Platform for Autonomous Service Composition

Krasimir Baylov, Dessislava Petrova-Antonova, Aleksandar Dimov
Department of Software Engineering
University of Sofia "St. Kliment Ohridski"
Sofia, Bulgaria
e-mail: krasimirb@uni-sofia.bg, d.petrova@fmi.uni-sofia.bg, aldi@fmi.uni-sofia.bg

Abstract — The increasing complexity of business processes requires improved methods for composition of web services. In order to fulfill this, it is difficult for administrators to keep up with the growing demand and the enormous amount of customizations required by the users. A possible solution that will help in this situation is to develop methods and technologies that support autonomous compositions of web services. Such compositions should adapt dynamically to changes in the requirements or the environment. This paper describes a platform which implements such solution, based on Quality of Service (QoS). The platform prototype, which is presented here, is able to monitor web service QoS and determine whether the service composition fulfils the overall quality required by the end users.

Keywords - Quality of Service; Web services; SOA; Dynamic web services composition

I. Introduction

Service Oriented Architectures (SOA) play an important role in enabling integration of business with IT [25]. Services are a key concept in SOA and they represent reusable entities that should minimize the development effort and provide means for information exchange for both service consumers and providers. On the other hand, the complexity of business problems is increasing and to solve them, users could employ a number of services into a composition to execute a business process. However, as business gets more and more flexible today, consumers require additional functionality and customizations towards the services they use. In other words, a static composition is not capable to fulfill all user requirements in a long term perspective. This makes service providers search for ways to deal with the increasing number of service demand while at the same time providing personalized Service Level Agreement (SLA) management [26].

A solution to this problem is to provide compositions that are autonomic and are capable to adapt to changes in user requirements or the environment. Such compositions can adapt according to some measurable rules. Let us consider that, for a composition, one should choose a service out of a set of services that share similar functionality. To solve that, it is possible to choose a service that offers the best Quality of Service (QoS). Moreover, it would be better if the composition is not static but changes dynamically according to changes in the QoS of services (based on changes in workload, number of requests, etc.) or in user requirements. For example, if more users send requests to a service, its response time may raise to an undesirable level, and then another service should be found and integrated into the composition. The goal is to make this with minimal human intervention and implement the change dynamically and transparently for the user.

This paper presents a platform for building autonomous web service compositions based on QoS. The platform provides means for gathering data for evaluation of service QoS characteristics (like performance, availability, reliability, cost, etc.). Such means include:

- An extended service registry, which is used as a repository for collection of service QoS data and enables easy web-services search and selection
- A model to calculate service QoS, according to the data in the registry
- An algorithm to find and select the services that will best meet the agreed SLA of the composition
- Automatically and transparently integrate selected web services into a working composition

A key aspect of the proposed platform is that service compositions are determined and updated dynamically at runtime. This frees administrators and developers from implementing any QoS related changes.

The rest of this paper is organized as follows: Section 2 makes an overview of the related work; Section 3 presents the model that we use for evaluating the quality of web service compositions and determining the best composition; Section 4 introduces the design and implementation of our platform for autonomous web service composition. Section 5 presents a simple case-study to illustrate usage of the platform and validate it, and finally, Section 6 concludes the paper and states directions for further research.

II. Related Work

Quality attributes are very important in terms of design and reasoning about of software systems. They are regarded as key concerns in software architecture design [3] and selection of relevant web services [9]. Many researchers have also managed to solve the problem with formal definition and management of software quality in general. For example, there exist a lot of theoretical models for evaluation of reliability [6][24], performance [2][7], complexity, etc. However, such models tend to be relevant only at theoretical level as they are either quite general and have some unrealistic assumptions that make them inapplicable in practice or too complex to be applied in a broad range of domain areas.

The work that relates to ours may be split into two main directions: first one is related to models for calculation of software QoS and second one – to methods for dynamic and autonomous web service composition. Many research efforts combine the two directions and use the overall QoS of a service composition to determine whether it should change or not [11][13]. There are a lot of models available that consider the quality characteristics and based on them provide the best service composition.

