The Unified Service Query Language Technical Report

Michael Pantazoglou and Aphrodite Tsalgatidou
National and Kapodistrian University of Athens, Hellas

{michaelp, atsalga}@di.uoa.gr

July 19, 2009

Abstract

This report presents the specification of the Unified Service Query Language (USQL), which is aimed at supporting the discovery of heterogeneous Web, Peer-to-Peer, and Grid services. At the conceptual level, USQL establishes an abstract and service type-independent viewpoint of the service properties, which are constrained by service requesters through their search criteria. At the grounding level, the language provides an XML-based syntax for the formulation of service discovery requests and responses. Overall, USQL is characterized by a high-level of abstraction, which ensures compatibility with existing service description and discovery standards, and extensibility, which allows for easy accommodation of additional service properties and/or support for emerging service technologies.

1 Introduction

Service orientation fosters a new breed of distributed, interoperable software systems and applications. In a Service-Oriented Architecture (SOA), *providers* expose their services through standards-based interfaces, and publish their descriptions in appropriate *brokers*, thereby rendering them available for discovery. In turn, *consumers* search the contents of these brokers in order to discover services, which meet their requirements. As soon as a matching service is selected, a negotiation interaction may begin between the provider and the consumer, after which the service can be directly invoked and utilized within the context of the consumer's application.

The aforementioned interactions may take place during designing of a system, be dynamically applied at runtime, or be performed in an ad-hoc manner by end users, e.g. for finding and consuming a single service. Hence, service discovery is of paramount importance, spanning the whole life cycle of a service-oriented application, and is iteratively applied to improve its functionality and performance. Moreover, it plays an important role in driving the adoption of Service-Oriented Computing (SOC) by the industry, as it enables businesses and organizations to communicate and interact with each other in terms of the services they provide, or need.

Within the last years, the growing popularity of service orientation led to the emergence of a number of instantiating technologies. To date, Web services are the most prominent one, and are based on a set of widely accepted standards governing their description (WSDL [5]), publication & discovery (UDDI [6]), and invocation (SOAP [20]). Besides, other computing technologies also adopted the service-oriented approach. In Grid Computing, the *Web Services Resource Framework* (WSRF) [22] extended the Web services model in order to meet Grid requirements towards the notion of state. The WSRF specification comprises a set of protocols for modeling state in a Web services context, thereby defining a *WS-Resource* as a set of a Web service and a well-described resource. Further, in service-oriented Peer-to-Peer (P2P) technologies, such as JXTA [10] or Edutella [21], peers expose functionality as P2P services so as to allow other peers in their belonging peer group to discover and use them. Each peer in a service-oriented P2P network is capable of both providing and consuming services, as well as hosting service advertisements. Thus, peers generally play all basic roles that are defined by an SOA (i.e. provider, consumer, and broker).

Web services are commonly discovered by searching private registries/repositories, which comply with the UDDI or ebXML [12] standards. P2P services are discovered in a decentralized manner, and the respective mechanisms primarily depend on the P2P infrastructure. For instance, in JXTA, service advertisements are discovered with the use of a Discovery service provided by the JXTA infrastructure, which effectively propagates queries within the peer group of the requester peer. Discovery of Grid services is also strongly associated with the underlying Grid middleware. Virtual Organizations (VOs) implemented on top of the Globus infrastructure (GT4) [9] utilize the provided Index Service for discovery of the available Grid services and resources, according to the WS-ServiceGroup specification. Besides, Grids relying on the emerging gLite middleware [17] employ realizations of the GLUE Schema [1] such as R-GMA, LDAP, or XML, to share information regarding Grid services and other kinds of entities commonly met in Grid environments.

Apart from the aforementioned diverse discovery mechanisms, heterogeneity also exists in the languages being used for the description of Web, P2P, and Grid services. Web services are commonly described by means of the WSDL language. The WSDL standard is also conveniently used for the description of Grid services, although certain extensions have been introduced by the WSRF specifications to cater for the description of resource properties. As regards P2P services, there is currently no common format for their description, meaning that each P2P technology freely defines its own language to describe and advertise services offered by its peers. For example, JXTA peers describe their services with the use of an extensible XML document called Module Spec Advertisement (MSA). Nevertheless, the need to semantically describe all types of services led to the emergence of additional, ontology-based description formats such as OWL-S [19], WSML [26], or the latest SAWSDL specification [8], as complements to WSDL. In addition to these technologies, numerous languages have been proposed for the description of application-level, qualitative characteristics of services, such as WS-QoS [30], WSLA [14], WSOL [31], etc.

In various application domains, business requirements impose the need to utilize other types of services, such as P2P or Grid services, in addition to Web services, so as to deliver the desired

functionality. However, the above presented discrepancies render their combined discovery a cumbersome task. In order to better illustrate this fact, let us briefly describe a real-world scenario from the domain of e-Health, as it was shaped in the context of the SODIUM project [32].

1.1 Use Case Scenario

The IT department of a private clinic has decided to build a service-centric system that will facilitate interactions between patients, doctors and other associates. The department has already exposed parts of its legacy software related to administrative and data management tasks as Web services, which have been published to the clinic's private UDDI registry. The clinic has also set up a JXTA-based system supporting P2P services such as patient monitoring, mobile telemedicine, and information sharing between patients and doctors. Further, the clinic is involved in a Grid Virtual Organization (VO) that has been collaboratively set up by a number of partner hospitals with the aim of supporting resource-demanding Grid services with complex computations, such as drug analysis, tissue compatibility checks, etc, in an interoperable manner.

