

# QoS composition of Services for Data-Intensive Application

Antonio Bucchiarone

IMT Lucca Institute for Advanced Studies  
Piazza S. Ponziano 6, 55100 Lucca-Italy  
antonio.bucchiarone@imtlucca.it

Luigi Presti

IBM Software Group, Tivoli Rome Lab  
Via Sciangai 53, 00144 Rome (Italy)  
luigi.presti@it.ibm.com

**Abstract**—Service-Oriented Computing (SOC) is a promising means to integrate heterogeneous systems. Services from different providers can be integrated into a composite service regardless of their locations, platforms, and/or execution speeds to implement complex business processes. In this paper we propose an approach for the services composition that takes into account the QoS that the composite system must satisfy. After that we design the service selection algorithm used to construct the optimal composite service with the objective to maximize the values of the well-defined QoS characteristics. In this paper we are interested on the Data-Intensive applications where the QoS attributes are very important for the reliability and performance of these systems.

## I. INTRODUCTION

Service Oriented Computing (SOC) utilizes services as the constructs to support the development of rapid, low-cost and easy composition of distributed applications. Services are autonomous, platform-independent computational entities that can be described, published, discovered, and dynamically assembled for developing distributed and evolvable systems.

This "service-oriented" approach is independent of specific programming languages or operating systems. It allows organizations to expose their core competencies programmatically over the internet or a variety of networks, e.g., UMTS, XSDL, Bluetooth, etc., using standard (XML-based) languages and protocols, and implementing a self-describing interface. The visionary promise of services technologies is a world of cooperating services where application components are assembled with little effort into a network of services that can be loosely coupled to create dynamic business processes and agile applications that span organizations and computing platforms [1]. The key of this concept is the service-oriented architecture (SOA). SOA is a logical way of designing a software system to provide services to either end-user applications or to other services distributed in a network, via published and discoverable interfaces. A well-constructed, standard-based SOA can empower a business environment with a flexible infrastructure and processing environment. Efficiencies in the design, implementation, and operation of SOA-based systems can allow organizations to adapt far more readily to a changing environment.

Currently, the service oriented software development paradigm is mostly used for internal integration and not so many services are exposed to outside world. This is partly caused by the extant difficulties to use (web-) service technology, and by the lack of trust in using someone else's services. More and more important web companies are making web services available through which third parties can add value. Examples include Amazon and Google. With the Google Web APIs service, software developers can query more than 8 billion web pages directly from their own computer programs, and develop their own applications based on the queries<sup>1</sup>. The aim of the research carried out in the area of services is for companies to have the software infrastructure and tools to easily create and deploy new types of e-services. There is a big gap between the current state of the art in service research and the future vision where services are all around us providing useful assistance. In order to bridge that gap the following questions and issues have to be resolved:

- 1) How a service is described to users and application developers that want to use a service as it is, or want to integrate that service with others to develop a new service (service description).
- 2) If we want to compose a new service on the basis of existing services, how can we do that (service composition), especially on the fly, according to the QoS needs of a user? (QoS dynamic composition).

In order to resolve these questions we have defined a QoS approach for the services composition. Each provider describes a service with QoS informations (i.e., time and cost) and put this on the net. A front-end application, that we will see in Section 3, composes these services in order to satisfy the client requests. For these reasons we have defined a front-end application and the QoS services composition algorithm. In this work we have concentrated the attention to Web Services and Grid Services but other kinds of services (i.e., P2P) could be considered.

The rest of this paper is organized as follows. Section 2 presents a background of the main arguments that are involved

<sup>1</sup><http://www.google.com/apis/>

in this paper. Section 3 provides the QoS model for services composition and the QoS services composition algorithm. Section 4 presents some related work. The paper ends with future work.