Liu et al. [22] proposed of QoS model that is open and extensible. They provide an implementation of a QoS registry that stores the web services QoS data and allows consumers to search against it. A key point here is that such data is obtained through user feedback, i.e., consumers that use the services rate them and provide their feedback to the QoS registry.

Ran [18] proposed a model for web service discovery based on QoS. They argue that current web service registries limit service discovery to functional requirements only and non-functional properties should be paid more and more attention. They extend the current web services registration and discovery model by introducing a new role – *Web Service QoS Certifier*. The concept of a certifier is also covered in [12]. The certifier is responsible to certify/verify the claimed non-functional properties of the web service providers.

The DYSCO platform [15] provides a complex solution for dealing with dynamic web service composition. The platform allows automatic generation of executable business processes and SLA for each web service. It also provides mechanisms for monitoring the used web services and updating the business process when SLA deviations are discovered.

AgFlow [10] is a middleware platform that allows quality-driven dynamic web services composition. The platform provides a multidimensional QoS model that is responsible for capturing the non-functional properties of the web services. This work introduces two approaches for selecting web services – *local optimization* and *global planning*. An adaptive execution engine is responsible for the runtime adaptation of the web services composition. It replans the execution any time when any of the services is unavailable or the quality properties exceed predefined thresholds.

The web service composition algorithm proposed by Lu et al. [19] is based on seven QoS properties – running cost, runtime, success ratio, usability, trustworthiness, degree of security and degree of semantic correlation. A limitation of the algorithm is that only semantic (immeasurable) QoS properties are considered. In addition, it is not clear how the QoS properties are assessed. In contrast, Yu et al. [20] rely on measurable QoS properties and especially on latency, execution cost, availability and accuracy. The advantage of the proposed solution is that it is applicable to data intensive web service compositions. It combines the tabu search and the genetic programming techniques. The last one is applied also in the web service composition approach presented in [1].

An approach for self-healing web service composition is introduced by Aziz et al. [14]. It repairs the web service composition when some of its components violate the QoS constraints. The headers of the SOAP messages are extended in order to provide information about QoS properties. The approach includes three main phases: monitoring, diagnosis and repairing. When QoS degradation is detected during diagnosis phase, a repairing procedure is started. As a result the failed web service is replaced with another one obtained from the UDDI registry. A possible drawback of the approach is that it relies on SOAP as communication protocol and is not clear how it could be applied when the composition includes REST web services.

The web service composition system presented by Brahmi and Gammoudi [23] is based on cooperative agents. The agents are organized as a social network and cooperate to find the optimal composition with respect of QoS. The approach proposed by Xia and Yang [21] is focused on QoS optimization and redundancy removal. An advantage of the approach is that it removes most of the redundant web services minimizing total execution cost of the composition. Unfortunately, the QoS optimization is based only on two QoS properties – response time and throughput.

Birgit and Marchand-Maillet [5] solved the web service composition problem partially by providing an algorithm for QoS-aware selection. The algorithm uses a rank aggregation instead of direct measures of QoS values. Its core includes so called abstract voter that sorts the web services according to a particular QoS property, named QoS factor. In [8], the web service composition problem is formalized as problem of traversing a Petri Net. The estimation of composition's quality is performed through utility function that aggregates the functional, QoS and transactional properties of the web services.

Currently, there is no universal approach for autonomous management of dynamic web service composition based on QoS. In this work, we propose a platform that deals runtime with QoS monitoring, adaptation and discovery of web services, in order to determine the best possible composition. Another advantage of the approach presented here is that it is compatible with the Business Process Execution Language (BPEL) standard and is capable of implementing an executable composition.