One of the intended applications of the system is to electronically assist the diagnoses made by the clinic's medical staff. The respective process will start with the retrieval of the specified patient's medical file from the clinic database, through an appropriate Web service. Next, the process splits in two parallel execution branches:

- In the first branch, a Web service is needed to extract the details of the specified medical episode from the patient‘s file, and a P2P service is required to get a list of online specialist doctors.
- In the second branch, the medication list of the patient is retrieved with the use of a Web service, and is processed with the use of a drug analysis Grid service.

Then, based on the collected information, the clinic's doctor consults one of his/her online specialist colleagues through a P2P service in order to get a second opinion. Finally, the doctor is in a position to make a diagnosis on the specific episode, send it directly to the patient by means of another P2P service, and accordingly update the patient‘s record in the database, through the corresponding Web service.

Naturally, in order to implement the designated diagnosis support application, developers need not to start from scratch. Instead, they should first determine whether the specified tasks could be fulfilled or not, by existing Web, P2P, and Grid services. However, in doing so, they have to separately perform service discovery in the clinic's UDDI registry, the JXTA network, and the Grid VO, which the clinic has access to. This presumes that, developers need to (1) be familiar with the low-level technical details of each technology and its provided search APIs; and (2) formulate their queries depending on the formats of the corresponding service descriptions advertised in the aforementioned broker types.

1.2 Challenges and Contribution

As the above scenario demonstrated, heterogeneity among the various services technologies gives rise to the necessity of addressing the following challenges in the field of service discovery:

- *Heterogeneity in service types and service descriptions*: Service requestors should be able to express their requirements independently from the service type and the format(s) used by service providers in describing their services
- *Heterogeneity in service broker technologies*: Service discovery queries should be uniformly constructed and processed, independently from the environments in which the various service descriptions have been published
- *Multi-dimensional query formulation*: Queries should accommodate the multiple functional and non-functional service properties
- *Flexibility and sustainability*: The established service discovery solution should be immune to advances in the related technologies, to the maximum possible extent

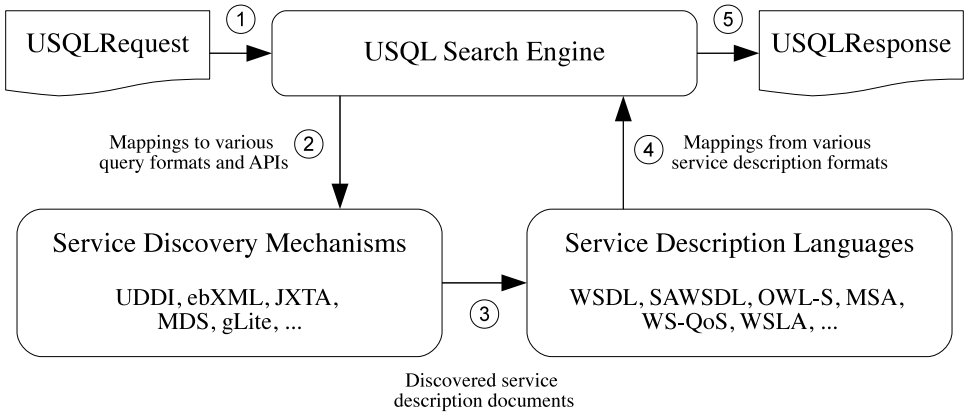

Figure 1: Using USQL to uniformly discover heterogeneous Web, P2P, and Grid services

To address these challenges, we propose a high-level, XML-based service query language called *Unified Service Query Language* (USQL), which is reported in this document. In a nutshell, USQL:

- supports the unified formulation of service discovery requests, and their responses, regardless of the desired service type, and/or search target;
- provides a rich set of elements supporting the expression and effective compilation of both functional and non-functional service requirements, which can be evaluated against heterogeneous service descriptions;
- is interoperable with an open set of service types, including but not limited to Web, P2P, and Grid services, and their respective technologies, thanks to its abstract and flexible design; and
- can be seamlessly extended to keep up with future developments in the respective technological areas.

The remainder of this report is dedicated to the detailed description of USQL. Along the lines, we also give simple examples of the language, which are intended to clarify the semantics and use of its various features.

2 Language Overview

USQL provides the necessary structures for the formulation of service type-independent service discovery request and response documents. Through a rich set of search criteria, service requesters are given the means to express their various requirements towards the desired service, irrespectively of the way it has been described, or the discovery mechanism of the broker where it has been published.

Figure 1 gives an overall picture of how USQL is applied to the discovery of heterogeneous services, and also illustrates its association with the various services technologies. As it can be seen in the figure, the discovery process involves the following steps:

1. The requester formulates the USQL request document
2. The search criteria of the USQL request document are appropriately mapped to the query formats and APIs supported by the various service brokers
3. The service brokers are searched, and the discovered service description documents are retrieved
4. The content of the retrieved service descriptions is appropriately mapped to the USQL search criteria to facilitate the matchmaking process

5. The results of the matchmaking process are consolidated into a USQL response document, which is returned to the requester

Thus, unification in the discovery of heterogeneous Web, P2P, and Grid services is achieved by the high level of abstraction of USQL, which makes it possible to support an open set of service description and discovery technologies, by means of appropriate mappings.

2.1 Why USQL?

As one might question the need for introducing a new language instead of using a standard query language for the purposes of service discovery, such as XQuery [3] or SPARQL [25], it is important to emphasize on their different scopes and purposes. USQL is used to express the requirements set by a service requester upon service discovery, as well as to include the information that is required for invoking the discovered services. XQuery is an expression language designed for processing and formatting XML data, which could be used in a complementary fashion by implementations of the USQL specification, in order to process XML-based service advertisements according to the requirements specified in a USQL request. SPARQL is a language used for constructing queries against resources semantically described with the use of RDF. As such, it could also be used complementary by the implementation of USQL, in order to retrieve information from RDF-based service advertisements.

