A Framework for SLA-Based Cloud Services Verification and Composition

Asma Al Falasi, Mohamed Adel Serhani

Faculty of Information Technology, UAE University
Al Ain, UAE

{asma.alfalasi, serhanim}@uaeu.ac.ae

***Abstract*— Cloud computing is becoming a key element in service provision on the Internet. Businesses are seizing the availability of infrastructure as service to obtain cost effective infrastructure solutions. However, in businesses the power of services emerges from the ability to combine different services in order to obtain some value added services. Cloud services composition presents some challenges like, service discovery and real time service evaluation; which are addressed in this paper by introducing a primal framework that enables dynamic specification of SLAs, in addition to SLA-based verification and composition of services on the Cloud. The verification is used as input to the composition and consists of verifying the functional and non-functional properties of the cloud service under test. An ongoing prototype implementation will evaluate the verification scheme and prove its importance in composing and selecting services on the cloud.**

***Index Terms*— Cloud Services Composition, Genetic Algorithm, Verification, SLA, Web services.**

I. INTRODUCTION

Cloud computing is an emerging trend for the provision of IT infrastructure as services, with the potential of transforming the way of offering business services and software development to become prominent and accessible for all without the hassle of investing in expensive hardware resources nor of managing and maintaining them.

Cloud computing is of a great benefit for enterprises that need to maintain plenty of applications for many users within certain quality measures to insure reliability of business processes. However, for most enterprises a single business process is typically composed of number of tasks to be carried out by different applications or services; thus the need for a steady way of selecting quality services to perform a business process is essential.

In order for Cloud providers to supply clients with services that meet their quality constraints, they both need to negotiate the client's request and the provider's infrastructure capabilities, and then agree to certain conditions [1]. This agreement is known as Service Level Agreement (SLA), and it is what the provider conforms to in order to deliver the client with what he needs. This concept is well known in web services provision, but extended with Cloud computing with the ability to flexibly adhere to modified SLAs at run time; because of its scalable nature.

The research problem addressed in this paper is focusing on defining a framework that overcomes some Cloud-inherited issues in service composition; in order to enable dynamic specification of SLAs in addition to SLA-based composition of services on the Cloud using a declarative [2] genetic algorithm; that applies some Quality of Service constraints (QoS) to determine if services can be composed.

The rest of the paper is organized as follows: Section two provides relevant information on Cloud computing, service oriented architecture and web services composition, in order to understand the characteristics of Cloud services composition. Section 3 discusses some related work to SLA specification, and genetic algorithm implementation for QoS-based web services composition. Section 4 motivates the contribution of this paper and describes the proposed framework and its components. Section 5 explains the services' verification approach. Section 6 depicts the requirements for the proposed framework implementation. Section 7 concludes with some discussion on future work.

II. OVERVIEW OF RELEVANT TECHNOLOGIES

A. Cloud Computing

Computing on the cloud is perceived as an evident outcome of the recent expansion of the web as it grows to turn into web of services. It is defined as:" *a set of network enabled services, providing scalable, QoS guaranteed, normally personalized, inexpensive computing infrastructures on demand, which could be accessed in a simple and pervasive way."*[3] Which simply means accessing applications, services and IT infrastructure through QoS guaranteed web services. Hence, Cloud computing enables users to utilize services without having to be aware of their complexity, nor needing to acquire the knowledge and expertise to actually consume the services. Basically, cloud computing provide users with services to access hardware, software, and data.

Cloud computing is enabled by the still evolving technologies of Web and service oriented computing. It became popular nowadays because of the emergence necessity to provide complex IT infrastructure, for users to be able to manage different software requirements, and to handle the exponentially rising data size on the internet. Further, the common adoption of service oriented architecture (SOA), and web application has lend a hand to the increasing adoption of the cloud computing. SOA is considered as the underlying concept of the Cloud computing; as it enables remotely integrated services to be provided based on some specific end user requirements.

