Incorporating QoS Specifications in Service Discovery

V. Deora, J. Shao, G. Shercliff, P.J. Stockreisser, W.A. Gray, and N.J. Fiddian

School of Computer Science
Cardiff University
Cardiff, UK
v.deora@cs.cf.ac.uk

Abstract. In this paper, we extend the current approaches to service discovery in a service oriented computing environment, such as Web Services and Grid, by allowing service providers and consumers to express their promises and requirements for quality of service (QoS). More specifically, we allow service providers to advertise their services in an extended DAML-S that supports quality specifications, and we allow service consumers to request services by stating required quality levels. We propose a model here for incorporating QoS specifications and requirements in service discovery, and describe how matchmaking between advertised and requested services based on functional as well as quality requirements is supported in our model.

1 Introduction

There is a growing interest in service oriented computing (SOC) in recent years [1,2,3,4]. Central to SOC is the notion of a *service* which can broadly be considered as a software component that represents some computational or data-offering capability. By allowing services to be advertised declaratively, discovered dynamically and invoked remotely, SOC makes it possible for users to locate, select and execute services without having to know how and where they are implemented. As such, this new computing paradigm offers great potential for distributed applications in an open environment and on a large scale.

In this paper, we consider the problem of how to support dynamic service discovery within an SOC environment. This is an important issue because an SOC environment can potentially contain thousands of autonomous services which may come and go at any time. It is desirable therefore that the user is able to request a service by stating declaratively what is required, rather than having to specify how to obtain or access a specific one at a pre-determined location. In other words, discovering which services are available to meet the requirement of a specific service request should be performed dynamically at the time the request is made.

A general approach to supporting this type of service discovery is based on the *register-find-bind* model [5] outlined in Figure 1. Here, service providers (SPs) make their services available by registering their capabilities in a centralised

C. Bussler et al. (Eds.): WISE 2004 Workshops, LNCS 3307, pp. 252–263, 2004.

service registry. Service requesters (SRs) then locate their required services by sending their requirements to the registry. A matchmaking component is often implemented as part of the registry to match an incoming service request with those available in the registry. If a match is found, the details of the advertising SP are returned to the requesting SR. The SP and SR are then said to be *bound* to complete their task.

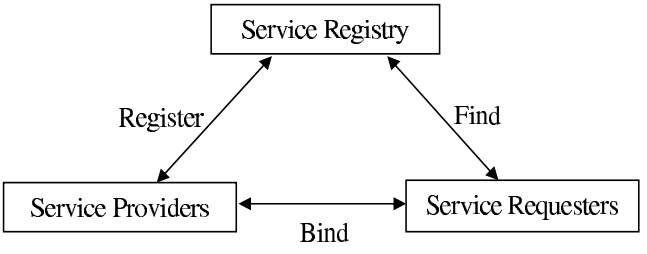

Fig. 1. A Generic Service Discovery Model

Currently, matchmaking between requested and registered services is largely based on their functional attributes. To illustrate this, consider the following example. Suppose that there are several SPs who offer multimedia services to PDA or mobile phone users, and we have an SR who wishes to purchase a monthly subscription package for science-fiction movies and news services, including 30 free text messages and at least 50 free talk minutes. This service request may be represented abstractly as follows:

subscription_type = monthly
+ Video_Content where media_style = science_fiction
+ Html_content where html_content_type = news
+ text_messages where number_of_free_messages = 30
+ Phone_Calls where number_of_free_minutes ≥ 50

Fig. 2. A Request for Service

Here, the request is made in terms of the required functions, for example, the qualifying SP must offer `science fiction` movies. To determine which SP(s) can offer the required service, the matchmaking component searches through the registry, typically using a string comparison method, to see if any registered services match some or all of the functional attributes listed in the request. If, for example, SP1 has advertised that it offers `science fiction` movies or

`cartoons`, then SP1 is identified as a potential provider for the required service and its details are returned to the SR.[1]