2.2 Conceptual Model

The specification of USQL has been based on a high-level conceptual model. That model was based on the results of an extensive survey, which we conducted in the state-of-the-art of Web, P2P, and Grid services technologies. Its goal was to provide a unified conceptual service viewpoint tailored to the needs of unified service discovery. In other words, the model focuses on the description of service concepts and properties, which are of particular interest to service requesters, and describes them in a service type-agnostic manner.

The conceptual model is depicted in Figure 2. In general, a service provider provides one or more services, which are published in one or more brokers. Each broker has a set of properties, the acquisition of which is required for publishers/requesters to be able to access it. The names and corresponding values of these properties depend on the concrete broker type (e.g. UDDI, ebXML, JXTA, etc.).

A service belongs to a specific classification and realizes one or more capabilities. Each concrete capability exposes an interface, which describes the structure of the involved input and/or output messages. For the service to provide a capability, it will accept at most one input message (the request), and will return at most one output message (the response). Both input and output messages may be empty, or else they comprise a number of elements, each adhering to a specific data type. Data types are identified by their name and the namespace, which they have been defined in. Each service capability is also associated with a set of information necessary to support its remote invocation. The actual content of invocation details depends on the type of the respective service (e.g. Web, P2P, or Grid service).

A service capability may be associated with a specific resource, which is required for its realization. A resource is basically a group of resource properties, each having a specific data type. Service capabilities, input/output messages, message elements, resources, and resource properties can be described with the use of text, or additionally by means of semantic, ontology-based annotations.

Besides its functional aspects, a service is also characterized by its non-functional properties, specifically the various Quality-of-Service (QoS) dimensions. Each QoS property of a service has a name, which informs of its semantics, and a value. The latter may be additionally qualified (or constrained) by one or more related qualifiers. For instance, the price of a service is semantically qualified by the currency unit. Likewise, the availability, or reliability, of a service can be constrained by the minimum and maximum allowed values.

Most entities in the conceptual model of USQL are generic, and thus they can be easily customized to accommodate technology-specific properties. For instance, the *ServiceBroker* concept can be used to describe any type of search target. Likewise, the *QoSProperty* concept provides

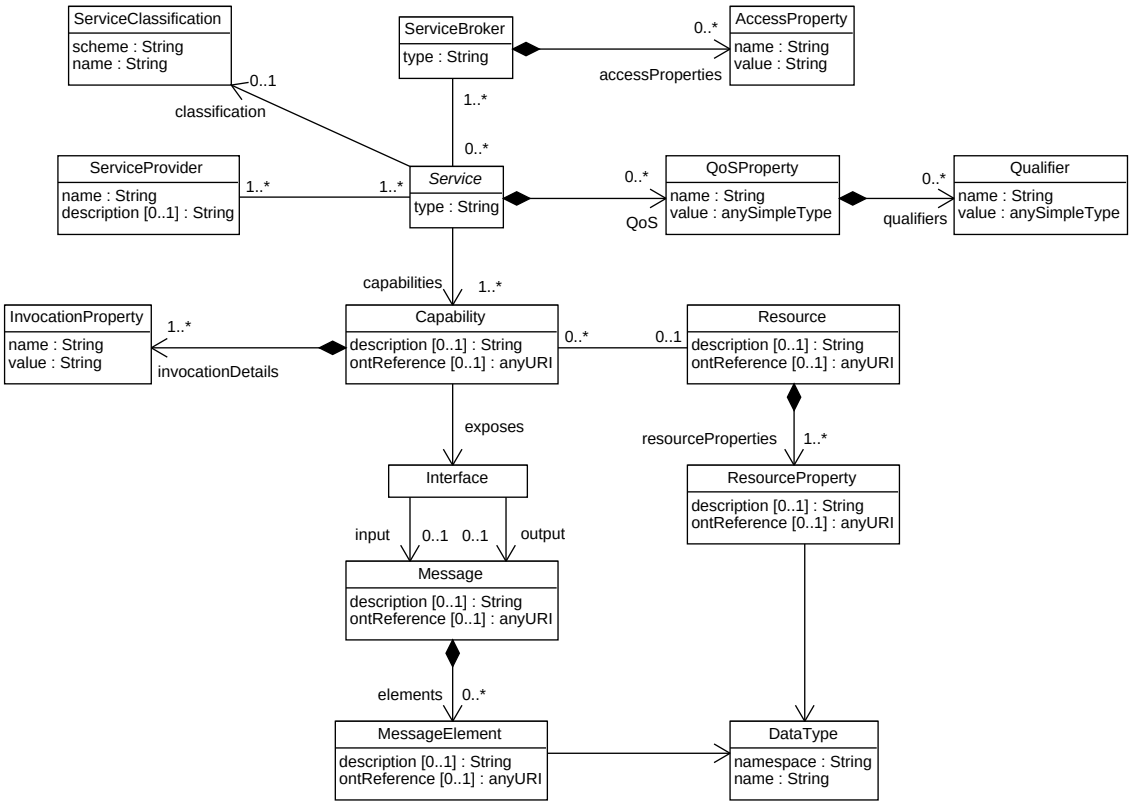

Figure 2: Conceptual model of the USQL language

a vehicle for describing an open set of application-level QoS attributes, such as the service availability, reliability, response time, cost, etc. Finally, the *InvocationProperty* entity provides generic support for the description of the service type-dependent invocation details of a service capability.