## II. BACKGROUND

### A. What are services?

The word "service" means different things to different people. A general definition is that a service is useful labour that does not produce a tangible commodity. Within the context of this paper we are interested in service that are mainly delivered by software, i.e., applications that provide computational resources on request. We may categorize software based services according to their functional use:

- *Context adaptive and intelligent user services*: These are "ambient intelligence" type of services providing users the ability to access the services they need, anywhere from any device.
- *Information services*: services that provide (personalized) information, for instance by comparing, classifying, or otherwise adding value to separate information sources.
- *Intermediary services*: services that help in finding appropriate services, for instance search services.
- *Location-based services (LBS)*: is a family of services that depend on the knowledge of the geographic location of mobile stations. The driving force behind the development of LBS is accelerating market demand for more advanced, innovative personalized services. Increasingly connected consumers are becoming dependent on LBS to support busy lifestyles and businesses are discovering their flexibility and efficiency.

But quite often also a classification is made according to the technical protocols or business model used:

- *Web services*: A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL [2]). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages [3], typically conveyed using HTTP [4] with an XML serialization in conjunction with other Web-related standards <sup>2</sup>.
- *Grid Services*: Grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed "autonomous" resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements [5]. Grids aim at exploiting synergies that result from cooperation - ability to share and aggregate distributed computational capabilities and deliver them as service <sup>3</sup>.

Without providing a generic definition of services, we can say that a service provides some useful functionality through

a well defined interface, and that it is possible to combine services to produce useful 'composite' services. Two new roles are introduced when talking about services: the provider and consumer. A provider makes a service available, and a consumer uses it without worrying about its internal details. Consumers can at the same time be providers, when they make available a composite service based on already existing services. Future services will be robust, simple and easy to use with high perceived value and capable of being adaptive to situation and environment. They will provide "intelligent" support while leaving the user in control and being simply accessible through different infrastructures and devices.

### B. Services Composition

The service composition defines rules and functionalities for the aggregation of multiple services into a single composite service. Resulting composite services may be used by service as basic services in further service compositions or may be offered as complete applications/solutions to service clients. Web services provide interactions and collaborations implementation of business process within and across organizational boundaries. Web services technology is based on a series of related standard protocols to describe and interact with services [6]. SOAP [3] is an XML based protocol for information exchange designed to allow communication in a decentralized, distributed environment, which supports web service's binding for invocation purpose. WSDL [2] is an XML format for describing network services based on a standard message layer like SOAP, which supports the definition of Web Services. UDDI [7] is the definition of a set of services supporting the description and discovery of the web services, business, organization and other web services providers and the technical interface that may be used to access those services, which supports web service's advertisement to the community of potential users. Grid [5] provides users the ability to utilize the power of a large variety of heterogeneous, distributed resources, computing resources, data storage systems, instruments etc. Numerous grid applications, tools, and systems have been developed over the past years, which support the sharing and coordinated use of different resources in dynamic virtual organizations. Grid computing is becoming a new computing infrastructure for scientific and cooperation and a mainstream technology for large-scale resources sharing and distributed system integration. The Grid and Web Services are converging in the WSRF (The Web Service-Resource Framework) [8] that is a series of specifications for performing grid computing on top of web services is proposed, so the grid will not only provides the computing as resources sharing pools, but will also supports the electronic business and enterprise applications integration with the composition of grid services.

In the WSs world the terms "orchestration" and "choreography" have been widely used to describe business interaction protocols comprising collaborating services. Orchestration describes how services can interact with each other at the message level, including the business

<sup>2</sup><http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice>

<sup>3</sup><http://www.gridcomputing.com/gridfaq.html>

execution order of the interactions from the perspective and under control of a single endpoint. Orchestration refers to an executable business process that may result in a long-lived, transactional, multi-step process model. With orchestration, the business process interactions are always controlled from the (private) perspective of one of the business parties involved in the process.

Choreography is typically associated with the public (globally visible) message exchanges, rules of interaction and agreements that occur between multiple business process endpoints, rather than a specific business process that is executed by a single party. Choreography is more collaborative in nature than orchestration. It is described from the perspectives of all parties (common view), and defines the complementary observable behavior between participants. Choreography offers a means by which the rules of participation for collaboration can be clearly defined and agreed to, jointly. Choreography tracks the sequence of messages that may involve multiple parties and multiple sources, including customers, suppliers, and partners, where each party involved in the process describes the part they play in the interaction and no party "owns" the conversation. Orchestration is targeted by a family of XML-based process standard definition languages. The most representative is the Business Process Execution Language for Web Services (BPEL4WS) [9]. Services choreography is targeted by Web Services Choreography Description Language (WS-CDL) [10], which specifies the common observable behavior of all participants engaged in business collaboration.