The above model works fine if we assume that the SR is only interested in the functional aspects of a service. In practice, however, it is quite possible that we may have several SPs offering the same service, and just like in a common marketplace, an SR may wish to select a service based on functional as well as other attributes such as quality and cost. Thus, it would be desirable that the SR can pose the following request, where `frame rate = 24` and `availability = 7 days/week` are QoS requirements:

subscription_type = monthly
+ Video_Content where media_style = science_fiction
(frame_rate = 24 & availability = 7 days/week)
+ Html_content where html_content_type = news
+ text_messages where number_of_free_messages = 30
+ Phone_Calls where number_of_free_minutes ≥ 50

Fig. 3. Service Request with QoS Requirements

The current function-based approaches are not sufficient to support this more advanced form of service discovery. In this paper, we extend the existing approaches to service discovery by incorporating QoS specifications and requirements into matchmaking. That is, we let SPs advertise their functional capabilities and QoS promises to the registry, and monitor the actual quality of the services offered by the SPs. An SR can then request a service by specifying not only functional requirements, but also QoS expectations. We propose a two-step service discovery model here, and describe how matchmaking between advertised and requested services based on functional as well as QoS requirements is supported in our model.

The rest of the paper is organised as follows. Section 2 discusses the related work. Section 3 introduces our service discovery model. We will explain how service advertisement, request and matchmaking are supported in our model. Section 4 presents conclusions and discusses future work.

2 Related Work

The progress on supporting dynamic service discovery in SOC is largely represented by the development of UDDI [6], which is currently a *de facto* standard for service advertisement and discovery. The standard UDDI does not offer any

[1] Note that it is not necessary to find a single SP who can satisfy the SR's request completely, and the matchmaking component will search for all the SPs who can serve any part of the request.

support for QoS specification and relies largely on keyword matching for service discovery. Recently, some extensions have been made to UDDI to allow inclusion of QoS in service specifications [7], but they support only a very limited set of specific attributes, for example, network response time. Our approach differs from these extensions in that we employ a QoS ontology in service specification. This allows richer semantics of services to be specified and more sophisticated matchmaking between advertised and requested services to be performed, using a wide range of QoS attributes that require different means of representation, monitoring and assessment.

Integrating QoS with service description has also been considered by the work on deriving service level agreement (SLA) for web services. IBM, for example, has recently introduced a framework which allows SRs to form contracts with SPs using an XML-based language [8]. This framework also allows a representation with which the QoS provision agreed by the SPs can be monitored. While these works also attempt to integrate QoS with service descriptions and store such extended service descriptions in a repository for reference, they differ from our work in that they consider QoS representation and integration for services after they have been discovered, rather than in the initial discovery of relevant services. In other words, we attempt to use QoS information as part of a service-selection process, whereas the works on SLA consider it from the service provision perspective.

Various service description languages, such as WSDL [9], DAML-S [10] and OWL-S [11], have been proposed. WSDL is used currently in conjunction with UDDI as a standard for describing web services. This is a low level mechanism which does not support semantic description of services. DAML-S, and its newer version OWL-S, are developed by the semantic web community for describing services at a higher level. They support the use of service ontology in service description and allow some form of reasoning in service discovery [12,13]. Although these languages are quite powerful in describing the functional aspects of a service, they offer little support for specifying the non-functional aspects. We also use DAML-S for service descriptions, but extend it to include explicit, separate QoS specifications.

3 QoS-Based Service Discovery

In this section we introduce our QoS-based service discovery model. Our approach is outlined in Figure 4, which is an extension to the generic service discovery model given in Figure 1.