In addition to the above, the abstract *Service* entity can be easily extended, to accommodate additional properties. Thus, thanks to its extensible design and its high level of abstraction, the conceptual model of USQL attains resilience to changes and future developments in the areas of Web, P2P, and Grid services technologies. Further, it becomes much easier to conceptually accommodate additional properties and/or other emerging types of services in the future.

3 USQL Documents

The USQL specification has been formally defined with the use of the XML Schema. However, for the sake of convenience, we have chosen to use the EBNF notation [27] in this section, to present the language syntax.

USQL defines two types of documents, namely the *USQLRequest* and *USQLResponse*, the syntax of which is described in the following paragraphs.

3.1 The USQLRequest

The *USQLRequest* document is used for the formulation of service discovery requests, in a unified and technology-independent manner. Its detailed syntax is given below:

```
<USQLRequest id=xs:string
        [minMatch=xs:double]
        [maxResults=xs:int]>
        (<SearchCriteria>, <SearchTargets>)
</USQLRequest>
```

Each *USQLRequest* document is identified by a unique *id* and may optionally specify (a) the minimum degree of match (*minMatch*); and (b) the maximum number of matches that should be included in the search results (*maxResults*). The content of *USQLRequest* documents allows requesters to express a rich set of search criteria (*SearchCriteria*), as well as to explicitly specify the search target(s) (*SearchTargets*).

Let us proceed with the syntax of the included *SearchCriteria* element:

```
<SearchCriteria>
    (
        [<Providers>],
        [<Classifications>],
        [<Capability>],
        [<Interface>],
        [<Resource>],
        [<QoS>],
        {any}
    )
</SearchCriteria>
```

The *SearchCriteria* element acts basically as a container of all types of requirements that can be expressed in USQL. In compliance with the conceptual model, the reporting version of the USQL specification provides support for expressing requirements towards the desired service provider, classification, capability, interface, resource, and Quality-of-Service (QoS). Nevertheless, the *SearchCriteria* element can be extended to accommodate additional types of requirements, should that be necessitated by the user.

3.1.1 Search Filters (Hard Constraints)

The *Providers* and *Classifications* container elements can be regarded as filters, or *hard constraints*, and are optionally specified by the service requester to limit the number of search results. In particular, with the use of the *Providers* element, the service requester may set a filter on the provider of the desired service, by requiring that the desired service is provided by one of the specified service providers.

```
<Providers>
    {<Provider name=xs:string [description=xs:string]/>}-
</Providers>
```

In a similar fashion, the *Classifications* container may be used to specify a filter in regards to the category of the requested service.

```
<Classifications>
    {<Classification scheme=xs:string name=xs:string/>}-
</Classifications>
```

For each *Classification* element in the *Classifications* container, the requester needs to specify the corresponding classification's *scheme* and *name*. Thus, with the use of this abstracted structure, the USQL request may leverage various existing classifications and taxonomy schemes, e.g. UNSPSC, NAICS, etc.

In Listing 1, we illustrate the combined use of the *Providers* and *Classifications* filters in a USQL request.

In the example, the requester has constrained the query to services being provided by a company named "Medisystem", and classified as "Private specialized clinic services", according to the UNSPSC classification scheme.

```
<USQLRequest id="1">
  <SearchCriteria>
    <Providers>
      <Provider name="Medisystem"/>
    </Providers>
    <Classifications>
      <Classification scheme="UNv111201" name="85101502"/>
    </Classifications>
  </SearchCriteria>
  <SearchTargets/>
</USQLRequest>
```

Listing 1: Using the *Providers* and *Classifications* filters

```
<USQLRequest id="1">
  <SearchCriteria>
    <Capability ontReference="http://www.s3lab.com/healthcare.owl#GetMedication">
      Retrieve the list of medications of the patient identified by the given id
    </Capability>
  </SearchCriteria>
  <SearchTargets>
</USQLRequest>
```

Listing 2: Expressing the desired service capability

3.1.2 Capability Requirements

In the most common usage scenarios, a *USQLRequest* document is expected to support users in specifying the desired service capability. Such requirement is expressed with the use of the *Capability* element, the syntax of which is shown below:

```
<Capability [ontReference=xs:anyURI]>
    [<Description>]
</Capability>
```

As it can be seen, USQL provides two ways for describing the desired capability of a requested service. In the most common case, users may use the *Description* element to specify the service capability through free text. Alternatively, or in a complementary manner, the *ontReference* attribute may be applied to semantically annotate the requested capability with a concept defined in some formal ontology. Along the lines of the previous example, Listing 2 demonstrates both ways of specifying the service capability in a *USQLRequest* document.

As it can be seen, the requested service capability is related to the retrieval of medication information for a specific patient. The capability has been described both via free text and semantic, ontology-based annotation.