### C. Services Level Agreement

A Service Level Agreement (SLA) [11] complements a service definition. While a service description, for example, using WSDL [2] defines the service interface relationship between a service and its using application, the SLA defines the agreed performance characteristics and the way to evaluate and measure them. SLA complements service description languages. Service descriptions are input to the design and implementation of the service system and the client application using its service. The SLA provides input to the measurement and management system of an organization that checks and manages an organization's compliance with a SLA. The instrumentation of service and service-using application can be instrumented to gain measurements for the evaluation of the SLA. Both service provider and service customer may run their own instrumentation and measurement and management systems. Each organization may access measured metrics from various sources, such as server-side metrics from the provider and client-side metrics from the customer. This allows parties to determine both a service's performance within a service provider's domain and its performance as experienced by a user. The relationship between a Service Level Agreement and the measurement and management system that deals with it is more dynamic than the relationship between a service description and its implementing service.

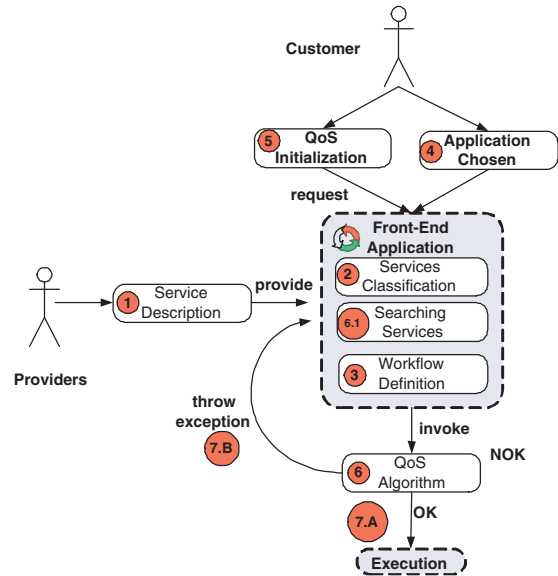


Fig. 1. Front-End Application for QoS Service Composition.

## III. QOS MODEL FOR SERVICE COMPOSITION

### A. A Front-End application

Before entering in the details of the QoS service composition algorithm we want to explain the innovation idea of this paper. In our mind we have a system that is able to put together different services from various providers (i.e., companies distributed on the net) in order to construct and execute a data-intensive application (shown in figure 1). In the following we explain which are the actors and the main functionalities of this application.

The main actors of the application are:

- **Providers:** are the providers of the services (i.e. companies distributed on the net).
- **Customer:** is the client of the application that wants to execute a new application that satisfies some QoS characteristics.
- **Front-End Application:** is a system that from the provided services (i.e., companies) realizes the application with the aid of the QoS Service Composition algorithm and other functionalities as *Service Classification*, *Searching Services* and *Workflow Definition*.

The main functionalities are defined in the following and with a precise order that they must be invoked (i.e., figure 1) moreover for each of these we explain the actors involved and what it does.

- 1) *Service Description:* It is used for the service description with QoS parameters, for the Service Classification and in order to guarantee the SLA of the providers. The main actor involved is the Provider.
- 2) *Service Classification:* The Front-end application subdivides the services supplied from the providers in various classes based on their description and QoS attributes.
- 3) *Workflow Definition:* The Front-end application describes the services composition structure. It

a meta-workflow (i.e., Data-Flow, Or and Pipeline) that will be instantiated with real services when the application chosen from the customer is executed.