The service registry in our model consists of two components: the Yellow Pages (YP) component and the QoS component. The YP component is similar in purpose to the matchmaking component that we described in Section 1, and is used to determine which advertised services match the requirements stated in an incoming service request. Note however that in our model both functional and quality properties may be specified. The QoS component, on the other hand, is used to calculate the quality of a service on demand, using the QoS ratings

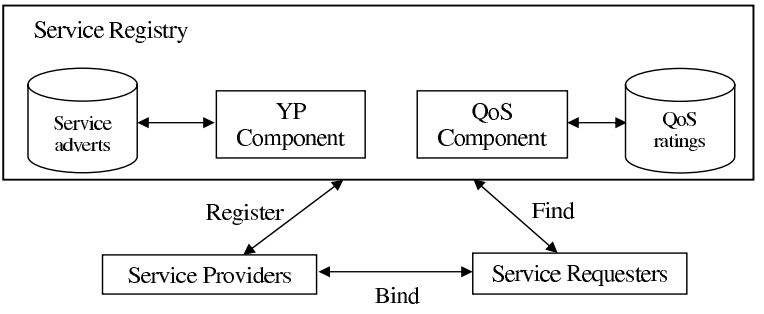

Fig. 4. QoS based Service Discovery

collected from the previous uses of the service. By interacting with these two components, an SR can find a required service in two steps:

- **Searching.** In this step, the SR sends a service request containing functional and/or QoS requirements to the YP component. For example, SR1 may request a service that can offer `science fiction` movies (a functional requirement) and the `frame rate` of movie delivery is required to be 24 frames per second (a QoS requirement). The SR uses this step to search for SPs who *claim* to be able to offer the required service. The YP component will return a list of SPs whose advertisements match what is required in the SR's request.

- **Evaluation.** Following the searching step, the SR can ask the QoS component to assess *how well* each SP returned by the YP component can actually provide the service in this step. This is necessary because service advertisements can not entirely be trusted, and some SPs may not honour what they have promised. For example, the YP component may find, from the advertisements, that SP1 can provide the movie service that SR1 has requested, but the past experience by other users may suggest that SP1 is more likely to provide, on average, 22 instead of 24 frames per second for its movie delivery. Allowing QoS assessment by the QoS component in this step therefore gives the SR the opportunity to establish what can really *be expected* from a particular SP.

We consider this two-step service discovery model being a significant improvement over the existing approaches. By allowing QoS attributes of a service to be specified (by SRs and SPs), searched (by the YP component) and evaluated (by the QoS component), our approach supports a more meaningful, accurate and relevant service discovery in SOC. In the rest of the paper, we will primarily consider the searching step, and will discuss how service advertisement, request and matchmaking are supported in our model. The reader is referred to [14] for a detailed description of the evaluation step.

3.1 Description of Services

To support dynamic and automatic service discovery, it is first necessary to be able to describe services in a machine-processable way. To enable this, we need an expressive service description language and some ontologies for common terminology. In our model, we use DAML-S [10] for service description. We choose DAML-S because it allows us to describe not just the low-level service functionality and requirements in terms of message format, data types and protocols, but also attach semantic information, such as service classification and descriptions, to the services. For standardising terminology in service description, we create two ontologies. The *service ontology* provides SPs and SRs with a common terminology for advertising and requesting services, and enables the YP component to match advertisements with requests. The *quality ontology*, on the other hand, specifies what QoS attributes are and how they are related to the services.

Creating a service ontology is supported by DAML-S and is relatively straightforward, but is domain dependent. Thus, different service ontologies are needed for different domains. For a simple `media` application, for example, a sample service ontology is given in Figure 5 (for simplicity of presentation, we have expressed the ontology here as a class diagram, rather than its implementation in RDF).[2]

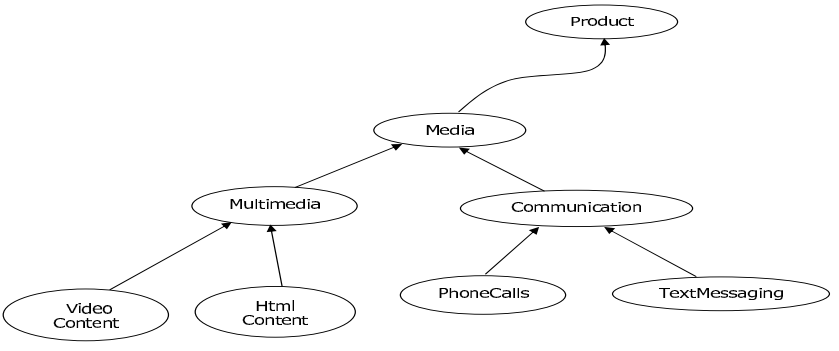

Fig. 5. A Service Ontology for a Simple Media Application

Creation of a quality ontology is, however, not currently supported by DAML-S, and requires some explanation. For different classes of services, a large number of attributes may be used to describe their QoS properties. Some are domain dependent and will only be relevant to a specific class of services. For example, `frame rate` is only relevant to a movie service. Others are domain independent and are applicable to all types of service. For example, `availability` [15] is applicable to movie as well as other services.