3.1.3 Interface Requirements

Though service discovery based on the desired capability may suffice in some cases, e.g. at early stages of the development process where requirements are formulated, assessed, and iteratively elaborated, it is not generally sufficient to cover the complete development lifecycle. As developers start to design their service-oriented solution, the interfaces of the constituent services attain importance. USQL adopts a simple model for the formulation of more structured requirements towards the service interface. Specifically, interface related search criteria are set on the anticipated service input and/or output.

```
<Interface>
    ([<Input>], [<Output>])
</Interface>
```

The requested service input and output take the form of messages consisting of zero or more elements. The USQL syntax for the formulation of input/output messages has as follows:

```
<USQLRequest id="1">
  <SearchCriteria>
    <Capability>...</Capability>
    <Interface>
      <Input>
        <Element>
          <Description>The patient identifier</Description>
          <Type namespace="http://www.w3.org/2001/XMLSchema" name="string"/>
        </Element>
      </Input>
      <Output>
        <Element>
          <Description>The patient medication list</Description>
          <Type namespace="services.medisystem.ro" name="XListOfMedication"/>
        </Element>
      </Output>
    </Interface>
  </SearchCriteria>
  <SearchTargets/>
</USQLRequest>
```

Listing 3: Specifying the desired interface of the requested service

```
<Message [ontReference=xs:anyURI]>
    ([<Description>], {<Element>})
</Message>

<Element [ontReference=xs:anyURI]>
    ([<Description>], [<Type namespace=xs:string name=xs:string>])
</Element>
```

In a way similar to the *Capability* element, requesters may describe the input/output message, as well as its consituent elements, by means of their corresponding *ontReference* attribute and/or the *Description* element. In addition, the data type of each message element may be specified through the optional *Type* element. Each data type is distinguished by its name and the namespace it belongs to. The genericity of the *Type* element renders USQL independent from the various schemes (e.g. XML Schema, JSON, etc.) that are currently used by service description technologies to define the various data types.

In continuation to our running example, Listing 3 demonstrates the use of the *Interface* element in a USQL request to specify the desired input and output of the "Get Medication" service.

3.1.4 Resource Requirements

As it is showcased by the latest developments in the area of Grid-based services, a service may be tightly coupled with a specific resource in delivering its functionality. To efficiently discover such services, it is therefore important that the query language allows requirements to be set on the resource type and its properties. USQL provides support for expressing such requirements by means of the *Resource* element.

```
<Resource [ontReference=xs:anyURI]>
    ([<Description>], {<ResourceProperty>}-)
</Resource>

<ResourceProperty [ontReference=xs:anyURI]>
    ([<Description>], [<Type namespace=xs:string name=xs:string>])
</ResourceProperty>
```

The *Resource* element syntactically resembles the *Message* element. Indeed, a requested resource can be described with free text and/or ontology-based annotation. Further, a resource will contain one or more *ResourceProperty* elements, which may also be described in the same way. Additionally, in analogy to the message elements, the requester may specify the desired data type of each resource property.

```
<USQLRequest id="1">
  <SearchCriteria>
    <Capability>...</Capability>
    <Interface>...</Interface>
    <QoS>
      <QoSProperty name="Availability" value="0.99" priority="2">
        <Qualifier name="min" value="0" />
        <Qualifier name="max" value="1" />
      </QoSProperty>
      <QoSProperty name="Cost" value="5" priority="1">
        <Qualifier name="Currency" value="EUR" />
        <Qualifier name="Context" value="day" />
      </QoSProperty>
    </QoS>
  </SearchCriteria>
  <SearchTargets />
</USQLRequest>
```

Listing 4: Expressing requirements towards multiple QoS properties

3.1.5 QoS Requirements

The eventual selection of a service over a set of candidate services with similar functionality and/or interface is to a great extent influenced by its non-functional properties, and especially those, which define its application-level qualitative characteristics such as availability, reliability, response time, etc. USQL defines a generic structure that can be leveraged to express constraints towards an open set of application-level QoS properties.

```
<QoS>
    {<QoSProperty>}
</QoS>

<QoSProperty name=xs:string value=xs:string [priority=xs:int]>
    {<Qualifier name=xs:string value=xs:string>}
</QoSProperty>
```

Each specified QoS property in the USQL request is characterized by a name, has a specific value, and may be assigned a priority in the form of an integer value. Prioritization of different QoS properties is important to accurately reflect the user preferences in the USQL request. Further, each QoS property may specify a number of qualifiers in order to clearly define and/or constrain the semantics of its value.

In the example that is displayed in Listing 4, the *QoSProperty* element has been utilized to express requirements towards the availability and cost of the requested service. In the case of the Availability criterion, the *Qualifier* elements have been introduce to define the value set, which the indicated value has been selected from. Thus, the given value '0.99' implies that the service should be at least 99% available. In the case of the Cost criterion, the requester has introduced qualifiers to specify the currency metric of the given value. as well as the context which the requested price applies to.

The *QoSProperty* and *Qualifier* constructs actually act as generic placeholders and can be populated with content of many forms. Although in the example the QoS property types and qualifiers have been specified with the use of simple words, it is also possible to use concepts extracted by specialized QoS ontologies, such as the QoSOnt [7] for instance, in order to better align and utilize USQL for service discovery within semantically enhanced environments.

3.1.6 Search Targets

In many cases, service requesters want to direct their queries in specific private registries, networks, etc., instead of looking up for suitable services out the Internet. Such cases could be related to the security restrictions imposed by a company, or the explicit knowledge of developers on where to find the services, e.g. due to service level agreements established between their company and its partners. USQL addresses this need by allowing users to explicitly specify the targets of their

```
<USQLRequest id="1">
  <SearchCriteria>...</SearchCriteria>
  <SearchTargets>
    <SearchTarget id="uddi@jemini.di.uoa.gr" type="uddi">
      <AccessProperty name="InquiryURL"
          value="http://jemini.di.uoa.gr/juddi/inquiry"/>
    </SearchTarget>
    <SearchTarget id="p2p@s3lab" type="jxta">
      <AccessProperty name="PeerGroupID"
          value="jxta:uuid-4d6172676572696e204272756e6f202009"/>
    </SearchTarget>
  </SearchTargets>
</USQLRequest>
```

Listing 5: Setting heterogeneous brokers as search targets for the query

service discovery request. More specifically, the language defines a generic structure, which is used for this purpose and conveniently accommodates the specification of any type of service discovery target.

```
<SearchTargets>
    {<SearchTarget>}
</SearchTargets>

<SearchTarget id=xs:string type=xs:string>
    {<AccessProperty name=xs:string value=xs:string>}
</SearchTarget>
```

Each *SearchTarget* element has an *id* that is expected to uniquely identify the corresponding broker within the search engine, whereas the *type* attribute is used to specify the kind of the broker (e.g. UDDI, ebXML, etc.). A *SearchTarget* may contain a number of properties (*AccessProperty* element), which are basically name-value pairs and are required to gain access and query the corresponding broker.

