Modeling Quality of Services in Service Oriented Environments

Jens Hündling, Mathias Weske

Hasso Plattner Institute for Software Engineering at the University of Potsdam
Prof.-Dr.-Helmert-Strasse 2-3, 14482 Potsdam, Germany,
Phone: +49(0)3 31 55 09-1 96
{huendling,weske}@hpi.uni-potsdam.de

Abstract. Information technology is the core enabler for global businesses. Based on an integration of services provided by companies, virtual enterprises are built to achieve business goals. This integration can be supported or even automated, if companies publish a standardized description of the services they offer in a formal, computer-readable way. This paper argues that service orientation offers a promising approach to achieve this goal. Furthermore, the paper introduces a formal model for enriched service description that goes beyond technical interface specifications by including Quality of Service properties. We illustrate the model in usage scenarios like automated discovery and substitution of services as well as for optimized composition of services to support service oriented processes. Furthermore, the Quality of Service properties will be integrated in the general service model.

1 Introduction

Nowadays, companies are coalescing, building virtual enterprises and are achieving common business goals together. Information technology is the core enabler for tightening integration of companies, and service orientation is the arising paradigm to handle the complexity of this task. Using the service oriented paradigm, this integration can be supported or even automated, by companies publishing a standardized description of the services they offer in a computer-readable way. This paper argues that service orientation offers a promising approach to achieve this goal, but still is in its infancies since important features are missing. One key aspect is a model for enriched service description that goes beyond technical interface specifications by including Quality of Services (QoS) properties. Throughout this paper, it will be shown how and where QoS properties can be used in the context of automated discovery and substitution of services as well as for optimized composition of services to enable service oriented processes.

The basic principles of service orientation is manifasted by the Service Oriented Architecture (SOA) [2, 8] defining three roles and their interactions as shown in Fig. 1. *Service Providers* publish descriptions of their services in a *Service Repository*. *Service Requestors* can find the published descriptions at the repository, where several descriptions are stored, e.g. in a categorized catalogue. A service description allows the requestor to use the service. This is done by automatically generating a message and send it to the provider for invoking the service; this is called bind.

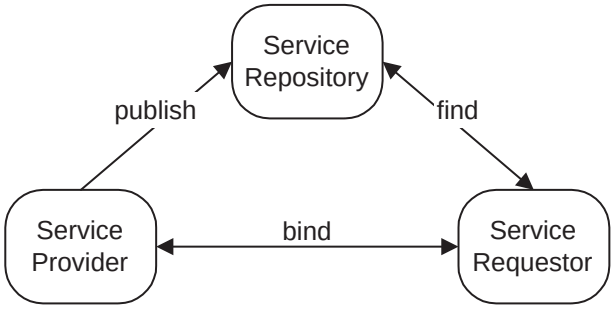

Fig. 1. The Service Oriented Architecture (SOA) – roles and interactions.

These basic roles and interactions imply the availability of standardized description languages and standard communication protocols as well as a common understanding of the querying, the categories and other objectives of the service repository. Current standardization efforts like SOAP for XML messaging, WSDL for service descriptions and UDDI for service repositories are de-facto standards for todays most popular implementation of service orientation: Web services [1]. Additionally, for specific industrial sectors, standardized names for categories and services are evolving or are already existing, and the semantic web is an emerging technology to handle this. This implementation of the basic principles allows loose coupling between business partners and since all interactions can be automated, very late binding, i.e. just-in-time queries to find available services, becomes feasible.

Since paper concentrates on enhancements of service descriptions, the Web Services Description Language (WSDL) [4] will be introduced in more detail. WSDL provides a model, shown in Fig. 2, and an XML format for describing Web services that enables separating the description of the abstract functionality offered by a service from concrete details of a service description such as 'how' and 'where' that functionality is offered. Thus, WSDL describes a Web service in two stages: At an abstract level, WSDL describes a Web service in terms of the *messages* it sends and receives; messages are described independent of a specific wire format typically using a XML Schema type system. One or more messages, their exchange pattern and the logical roles of the participants are aggregated in an *operation*. Finally, an *interface* groups logically related operations still without any commitment to communication protocols. At a concrete level, a *binding* specifies transport and wire format details for interfaces. Adding a network address with a binding results in an *endpoint*, and finally, a *service* groups together endpoints implementing a common interface [4]. The different levels of abstraction will also be relevant for the QoS model introduced in this paper.