[2] Note that due to naming conflicts with DAML-S our top-level service is of class `Product` – DAML-S itself defines `Service` as a class from which all services inherit.

Thus, it is important that our quality ontology distinguishes between these two types of QoS attributes, so that we do not repeat ourselves in specifying service independent QoS for each individual service. Motivated by this observation, we group all QoS attributes into *service specific* and *service independent*, as shown in the example ontology given in Figure 6 for the simple `media` service. As can be seen, a service must include a service specific QoS attribute explicitly (e.g. `VideoContent` has `FrameRateCategory` as one of its QoS property), but will include service independent QoS attributes implicitly (e.g. `VideoContent` also has `AvailabilityCategory`, `PerformanceCategory` and `ReliabilityCategory` as its QoS attributes).

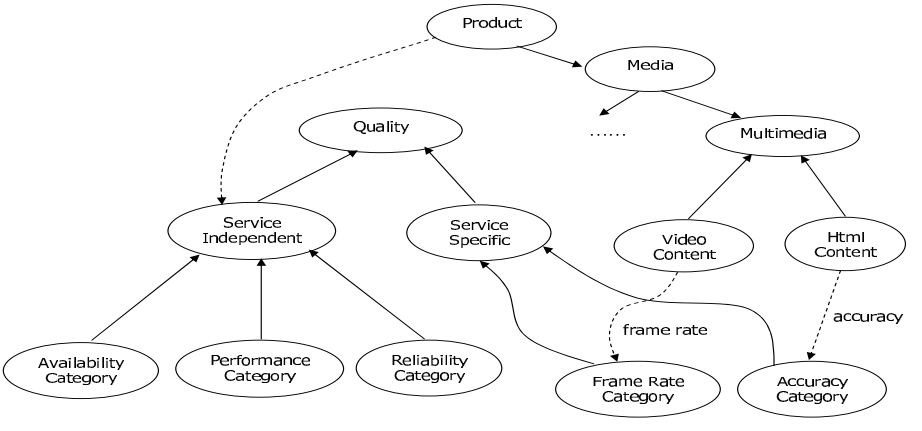

Fig. 6. A Quality Ontology for the Simple Media Application

The two ontologies are then integrated with DAML-S to facilitate service description. This is achieved by making our `Product` class to inherit from the `ServiceProfile` class provided by DAML-S. This is illustrated in Figure 7. Note that DAML-S defines its `Service` class in terms of `Service Profile`, `Service Model` and `Service Grounding`. For our work, we have only used `Service Profile` as this is sufficient for service discovery purposes.

It is worth noting that in the extended service description in DAML-S, we have also included the `QualityPreference` class. This is to allow an SR to specify preferences in searching for the required service. By stating whether it wishes to maximise service quality or perform some user specified tradeoff between quality and cost, a flexible service discovery can be supported. More detailed discussion on this is, however, beyond the scope of the current paper.

SPs and SRs can then advertise and request services using the extended service description facilities given in Figure 7. To ensure that service advertisements and requests adhere to the service and quality ontologies, we introduce advertisement and request schemas. Due to space restrictions we will not dis-