III. A MODEL FOR AUTONOMOUS WEB SERVICE COMPOSITION

In this section, we present the model we use for autonomous web service composition. It includes analyzing the quality data for each eligible web service and determining the best composition that matches a predefined set of quality requirements. The presented model is based on [16] and [17], but adds the following additional features to achieve the goal:

- Introduces the concept of a web service category as an abstract entity that may refer to multiple web services, providing the same functionality and interface.

- Uses weight (i.e., priorities) of quality attributes to determine the weight of actual user requirements.
- Analyzes all service compositions that may be integrated in order to find the one that best matches the user defined business process.

Moreover, the model introduces the concept of a *web service category*. Each category is defined by common functionality and an interface and may be associated with multiple web services. Therefore, the presented model requires that users define their business processes by specifying the web service categories rather than the concrete web service implementation. In other words, each business process is considered as a composition of multiple web service categories. Concrete web services are assigned after the model is applied and the best composition is known.

Currently, to our best knowledge, there is no unified standard for managing web service categories. For the purpose of this research, we have used a central service registry for discovering web service categories and associated web service implementations for each of them.

The first step in our model is determining the set of web service categories that build our business process. In this case, we are not interested in the sequence of the particular web service invocations but need to know the set of different web service categories like

$$BP = \{WSC_1, WSC_2, \dots, WSC_n\} \qquad (1)$$

WSC_n refers a single web service category that is used in the business process definition.

Additionally, we need to know the specific web service implementations associated with each category

$$WSC_i = \{WS_{i_1}, WS_{i_2}, \dots, WS_{i_m}\} \qquad (2)$$

In this case, WSC denotes a single service category and WS denotes a particular web service implementation. A single web service category WSC may include multiple web service implementations WS.

We also need the set of quality requirements *R* and their associated weights *C*.

$$R = \{r_1, r_2, \dots, r_l\} \qquad (3)$$
$$C = \{c_1, c_2, \dots, c_l\} \qquad (4)$$

Requirements and weights are set by the business process designer. They should reflect the end user needs. Requirements represent the quality characteristics under consideration like performance, availability, throughput, etc.

Weights are measured by relative values ranging between 0 and 1. Naturally, the sum of all weights should be equal to 1. l represents the number of quality attributes under consideration (performance, availability, etc.). Note that the requirements and the weights are paired. For each requirement r_i, there is an associated weight c_i.

Then, for each web service, the relevant quality data should be presented in a matrix. Each row represents an execution of the web service and each column represents a quality attribute $\{r_1, r_2, \dots, r_l\}$.

$$R_{WS_{ij}} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1l} \\ x_{21} & x_{22} & \cdots & x_{2l} \\ \vdots & \vdots & \ddots & \vdots \\ x_{k1} & x_{k2} & \cdots & x_{kl} \end{bmatrix}, i = 1 \div n, j = 1 \div m \quad (5)$$

In this matrix, k specifies the number of web service calls. Each row in this matrix represents the different quality characteristics for the related call like response time, throughput, available or not, etc. The data in this matrix is dynamic. It changes as new web service calls are invoked and quality data is updated. Note that we may not need to analyze the entire set of web service calls but only a subset of them. For example, in many cases it may be more practical to analyze only the last number of calls. This number may be updated dynamically based on the platform that uses the presented model.

The next step in this approach is to calculate the quality attribute values and normalize them so that they could be easily compared. For this purpose we need the average, minimum and maximum values for each quality characteristic from the $R_{WS_{ij}}$matrix. This means that we need separately process each column in the matrix. The average value is the sum of all x values for a quality characteristic divided by their number. The min and max values represent the lowest and highest values respectively. Therefore, for each column z ($z = 1 \div l$), we calculate the normalized value for the quality characteristics.

$$P_{zWS_{ij}normalized} = \frac{<average>}{|<max> - <min>|}, z = 1 \div l \qquad (6)$$

Once we have the normalized quality characteristics values, we should sum them in order to get a numeric representation of the web service quality.

$$R_{WS_{ij}} = \frac{\sum_{z=1}^{l} P_{zWS_{ij}normalized} \cdot c_z}{l} \qquad (7)$$

R_{WS} represents the normalized quality value of the j-th web service from category *i*. It is important to consider the weight/priority of each quality attribute. Therefore, the value representing the overall quality for a single web service would be the sum of the normalized quality values for each quality characteristic multiplied by the relevant weight factor.