Thanks to its genericity, the *SearchTarget* element can express any type of broker. In the snippet of Listing 5, we demonstrate this capability by specifying as search targets two heterogeneous brokers: a UDDI registry, and a JXTA peer group.

3.2 The USQLResponse

The *USQLResponse* document is used to convey the search results deriving from the execution of a given *USQLRequest*. Its syntax is given below:

```
<USQLResponse queryId=xs:string
        queryDuration=xs:long
        numberOfMatches=xs:positiveInteger
        truncated=xs:boolean>
    {<MatchedService>}
</USQLResponse>
```

A *USQLResponse* document is associated with its corresponding *USQLRequest* through the *queryId* attribute. Further, through a number of additional attributes, the requester is informed of the query execution duration (*queryDuration*), the number of matching services found (*numberOfMatches*), and receives an indication of whether the results were truncated or not (*truncated*), as a response to the *maxResults* specified in the *USQLRequest*.

Search results included in the *USQLResponse* take the form of zero or more *MatchedService* elements. Each such element corresponds to a service capability that was found to meet the user requirements, and its overall rank passed the *minMatch* threshold specified in the *USQLRequest*. The following syntax is used to describe a matched service entity:

```
<USQLResponse queryId="1" queryDuration="4561" numberOfMatches="1" truncated="false">
  <MatchedService rank="1.0" type="WebService">
    <Name>PatientMedications</Name>
    <Provider name="Medisystem"/>
    <InvocationDetails>
      <Property name="Service" value="PatientMedications"/>
      <Property name="Port" value="PatientMedicationsSoap"/>
      <Property name="Operation" value="GetAll"/>
      <Property name="WSDL" value="http://services.medisystem.ro/InternalWS/
          PatientMedications.asmx?WSDL"/>
    </InvocationDetails>
  </MatchedService>
</USQLResponse>
```

Listing 6: Example of a matched Web service

```
<USQLResponse queryId="2" queryDuration="3973" numberOfMatches="1" truncated="false">
  <MatchedService rank="1.0" type="JXTAService">
    <Name>SecondOpinionService</Name>
    <Provider name="PeerX"/>
    <InvocationDetails>
      <Property name="PipeType" value="JxtaUnicast"/>
      <Property name="PipeId" value="urn:jxta:uuid-9CCCDF..."/>
    </InvocationDetails>
  </MatchedService>
</USQLResponse>
```

Listing 7: Example of a matched JXTA P2P service

```
<MatchedService rank=xs:double type=xs:string>
    (<Name>, [<Description>], [<Provider>], <InvocationDetails>)
</MatchedService>

<InvocationDetails>
    {<Property name=xs:string value=xs:string>}-
</InvocationDetails>
```

Each *MatchedService* denotes the type of the service (*type* attribute), and is given a *rank* that represents its degree of similarity to the search criteria of the corresponding *USQLRequest*. The rank values range between 0 and 1: the higher the rank value is, the more suitable the specific service is with respect to the user requirements. Further, each *MatchedService* element contains the name of the corresponding service, and may optionally include a textual description as well as information about its provider. Finally, the *InvocationDetails* container provides all required information to enable invocation of the corresponding service capability. Such information takes the form of a collection of name-value properties. Thanks to its generality, the *InvocationDetails* element can be seamlessly customized depending on the type of service.

Listings 6 - 8 illustrate the use of the *MatchedService* element to represent three different types

```
<USQLResponse queryId="3" queryDuration="5924" numberOfMatches="1" truncated="false">
  <MatchedService rank="1.0" type="GridService">
    <Name>DrugInfo</Name>
    <Provider name="DemoVO"/>
    <InvocationDetails>
      <Property name="Service" value="DrugInfo"/>
      <Property name="Port" value="DrugInfoSoap"/>
      <Property name="Operation" value="GetDrugAnalysis"/>
      <Property name="WSDL" value="http://grid.example.org/services/DrugInfo?wsdl"/>
      <Property name="EndpointReference" value="https://grid.example.org/resources/drug/
          abcdef"/>
    </InvocationDetails>
  </MatchedService>
</USQLResponse>
```

Listing 8: Example of a matched Grid service

Table I: Interoperability of USQL with various service discovery technologies

USQL Search Criteria	**Service Discovery Technologies**				
	UDDI	ebXML	JXTA	MDS	gLite
Providers	X	X	X		X
Classifications	X				X
Capability	X	X	X	X	X
Interface	X				
Resource				X	X
QoS					

Table II: Compatibility of USQL with various service description technologies

USQL Search Criteria	**Service Description Technologies**						
	WSDL	WSRP	SAWSDL	OWL-S	MSA	WS-QoS	WSLA
Providers				X	X		
Classifications			X	X			
Capability	X		X	X	X		
Interface	X		X	X			
Resource		X					
QoS						X	X

of matched services: a Web service, a JXTA P2P service, and a Grid service.