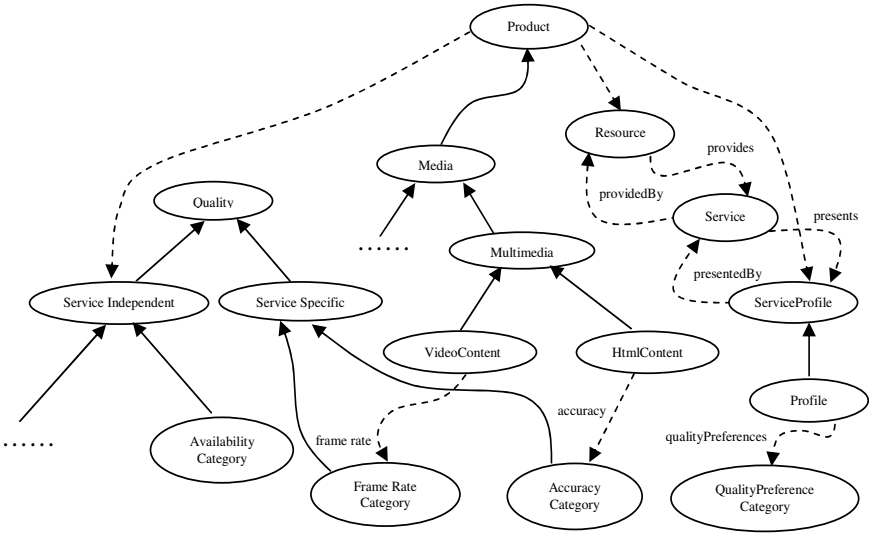

Fig. 7. The Extended Service Description in DAML-S

cuss these schemas further. However, it is worth noting that SPs and SRs do not have to specify values for all QoS properties that are listed in the quality ontology. If some QoS property is unspecified, we will treat it as *unknown* (if it is in an advertisement) or *uninterested* (if it is in a request). Once created, service advertisements and requests are sent to the YP component for registration and matchmaking, respectively.

3.2 Matchmaking

Matching a service request with the advertised ones is performed by the YP component. In this section, we explain how this is done. Suppose that we have a set of advertised services A and a service request R, respectively, as follows

$$A = \{s_1^a, s_2^a, \ldots, s_n^a\} \qquad R = \{s_1^r, s_2^r, \ldots, s_k^r\}$$

where each s_i^a $(1 \leq i \leq n)$ is an advertised service and each s_j^r $(1 \leq j \leq k)$ is a requested service. Note that a single service request may ask for several services.

For matchmaking purposes, we assume that an advertised service (s^a) and a requested service (s^r) are represented as follows:

$$s^a = \langle sn, sp, fs, qs \rangle \qquad s^r = \langle sn, fs, qs \rangle$$

where sn is the service name, sp is the service provider, fs is the set of functional specifications and qs is the set of QoS specifications. We refer to these components using the "." notation, that is, $s^a.sn$ refers to the service name of s^a.

Our matchmaking task is to find a set of SPs who offer services that will match any subset of R. That is, we search for

$$M = \{s_i^a \mid s_i^a \succeq R' \subseteq R, 1 \leq i \leq n\}$$

where $s_i^a \succeq R'$ denotes that s_i^a provides a service that satisfies the functional and QoS requirements of each $s^r \in R'$. The following steps describe how M is found.

1. *Determine which advertisements are relevant to R.* This is not simply a process of comparing $s_i^a.sn$ to $s_j^r.sn$. Since our service ontology organises classes of services as a hierarchy, it is necessary to traverse the service ontology too. For example, the example ontology given in Figure 5 defines `VideoContent` and `HTMLContent` as two sub-classes of `Multimedia`. If an SP advertises to offer `Multimedia` service, then this SP is considered to offer *both* `VideoContent` and `HTMLContent` services together. Thus, to search for SPs who can provide `VideoContent` and `HTMLContent` services, it is necessary to consider the advertisements that offer `Multimedia` services too. The following procedure explains how this is performed by the YP component:

```
input A = {s_1^a, s_2^a, ..., s_n^a}, R = {s_1^r, s_2^r, ..., s_k^r}, ont = service ontology
output Rel ⊆ A
1   Rel = ∅
2   for each s_j^r (1 ≤ j ≤ k)
3       Rel = Rel ∪ {s_i^a.sp | s_i^a.sn = s_j^r.sn, 1 ≤ i ≤ n}
4       p = s_j^r
5       while (p ≠ null)
6           p = getParentClass(p, ont)
7           S = {s_i^a | s_i^a.sn = p, 1 ≤ i ≤ n}
8           for each s ∈ S
9               c = getComponentServices(s)
10              Rel = Rel ∪ {s_i^a.sp | c ⊆ R, s_i^a.sn = p}
11  return Rel
```