By now, we should have a numeric representation of the quality of each web service. Next we analyze all combinations of web services in the composition to find the best one. For each possible composition we calculate the related quality by summing the quality values for each service that builds it.

$$Q_{BP} = \sum R_{WS_{ij}} \qquad (8)$$

Once the quality value for each web service composition Q_{BP} is calculated, we analyze them and select the one that has achieved the highest score. As a result, a single web service composition is selected. This is the one that best meets the user requirements according to the model.

This model allows us to find the best web service composition based on a predefined set of quality requirements and their associated weights. It could easily be applied to any number of web services and extended to support various types of quality attributes. This model is applied in the implementation of our autonomous service composition platform. The technical implementation of the platform is presented in Section 4.

IV. QOS BASED PLATFORM FOR AUTONOMOUS SERVICE COMPOSITION

In this section, we present the design and implementation of our platform. It is based on the model described in the previous section and follows the architecture presented on Figure 1. The model for determining the best service composition is implemented in the *BPEL Extension* component. The current platform allows runtime updates to the deployed service compositions with no human supervision.

Our platform also provides an extended web service registry that allows consumers to search for the services they need and also inquire information for their QoS characteristics. Finally, all these data are processed by a BPEL extension tool which is part of our previous work and it allows dynamic binding of the selected web services in the defined composition [4]. The platform consists of the following components:

1. **BPEL Extension** – extension deployed on business process server allowing to perform the runtime composition of web services and adjust to the quality requirements of each user.
2. **Extended Service Registry** – a standard service registry with a DB extension for persisting quality attributes data for the web services. Access to this data is exposed as part of the registry interface allowing service consumers to use it for their composition analysis.
3. **Web Service Interceptor** – tool that is able to intercept any web service call and collect the needed quality attributes data. This data is then stored in the extended service registry and made available of other service consumers.

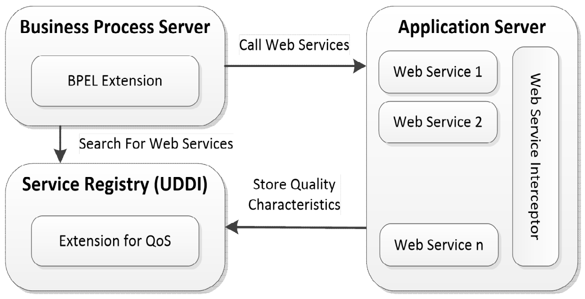

Figure 1. Architecture of the platform for dynamic web service

In the next subsections, we provide a detailed description for each of the platform components.

A. BPEL Extension

Our BPEL extension allows updating a BPEL process at runtime. It is developed according to the BPEL extension specification and can be deployed and plugged in any BPEL compliant server. In this work, the WSO2 BPS server is used to test the extension.

B. Extended Service Registry

The extended service registry provides the standard UDDI (Universal Description, Discovery and Integration) interface. Already existing *Apache jUDDI v.3.0.4* registry is used for this purpose. All web services that the platform can work with are registered there.

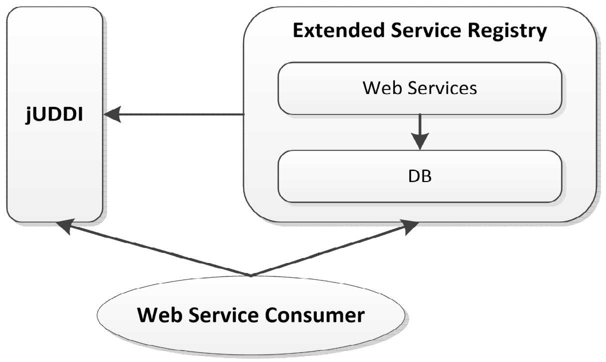

Figure 2. Extended service registry design approach

As stated in the name of the component it provides extended functionality. We have deployed a database that stores the quality characteristics for each web service invocation. This data is stored in raw format so that it can be used with various models. To make this data accessible we have developed Apache jUDDI-like services (https://juddi.apache.org/), so that consumers could obtain the quality data they need for building their service composition. Those web services are exposed as SOAP services.