The remainder of the paper is organized as follows: For further motivation and better understanding, an example and general usage scenarios for a QoS model are given in section 2. Afterwards, the QoS model is developed in section 3, starting with a definition of QoS, categories of aspects of the properties and the model itself. Additional related

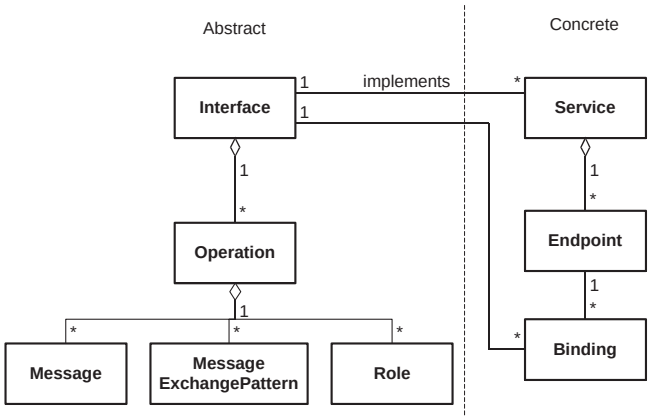

Fig. 2. The core elements of WSDL, version 2.0 (excerpt).

work is pointed to in section 4, and the paper will be closed with an outlook of future applications and refinements of the QoS model.

2 Usage Scenarios for a QoS Model

Using a WSDL description of a service, it is possible to create a correct invoking message and send it to the service providers network address, who in turn conducts the service and returns the result. Nevertheless, for a fully-fledged service oriented environment, this is not enough. First, a service has to be found and the requestor has to decide, whether it uses a specific service implementation or another implementation from a competitor. WSDL concentrates on technicalities like messages, message exchange patterns and URIs of the endpoints. Additionally, a UDDI repository only offers keys for arbitrary technical issues (so called tModel-keys) and category bags aiming at offering categorized searching capabilities like yellow pages. This is not enough as will be shown in the following usage scenarios of typical interactions for Quality of Services enhanced descriptions [13]. Some of these will benefit from and some are not possible without a QoS-Model.

2.1 Example

For better understanding, a short example of a transportation service will be introduced. The service offers transportation and delivery of goods. It is invoked by a message containing procurement information, start and destination address. Additional information like possible climatic conditions of the transportation, e.g. temperature and humidity in climatic containers, are available and important for users of such services. Cost and delivery time can also be provided and security options for the messages like encrypted

payment issues are available. Furthermore, some providers offer delivery state monitoring, e.g. by a web portal (web), per automatic mails at special events like delay or delivery (mail) or by a 24h phone-hotline (phone).

Service	Climatic Temperature	Humidity	Cost	Time	Security Option	State Monitor
S_{11}	–	–	100-150 €	12-48 hours	–	web portal
S_{12}	$\leq$-8°C	$20-60\%$	275 €	24 hours	–	web portal
S_{21}	–	–	150 US-$	1 day	–	mail, phone
S_{22}	max. 20°F	–	320 US-$	8 hours	128-bit key	mail, phone
S_{31}	$\leq$+4°C	$20-80\%$	400 €	24-36 hours	512-bit key	phone, web portal
S_{32}	$\leq$-4°C	$20-40\%$	450 €	48 hours	512-bit key	phone, web portal

Table 1. Sample transportation services and their properties

Table 2.1 shows service implementations S_{11} to S_{32} with additional properties. These services are implementing a common transportation interface I_T (not described in more detail) and are offered by three different transport service providers (T_1, T_2 and T_3). The service names S_{ij} means service is offered by provider T_i; j is a counter for the services of one provider. As could easily be recognized, the diversity of the properties show the complexity of a complete QoS model, and the next sections depicts usage scenarios for the properties using the sample services.