- 4) *Application Chosen*: The Customer chooses the kind of application (data mining, physics, satellite data processing, astronomy, travelling, tourism, etc) that he wants to develop from a set of possible applications available in the front-end list.
- 5) *QoS Initialization*: The customer inserts two QoS parameters: *max-cost*: maximum cost that the services composition (i.e. the application) must have and *min-perf*: minimum required performance, measured by the medium departure time of each result in the last node of the workflow ( $T_p$ ).
- 6) *QoS Algorithm*: It is executed from the Front-end application. It is used for the QoS Services Composition and takes in input the workflow definition and QoS parameters. The output is the workflow in which the services are instantiated, that is the final application that can be executed and satisfies the customer input, according to the services providers SLAs.
- 7) *Searching Services*: It is used from the Front-End application in order to search a service inside the classes based on the QoS parameters based on the parameters supplied from the customer and the application that must be developed.
- 8) *Execution*: If there are no problems the application will be executed otherwise
- 9) *Throw Exception*: the Front-End application raises an exception.

## B. Queuing Model

The queuing model that we consider is a classical model described in [12]. It is characterized by:

- *The arrival process of customers*: Usually we assume that the inter-arrival times are independent and have a common distribution. In many practical situations customers arrive according to a Poisson stream (i.e. exponential inter-arrival times). Customers may arrive one by one, or in batches. An example of batch arrivals is the customs once at the border where travel documents of bus passengers have to be checked.
- *The service times*: Usually we assume that the service times are independent and identically distributed, and that they are independent of the inter-arrival times. For example, the service times can be deterministic or exponentially distributed. It can also occur that service times are dependent of the queue length. For example, the processing rates of the machines in a production system can be increased once the number of jobs waiting to be processed becomes too large.
- *The service discipline*: Customers can be served one by one or in batches. We have many possibilities for the order in which they enter service. We mention:
  - 1) first in first out, i.e. in order of arrival;
  - 2) random order;

- 3) last in first out (e.g. in a computer stack or a shunt buffer in a production line);
  - 4) processor sharing (in computers that equally divide their processing power over all jobs in the system).
- *The queue occupation rate or server utilization*: it is the factor that identifies the congestion degree of the service.

In this paper we consider that each service receives flows from one or more queues in income in data-flow way. This allows to manage data-intensive applications. A key factor is to guarantee that in the composition the server utilization parameter  $\rho$  must be  $\rho = T_s/T_a < 1$ , where  $T_s$  is the medium time of answer (or service time), while  $T_a$  is the medium inter-arrival time. From the queuing theory [12] if  $\rho$  is near to 1, the data-flow in input becomes infinite with serious consequences on the answer time of the service. For this reason we have to guarantee that  $\rho < 1$ .

## C. Services Composition Flow Models

In our approach we define four types of services (shown in figure 2)

- *Beginning application service* is the service that starts the services composition.
- *One-queue service* is a service that has a single data queue in input.
- *Multi-queue service* is a service that has a multi-queue in input.
- *Final application service* is the last service of the workflow.

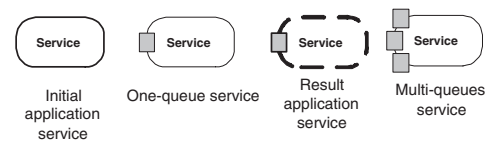


Fig. 2. Types of Services.

QoS management covers a wide range of issues to match the needs of service requesters with those of the service providers. QoS has been a major area of interest in communication networks, real-time computing and multimedia systems. For services in general (i.e. Web Services, Grid Services, etc..), QoS guarantee and enhancement have started to receive great attention. In our approach, we consider three quality attributes as part of the service parameters (shown in figure 3). Since our goal is to build an application from an automatic composition of services, the QoS attributes that we consider must be intuitive to understand and easy to measure. In a typical customer/provider scenario a customer demands and pays for a certain Quality of Service (QoS) laid down in so-called Service Level Agreements (SLAs). In order to observe the quality of delivered services it is necessary to negotiate QoS parameters as well as means for the measurement and evaluation of these parameters. For this reason in our approach, each service is identified from a cost of the service  $C$ , the service-class and the service-time  $T$ . In the table in figure 3 we can see the detailed description and the value of each QoS Service parameter.

