3 Related work

In the last few years, QoS issues in Web service technology gained importance in ongoing research. This chapter gives an overview of selected major industrial and academic efforts towards the specification and management of QoS for Web services [20]. After discussing the state of the art in Web service technology, five examples will give an impression of the variety of research in this area.

3.1 State of the art

In the standard Web service interaction model [21], a.k.a. Service Oriented Architecture (SOA) [22], the functionality and information on where and how to access Web services (i.e. under which universal resource identifier (URI) using which protocol) is described in a WSDL file. This file is commonly referenced from a UDDI registry for standardized service discovery.

A UDDI registry is basically a data base with UDDI logic and interfaces for publishing and searching Web services [10]. Industrial categorization, contact information, and technical information about services such as contact information of a company or industry categories, can be stored and viewed there by either using a Web interface or an application program interface (API). However, entries are often inaccurate or expired [23] and UDDI does not provide

a mechanism for automatically updating the registry as services (and service providers) change [24].

Since the communication between clients and UDDI is a client/server interaction, UDDI can become a performance bottleneck in case of overloading or even unavailability. Furthermore, [26] shows that performance considerations of current UDDI implementations do have a major impact on the overall acceptance of UDDI. Section 4.3 introduces the WS-QoS broker, which improves the standard Web services interaction model.

A Web service can be described by its static functional attributes and its dynamic non-functional parameters including QoS aspects. The different QoS aspects are for example

- server performance including throughput, availability and reliability,
- network performance including bandwidth, jitter and delay,
- security and transactional support,
- configuration and management capabilities as well as
- cost.

With the emergence of functionally equivalent services implementing a common service type (e.g. *tModel in terms of UDDI*), the non-functional QoS properties associated with services will become distinctive criteria for the success of a company offering e-business through Web services. It is also imaginable that a business will offer its services in various classes of quality to meet different customers' requirements, according to what they might be willing to pay in return. Therefore, the need to unambiguously specify both QoS properties and different QoS levels in some kind of contract such as SLA and to prove the Web services' compliance arises.

In terms of SOA, Web services are building blocks from which more sophisticated applications can be created. An introduction to current specifications for Web service composition is given in [27]. Our WS-QoS framework does not deal with service composition but can be well applied for looking up appropriate services in a service composition process.

Three different phases of QoS management can be identified. Firstly, QoS constraints on certain parts of a provided service are formulated in a *specification*. Since the purpose of such a specification is to reinforce a certain level of QoS, parameters are monitored by constant *measurements* at runtime. Finally, the values measured have to be tested against the negotiated specification and appropriate activities should be carried out in order to *control* the QoS conformance ensuring a low violation rate. In case of violations, compensation may be refunded to the service consumer.

The following subsection gives an overview of five selected major approaches towards QoS specification and management for Web services, coming from both the industry and the academia. The selected approaches cover SLA for Web

services, formal specifications of classes of services, Web service reputation, and UDDI extension. These approaches are

- the Web Service Level Agreement (WSLA) developed by IBM [28],
- the Web Service Offering Language (WSOL) developed at Carleton University, Canada [29],
- SLAng developed at University College London, UK [30],
- a UDDI eXtension (UX) developed at Nanyang Technological University, Singapore [31], and
- UDDIe developed at Cardiff University, UK [24].

3.2 Approaches towards QoS specification and management for Web services

All approaches that will be presented in this section deal with the specification and management of QoS for Web services. Although they all target the same issues, the proposed concepts are very different, which makes a direct comparison and evaluation difficult. Generally, two basic ideas of a QoS-aware service selection can be identified: The first type of concepts proposes a new infrastructure for specifying QoS issues associated with Web services. The second type deals with extending the functionality of UDDI either within or outside UDDI by introducing an additional server or broker. While WSLA, WSOL and SLAng are of the first type, UX, UDDIe, and WS-QoS belong to the second.