2.2 Usage Scenarios

Discovery The interaction starting with a service requestor searching for a service is called discovery. The requestor generates a query or searches the service repository interactively. This usage scenario surely works with the service descriptions as defined by WSDL and with the service entities described in Fig. 2, when the requestor searches for implementation of an interfaces. Considering the example, all services from S_{11} to S_{32} are in the result set when querying for implementations of I_T. Additional information in an UDDI, like categories etc. are intended to describe this interface or the providers.

Nevertheless, enhancements are more than helpful. Consider a service requestor that has exact restrictions like delivery within 24 hours or frozen goods that have to be transported. Even more, optimization criteria could be given, e.g. the cost. With these optimization criteria, it is possible to select the best service out of a set of restriction matching implementations for the interface.

During automated discovery, additional properties that the requestor did not mention in its query might be taken into account, when more than one service matches the initial query. All of this leads to a more accurate and rapid (i.e. automated) discovery.

Negotiation Service requestor and provider can negotiate on the concrete QoS properties for a specific service or even for a single service execution. This negotiation takes

place after the requestor has located the service description in the repository. For instance, a provider can offer a service with a cost- and time-range like service S_{11} and S_{31} in the example. On each execution the precise costing and time-guarantees can be negotiated.

This allows the provider to take its current workload and available resources into account. On the other hand, the requestor for instance might decide ad hoc whether a very fast service execution is necessary and ask the provider if he can offer this. The negotiation itself can also be automated, for instance by intelligent software agents relying on a QoS model, but this is out of the scope of this paper.

Substitution A specific service often is one step towards a bigger business goal and thereby part of a complex business processes underneath. In special occasions like unforeseen exceptions it becomes necessary to substitute this service, e.g. by a better, faster or cheaper one. Thus, comparing service implementations of a common interface allows more flexibility in the business process and could be a foundation for negotiation by recognizing the boundaries (i.e. limitations or restrictions) and bottlenecks. Consider a company normally using S_{21} for delivery of their products. Suddenly one of the suppliers of the company has problems, causing a 10 hour production delay. To deliver in time, very fast transportation is necessary and thus the faster service S_{22} is used by the company instead.

Composition The advantages of service orientation are multiplied, when services are aggregated to build a new higher-level, value-added service; this is called service composition [9]. The business logic is expressed in a process model that is carefully designed by human experts. QoS properties have to be taken into account when selecting concrete services [20]. Obviously, the QoS of this process that will be offered as a new service depends on the QoS of the individual services. The overall QoS has to be calculated or estimated by the QoS of the composed services due to algorithms, taking into account the control flow, uncertainty etc [3]. These algorithms are out of the scope of this paper, but nevertheless this implies another application of the QoS model for single services.

Process (re-)planning As already mentioned, service composition is normally done by human experts modeling a business process. In an integrated process planning and enactment system as presented in [15], AI planning techniques support the business process modeling. Thereby, it is vital to have a sound QoS model to generate optimized process plans taking time, cost and other QoS properties into account [16]. Obviously, these properties have to be defined in a way the planning algorithm can use them. Taking Replanning [17] in a volatile service oriented environment into account, these optimization can be flexibly done during runtime with up-to-date information. From the viewpoint of usage scenarios the distinction between composition/substitution and planning/replanning is blurred, but the requirements for the QoS model are enhanced, because an AI planning algorithm as to handle the QoS properties.

Services Management Service management is done by providers of services. Providers normally offers a whole set of different services, e.g. at different locations and with different QoS. With measuring and metering of the current execution behavior of its services, it is possible for the provider to adjust resources for the services. In the example, this could be done by recruiting additional resources like drivers and delivery vans or allowing overtime work. The provider has to react flexible to changing demands of requestors as well as to competitor offering service implementation of common Interfaces interfaces, e.g. at a cheaper price.

Furthermore, with a sound QoS model a provider can offer a service at different levels, like T_2 does with the expensive and fast premium level service S_{22} and the cheaper and slower standard level service S_{21}. Thereby, the provider can optimize his workload and cost. Even more, with automatically calculated and managed QoS, dynamic and flexibile negotiation for individual service request is feasible, as already mentioned above. For instance, if its resources are nearly used at full capacity, additional requests might only be answered if the negotiated price is very high.