4 Interoperability and Compatibility

The generality of USQL and the high level of abstraction, which it has been defined at, ensure its interoperability with a variety of service discovery mechanisms, and its compatibility with various service description languages. Table I outlines the mappings of USQL search criteria to a non-exhaustive set of Web, P2P, and Grid service discovery technologies. In general, we can make the following observations:

- USQL can be applied to various heterogeneous environments, in terms of the technologies being used for service discovery.
- With the use of USQL, service requesters can express their search criteria in a unified way, regardless of the various service discovery technologies. The search criteria can then be appropriately mapped to the corresponding query formats and interfaces that each service discovery technology supports.
- Compared to the various Web, P2P, and Grid service discovery technologies, USQL provides a much richer interface for query formulation. Indeed, each one of the investigated service discovery technologies support only a subset of the search criteria that can be expressed with USQL.
- None of the investigated service discovery technologies provides built-in support for QoS-based search criteria, despite their significant role in service sorting and selection.

In Table II, we also outline the compatibility between USQL and a non-exhaustive set of service description technologies. The table specifically shows which of the USQL-supported search criteria the contents of each service description technology can be mapped to. In this case, we can make the following observations:

- USQL can be used to formulate service discovery queries in various heterogeneous settings, in terms of the technologies being used for service description.

- Each investigated service description technology supports the description of only a subset of the service properties, which can be constrained by the USQL search criteria. This fact showcases the superiority of USQL as a query language, in comparison to various other approaches, which formulate their queries by utilizing the same language that is used for the description of services.
- Seemingly, service providers need to use more than one technologies to adequately describe the various properties of their services. In contrast, with the use of USQL, service requesters are lifted off of this burden, as they are given a high-level, unified interface for expressing their requirements.

Concluding this section, we would like to remind that, USQL can be easily extended to accommodate additional types of search criteria. Thanks to such flexibility, it becomes possible to establish an even wider coverage of currently existing service discovery and description technologies. For instance, appropriate extensibility elements could be introduced into the *SearchCriteria* element of the *USQLRequest* document, to support requirements towards pre-conditions and post-effects of the desired service, as these properties can be found in a number of service descriptions (e.g. services described in OWL-S).

5 Related Work

Over the last years, much research has revolved around service discovery producing significant results. Various discovery mechanisms were proposed and many ways of formulating service queries were established. An early effort to support the expression of syntactic and semantic requirements in service discovery is described in [29]. Therein, the LARKS framework provides a number of structures used for applying syntactic and semantic search criteria in service discovery. Overall, the approach complies with a rather static service description schema. Our work overcomes such limitation by relying on a high-level conceptual meta-model built to comply with most of the existing service-oriented technologies and standards.

In [4], an intuitive XML-based language called Quilt was proposed, which brought together concepts from many popular query languages, such as SQL, XPath, etc.. Quilt is characterized by its expressiveness and comprises a wide set of elements, expressions, and functions. Given its support of formulating queries against a broad spectrum of XML-based information sources, Quilt can also be put to practice in the field of service discovery, where most of the service description formats are XML-based. However, once it is formulated, a Quilt query can only target descriptions abiding by a specific format. On the contrary, our approach totally abstracts queries from their targets. USQL is independent of the way service descriptions have been constructed, and compatibility with the various service description protocols is obtained by means of appropriate mappings, which nevertheless are considered an implementation issue.

The Web Service Discovery Architecture (WSDA) proposed in [11] defines a simplified version of WSDL for service description (namely SWSDL) and then leverages the expressiveness of XQuery in order to formulate queries against such descriptions. Still, the resulting XQuery expressions are bound to the schema of the target SWSDL documents and consequently the same query cannot be evaluated against other types of service descriptions. Thus, the WSDA approach lacks that kind of flexibility, which instead is supported by the USQL approach.

Adopting an Architecture-driven Service Discovery (ASD) approach, the ASD framework described in [16] expresses service queries and their respective results with the use of UML 2.0. More specifically, a UML profile is introduced that defines a set of elements and query constraints, which, like USQL search criteria elements, can be given different weights (i.e. priorities). Then, graph-based matchmaking is applied with the use of a set of distance measure functions and results are ranked according to their similarity. Our approach differs from the ASD framework in that it provides support for expressing semantic and QoS requirements, and goes beyond Web services, as it already supports the formulation of discovery requests for P2P and Grid services.

In [15], the authors describe the Process Query Language (PQL) that is used for expressing process requirements in service discovery. PQL queries are specified by looking for particular entity-relationship patterns, using a small set of clause types. Although PQL is expressive and

can improve precision and recall in service retrieval, it suffers from two shortcomings: first, PQL queries can only be matched against service advertisements abiding by a specific type of process model annotations; second, the PQL enactment requires that these, rather complex, annotations are applied by service providers. As opposed to this approach, USQL retains its independence from various service description protocols and standards. Thereby, USQL queries can be matched against a wide range of service advertisements and annotations, without necessarily imposing specific kinds of annotations on service providers.

In [18], the authors propose a peer-to-peer overlay network upon which service discovery can be applied with substantial scalability and robustness. Within the proposed framework, WSDL service descriptions are further annotated with meta-data, provided by the service provider, and published on a specific peer, based on a hash key mechanism. Accordingly, XML-based queries are mapped to specific hash keys and thus forwarded to the appropriate peer hosting the desired service. In that approach, operators can be explicitly specified in a query; however, requirements may only be expressed at the syntactic level, resulting in poor expressiveness. The effectiveness of such an approach could be enhanced by the use of a rich service query language such as USQL, which provides the means to express semantics and quality conditions within service queries. In this regard, the two approaches can be considered complementary.