QoS parameters	Description	Value
Cost (C)	$Service\_cost$ : the cost for executing the service; $Transmission\_cost$ : the cost for transmitting result data from a server to a requester; $Level$ : i.e., service level or parallelism degree	$C = Csr + Ctr(n) + Cleve(l)$ $Csr$ : service cost; $Ctr$ : transmission cost; $Cleve$ : level cost; $l$ : service level, parallelism degree or speed. $n$ : the network
Service-class	It identifies the class of each service. Each service in a class provide the same functionality.	The Service class can be mined by the service description of the provider.
Time (T)	It includes the time interval between when a user requests the service and when the data go out and the $level$ of the service.	Service time is specified by service provider SLA. $T = Tsr + Tlev(l)$ $T_{sr}$ : service time; $l$ : service level, parallelism degree or speed

Fig. 3. QoS Parameters.

The QoS attributes of a services composition are decided by the QoS attributes of individual services and their composition relationships. There are different ways that individual services can be integrated to form a services composition. We define three basic relationships that are (1) Data-Flow, (2) OR and (3) Pipeline that we explain in the following.

- **Data-Flow:** When a service generates more than one output each time. For this reason each service can have one or more input queues (One or Multi-queues service in Fig. 2).
- **OR (Choice):** In this case a service gives in output the data. These data can go in input at a service that we can choose in the Class of services, each branch of the OR has associated an execution probability (shown in Fig. 4).
- **Pipeline:** If the composite service has only one execution path, it's considered as a pipeline.

In this paper we suppose that more than one execution path can exist in fact we consider that the composite service is structured as a DAG (shown in Fig. 4).

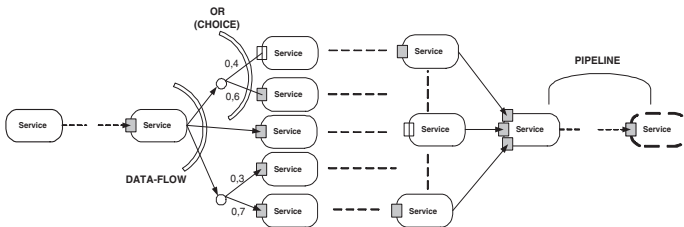


Fig. 4. Services Composition Flow Model.

#### D. QoS Service Composition Algorithm

In this section we introduce the QoS service composition algorithm used by the application developer in order to compose services that have QoS attributes. In the following we define the input the output and the pseudo-code of the algorithm. After the pseudo-code we explain each single function that is involved in the algorithm for better understand it.

- **Input:** The user that want to develop a new application chooses it in the front-end. This application is described by a workflow that defines a services composition. Moreover these services have a certain type (or class).
- **Output:** The algorithm gives in output a workflow where the services are instantiated and the QoS attributes defined by the customer are satisfied.

#### Algorithm 1 QoS-Composition\_Algorithm

**Require:** *workflow*, *MaxCost*, *MinPerf*

**Ensure:** *iworkflow*

```

1: Init(iworkflow);
2: Cost = 0;
3: while iworkflow.isNotCompleted() do
4:   Class = extract(iworkflow);
5:   Ta = computeTa(Class, iworkflow);
6:   service = choice(Class, Ta);
7:   compose(service, Class, iworkflow);
8:   for all e : outedges(Class, iworkflow) do
9:     assign(e, Ta);
10:  end for
11:  Cost = Cost + service.price;
12:  if Cost > MaxCost then
13:    throw exception;
14:  end if
15: end while