In order to make the extension as loosely coupled to the UDDI registry we have implemented it as a separate tool that end users could integrate with. Figure 2 represents our design approach.

This approach allows service consumers to use the UDDI registry in a standard way and only those who are interested in the QoS data could trigger the relevant queries against the extension. In addition, our extension is aware of the service categories.

A key point here is that we try to avoid the concept of using a web service QoS certifier. We would not let service providers publish any QoS data for their web services. Rather, we would expect every provider that is interested in providing such data to install the so-called web service interceptors that we provide. They will store the relevant data in the extended registry. This way the interceptors (presented in the next section) act like a certified web service QoS data provider.

C. Web Service Interceptor

The web service interceptor is a module, responsible for gathering quality related data for each web service invocation. This module is deployed on each server that hosts the implemented web services. Because there are multiple technologies for implementing and exposing web services we have limited ourselves to using the Apache Axis 2 framework. It provides mechanisms for extensibility and we could easily integrate our custom logic. What's more the Apache framework design includes mechanisms for developing custom handlers for the supported web services.

We take advantage of this functionality and we have developed custom handlers that intercept the web service invocations. There are two types of handlers – *message flow handlers* and *error flow handlers*. The *message flow handlers* process the standard web service invocation while the *error flow* ones are activated when the web service fails. Figure 3 shows how the interceptors fit into the process of a web service invocation.

For each web service invocation we get the following data – S*OAP message size*, *processing time* and identification data like operation correlation ID, IP addresses, etc. This data is then stored in our database and exposed for calculation of quality data.

One of the major design goals for web services interceptor module is modularity. Therefore, it is implemented in an easy to configure and customize way. The interceptor module itself is packaged as a ".mar" (module archive) file. This file is deployed on the servlet container by creating a folder named "modules" in the "webapps/axis2/WEB-INF" directory.

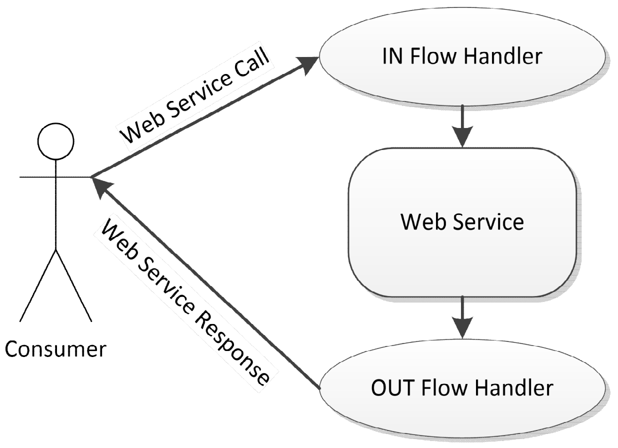

Figure 3. Intercepting web services invocation

V. EXPERIMENTS

In order to show the benefit of the Autonomous Service Composition Platform, this section presents experiments that show how it performs in selection of the best (by QoS) web service. The current implementation is still a prototype and additional validation will be made when it matures. The experiment focuses on two quality characteristics – performance and availability. We have set the weight for each for the quality characteristics to 0.7 for performance and 0.3 for availability. For the purpose of this experiment, we have defined three *web service categories*, each representing a mathematical operation – *Multiply*, *Power* and *Add*. For each category we have developed a set of three web services with the same functionality and interface but simulating different quality characteristics – *standard*, *slow* and *randomly available*.