B. Service Oriented Architecture

Service Oriented Architecture (SOA) [4] is a software architecture paradigm which abstracts application logic into services, and exposes them with well-defined interfaces, and making these service publically available through discovery mechanisms [4]. When used in the context of Web applications, these services are labeled as Web services. A web service is defined as an application programming interfaces (API), that surpasses programming languages,

operating systems, network protocols, and data representation dependencies and issues [5]. SOA architecture identifies three roles:

1) *Service Provider:* is the owner of the Web Service and its hosting platform.
2) *Service Requestor:* represents the entity which makes use of the provided Web Service.
3) *Service Registry:* this is a yellow pages-like database of Web Services' descriptions. Each record in this database contains all the required information to use the listed Web Service.

In addition, there are three operations defined by SOA:

1) *Publish:* in order to be known by Web Services' requestors, an XML-based description of the services is published in a standard document construct specifically defined for that purpose called WSDL or Web Service Description Language [6].
2) *Find:* Used to locate and return the Web Service's description (i.e. WSDL document) to a requestor. It is executed in two steps in the Web Service life cycle: during design of the Web Service's client to retrieve the Web Service interfaces' description, and during runtime to retrieve the Web Service's binding and location information.
3) *Bind:* this operation is used to invoke operations from the desired Web Service.

The relationship between the roles and the operations are illustrated in Fig.1.

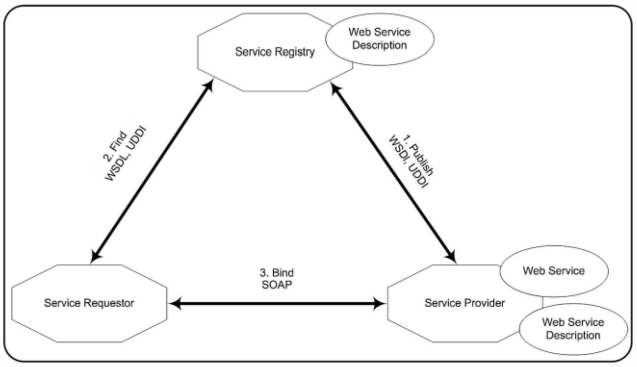

Fig. 1: Service Oriented Architecture

C. Web Services Composition

Nowadays, web services are playing a vital role in the world of businesses integration and collaboration; providing a distinct aspect to collaboration through their ability to be composed. Composition of web services is defined as a set of atomic services together with the control and data flow among the services [7]. Composing services rather than accessing a single service ensures cost effective solutions to businesses. Most organizations only implement their core business and outsource other application services over the Web via web services. Conversely, Web service composition is considered a very complex task to be handled manually by humans. Generally, the complexity is due to the following reasons [2]:

1) There exists a huge amount of web services on the web, and the number is increasing everyday, which leaves us with an enormous web services repository to dig into.
2) The ability to create and update web services on the fly forces composition systems to perceive such updates at runtime; since decisions should be made based on latest information.
3) Web services can be described by different concept models, as they are developed by different organizations.

Hence, the ability to automatically search UDDI's to select and integrate most appropriate web services at run time is important to achieve efficient service integration; as it allows businesses to construct value added services from concrete ones, thus creating new abstract services that were not previously defined. This trend has caused a significant number of research efforts on the selection and composition of Web services both in academia and in industry, putting forward various compositions' methods, techniques and algorithms that can be based on wrappers, workflows, languages, ontologies and declarations [8].

D. Genetic Algorithm

Genetic algorithm (GA) is a particular class of evolutionary algorithms that adopts techniques inspired by evolutionary biology, such as inheritance, mutation, selection, and crossover. It is defined as a search method used in computing to reach optimized solutions of search problems in many fields [9].

III. Related Work

QoS-based service composition has received significant research interest, work on specification of QoS attributes for web services can be classified based on their main objectives: for services and components description WS-Policy [10] which specifies services policies on QoS attributes and security capabilities. In addition to DAML-S [11] providing ontology-driven description of web services semantics including QoS constrains.

For SLA-centric models; the IBM Web Service Level Agreements (WSLA) [12] is used for specifying and monitoring SLAs for web services. In addition to Web Service Offerings Language (WSOL) [13] that enables multiple classes' specification of a single web services to include functional and non-functional requirements, besides access control and relationship with other services. There is also the Service Level Agreement definition language (SLAng) [14] which enables expressing of service's features and QoS constraints for SLA validation, monitoring and enforcement.