3.2.1 Web service level agreement

IBM's Web Service Level Agreement (WSLA) framework fosters the idea of individually negotiated customized service level agreements. A WSLA is an agreement between a service provider and a customer and as such defines the obligations of the parties involved. Primarily, it is the obligation of a service provider to perform a service according to agreed-upon guarantees for the service parameters on the technical level (such as availability, response time, and throughput) [28].

The design goals of WSLA are a formal and flexible XML-based language for SLA definitions between different organizations, a wide acceptance and applicability to existing e-business systems and standards, nested relationships of service clients and providers, delegation of monitoring tasks to third parties, and an SLA-driven configuration of the managed resources, i.e. deriving configuration settings directly from SLAs.

3.2.1.1 Service level specification

While a great variety of SLAs with different semantics of QoS parameters exist, an SLA generally includes information on the parties involved, a reference to the operational description of the service covered by the SLA, SLA parameters to be

monitored, the metrics and algorithms used to compute the SLA parameters as well as the intended service level objectives (SLO) and appropriate actions to be executed in case they are violated [28].

According to the WSLA XML schema a WSLA is divided into three sections:

In the "parties" section, *signatory* and *supporting parties* are introduced.

The "service description" section yields information on the service's characteristics and the parameters to be observed. *Resource Metrics* are derived directly from the managed resources as specified in a *Measurement Directive*. They can then be aggregated to *Composite Metrics* where a *Function* explains how a composite metric is computed from input parameters which are either resource metrics or composite metrics themselves. In an SLA *Parameter* the retrieved metrics are put into the context of a specific customer, adding information on who supplies the value and to whom it will be reported.

Finally, the third section "obligations" describes guarantees and constraints imposed on the SLA parameters in the form of *service level objectives (SLOs)*, i.e. high/low watermarks for associated SLA parameters which are promised to be met for a certain period of time. In the case of SLA violations, appropriate compensating activities are defined in *Action Guarantees*.

3.2.1.2 Deployment and monitoring – the SLA lifecycle

The service customer and service provider are the signatory parties of a WSLA. A trustworthy third party may be entrusted with the monitoring of the SLA compliance. A WSLA's management lifecycle consists of five stages, as shown in Figure 4. These stages are:

1. The establishment of an SLA is negotiated with the help of an SLA *Establishment Service*.

2. After the signatory parties have reviewed the SLA, relevant information in form of *Service Deployment Information* (SDI) is *deployed* to the supporting parties.

3. Based on the received SDI a *Measurement Service* collects the metrics specified and reports them to a *Condition Evaluation Service*, which in return computes the grade of compliance of the measured values with the SLA parameters.

4. On reception of this information the *Management Service* will take corrective management actions to deal with the occurrence of SLA violations. Before carrying out any corrective action, the *Management Service* has to consult the *Business Entity* for approval. The *Business Entity* holds critical business information which is usually confidential.