To make the experiment we created a business process that uses the three web service categories. Each of them is called one after another. In this case, we are not interested in the final result of the calculation but we pay close attention to the quality characteristics of the executed business process. Figure 4 represents the business process we use in our experiments.

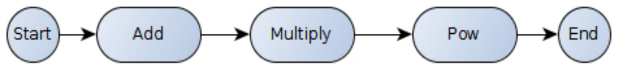

Figure 4. Experimental business process

From a model perspective, our composition can be presented in this way

$$BP = \{WS_{Multiply}, WS_{Power}, WS_{Add}\} \qquad (9)$$

After a series of service invocations we collected data regarding the quality characteristics of each web service. Table 1 presents the obtained average values.

TABLE I. AGGREGATED QOS DATA FOR EXPERIMENTAL SERVICES

Category	Web service	Avg. performance	Avg. availability
Multiply	Multiply Slow	0.493	1
	Multiply Available	0.243	0.64
	Multiply Standard	0.239	1
Power	Power Slow	0.539	1
	Power Available	0.231	0.65
	Power Standard	0.225	1
Add	Add Slow	0.543	1
	Add Available	0.253	0.67
	Add Standard	0.246	1

In this case, there are 27 possible compositions that could be built. However, each composition will have different value for the entire quality and the platform should select the one that has the highest score. Figure 4 presents a graphics of the calculated values for the quality of compositions for the possible combinations. To run this simulation we have set the weights for performance and availability to 0.5 and 0.5. The composition on the top of the graphics has highest score compared to the rest. This is the composition *{Multiply Standard, Power Standard, Add Standard}*.

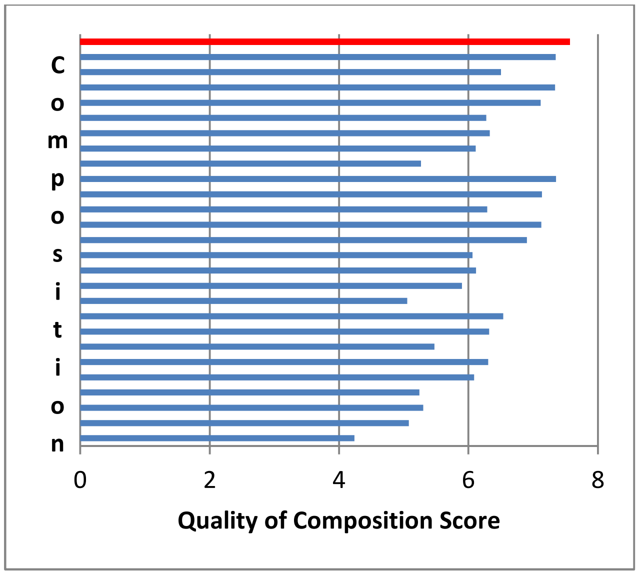

Figure 5. Scores for the Quality of Analyzed Compositions

When the experiment started our platform analyzed the defined business process and the associated quality goals. Based on the defined web service categories the relevant service implementations were discovered and the final composition was set. Table 2 presents the web services that were selected as a result of our experiment QoS data for selected services after experiment.

TABLE II. QOS DATA FOR SELECTED SERVICES AFTER EXPERIMENT

Category	**Web service**	**Avg. performance**	**Avg. availability**
Multiply	Multiply Standard	0.239	1
Power	Power Standard	0.225	1
Add	Add Standard	0.246	1
Total Quality		0.71	1

The total time for the execution of the business process is 0.71 seconds and the availability remains 100%. This is the best possible composition that fits the predefined quality requirements and the associated weights for each of them.

VI. CONCLUSION AND FUTURE WORK

In this paper, we have proposed a platform for autonomous web service compositions. This platform is able to monitor the web services execution and gather data for evaluation of service quality characteristics. This data is available through web service registry extension. The paper also proposes a model for analysis of the quality data and determining the best service composition. A key aspect of this model is the introduction of *web service category* as an abstract way of defining a set of services providing the same functionality and interface.