```

- *Init*: It sets the *iworkflow* and sets the *Ta* for the first service in the composition. *iworkflow* will be the final workflow.
- *extract*: It extracts from *iworkflow* a class of services. These services have the  $Ta_i$  in input already calculated.
- *computeTa*: From queues theory it calculates  $Ta$  from the set of  $Ta_i$ . For each queue in input the algorithm calculates the  $Ta$  that is  $Ta = 1 / \sum_{i=1}^n (1/Ta_i)$ . The global  $Ta$  is the maximum of the  $Ta$  of each queue in input. For example if we have a *one-queue service* (i.e., figure 2  $Ta = 1 / \sum_{i=1}^n (1/Ta_i)$ ).
- *choice*: It chooses a service with less cost. It belongs to the services class and must have time  $T_s < T_a$ .
- *compose*: It updates *iworkflow* with a new service. Moreover the value of  $Tp$  is equal to the global  $Ta$  previously calculated because  $T_s < T_a$  and since that  $Tp = \max(T_s, T_a)$  we have  $Tp = T_a$ .
- *assign*: It assigns at each edges the  $Ta_i$ . It depends of the composition typology, if we have a *data-flow* each edge in output will have the same value (i.e.,  $Tp$ ), else if we have an *OR* then  $Ta_i = Tp / prob_i$  where  $prob_i$  is the probability that the edge  $i$  is executed.

#### IV. RELATED WORK

Many projects have studied service composition (or workflow). The SWORD project [13] provides a simple and efficient mechanism for Web service composition. It uses a rule-based expert system to check whether a

service can be realized by existing services and to generate the execution plan. SWORD performs the planning at the composition time, not at run time. The advantage of offline planning is predictability and efficiency since no overhead is incurred at run time. However, the dynamic nature of Web services may cause an offline plan useless or inefficient since a pre-selected services may no longer be available or some new powerful services may become available. In VanderMerr et al [14], authors describe a framework, called FUSION, for dynamic Web service composition and automatic execution. They stress on automatically generating an optimal execution plan from the abstract requirements that a user may specify, executing according to the plan, verifying the result against a user's stated satisfaction criteria and, finally, initiating the appropriate recovery procedures in case of satisfaction failure. The satisfaction criteria are defined by users, which are used to measure whether the actual result is acceptable or not. Neither SWORD nor FUSION addresses QoS issues such as service latency, availability, reliability, that are critical to dynamic services composition and determine the performance of the whole system. Patel, Supekar and Lee [15] propose a QoS-oriented framework WebQ for adaptive management of Web services based workflows. WebQ conducts the adaptive selection process and simultaneously provides binding and execution of Web services for the underlying workflow. Their proposed QoS selection criterion only considers service load and makes only local decisions. None of the previously mentioned projects consider the QoS service composition. Zeng et al [16] gives a quality driven approach to select component services during execution of a composite service. They consider multiple QoS criteria such as a price, duration, reliability and take into account of global constraints. The linear programming method is used to solve the service selection problem, which is usually too complex for run-time decisions. Shankar et al [17] give an end-to-end QoS model and management architecture for complex distributed real-time system. In their model, a system contains several dependent components (tasks), the outputs of some tasks are the inputs of others and the output quality of every task depend on the the input quality. QoS requirements are end-to-end in the user's point of view. They focus on how to do the service establishment in this architecture and give the QoS negotiation and establishment protocol. EFlow [18] is a platform for the specification, enactment and management of composite services. EFlow uses a static workflow generation method. A composite service is modeled by a graph that defines the order of executing among the nodes in the process. The graph is created manually but it can be updated dynamically. The graph may include service, decision and event nodes. Service nodes represent the invocation of an atomic or composite service, decision specify the alternatives and rules controlling the execution flow, and event nodes enable services processes to send and receive several types of events. Relative limited research about the grid services composition. GSFL (Grid Service Flow Language) [19] analyzes existing technologies that address workflow for Web Services, and tries to leverage them for Grid

services, which have different needs from standard Web services, and addresses them for Grid services within the OGSA (Open Grid Service Architecture) framework. [20] presents the service selection under a Grid environment and proposes an uncertainty framework to address the issue attempts to investigate the underlying criteria in practice, propose an initial solution using a bidding like mechanism. It shows that the grid service efficiency of composition and scheduling is very crucial to grid applications.

## V. CONCLUSIONS AND FUTURE WORK

In this paper we have proposed a front-end application in order to develop data-intensive applications starting from services that are developed by different owners on the net. This application uses a QoS Composition Algorithm that takes into account a lot of characteristics (Workflow, QoS Parameters, Class of Services, etc.) in order to compose services respecting QoS parameters. From this paper there are a lot of open points that will be considered in future that we are going to list in the following:

- To enrich composition model with more complex workflows and consequently update the composition algorithm.
- To enrich the QoS composition algorithm considering the transmission time parameter that can be variable.
- To implement the front-end application with the QoS composition algorithm in order to use it in a real case study.