3 Modeling Quality of Services

These usage scenarios showed where and how a QoS model is needed and how it can be used. Additionally, the main requirements can easily be derived. In this next section, a QoS model will be introduced. First a definition of the term Quality of Services will be given and after a categorization of property scales, the model and its relation to the service entities are explained.

3.1 Definition of Quality of Services

To state the term Quality of Services more precisely, a short discussion is started, heading to a term often considered as closely related or even synonym for QoS [13]: non-functional properties. First of all, a suitable definition of *property* for the purposes of this paper is given by Merriam-Webster OnLine Dictionary [12]:

a.) quality or trait belonging and especially peculiar to an individual or thing
b.) an effect that an object has on another object or on the senses (...).

This definition fits if the thing in definition a.) is the service and the effects on the objects mentioned in b.) are the service execution for some asset between provider and requestor. Interestingly the definition sees *quality* as a synonym for property. As stated earlier, a whole set of different properties about different aspects is related to a service. These properties are often categorized in functional and non-functional properties. Generally speaking, *functional properties* describe what the service does and *non-functional properties* are used to describe how the service does it. Separating properties into functional and non-functional seams feasible and useful, because functional equivalence can be defined on two services with the same functional properties. Thus, a service can easily be replaced by a functional equivalent service with more suitable non-functional properties, e.g. lower cost, faster execution, and higher security. This goes along with the basic service oriented principle of loose coupling.

Nevertheless, in a service-oriented environment it is often hard to decide whether a certain property is functional or not. Take for example the property execution duration of a service, which is a typical example for a non-functional property. Consider a request for "a transportation service within Germany in 24 hours". Here the service's functionality is to transport something. The functionality is specified more precisely by stating "within Germany"; considering the example in section 2.1, this would be part of the interface I_T. The restriction of delivery within 24 hours is also a functional requirement, because a functionally equivalent service has to meet this mandatory property; in the example S_{31} will not meet this criteria and for S_{11} it is not sure since a time-range is given. Similar examples can be found for other properties like costs, security or quality that are often regarded as non-functional properties. To sum up, the decision whether a property is functional or not has to be made individually for each request and is not dependent on the property itself.

For the remainder of this paper we use the term Quality of Services (QoS) for all properties without further distinction between functional or non-functional. Nevertheless, we separate QoS from the description of the service, like a WSDL document or the categories of an UDDI repository.

3.2 Scales of Measurement for QoS Properties

The first aspect to categorize QoS is the scales of measurement of the property. Generally speaking, scales of measurement as know from statistics [19] refer to ways in which variables or numbers are defined and categorized. These scales are ordered, i.e. each higher level of measurement includes the principles of the lower level which implies the allowed arithmetics are also allowed in the higher levels; the categories start with the lowest level and examples for QoS of each scale are given.

Nominal Scale The nominal scale just assigns labels like numbers to values and does not express relationships between values. Thus, the same scale values for a specific nominal scale QoS of two services implies that they are equivalent concerning this QoS. Nevertheless, the values have no numeric meaning. Considering the example in section 2.1, labeling the state monitoring option *web portal* as 1 and *mail* as 2 does not mean *mail* is twice something compared to *web portal*. Nor does it suggest that 1 is somehow better than 2, although a specific requestor might prefer an option.

Consequently, the only arithmetic or statistical operations that can be performed on nominal scales is a frequency run or count. An average can not be determined, except for the mode – that number which holds the most responses –, nor can numbers be added and subtracted. It is important to mention that possible values must be assigned in advance; in order to be exhaustive, an category such as *other* or *unknown* might be applicable. It is also important that the categories provided are mutually exclusive; but multiple options can be offered.

Ordinal Scale QoS properties of ordinal scale have a logical or ordered relationship to each other. This scale permit the measurement of degrees of difference, but not the specific amount of difference, i.e. a service with a higher scale value of a ordinal QoS

has *more* of this property, but the intervals between adjacent scale values are indeterministic. For example, service S_{22} offers message encryption with a 128-bit key, which is less secure than service S_{32} using a 512-bit key.