In addition, our platform is based on open source software and is designed for easy extendibility and modifiability. In the long term such an approach could save a lot of administrative work and increase the level of customer satisfaction. The platform provides means for autonomously adapting the running business processes based on predefined user goals in terms of SLA. However, we can state the following directions for future research, in order for the platform to provide a fully functional end-to-end solution:

1. **Extending the scope of the web service interceptor** – currently, we support Apache Axis2 based web services but we plan to develop interceptors for other web service frameworks that can be extended.
2. **Extending the number of quality attributes** – currently, we have focused our research on *performance*, *availability* and *throughput*. We consider extending the number of supported quality attributes within the interceptors and the extended web service registry.
3. **Improving the model for selecting best web service composition** – a weak point for our model is the selection of the best web service composition. It is expected that all possible compositions are analyzed and then the best one is selected. As a point of improvement, we consider optimizing the selection algorithm to work in a more efficient way.
4. **Perform detailed validation of the platform –** The presented platform is still a prototype. As the platform gets more mature, additional validation and experiments should be performed.

ACKNOWLEDGEMENT

The research presented in this paper was partially supported by the DFNI I02-2/2014 (ДФНИ И02-2/2014) project, funded by the National Science Fund, Ministry of Education and Science in Bulgaria.

REFERENCES

[1] A. Silva, H. Ma, and M. Zhang, “A GP Approach to QoS-Aware Web Service Composition and Selection”, G. Dick et al. (Eds.): SEAL 2014, LNCS 8886, 2014, pp. 180–191.

[2] A. Machado and C. Ferraz, “Guidelines for performance evaluation of web services”, In Proceedings of the 11th Brazilian Symposium on Multimedia and the web (WebMedia '05), Renata Pontin M. Fortes (Ed.). ACM, New York, NY, USA, 2005, pp. 1-10.

[3] L. Bass, P. Clemens, and R. Kazman, Software Architecture in Practice, Addison Wesley, 2013.

[4] K. Baylov, D. Petrova-Antonova, and A. Dimov, "Web service QOS specification in BPEL descriptions", In Proceedings of the 15th International Conference on Computer Systems and Technologies (CompSysTech '14), Boris Rachev and Angel Smrikarov (Eds.). ACM, New York, NY, USA, 2014, pp. 264-271.

[5] H. Birgit and S. Marchand-Maillet, "Rank Aggregation for QoS-Aware Web Service Selection and Composition", 6th IEEE International Conference on Service-Oriented Computing and Applications, 2013, pp. 252-259

[6] B. Buhnova, S. Chren, and L. Fabriková, "Failure data collection for reliability prediction models: a survey", In Proceedings of the 10th international ACM Sigsoft conference on Quality of software architectures (QoSA '14). ACM, New York, NY, USA, 2014, pp. 83-92.

[7] R. Douglas, D. F. Pigatto, J. C. Estrella, and K. Branco, "Performance evaluation of security techniques in web services", In Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services (iiWAS '11), ACM, New York, NY, USA, 2011, pp. 270-277.

[8] E. Blanco et al., "A Transactional-QoS Driven Approach for Web Service Composition", Z. Lacroix and M.E. Vidal (Eds.): RED 2010, LNCS 6799, 2012, pp. 23–42.

[9] L. O'Brien, P. Merson, and L. Bass, "Quality Attributes for Service-Oriented Architectures", In Proceedings of the International Workshop on Systems Development in SOA Environments (SDSOA '07), IEEE Computer Society, Washington, DC, USA, 2007, pp. 3-

[10] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-Aware Middleware for Web Services Composition", IEEE Trans. Softw. Eng. 30, 5 (May 2004), 2004, pp. 311-327.

