very competitive task, as well as, providing service with variable quality that do not exceed specified budget. Our idea is to estimate the number of invocations to the end of accounting period basing on the number of invocations until now. Having this information and the amount of money which left to spend, we can change the offer to the cheaper one. Customer has priorities what sorts of metrics are more important then others. Providing exact weights of each metric is multi dimensional decision problem and it is not a trivial task. We have found that decomposing problem of assigning weights makes it easier to deal with. *Analytic Hierarchy Process* [2] is a technique based on mathematics and human psychology for prioritizing elements of decision problem. For each pair of metrics user specifies which one is preferred in the form of fraction between 1/9 and 9/1. The result of AHP is a vector of weights w_i for each metric i . We will be referring to fitness factor of *QoS* with additional information of metrics importance as:

$$\text{fitness}(QoS) = \sum_i w_i \times QoS^i$$

Customer chooses set of *SLA* in which is interested. After estimating cost per single invoke, system choose the *SLA* with the best fitness and the price per invoke less than calculated.

3.4 Service Level Agreement

Client agreement is represented as tuple:

$$SLA = (QoE_{SLA}, price, penalties, time)$$

where *QoE* is the quality of user experience, *price* is the amount of money which client has to pay for each service invoke, and *penalties* is the price which provider will pay in the case of any deviation of *QoE*, and *time* is the accounting period.

Price for using service *S* is defined as follow:

$$bill = n * price - missed * penalties$$

$$missed = \begin{cases} 0 & , \text{if } QoE \geq QoE_{SLA} \\ \max_i \left(\forall : (QoE_{SLA}^i - QoE^i) n \right) & , \text{if } QoE < QoE_{SLA} \end{cases}$$

where n is the number of invocations. Accounting algorithm might be illustrated by simple example. Let us say that availability agreed by SLA is 50%, and on the end of month, on 20 invokes, only 6 was successful. It means that customer was overcharged unfairly for 4 invokes which has to be compensated. It is a company politic which defines how value of *price* is related to value of *penalties*.

Accounting for variable *QoS* is quite different because client agreement is represented as a set of SLAs and the metrics weights calculated using AHP from *preferences* set:

$$SLA_{var} = (preferences, \{SLA_1, \dots, SLA_i\}, budget)$$

System dynamically changes *SLA* to assure that bill will not be greater then *budget*. On every accounting period, client is charged independently for every *SLA* contract but total sum is not grater than *budget*. Client is eligible to receive compensation if any of *SLAs* is violated.

4 Motivating Scenario

Let us assume that we have Internet website that provide information for amateur pilots and we want to include very accurate weather forecast for nearest airports on the main site. Composite service consists of two base services i.e. service which takes city name on input and returns airports in the near proximity, and the second service that takes name of the places and returns weather forecast. We have to define the metric for describing quality of provided data. Moreover, our portal is a growing up business hence, we have limited budget to maintain system, so we do not like to spend more than assumed price for the service. For this type of service, it is better from customer point of view, to achieve very accurate data than has high availability of poor information. To verify our concepts we have developed simulation environment that is flexible enough to implement different adaptation strategies.

4.1 Metrics

We have assumption that base services are invoked in a sequence. Below are described metrics used in this example scenario and algorithms for calculating values for composite services.

Availability. Availability is the probability that service is accessible. For our composite service, availability is the product of availabilities of the base services.

Execution time. Execution time is the time between invoking service, and the time of receiving response. For composite service, execution time is the sum of execution time of each base service.

Data Quality. For this example we have provided Data Quality metric which defines the quality of received data. Data quality means simply the accuracy in kilometres of

weather information. Value of this metric for composite service is the minimum value of the base services.

As it was mentioned before, our service is very specific, so customer has to decide what is more important and what is not:

- *availability* is three times as important as *execution_time*;
- *availability* is five times less important than *data_quality*;
- *execution_time* is five times less important than *data_quality*.

Fig. 5. Evaluated metrics weights

Results of metric weights evaluation with AHP are presented in Fig. 5. Customer agreed to use three SLA. Each contract has the same data quality metrics, but differs in response time and availability. In the table below, contracts are listed in the order with respect to metrics weights.