This allows requestors to specify its needs with = or $\leq$ and providers to give offer values at ranges or using *max*-expressions, but S_{22} is neither 384 bits nor four times more secure than S_{32}. It is not specified that distances between values are equal and difference in the key-length has no numeric meaning.

Interval Scale For QoS of interval scales the distance between adjacent points on the scale are equal, e.g. the difference between 4 and 5 is the same as the difference between 27 and 28. For example, the temperature scales like °Cis an interval scale, since each degree is equal but there is no absolute, meaningful zero point. Consider service S_{12} offering guaranteed -8°C, which is 4°Ccolder than S_{32} offering -4°C, but S_{21} is not twice as cold as S_{32} (think of S_{31} offering +4°C).

At interval scale, addition and substraction is now allowed, and calculating of averages mode, median and mean, the range and standard deviation is possible. This might be useful for estimations of executions, especially when composing services to processes or for service management, when competitors are investigated.

Ratio Scale The ratio scale is applicable where a meaningful zero point exist. For instance, the delivery time in the example is a ratio scale value: S_{12} takes 24 hours for delivery and S_{32} takes 48 hours, which indeed is twice as long. Other examples are weight, costs, or consumption of certain resources like fuel. Multiplication and division of scale values is allowed. As a result, a large number of descriptive calculations are applicable to ratio scale.

3.3 Quality of Services Properties

As mentioned in section 2, service discovery is done by specifying the needs, e.g. services implementic interface I_T. Additional restrictions on the QoS property, like budget restrictions, temporal deadlines and necessary security protocols can be added to the discovery query. On the other side, a requestor can offer his service not with precise QoS properties, but with ranges or at different levels. All of these can be found in the example in Table 2.1. Obviously, several aspects of these restrictions must be handled by a QoS model:

- A QoS property can be a *fixed value* or a *set of possible values*. When the QoS property is at least of an ordinal scale, a *value range*, *min* or *max* are also possible restrictions.
- A QoS property can be *mandatory* or *optional*. In a more sophisticated QoS model, a prefering metric, e.g. an ordering or a weighting and scoring of possible values, can be added at the correct scale.
- Taking into account a set of restrictions, these can be combined *disjunctively* or *conjunctively*.
- *Negated* QoS properties can be described by *complemental* notions.

If a requestor searches a service, the restrictions should be described in a declarative way. Thereby methods well known from Description Logic could be applied for matchmaking, like described in [10]. Standardization efforts like the Web Ontology Language (OWL) [18] are suspected to build a semantic foundation for describing such restrictions.

This leads to the following model: The set of QoS for the service interface I is defined as a set of properties P_i as follows:

$$Q_I = \{P_1, ..., P_n\}.$$

Each $P_i \in Q_I, 1 \leq i \leq n$ has a domain $dom(P_i)$ and the values are of a scale $scale(P_i)$ as defined in section 3.2. Also, for each property a metric $metric(P_i)$ has to be defined, how, where, and when the property is measured and the numbers or labels assigned to the measured results. Furthermore, let $dom(Q_D) = \bigcup_{P \in Q_D} dom(P)$. Then, a concrete QoS level L_S of a service S_I implementing an interface I is a transformation:

$$\lambda : Q_D \rightarrow dom(Q_D) | (\forall P \in Q_D) \lambda(P) \in dom(P).$$

Thus, for each service implementation the provider can define a service level with the applicable properties. This service level should be stored as part of the service description in the repository and requestors can define queries with desired mandatory and optional properties. A key requirement for this model is that a predefined and commonly agreed on domain of possible values and the corresponding scale. Thereby, checking correct values and applicable comparisons like *at least*, *inbetween* or *max* are possible. For example, a requestor can query for a Service with the properties $P_{time} \leq 30hours$, $P_{securityoption} \in$ { (128-bit key),(512-bit key)} and so on.