The problem of service composition based on multiple QoS criteria to select from multiple concrete services is a combinational optimization problem that is known to be an NP-Hard [15]; as it requires considerable amount of time and costs to find an optimal permutation of concrete services that matches the required QoS criteria. Using GA to find optimal web services composition has been an active research area recently. [16] explores the problem of web services heterogeneity in terms of description and interfaces and suggests a model that works on classifying and clustering the

web services components within UDDI's and semantically annotate them in order to be searchable efficiently, and then statically create different execution plans that meets the user QoS requirements in a weighted multistage graph; thus, transforming the use of GA from selecting the optimal execution plan, into the selection of the longest path of a graph. Which creates a lot of overhead that is not practically preferable; because during the time of preparing web services, designing execution plans; a single service my no longer be available or some would have their QoS attributes fluctuating.

Besides, the optimization of the web services composition problem can be provided either by defining a single objective function as in [15] and [17]; where they model a GA genome by a set of abstract services that are implemented by one or more concrete services providing the same functionality. And based on a fitness function which is composed of an aggregation of the weighted QoS attributes required by the user, they provide a single optimal workflow of the composed abstract services. Also in [18], web services composition is exemplified through a suggested improvement of the GA performance, aiming to reduce the domain range of the initial population and to constrain crossovers by incorporating the rough set theory [19].

On the other hand, an optimal solution can be based on aggregating single functions to come up with a multiple-objective function; which is more suitable for the problem of web services composition, as the composed service is required to meet multiple predefined criteria. [20] models web services components collected from available UDDIs using a scenario-based work flow which describes the tasks required to be accomplished by the composition. And in order to assess the fitness of the composition they define a multi-objective function of the commonly used metrics of time, reliability and cost as their evaluation attributes; however their approach is not applicable in the context of the Cloud since there are no UDDIs for Cloud services. Conversely [21] solves the problem of web services composition within the context of Cloud computing, using a multiple objective function that provides three optimal solutions supporting three different SLAs. In this paper we adopt their approach to compose services on the Cloud; as it is flexible enough to cater for the adaptive nature of the Cloud.

The process of Cloud services composition basically starts with the service requester defining his requirements at the business process level. The service provider then composes the service components that meet the defined requirements. In SOA this definition of requirements or SLAs -which guarantee an estimated range for required QoS attributes-, is carried out at design time of the service composition and those SLAs would remain fixed as long as the composed service is being consumed. However, when using services on the Cloud a client can modify the predefined SLAs to reflect his needs at any time; scaling up or degrading the service, because of the Cloud computing infrastructure that can provide such flexibility. Clouds provides services based on runtime QoS attributes, that are performance oriented and can change during runtime; like when the average penetration rate for CNN.com surges suddenly after the emergence of some big news, the site admin can change their SLAs to accommodate higher penetration rate. However, this is not the case with design time constraints used in SOA.

Besides, currently Cloud providers offer SLAs that includes QoS measures of their services capabilities based on historical data of how their services have previously performed; which is not a reliable practice. Since historical QoS measures may not be reflecting the service actual performance; web services performance can vary between different invocations as it can be influenced by the network or server performance. What is more, Cloud providers may offer SLAs that claim unrealistic or incorrect QoS measures; because there are no means of SLAs validation procedures to insure consistency of provided SLAs. Additionally, services on the Cloud are distinct from SOA services in a sense that there are no repositories maintaining information on Cloud services for clients to search and pick from. Thus there is a need to formally specify clients' requirements to allow automatic discovery of services; allowing clients to compare and evaluates different SLAs provided from different Cloud providers.

In this work, a primal Cloud services composition framework is introduced to address all aspects of the service composition process; starting from the dynamic SLAs negotiation, in addition to real-time services verification and composition, up to SLA agreement.