   The YP component will first find all the advertisements that have the same service names as those requested (line 3). Then, the YP component recursively traverse the service ontology up (line 6) to find those services that are more general than s_j^r, but contain no component services that are not requested in R (lines 9 & 10). This "no more than required" restriction is necessary because currently we assume that an advertised service must be taken in its entirety. For example, one is not allowed to take the `HTMLContent` service alone from an SP if it has advertised to offer `Multimedia` service. Clearly, SPs who offer more than necessary (and perhaps will charge more) are undesirable.
2. *Determine which advertisements meet functional requirements.* In this step, the functional requirements specified in each $s^r \in R$ are used to determine

which advertised services that the YP component discovered in Step 1 must not be returned to the SR. That is, the YP component performs the following:

input $Rel = \{s_1^a, s_2^a, \ldots, s_m^a\}$, $R = \{s_1^r, s_2^r, \ldots, s_k^r\}$
output $RF \subseteq Rel$

1 $RF = \emptyset$
2 for each s_j^r $(1 \leq j \leq k)$
3 $\quad RF = RF \cup \{s_i^a.sp \mid s_i^a.sn = s_j^r.sn, s_i^a.fs \geq s_j^r.fs, 1 \leq i \leq m\}$
4 return RF

where $s_i^a.fs \geq s_j^r.fs$ expresses that advertised functionality ($s_i^a.fs$) must be equal to or better than the requested ($s_j^r.fs$). For example, if the SR requires `Media style = science fiction`, then movie services advertisements that offer `cartoons` will be disregarded at this stage. This is a fairly straightforward process. If no advertisements can be found, the YP component will send `failed` message to the SR.

3. *Determine which advertisements meet QoS requirements.* This is similar to Step 2, except that the conditions for matching are different. Assume that $qa \in s_i^a.qs$ is one of the advertised qualities for s_i^a and $qr \in s_j^r.qs$ is one of the requested qualities for s_j^r. The YP component will match qa with qr according to the following:

advertised (qa)	requested (qr)	matching condition
specified	specified	$qa \geq qr$
specified	unspecified	matching
unspecified	specified	matching

That is, if either qa or qr is unspecified, then the YP component considers the two qualities unconditionally matching. This is justified because they represent the cases where either the SR is not interested in some QoS properties or whether its QoS requirements can be met or not cannot be verified. At the end of this step, any advertisements that do not meet the required QoS properties will be dropped from RF, and the details of SPs for the remaining advertisements are returned to the SR.

The above description explained how the YP component performs matchmaking between advertised and requested services during the searching step. As we have outlined earlier in this section, our service discovery model also has a second step – the evaluation step. The SR may decide to use the QoS component in the evaluation step to establish what can really be expected from the SPs returned by the YP component. This is particularly useful in cases where some SPs cannot be trusted or some QoS requirements specified by the SR are unspecified by the SPs. The QoS component can in such cases help to establish some "facts" about the "unknowns", based on other users' experience with the services. We have developed an expectation-based QoS calculation model and for details of our algorithm, the reader is referred to [14].

4 Conclusions and Future Work

In this paper we presented a framework for incorporating QoS specifications in service discovery. The proposed service discovery model is currently being implemented as part of the CONOISE project [16], which aims to develop agent-based technology for forming and managing virtual organisations (VOs) in an SOC environment. In our model we extended the standard DAML-S to allow SPs and SRs to advertise and request services with both functional and QoS requirements, and described our matchmaking mechanism. It is important to note that the service and quality ontologies developed here are not fixed schemas. They can be created and modified as required, without affecting the underlying service discovery mechanism. Thus, our model provides a scalable, dynamic service discovery in an open, distributed computing environment.