5. The last stage is the termination of the customer/provider relationship.

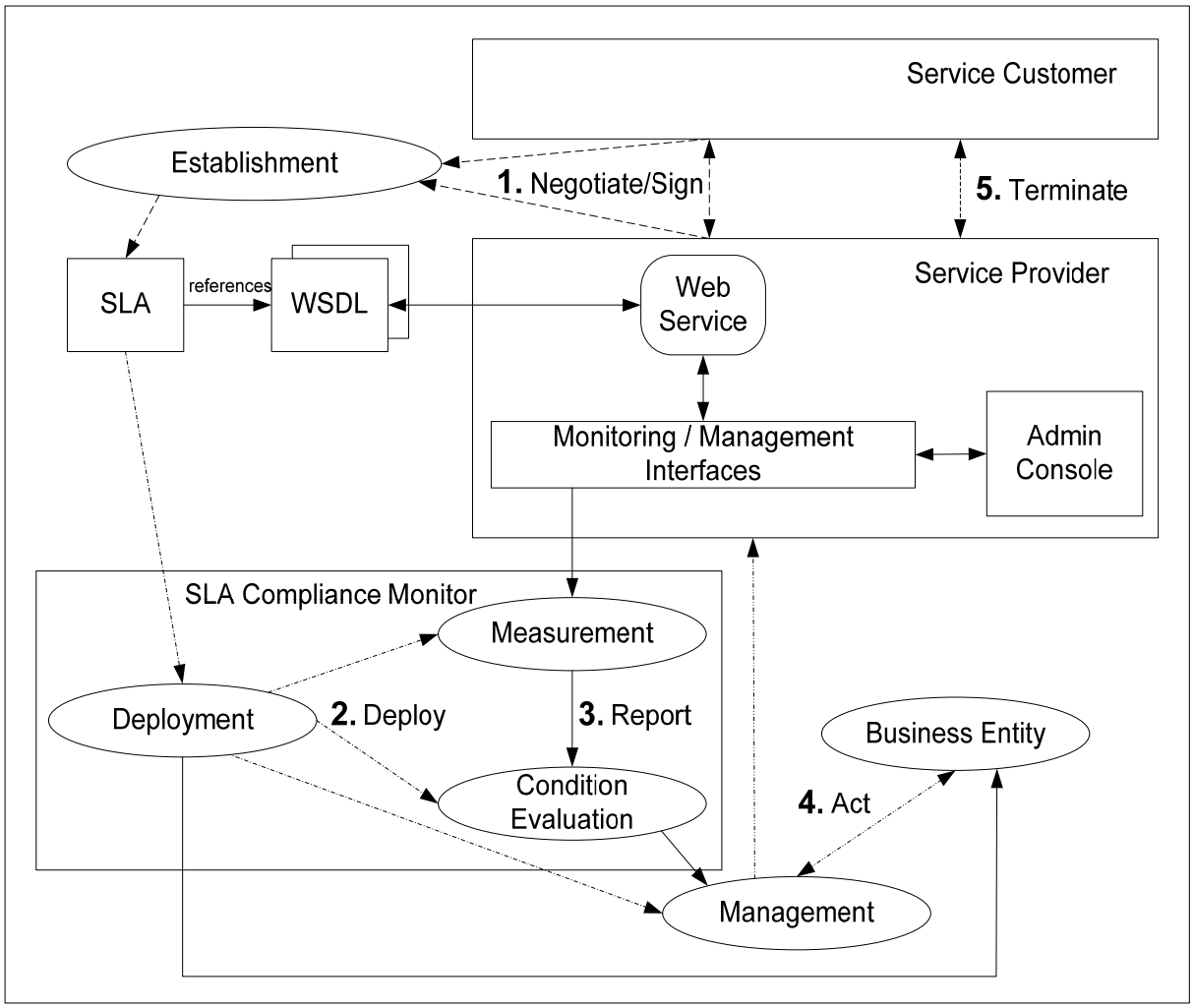

Figure 4. The five stages of an SLA management lifecycle [32]

IBM's SLA Compliance Monitor implements the Measurement, Condition Evaluation and Deployment Service as described above. It is part of IBM's Emerging Technologies Toolkit [33].

3.2.2 Web service offering language (WSOL)

A research group from Carleton University in Canada has developed the notion of providing various *classes of service* for one and the same functional service specification, which differ in QoS level and management efforts. WSOL allows the formal and unambiguous specification of prices, monetary penalties, management responsibilities and third parties, especially accounting parties.

The main targets of the WSOL project are the creation of service offerings, definition of QoS constraints, management statements, reusability, and a mechanism called service offering dynamic relationship (SODR) allowing for switching between services [29].

Another important design goal is a low run-time overhead achieved through defining classes of services instead of individually managed SLAs. WSOL also supports reusability of specifications. This is realized by means of the concept of

constraint groups and constraint group templates to include formerly defined elements and import of elements defined in other WSOL files.

3.2.2.1 Service offering

Classes of service are a mechanism for the description and the differentiation of a Web service and QoS associated with that Web service. While classes of service of one Web service refer to the same functionalities which are defined in the same WSDL file, they differ in QoS constraints and management statements. Different classes of a service may imply different utilization of the underlying hardware and software resources. A service offering can also be seen as a contract or SLA and consists of several components as listed in Figure 5.