IV. Framework for SLA-Based Cloud Composition

Since SLAs need to be discussed and settled on between the service requestor and the provider before enforcing them on the Cloud infrastructure capabilities, the proposed framework puts forward an efficient mode of communication between the two parties, so that dynamic SLAs modification can be possible. Also, the framework addresses the issue of SLAs reliability and the conformity of web services QoS measures through the use of a dedicated third-party broker for real-time testing and composition of web services on the Cloud. The framework is based on formally specifying the service level objectives (SLOs) required by the client, as well as formally specifying the services' performance capabilities of the Cloud provider, SLAs. Having these SLAs properly defined helps providing better response to SLA changes. Fig.2 illustrates the proposed framework, in an abstract level, the framework enables Cloud clients to easily search through a repository of Cloud services provider, and to specify their required QoS measures. The Cloud providers will be able to lookup their web services for service candidates that meet the clients' requirements, and then communicate a list of candidates service APIs to a trusted composition broker, who will carry on some verification techniques to validate the QoS measures of the candidate services. Candidate services that pass the verification tests are then used to come up with an optimal Cloud services composed solution that meets the client's QoS. The resultant optimal SLAs are negotiated back to the client to either agree on, select alternative solution or simply try out another Cloud provider. This framework adds a valuable improvement to the provision of Cloud services; as clients

would be able to obtain services that have been tested and are optimized to meet their requirements. Besides, the necessary process of monitoring the Cloud services QoS measures is delegated to a trusted third-party that assures real-time QoS measures of the service; hence reducing the load on the Cloud provider.

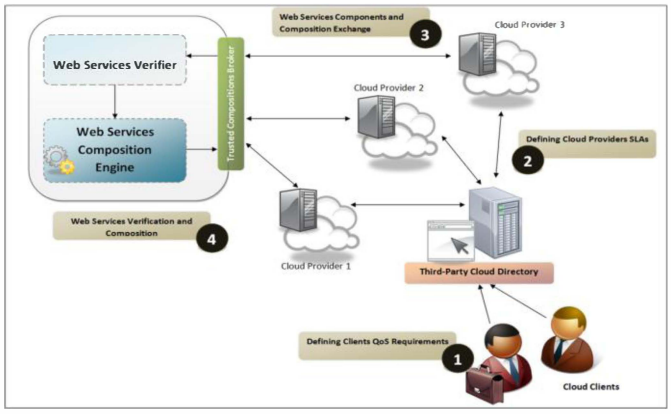

Fig. 2: Framework for SLA-Based Cloud Composition

A. *Framework Components*

Besides the Cloud service client, the framework is composed of three main components: a third-party cloud directory, cloud providers, and a trusted composition broker:

1) *A Third-Party Cloud Directory:* A trusted third-party plays the role of Cloud yellow pages; since there are plenty of Cloud providers offering their infrastructures as services, and with the absence of common Cloud repository; this directory will play an intermediary role between Cloud service requestors and providers. A Cloud provider has to sign up with the directory to promote its services, while a client can search the entire directory; providing his Service Level Objectives (SLOs), and be able to sign in and initiate an SLA negotiation with a selected Cloud provider. The Cloud directory utilizes the WSOL [22] to formalize the given SLOs and SLAs, which enables proper description of Cloud services QoS attributes; as it supports specifications in a multi-level manner. This unified specification of SLOs and SLAs enables an efficient and automated mapping among client's requirements and provider's capabilities. On the other hand, the directory assures secure access for the two parties; as it is considered the platform of SLA negotiations and agreements.

2) *The Cloud providers:* Cloud providers expose their infrastructure as web services for clients to explore and consume, like for example Amazon EC2, Salesforce, and Microsoft. They offer different service plans for clients; every plan is an abstract service that needs a composition of concrete services to carry out the different tasks of the offered plan. For instant, a web hosting service is actually an abstract service realized by a database service, a processing service and a memory management service. And every service plan is provided at three different levels with different QoS measures; *Silver*, *Gold* and *Platinum*; in order to provide client with service flexibility and preferences. When a client initiate an SLA negotiation with the Cloud provider directory, the Cloud provider looks up for candidate concrete services that realizes the service plan requested by the client and are able to meet his SLOs. After that the provider invokes the service of a composition broker including a list of candidate services APIs for a trusted composition broker, to come up with the optimal service SLA within the client's requirements.