## REFERENCES

- [1] F. Leymann, "Combining Web Services and the Grid: Towards Adaptive Enterprise Applications," in *In Proc. CAI/SE/ASMEA'05 (Porto, Portugal)*.
- [2] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. (2001) Web Services Description Language WSDL (Version 1.1). [Online]. Available: <http://www.w3.org/TR/wsdl>
- [3] World Wide Web Consortium W3C. (2003) SOAP. [Online]. Available: <http://www.w3.org/TR/soap/>
- [4] HTTP W3C. HTTP: HyperText Transfer Protocol Specification. [Online]. Available: [www.w3.org/protocols](http://www.w3.org/protocols)
- [5] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," in *Globus Project, 2002*, [www.globus.org/research/papers/ogsa.pdf](http://www.globus.org/research/papers/ogsa.pdf), 2002.
- [6] F. Curbera and M. Duftler and R. Khalaf and W. Nagy and N. Mukhi and S. Weerawarana, "Unraveling the web services web: An introduction to soap, wsdl, and uddi," *IEEE Internet Computing*, vol. 6, no. 2, pp. 86-93, 2002.
- [7] Universal Description, Discovery and Integration UDDI. [Online]. Available: <http://www.uddi.org/>
- [8] WSFR: Web Service-Resource Framework. [Online]. Available: <http://www.globus.org/wsr/>
- [9] IBM Corporation. Business Process Execution Language for Web Services BPEL4WS (Version 1.1). [Online]. Available: <http://www.ibm.com/developerworks/library/ws-bpel>
- [10] N. Kavantzias, D. Burdett, and G. Ritzinger, "Web Services Choreography Description Language WSCDL (Version 1.0)," 2004. [Online]. Available: <http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/>
- [11] IBM Corporation. Web Service Level Agreement (WSLA) Language Specification (Version 1.0). [Online]. Available: <http://www.research.ibm.com/wsla/>
- [12] L. Kleinrock, "Queueing Systems, Vol. I: Theory," in *Wiley* 1975.

- [13] S. McIlraith and T. Son, "Adapting Golog for Composition of Semantic Web Services," in *Conference Proceedings on Knowledge Representation and Reasoning*, 2002.
- [14] D. VanderMeer, K. Dutta, H. Thomas, K. Ramamritham, and S. Navathe, "FUSION: A System Allowing Dynamic Web service Composition and Automatic Execution," in *Proc. of IEEE International Conference on E-Commerce, Newport Beach, CA, USA.*, 2003.
- [15] C. Patel, K. Supekar, and Y. Lee, "A QoS Oriented Framework for Adaptive Management of Web service based Workflows Database and Expert Systems," in *In DEXA-2003 conference, Prague, Czech Republic*, 2003.
- [16] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Sheng, "Quality Driven Web Service Composition," in *Proc. 12th International World Wide Web Conference (WWW)*, 2003.
- [17] M. Shankar, M. DeMiguel, and J. Liu, "An end-to-end QoS management architecture," in *In Proc. 5th Real-Time Technology and Applications Symposium*, 1999.
- [18] "Adaptive and dynamic service composition in EFlow," in *In Proceedings of 12th International Conference on Advanced Information Systems Engineering (CAiSE)*, F. Casati, S. Ilnicki, and L. Jin, Eds., 2000.
- [19] S. Krishnan, P. Wagstrom, and G. von Laszewski, "GSFL: A Workflow Framework for Grid Services," in *In ANL/MCS-P980-0802, Argonne National Laboratory, 9700 S. Cass Avenue, Argonne, IL 60439, U.S.A.*, 2002.
- [20] W. Cheung, J. Liu, K. Tsang, and R. Wong, "Towards Autonomous Service Composition in A Grid Environment," in *In Proceedings of 2004 IEEE International Conference on Web Services*, 2004.