WSOL Service Offering:

- import / include of external specification
- subscription
- price
- penalty
- management responsibility
- constraints
  - domain (service-/port-/operation- name)
  - condition (as Boolean expression)
  - metrics (defined in external ontology)
  - rules for metrics aggregation
  - management entity for measurement
- statement
  - domain (service-/port-/operation- name)
  - <any definition>
- constraint group / instantiated CGT
  - <any item listed above>

Figure 5. Components of a WSOL service offering

3.2.2.2 Constraints

WSOL allows the specification of three categories of constraints: functional constraints, non-functional (QoS) constraints, and access rights. A constraint is a Boolean expression specifying a condition that is to be verified before and/or after the execution of a Web service's operation for which it is specified. Constraints

are defined by extending the generic *constraint* element. Functional constraints such as pre- and post-conditions check some characteristics of message parts of an invoked operation. Non-functional constraints describe guaranteed QoS parameters such as performance, reliability and availability of a service. QoS metrics and measurement units are defined in an external ontology. Finally, access control constraints define conditions under which a consumer is admitted for service invocation.

3.2.2.3 Management

Management information is described in WSOL *statement* elements including pay-per-use price, monetary penalty, and management responsibility. A pay-per-use price states the fee a consumer has to pay for the service usage and the monetary penalty regulates the condition when the service provider violates the assured QoS constraints. Management responsibility defines what party (the consumer, the supplier, or a trusted third party) is responsible for monitoring a particular constraint.

3.2.2.4 Reusability

There are a number of features enabling the reusability of existing specifications. Existing WSOL documents can be imported for the reuse with the *import* element. Constraints and statements that have already been defined can be applied to a different scope by referencing and renaming them in an *include* statement. Moreover, statements and constraints can be grouped into constraint groups (CG) that can easily be referenced for alternative domains. This *CG Element* allows for nesting definitions to an arbitrary level. An abstract *Constraint Group Template* is defined in a *CGT Statement* and is instantiated with concrete parameters.

3.2.2.5 Service offering dynamic relationship

Service Offering Dynamic Relationship (SODR), which is defined outside a WSOL file, ensures the initiation of predefined management activities in case of underperformance. The management activities are responsible for

- switching between service offerings,
- deactivating or reactivating existing service offerings, and
- the creation of new appropriate service offerings as well as
- the negotiation and selection of an appropriate replacement of service offerings.

A *Service Offering Relationship* element specifies a current service offering, a service offering appropriate for replacement and a sequence of constraints. A violation of the constraints will trigger the specified change in the service offering.

3.2.3 SLAng

As an XML-based language for defining service level agreements, SLAng, was developed at the University College London, UK. The main targets of SLAng are

to support inter-organizational service provisioning including storage, network, middleware, and applications as well as the specification of non-functional parameters at service level in order to enable QoS description and negotiation [30].

At the moment SLAng can be used only for static SLAs, since it does not support dynamic lookup of new services and update of non-functional service properties at runtime.

3.2.3.1 Service provision reference model

Figure 6 depicts the service provisioning model of SLAng. The 3-tier architecture consists of an application tier, a middle tier, and the underlying resources. Applications consume underlying components or Web services abstracted by the middle tier. The containers located in the middle tier support QoS negotiation, establishment, and monitoring, while the components are an abstraction for resources in the underlying resource tier. Network and storage facilities are typically representatives of this tier.