3.4 QoS Entities

The QoS model can be mapped to the service entities as shown in figure 3. The **Quality of Services** is a set of properties. For each **Property**, the **Domain**, a **Scale** and a **Metric** has to be defined as explained above. This abstract set of QoS properties is linked to a **Interface**. Thus, for each interface, the set of applicable QoS properties are specified. The concrete counterpart of this set is a **ServiceLevel** which in turn is related to a service. The cardinality of the relation between service and a service level is stated as 1-to-1, because with a different set of QoS properties, it becomes a different service implementation. Nevertheless, when a **Provider** stores its service and the service level in a repository ranges etc. can be defined. A **ServiceLevel**, a provider and a **Requestor** can negotiate on more precise QoS properties and create a **ServiceLevelAgreement** as a contract.

4 Related Work

As stated earlier, Quality of Services is a term used in many research areas. Service Level Agreements (SLA) are well know from network services as a contract offered by

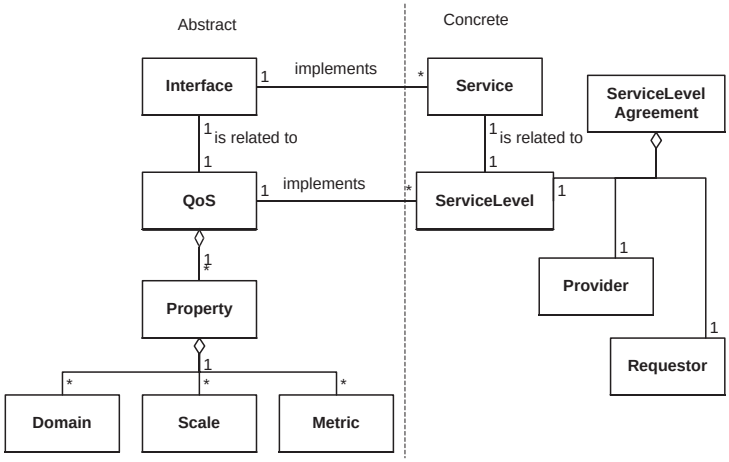

Fig. 3. Mapping of the QoS Model to the Service Entities.

a provider specified in measurable terms like throughput, availability (time) or number of simultaneously users. IS departments adopted the idea of SLA and allowed customer departments to measure, justify, and compare e.g. with outsourcing providers. Thus, it is necessary to exactly define, how each property metric can be observed or measured and what happens if a property is violated. This idea of contracts can also be used in a service oriented environment, where a SLA is seen as a formal (electronic) contract specifying the QoS of a service [14, 11]. The SLA can be stored in the service repository and thus included in the discovery [5]. Furthermore, a SLA can be the individual result from an automated negotiation phase between provider and requestor. Recently standardization efforts are underway to define a common language to define SLAs for Web services.

WS-Policy [7] is part of a greater framework focussed on sharing information securely between applications and organizations. This framework consists of two key groups of specifications: 1.) Technical concerns in security area like and 2.) Streamlining implementation of business policies in a Web services environment. The WS-Policy framework sees policies as collection of all acceptable policy statements for specific domain and thereby goes along with the definition of the set of QoS properties given in this paper. The framework specifies also how a policy can be attached to WSDL entities and how a policy is defined as a UDDI. Nevertheless, a policy itself are not clearly defined and quite vague and a sound model for the properties is vital for the possibilities of QoS.

In [13] an overview definition of possible non-functional properties are given and some of the use cases defined here are used for motivating precise modeling of non-functional properties. Nevertheless, it concentrates on specific properties rather than on an overall model.

Another very interesting approach is to take quality ratings of users of the service into account. [6] introduces a quality of services management framework based on user expectation. This approach as a different viewpoint, but the QoS models are not contradictory and integration of the ideas will be an interesting future work.

5 Conclusion and Outlook

In this paper a formal QoS Model is introduced. It is based on the principles of domains and scales of the possible properties. These properties are grouped as a set of properties that is related to an interface. For a concrete service implementation, the specific service level can be defined and enhanced requests for services can be generated.

Nevertheless, many things that are already in the example are not mentioned in detail in this paper. Considering the sample service S_{11}, it is likely that the cost depends on the delivery time negotiated as well as the distance between starting point and delivery point. Modeling this functional dependence is out of the scope of this paper and will be future work. Additionally, the metric has been left out of the discussion as well as the difference of the measurement units like US-$ and € or °C and °F.