We also introduced a two-step QoS-based service discovery model in this paper. With this model, we distinguish between what might be expected of a service (derived from advertisements, previous experiences or recommendations) and what might be materialised (calculated from previous uses of the service). We see this distinction being significant, as it mirrors service discovery in the real world and allows meaningful service discovery to be conducted.

There are several issues which will need further studies. Currently, we use a single YP component. It will be interesting to consider the case where multiple YP components, either distributed or working as a cluster, are used, and study how they may collaborate, especially if we allow each YP component to have its own local domain service and quality ontologies. It will also be useful to consider service composition, that is, to allow SRs to request composition of services, and to extend our matchmaking with a similar capability. Finally, our matchmaking mechanism is rather basic at the moment. There is a need to consider a more powerful matchmaking mechanism whereby ontological reasoning is fully exploited.

Acknowledgments. This work is supported by British Telecommunications plc, and we would like to thank the members of the CONOISE project team for their constructive comments on this work.

References

1. Casati, F., Shan, M.C.: Definition, execution, analysis, and optimization of composite e-services. IEEE Data Engineering Bulletin. Vol. 24(1) (2001) 29–34
2. Leymann, F.: Web services: distributed applications without limits. In: Proceedings of Tenth Conference on Database Systems for Business, Technology and Web. (2003)
3. Piccinelli, G., Stammers, E.: From e-processes to e-networks: an e-service-oriented approach. In: Proceedings of Third International Conference on Internet Computing. Vol. 3 (2002) 549–553
4. Rust, R.T., Kannan, P.K.: E-service: a new paradigm for business in the electronic environment. Communications of the ACM Vol. 46(6) (2003) 36–42

5. Kreger, H.: Web services conceptual architecture (WSCA). (http://www-4.ibm.com/software/solutions/webservices/pdf/WSCA.pdf) (2001)
6. Belwood, T. et al: UDDI Version 3.0.1 Specification. (http://www.uddi.org/)
7. Chen, Z., Liang-Tien, C., Silverajan, B., Bu-Sung, L.: Ux: An architecture providing qos-aware and federated support for uddi. In: Proceedings of the First International Conference on Web Services. (2003)
8. Ludwig, H., Keller, A., Dan, A., King, R.P.: A service level agreement language for dynamic electronic services. In: Proceedings of Fourth IEEE International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems. (2002) 25–32
9. Christensen, E. et al: Web services description language (WSDL). (http://www.w3.org/TR/wsdl)
10. Ankolekar, A., Burstein, M., Hobbs, J., Lassila, O., Martin, D., McIllraith, S., Narayanan, S., Paolucci, M., Payne, T., Sycara, K., Zeng, H.: Daml-s: Web service description for the semantic web. In: Proceedings of First International Semantic Web Conference. (2002)
11. Web-Ontology Working Group: Web ontology language - services (OWL-S). (http://www.daml.org/services/owl-s/1.0/)
12. Dogac, A., Cingil, I., Laleci, G.B., Kabak, Y.: Improving the functionality of uddi registries through web service semantics. In: Third International Workshop on Technologies for E-Services. (2002) 9–18
13. Dogac, A., Kabak, Y., Laleci, G.: Enriching ebxml registries with owl ontologies for efficient service discovery. In: Fourteenth International Workshop on Research Issues on Data Engineering. (2004)
14. Deora, V., Shao, J., Gray, W.A., Fiddian, N.J.: A quality of service management framework based on user expectations. In: Proceedings of the First International Conference on Service Oriented Computing. (2003) 104–114
15. Mani, A., Nagarajan, A.: Understanding quality of service for web services. (http://www-106.ibm.com/developerworks/library/ws-quality.html) (2002)
16. Norman, T.J., Preece, A., Chalmers, S., Jennings, N.R., Luck, M., Dang, V.D., Nguyen, T.D., Deora, V., Shao, J., Gray, W.A., Fiddian, N.J.: Conoise: Agent-based formation of virtual organisations. In: The Twenty-third SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence. (2003)