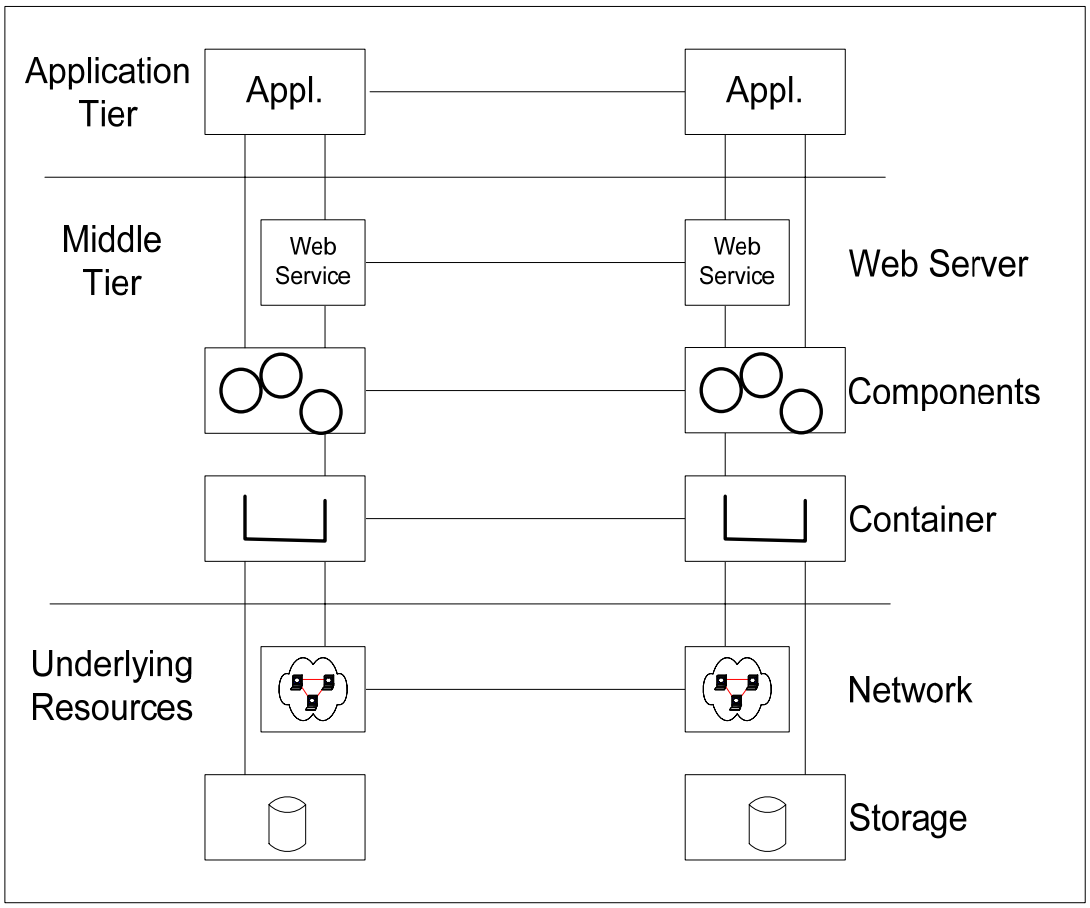

Figure 6. SLAng's service provisioning reference model [30]

3.2.3.2 Structures

SLAng supports both horizontal and vertical SLAs. While a horizontal SLA is a contract between peers in the same tier, a vertical SLA describes the usage of the underlying layer.

There are seven different types of SLA definitions in SLAng, as shown in Figure 7. The vertical SLAs are related to application, hosting, persistence, and communication, while the horizontal ones are related to services (between a component and a Web service provider), container, and networking. Each kind of SLA includes definitions of responsibilities of the service client and the service providers, and mutual responsibilities.