These aspects are future work, and defining QoS in OWL [18] seams to be a fruitful approach which we currently investigate. Additionally, using QoS for processes and process (re-)planning which was mentioned in the usage scenarios is still under research.

Acknowledgements: The authors would like to thank Hilmar Schuschel, Harald Meyer and Hagen Overdick for their comments on earlier versions of this paper.

References

1. G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services - Concepts, Architectures and Applications*. Springer Verlag, 2004.
2. S. Burbeck. The tao of e-business services. IBM Corporation, October 2000. Available at http://www-106.ibm.com/developerworks/library/ws-tao/.
3. J. Cardoso, A. Sheth, and J. Miller. Workflow quality of service. In *International Conference on Enterprise Integration and Modeling Technology and International Enterprise Modeling Conference (ICEIMT/IEMC)*, 2002.
4. R. Chinnici, M. Gudgin, J.-J. Moreau, J. Schlimmer, and S. Weerawarana. Web Services Description Language (WSDL) version 2.0 part 1: Core language. http://www.w3.org/TR/wsdl20/. W3C Working Draft 26 March 2004.
5. A. Dan, H. Ludwig, and G. Pacifici. Web services differentiation with service level agreements. Technical report, IBM Corporation, May 2003. http://www-106.ibm.com/developerworks/library/ws-slafram/.
6. V. Deora, J. Shao, W. A. Gray, and N. J. Fiddian. A quality of service management framework based on user expectations. In M. E. Orlowska, S. Weerawarana, M. P. Papazoglou, and J. Yang, editors, *Service-Oriented Computing - ICSOC 2003*, volume 2910, pages 104–114,. Springer, 2003.
7. M. Hondo and C. Kaler, editors. *Web Services Policy Framework (WSPolicy)*, May 2003. Available at http://www.ibm.com/developerworks/library/ws-policy.
8. J. Hündling and M. Weske. Web services: Foundation and composition. *EM - Electronic Markets Journal*, 13(2):108–119, June 2003.

9. F. Leymann, D. Roller, and M. Schmidt. Web services and business process management. *IBM Systems Journal*, 41(2), 2002.
10. L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Proceedings of the Twelfth International World Wide Web Conference (WWW'2003)*, pages 331–339. ACM, 2003.
11. H. Ludwig, A. Keller, A. Dan, R. P. King, and R. Franck. Web service level agreement (wsla) language specification. Available at http://www.research.ibm.com/wsla, January 2003.
12. Merriam-Webster, Inc. Merriam-webster online dictionary. Available at http://www.m-w.com, 2004.
13. J. O'Sullivan, D. Edmond, and A. Ter Hofstede. What's in a service? towards accurate description of non-functional service properties. *Distributed and Parallel Databases*, 12:117–133, 2002.
14. A. Sahai, V. Machiraju, M. Sayal, A. van Moorsel, and F. Casati. Automated sla monitoring for web services. In M. Feridun, P. Kropf, and G. Babin, editors, *13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM*, volume 2506 of *Lecture Notes in Computer Science*, pages 28 – 41, 2002.
15. H. Schuschel and M. Weske. Integrated workflow planning and coordination. In *14th International Conference on Database and Expert Systems Applications*, volume 2736 of *LNCS*, pages 771–781, Prague, Czech Republic, 2003. Springer.
16. H. Schuschel and M. Weske. Automated planning in a service-oriented architecture. In *2nd International Workshop on Distributed and Mobile Collaboration*. IEEE Computer Society Press, 2004. to appear.
17. H. Schuschel and M. Weske. Automated planning in a service-oriented architecture. In *8th East-European Conference on Advances in Databases and Information Systems*, LNCS. Springer, 2004. to appear.
18. M. K. Smith, C. Welty, and D. L. McGuinness. OWL Web Ontology Language Guide. Available at http://www.w3.org/2004/OWL/, 2004. W3C Recommendation 10 February 2004.
19. S. Stevens. On the theory of scales of measurement. *Science*, 103:677–680, 1946.
20. L. Zeng, B. Benatallah, M. Duams, J. Kalagnanam, and Q. Z. Sheng. Quality driven web service composition. In *The 12th International World Wide Web Conference*, Budapest, Hungary, May 2003. ACM.