3) *A Trusted Composition Broker:* A trusted third-party plays the role of a service composition broker. This broker exposes its functionalities as web services to be invoked by Cloud providers in order to propose optimal SLAs for a certain requested service plan. Two modules work together to achieve this goal:

a. The Web Services Verifier:
This module is used to test QoS measures of a given set concrete services; (database, processing and memory management) as in the previous hosting example. It basically invokes the services through provided APIs and run some conformity testing to conclude the actual performance metrics of the services. The module compares the test results with the required SLAs; and services which pass the conformity testing are then fed to the composition engine, while others are discarded. This module insures that only candidate services that perform well at the moment are included in the optimal solution.

b. The Composition Engine:
This engine aims to come up with three-level SLAs that meet the clients SLOs through adaption of E^3-MOGA [10]; a framework that defines a service composition model, and implements a multi objective genetic algorithm to find the optimal Cloud services composition based on multiple conflicting predefined SLAs. Given a certain business process definition from the service requestor, in addition to a set of concrete services that can implement the different tasks of the defined business process from the Cloud provider; the E^3 model can decide on what concrete services to include in the composition, and how many instances of each concrete services is needed in order to satisfy the client's SLA.

The model assumes that each concrete service has three QoS attributes to describe its performance capabilities: throughput, response and cost. Throughput is defined as the number of successfully served requests per second. Response time is the time space between sending a request and initiating the response in milliseconds, while the cost is the fees charged for using the service per hour. The concretes services used in the composition model are of similar functionalities but with different performance capabilities; which enables the Cloud provider to substitute any concrete service in the composition at any time with another service in order to meet a client's modification in the SLA. Computing the overall QoS measure of a service composition involves aggregating QoS measures of all concrete services in the composition. And since services on the Cloud have multiple

performance levels; the model finds the expected QoS measures of every single concrete service as an aggregation of probabilities of its performance levels. Based on the structure of the composition and the user defined QoS attributes in the SLA the model applies different aggregation functions to find the overall QoS measures of the composition; like for example if the composition is a sequential; meaning the concrete services are executed after each others; the aggregation function of the throughput will be the minimal throughput of the composed services, while the response time and cost will be the summation of all services' response time and cost. The QoS composition model is used to represent individuals in the multiple objective GA; every individual is composed of three genes each presenting a possible composition solution of different SLA, *platinum, gold* and *silver*; where for every level a set of overall QoS measures is set as its objective value. The algorithm uses a fitness function that is derived from an individual objective value and is based on evaluating two measures; *domination rank:* how an individual outperforms other individuals, and *Density:* how many other individuals provide similar objective values. The algorithm then performs its GA operations; where individuals evolve their genes over generations, and the process is repeated until maximum number of generation is reached and a set of feasible alternative solutions are given. The resultant optimized SLAs are communicated back to the Cloud provider who communicates it to the requested client via the Cloud directory. The client will have the option either to agree on the given SLAs and start consuming the services, or if the given SLAs were not satisfactory he can modify his SLOs again or simply move on to another Cloud provider; starting the negotiation process all over again in order to come up with new SLAs. It is worth mentioning that this framework enables client to dynamically alter their SLOs at run time while the service is being consumed to cope with certain needs or event.

4) *An Example of SLA*

Fig.3 describes an example of SLA that defines the main QoS along with their threshold values agreed up on selection of cloud services. It also, defines the period of service provision, the cost of using the service, and the possibility actions that should be taken if QoS provision is frequently violated.

V. SERVICE VERIFICATION APPROACH

Fig. 4, describes the service verification scheme used to verify the Cloud services and the main components involved in the verification process. Prior to selection and invocations of Service on the Cloud, we propose to verify the conformance of the claimed service's functional and non-functional properties (QoS). The verifier relies on the WSDL of the service and the pre-agreed SLA to starts the verification (Operation 1). These documents are parsed (Operation 2), and extracted relevant information is stored in the database (Operation 3.1).

```
<Cloud WS-name: name >
   <Class: GOLD>
      // QoS properties description and their thresholds
         <QoS>
      Reputation = 5
      RTmin = 8ms //minimum value of response time
      RTmax = 10ms //maximum value of response time
      Cost= "$0.1"
      Min Availability= 90%
         </QoS>
   </Class: GOLD>
<Class: SILVER>
         <QoS>
      Reputation = NULL
      RTmin = 15ms //minimum value of response time
      RTmax = 25ms //maximum value of response time
      Cost= "$0.05"
      Min Availability= 80%
         </QoS>
</Class: SILVER>
....
<Terms>
Terms and conditions of Service provision are included here. And it describes the necessary actions if the QoS is violated. It can take the form of assertion.
</Terms>
<Cloud WS-name: name >
```