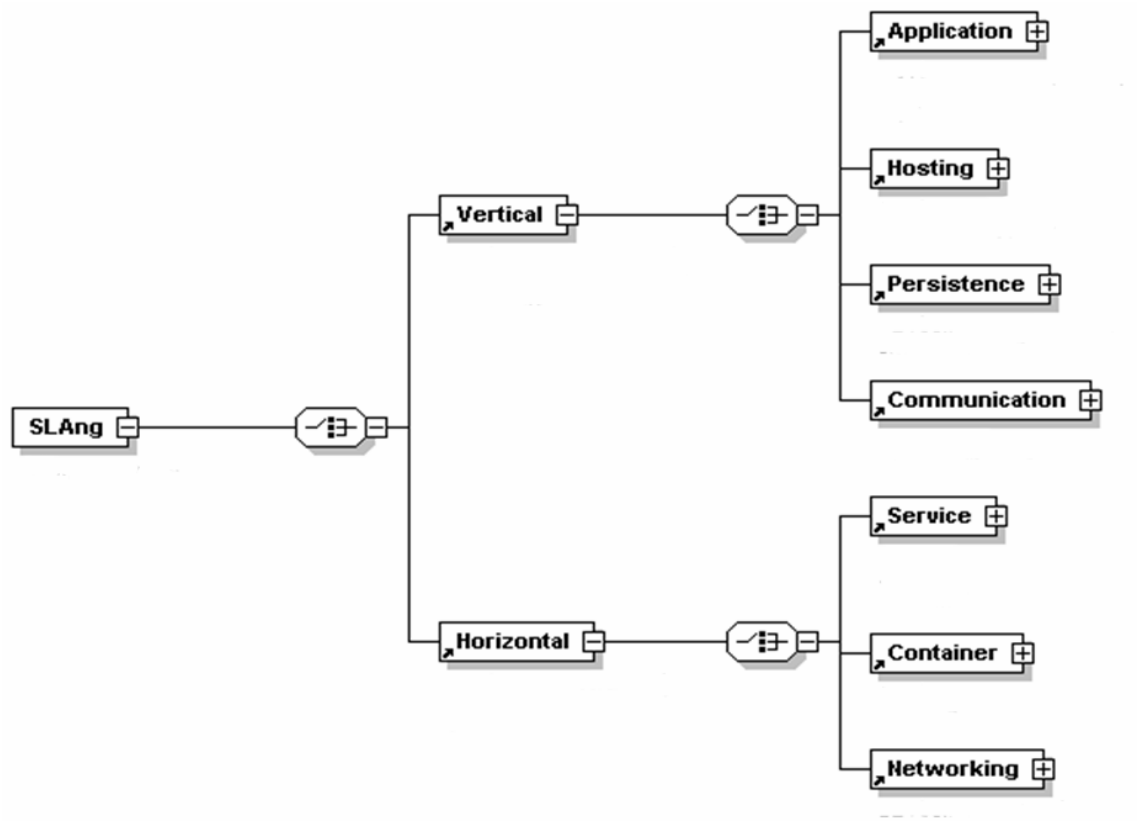

Figure 7. Vertical and horizontal SLAs [30]

3.2.3.3 QoS semantics

The QoS semantics in SLAng are defined according to the different SLAs of the performance properties. For example, the throughput of a database server and the throughput of a component server are different concepts, although both contribute to the overall QoS. Therefore, they are treated differently before being composed [30].

3.2.4 UX

UX is an architecture providing QoS-aware and cross organizational support for UDDI, developed at the School of Computer Engineering, Nanyang Technological University, Singapore. The first goal of UX is to rate services with reputation in order to allow service requestors to discover services with good quality. The second one is to share the ratings among UX servers – the proposed extension of UDDI – in different domains [31].

3.2.4.1 Approach

Reputation is measured through QoS feedback made by service customers. The proposed UX server uses the clients' reports containing response time, terminating state, and cost to generate summaries in order to predict the services' future performance. The generated summary containing response time, reliability, cost, timestamp, and report number is used to sort the query results.

The UX server is extended with an inquiry interface which is conforming to standard UDDI. A lookup interface in UX allows to discover and to distribute QoS summaries among different UX servers.

Figure 8 depicts the basic architecture of UX. When the UX server receives a request (1), it will first search the local UDDI server for services (2). If the number of appropriate services in the local registry is high enough, it will return a list of results back to the service requestor; otherwise it will initiate a so-called federated discovery in order to find more results.

Federated service discovery describes the ability to search QoS-aware services among different UX servers across different domains. UX applies a protocol that dynamically updates the links between cooperating servers over a WAN according to different events happening, either to some servers, or to the underlying WAN. The applied protocol is called Cooperating Server Graph (CSG) model [34].

In the UX architecture, the UX server returns a list of appropriate services to the service requestor, so that the service requestor has to process the suggested service offers on receiving them. This approach may result in additional processing time and increased configuration overhead at the client side.