Table 1. SLA contracts

SLA	Price per 100 invocations	Response time [ms]	Availability	Data quality [km]
0	1	100	0,9	5
1	0,5	300	0,8	5
2	0,2	200	0,7	5

4.2 Evaluation

Customer noticed that his website gets three thousands visits per month. As the website become more popular, number of invocations may significantly increase. In this situation client has decided to buy service with variable *QoS* and with maximum budget of 30€. Detailed statistic of user invocations is presented in Fig. 6. Larger number of page visits during third week we can explain that there was national flying contest.

Total number of invocations was not as expected three thousands but almost five thousands. During the month, system tried to estimate number of invocations until the end of an accounting period and this is depicted in Fig. 7.

Because number of invocations was greater then expected, system has to switch SLA, to makes the total sum less than assumed 30€. We can notice that on 14th day system has decided to switch SLA one down, but increased number of visits stayed for three days, so system has decided to switch one more down. Web site traffic started coming back to normal state, so system come back to the best SLA as depicted in Fig. 8.

To calculate expected number of invocations to the end of the month, system estimated it on the moving average basis from the last 7 days, and mean number of visits per day is multiply by the number of days to the end.

Fig. 6. Number of invocations per day

Fig. 7. Estimating number of invocations to the end of the month

Fig. 8. Selected SLA

Fig. 9. Partial bill

Fig. 10. Convergence of availability

SLA has been changed several times during accounting period hence, *QoS* metrics had to convergence to the values contracted in selected *SLA*. One can notice that convergence schema in Fig. 10 is very similar to the one known from the control theory.

4.3 Discussion

Without variable *QoS*, customer would be asked to pay 50€, which is a lot more than assured 30€. With variable *QoS* activated, system is able to encompass busy week on

the web site. As it has been stated in contracted *SLAs*, data quality was the same during the whole month, but availability and response time have changed as many times as *SLA* has changed. In Fig. 9 we can find partial bill during the whole month. The proposed approach does not take into account deviations caused by internet connections. Possible extension to incorporate it is to design remote invocation monitoring which would be deployed at the customer side. Secondly, accounting method might be unfair, especially when number of composite service invocation is very low or when invocations are not equally spread over the accounting period.

5 Conclusion

In this paper, we have presented novel approach to managing composite services. With the growing popularity of the Internet, providing services with guaranteed *QoS* has become increasingly important. Our algorithm integrates statistical methods for predicting quality of service for composite services invocations, as well as, automatic adaptation strategies to keep consumer's budget by changing *SLA* during execution. Presented case study verified usability of our method.

Many remaining issues are worth further research. The most interesting is how this model behaves in real applications. Another interesting aspect is how to improve accounting algorithm to prevent unfair charges. Finally, further examples, experimental tests and practical experience are needed to find true potential of applying adaptive quality control to different class of composite services. This remains an important focus for our future research.

References

1. Menascé, D.A.: Composing Web Services: A QoS View. *IEEE Internet Computing* 8(6), 88–90 (2004)
2. Forman, E.H., Selly, M.A.: *Decision By Objectives – How To Convince Others That You Are Right*. World Scientific Publishing Co. Pte. Ltd., Singapore (2001)
3. Cardoso, J., Sheth, A.P., Miller, J.A., Arnold, J., Kochut, K.: Quality of service for workflows and web service processes. *J. Web Sem.* 1(3), 281–308 (2004)
4. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering* 30(5), 311–327 (2004)
5. Arsanjani, A., Zhang, L.-J., Ellis, M., Allam, A., Channabasavaiah, K.: S3: A Service-Oriented Reference Architecture. *IT Professional* 9(3), 10–17 (2007)
6. Kokash, N.: A Service Selection Model to Improve Composition Reliability. In: *Proc. Proceedings of the International Workshop on AI for Service Composition* (2006)
7. Gu, X., Nahrstedt, K., Chang, H., Ward, C.: QoS-Assured Service Composition in Managed Service Overlay Networks. In: *Proc. ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*, Washington, DC, USA, p. 194 (2003)
8. Abdelzaher, T., Stankovic, J., Lu, C., Zhang, R., Lu, Y.: Feedback performance control in software services. *Control Systems Magazine*, 23(3), 74–90 (2003)