Fig. 3: Example of SLA-Description

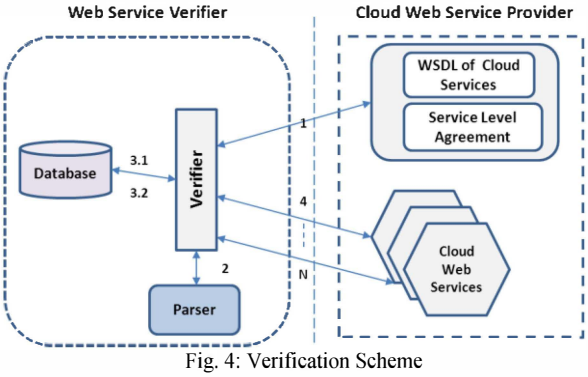

Fig. 4: Verification Scheme

Using this information the Web service verifier conducts verification in two phases: in the first phase the verifier generates test cases to verify service operations described in the WSDL by passing varying operation parameters (Operations 4 to *N*), *N* is the number of requests/responses exchanged between the service verifier and the Cloud web service for the purpose of verification. *N* varies based on many criteria for instance, the verified QoS property, and the number of operations available in the service. In the second phase the verifier generates QoS verification test cases using the information depicted in the SLA (Operation 4 to *N*). Once the verification cycle is completed the verification report is stored in the database (Operation 3.1). For example, to verify the availability of a given service the verifier generate varying number of requests over different period of time and measure the availability of the service, which is the percentage of received responses to the total of generated request to the Web service. Fig. 4 describes the set of steps of verifying both functional and non-functional properties of Cloud web services. The Cloud web services are granted verification if they passed the verification verdicts, However, they are rejected if they fail the verification test cases. The verification is used as input for the composition and only services that passed the verification are selected to be candidates for Cloud web service composition. The verification test cases are generated for each operation of the Cloud web services; it can

include also a series of test cases of the same operation but with different input parameters.

Using the verification data stored in the database, a verification report can be generated to summarize the overall verification of a given Cloud web services. The verification decision states clearly that the Cloud services are conforming to what have been claimed in the WSDL, and the SLA.

VI. Framework Implementation

The realization of the proposed framework is straightforward and it provokes minimum overhead on the Cloud service provisioning; since the communication is performed using web service, which uses lightweight XML messages. Besides the introduction of the verifier and composition broker will take off the load of monitoring services QoS measures from the Cloud provider. All what is required to implement the framework is: an online portal for the Cloud directory which enables clients and service providers' registration, communication and formalization of requirements, in addition to a verification and composition modules at the broker side for Cloud providers to invoke and utilize. The framework implementation requirements can be specified as follows:

1) *The Cloud directory:* Mediates between the Cloud service provider and requestor and it is composed of:
   a. Web server: needs to enforce secure access measures.
   b. Database management system: to maintain Cloud providers information
   c. Front-end application: for users to interact with.

2) *The composition broker:* A couple of web services that are invoked to verify and propose optimal SLAs for a certain requested Cloud service plan is made up of:
   a. Web server: for the modules to run on
   b. Back-end application: contains the programs to run the verification and composition processes.
   c. Published API for the verification module.

VII. Conclusion and Future Work

This paper addresses the issue of Cloud services composition; it discusses the distinctive aspects Cloud services have over traditional SOA services, and how this characteristics influence the process of the Cloud services compositions. Additionally, the paper introduces a framework for Cloud service composition that aims to overcome the issues caused by the open and flexible nature of Cloud services; by incorporating some trusted third-party entities to govern and optimize the service composition process.

For future work we will investigate the proper Cloud services verification technique to be implemented by the composition broker in order to achieve feasible and efficient conformity testing. Additionally we will be working on implementing the framework and setting up an appropriate environment to simulate the Cloud service composition and analyze the framework efficiency.