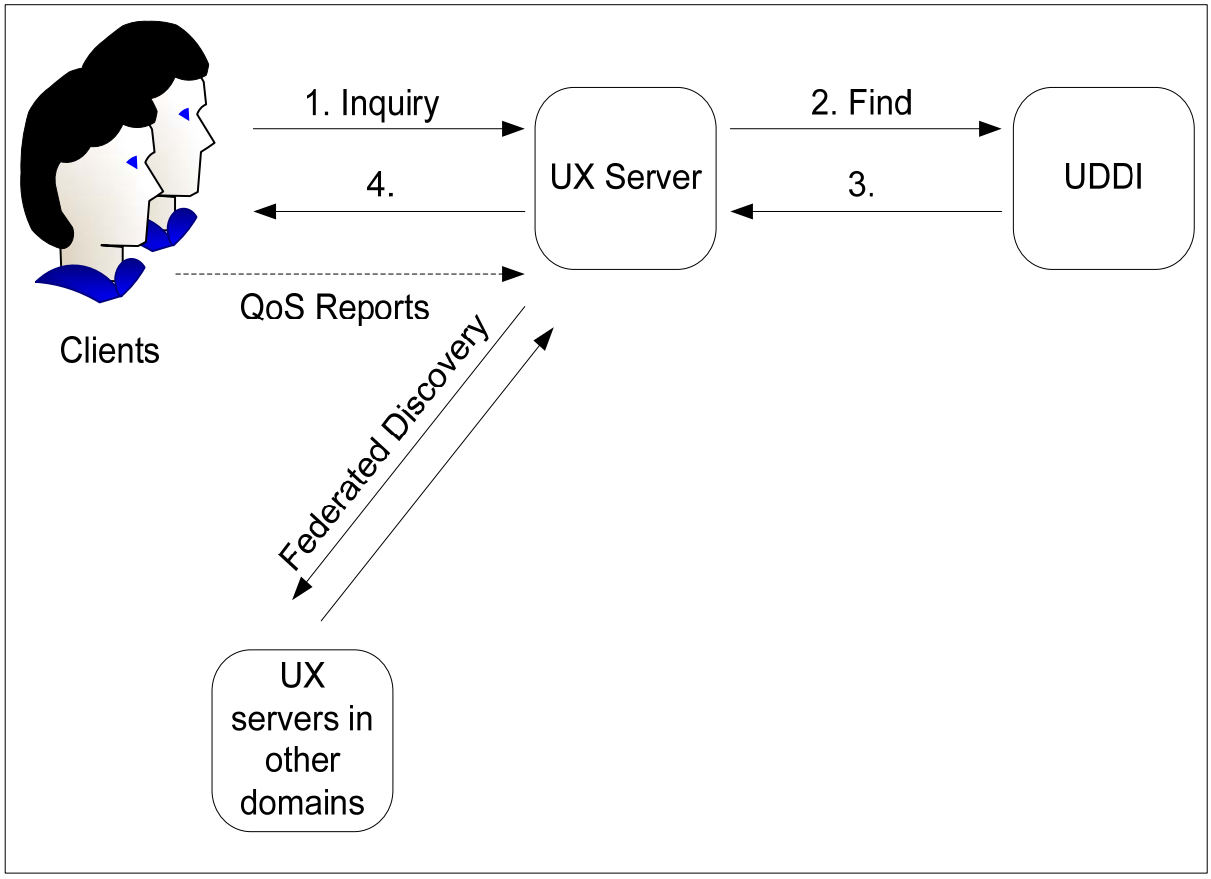

Figure 8. UX architecture [31]

3.2.4.2 Verity

UX extends UDDI by the ability to predict services' future performance based on reputation, which is measured through QoS feedback made by service customers. The users' experience is shared in a local and inter-domain way. Since reputation is mainly influenced by the user perception and can be furthermore manipulated easily, a research group from Monash University, Australia, extended the approach to reputation with a QoS attribute termed "verity" [35]. Verity is used as an indicator of trustworthiness for the quality driven selection and composition of services.