[11] L. Hideo, V. Nakamura, A. L. V. Cunha, J. C. Estrella, M. J. Santana, and R. H. C. Santana, "A comparative analysis of algorithms for dynamic web services composition with quality of service", In Proceedings of the 19th Brazilian symposium on Multimedia and the web (WebMedia '13), ACM, New York, NY, USA, 2013, pp. 217-224.

[12] M. A. Serhani, R. Dssouli, A. Hafid, and H. Sahraoui, "A QoS Broker Based Architecture for Efficient Web Services Selection", In Proceedings of the IEEE International Conference on Web Services (ICWS '05), IEEE Computer Society, Washington, DC, USA, 2005, pp. 113-120.

[13] M. Fluegge, I. J. G. Santos, N. P. Tizzo, and E. R. M. Madeira, "Challenges and techniques on the road to dynamically compose web services", In Proceedings of the 6th international conference on Web engineering (ICWE '06), ACM, New York, NY, USA, 2006, pp. 40-47.

[14] N. Aziz, J. Byun, and Y. Park, "A QoS-Aware Performance Prediction for Self-Healing Web Service Composition", Second International Conference on Cloud and Green Computing, 2012, pp. 799-803.

[15] D. Petrova-Antonova and S. Ilieva, "DYSCO: A Platform for Dynamic QoS-Aware Web Service Composition", IADIS International Conference on Theory and Practice in Modern Computing 2012, Lisbon, Portugal, July 17-19, 2012, pp. 91-94.

[16] D. Petrova-Antonova and A. Dimov, "A QoS Driven Approach for Probability Evaluation of Web Service Compositions", 6th International Conference on Software and Data Technologies, Volume 1, Seville, Spain, 18-21 July, 2011, pp. 321-326.

[17] D. Petrova-Antonova, "Cost Dependent QoS-based Discovery of Web Services", Proceedings of International Conference on Software, Services & Semantic Technologies, September 11-12, 2010, Varna, Bulgaria, ISBN 978-954-9526-71-4, 2010, p. 152-159.

[18] S. Ran, "A model for web services discovery with QoS", SIGecom Exch. 4, 1 (March 2003), 2003, pp. 1-10.

[19] Y. Lu, Z. Gao, and K. Chen, "A Dynamic Composition Algorithm of Semantic Web Service Based on QoS", Second International Conference on Future Networks, 2010, pp. 354-356.

[20] Y. Yu, H. Ma, and M. Zhang, "A Hybrid GP-Tabu Approach to QoS-Aware Data Intensive Web Service Composition", G. Dick et al. (Eds.): SEAL 2014, LNCS 8886, 2014, pp. 106–118.

[21] Y. Xia and Y. Yang, "Web Service Composition Integrating QoS Optimization and Redundancy Removal", 20th IEEE International Conference on Web Services, 2013, pp. 203-210.

[22] Y. Liu, A. H. Ngu, and L. Z. Zeng, "QoS computation and policing in dynamic web service selection", In Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters (WWW Alt. '04), ACM, New York, NY, USA, 2004, pp. 66-73.

[23] Z. Brahmi and M.M. Gammoudi, "QoS-Aware Automatic Web Service Composition based on cooperative agents", Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2013, pp. 27-32.

[24] Z. Zheng and M. R. Lyu, 2013, "Personalized Reliability Prediction of Web Services", ACM Trans. Softw. Eng. Methodol. 22, 2, Article 12 (March 2013), 2013, pp. 1-25

[25] J. Bih, 2006, "Service oriented architecture (SOA) a new paradigm to implement dynamic e-business solutions", *Ubiquity* 2006, August, Article 4 (August 2006), 2006, pp. 1-1.

[26] V. Muthusamy, H. Jacobsen, T. Chau, A. Chan, and P. Coulthard, "SLA-driven business process management in SOA", In *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research* (CASCON '09), Patrick Martin, Anatol W. Kark, and Darlene Stewart (Eds.). IBM Corp., Riverton, NJ, USA, 2009, pp. 86-100.