VIII. References

[1] I. Foster, Y. Zhao; I. Raicu, S. Lu. "Cloud Computing and Grid Computing 360-Degree Compared," *Grid Computing Environments Workshop. GCE '08* , vol., no., pp.1-10, 12-16 Nov. 2008

[2] S. Dustdar, W. Schreiner. 'A survey on web services composition', *International Journal of Web and Grid Services*, 2005. Vol. 1, No. 1, pp.1–30.

[3] L. Wang, et al. "Cloud Computing: a Perspective Study". *New Generation Computing*, 2010. Vol 28, pages 137-146.

[4] "Reference Model for Service Oriented Architecture V1.0." *OASIS*. 12 Oct. 2006. Web. 31 May 2010. http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html

[5] "Web Services Architecture." *World Wide Web Consortium (W3C)*. Web. 12 November 2010. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[6] "Web Service Definition Language (WSDL)." *World Wide Web Consortium (W3C)*. Web. 21 November 2010. http://www.w3.org/TR/wsdl

[7] D. B. Claro, P. Albers, and J. Hao. Selecting Web Services for Optimal Composition. In *IEEE Int'l Conf. on Web Services, Workshop on Semantic and Dynamic Web Processes*, July 2005.

[8] A. Alamri, M. Eid, and A. E. Saddik, "Classification of the state-of the art dynamic web services composition techniques," *Int. J. Web and Grid Services*, vol. 2, no. 2, pp. 148–166, 2006.

[9] D. Stauffer. "A Survey of Genetic Algorithms." *Annual Reviews of Computational Physics I|I*. Singapore [etc.: World Scientific, 1995. 87-118.

[10] "Web Services Policy Working Group." *World Wide Web Consortium (W3C)*. Web. 8 Jan. 2011. http://www.w3.org/2002/ws/policy/

[11] DAML-S Coalition, DAMLS-S, "Web Service Description for the Semantic Web", *In Proceeding of the International Semantic Web Conference*, June 2002.

[12] A. Keller and H. Ludwing, "The WSLA framework: Specifying and Monitoring Service Level Agreements for Web Services", *IBM Research Report*, May 2002.

[13] V. Tosic, K. Patel, and B. Pgurek. Wsol - web service offering language. In *Proc. of CAiSE International Workshop, Web Services, E-Business, and the Semantic Web (WES)*, LNCS 2512, pages 57–67. Springer, 2002.

[14] D. Lamanna, J. Skene, and W. Emmerich, "SLAng: A language for service level agreements," *In Proc. of the 9th IEEE Workshop on Future Trends in Distributed Computing Systems*. IEEE Computer Society Press, 2003, pp. 100–106.

[15] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani. An Approach for QoS-aware Service Composition based on Genetic Algorithms. *In Genetic and Evolutionary Computation Conference*, June 2005.

[16] Y. Gao, B. Zhang, J. Na, L. Yang, Y. Dai, and Q. Gong. Optimal Selection ofWeb Services with End-to-End Constraints. In *IEEE Int'l Conf. on Grid and Cooperative Computing*, October

[17] M. C. Jaeger and G. Mühl. QoS-based Selection of Services: The Implementation of a Genetic Algorithm. In *Conference on Communication in Distributed Systems, Workshop on Service-Oriented Architectures and Service-Oriented Computing*, March 2007.

[18] W. Liang, C. Huang. "The Generic Genetic Algorithm Incorporates with Rough Set Theory – An Application of the Web Services Composition." *Expert Systems with Application* 36 (2009): 5549-556.

[19] W. Ziarko. "Discovery through rough set theory". *Communications of the ACM*, 42(11):54-57, November 1999.

[20] W. Chang, C.Wu, and C. Chang. Optimizing DynamicWeb Service Component Composition by Using Evolutionary Algorithms. In *IEEE/ACM Int'l Conf. on Web Intelligence*, Septemer 2005.

[21] H. Wada, P. Champrasert, J. Suzuki, and K. Oba. Multiobjective Optimization of SLA-aware Service Composition. *In IEEE Workshop on Methodologies for Non-functional Properties in Services Computing*, July 2008.

[22] V. Tosic, K. Patel, and B. Pgurek. Wsol - web service offering language. In *Proc. of CAiSE International Workshop, Web Services, E-Business, and the Semantic Web (WES)*, LNCS 2512, pages 57–67. Springer, 2002.