3.2.5 UDDIe

UDDIe [24] was developed at Cardiff University, UK. It extends UDDI's functionalities within UDDI. Service providers can associate their services with QoS properties such as bandwidth, CPU, and memory requirements, which are encoded in the service interface. They can make their services available for a period of time by means of leasing. UDDIe introduced a concept allowing the definition of three leasing types including finite, infinite, and future lease. Furthermore, UDDIe supports qualifier-based search by introducing qualifiers such as *EQUALto, LESSthan,* and *GREATERthan*.

3.2.5.1 Approach

UDDIe is implemented in the context of the G-QoSM framework for grid service discovery [37]. A client application sends a request to the QoS broker of the G-QoSM system. The broker is not part of UDDIe. It processes the request and forwards them to the UDDIe registry. After receiving a list of services that implement the particular service type, the broker selects the most appropriate service for the client by applying a weighted average concept. Figure 9 shows a code fragment of a client request with QoS requirements.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
....
targetNamespace="http://MyService-Interface">
<wsdl:messagename="printNameResponse">
</wsdl:message>
....
<QoS>
<service_cost> 5 </service_cost>
<network_bandwidth> 256K </network_bandwith>
<memory> 48MB </memory>
....
</QoS>
</wsdl:definitions>
```

Figure 9. Client request with QoS requirements in UDDIe [24]

QoS properties associated with a service are provided together with further information about that service by the extended UDDI server. The QoS properties are encoded in the service interface.

Since non-functional properties of a service such as price and performance are subjects to frequent changes, repeated requests to a huge UDDI registry might prove to be a bottleneck in the performance of service lookups. Because of these facts the UDDI server is not the right place to store non-functional properties. Furthermore, encoding non-functional properties into service interfaces should be avoided, since otherwise the service interface would have to be recompiled each time non-functional properties are changed. WDSL is designed to hold information on functional aspects of a service which is not expected to be changed often once designed. Therefore, a more flexible way can be to delegate the non-functional description of a service from WSDL to an extra file so changes on non-functional properties can take place without changing the WSDL.

3.3 Conclusion

Section 3.2 has described five different approaches towards the QoS-aware specification and management of Web services. Common denominators of the approaches are the use of XML and the conformance with the existing Web service technologies such as WSDL and UDDI. While WSLA fosters individually customized SLAs, WSOL introduces a formal specification of classes of service. SLAng can be used for SLA specifications in general not only for Web services, aiming at a wide usage. UX is able to select services based on reputation among federated UX servers. UDDIe extends the standard UDDI API in order to associate QoS properties with Web services.

In the following, we compare the introduced approaches based on different aspects. We don't give an overall assessment of each approach. The table gives rather an overview of the main emphases of the introduced approaches.

Selected assessment criteria include requirement specification, class of service, QoS aspects, QoS mapping, and flexibility:

Requirement specification: Both Web service clients and providers need means to specify non-functional requirements and offers. The specification should ensure the compatibility and comparability of the specifications done by clients and service providers.

Class of service: QoS parameters differ in quality, quantity, and the corresponding monetary charge. Grouping similar parameters into a class or category that characterize a service will ease the utilization of the service.

QoS aspects: A Web services related framework should support more than the classical QoS parameters such as jitter and bandwidth. Aspects such as security, reliability, transaction as well as custom defined aspects should also be considered.

QoS mapping: An overall QoS support requires QoS support during the whole communication process, ranging from the QoS specification to monitoring at runtime. QoS has also to be considered through the different layers in terms of the Internet Model. Specifications in higher layers have to be carefully mapped onto lower layers.

Flexibility: An approach should be easy to use, extensible, and standards conforming.

The assessment of the introduced approaches is summarized in Table 1. The symbols mean:

"++": excellent concept

"+": good concept

"O": satisfying

"-": poor or not available

Table 1. Assessment of the introduced approaches

	Requirements specification	Class of service	QoS aspects	QoS mapping	Flexibility
WSLA	++	O	+	-	++
WSOL	++	++	+	-	+
SLang	+	-	+	-	+
UX	O	-	O	-	O
UDDIe	O	-	O	-	O