Due to the lack of appropriate query language specifications, many approaches in service discovery assume the use of the same format for both service descriptions and discovery requests [28]. Such assumption brings certain limitations to the expressiveness of the resulting queries, and also imposes the need to deal with the underlying details of that particular technology. The Unified Service Discovery Framework, presented in [13], provides appropriate models for service description and discovery and is applicable to a wide range of settings where service discovery is required, including Grid environments. The proposed framework supports QoS and contextual information both in service description and discovery process, and utilizes the XQuery language to construct the service discovery requests. This choice renders the resulting queries tightly coupled with the structure of the service advertisements. As opposed to all these approaches, USQL has been defined at a higher level with the primary aim of being independent of the various heterogeneous service description formats.

6 Discussion

In this report, we presented the Unified Service Query Language (USQL), as a solution towards the interoperability of heterogeneous types of services at the discovery level. USQL provides a set of high-level search criteria structures and the abstractions necessary for constructing service type-agnostic and service broker type-agnostic queries. Moreover, it retains its independence from the various service description languages, and can be easily extended to accommodate other types of specialized search criteria. USQL may be used to discover services at any stage of the development process, as well as during execution of a service-oriented application. The language specification has been fully implemented by the *PROTEUS* search engine [23], which provides an open framework for the discovery of Web, P2P, and Grid services. Further, it was used as the query language of choice by the DIRE [2] and PYRAMID-S [24] frameworks.

Just like human languages, computer languages are alive and need to evolve over time, in order to adapt to emerging needs. We are constantly evaluating the USQL specification against recent developments in service-oriented computing, in order to enrich the language with additional features, and render it a long-term, viable solution in service discovery. Such enhancements could include support for semantically expressing preconditions and post effects in the USQL request, elements for specifying context and location-based requirements, etc. We are also investigating the potential use of USQL for the discovery of additional types of services, such as RESTful services , OSGi services, etc.

Bibliography

[1] S. Andreozzi, M. Sgaravatto, and C. Vistoli. Sharing a Conceptual Model of Grid Resources and Services. In *Proceedings of the Conference on Computing in High Energy and Nuclear Physics (CHEP 2003), La Jolla, CA, USA*, March 2003.

[2] Luciano Baresi, Matteo Miraz, and Pierluigi Plebani. A flexible and semantic-aware publication infrastructure for web services. In *CAiSE*, volume 5074 of *Lecture Notes in Computer Science*, pages 435–449. Springer, 2008.

[3] S. Boag, D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie, and J. Simeon. XQuery 1.0: An XML Query Language. Technical report, W3C, November 2006.

[4] Donald D. Chamberlin, Jonathan Robie, and Daniela Florescu. Quilt: An XML query language for heterogeneous data sources. In Dan Suciu and Gottfried Vossen, editors, *WebDB (Selected Papers)*, volume 1997 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 2000.

[5] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Service Description Language (WSDL) 1.1 W3C Note. Technical report, World Wide Web Consortium (W3C), 15 March 2001.

[6] Luc Clement, Andrew Hately, Claus von Riegen, and Tony Rogers. UDDI Spec Technical Committee Draft 3.0.2. Oasis committee draft, OASIS, 2004.

[7] Glen Dobson, Russell Lock, and Ian Sommerville. QoSOnt: a QoS ontology for service-centric systems. In *EUROMICRO-SEAA*, pages 80–87. IEEE Computer Society, 2005.

[8] J. Farrell and H. Lausen. Semantic Annotations for WSDL and XML Schema. Online at http://www.w3.org/TR/sawsdl/, August 2007. W3C Recommendation.

[9] Ian Foster. Globus toolkit version 4: Software for service-oriented systems. In *IFIP International Conference on Network and Parallel Computing*, number 3779 in LNCS, pages 2–13. Springer-Verlag, 2005.

[10] Li Gong. JXTA: a network programming environment. *Internet Computing, IEEE*, 5(3):88–95, 2001.

[11] Wolfgang Hoschek. The Web Service Discovery Architecture. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–15, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[12] International Organization for Standardization. Electronic business extensible markup language (ebxml) part 3: Registry information model specification (ebrim). ISO/TS 15000-3:2004, May 2004.

[13] B. Jin, L. Zhang, and Z. Zang. A unified service discovery framework. In *GCC*, pages 203–209, 2007.

[14] Alexander Keller and Heiko Ludwig. The WSLA framework: Specifying and monitoring service level agreements for web services. *Journal of Network and System Management*, 11(1), 2003.

[15] M. Klein and A. Bernstein. Toward high-precision service retrieval. *IEEE Internet Computing*, 8(1):30–36, Jan-Feb 2004.

[16] Alexander Kozlenkov, George Spanoudakis, Andrea Zisman, V. Fasoulas, and F. Sanchez. Architecture-driven Service Discovery for Service Centric Systems. *International Journal of Web Services Research*, 4(2):82–113, 2007.

[17] E. Laure, S.M. Fisher, A. Frohner, C. Grandi, P. Kunszt, A. Krenek, O. Mulmo, F. Pacini, F. Prelz, J. White, M. Barroso, P. Buncic, F. Hemmer, A. Di Meglio, and A. Edlund. Programming the Grid with gLite. *Computational Methods in Science and Technology*, 12(1):33–45, 2006.

[18] Y. Li, F. Zou, Z. Wu, and F. Ma. PWSD: A scalable web service discovery architecture based on peer-to-peer overlay network. In *APWeb*, volume 3007 of *Lecture Notes in Computer Science*, pages 291–300. Springer, 2004.

[19] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. OWL-S: Semantic Markup for Web Services. Internet (http://www.w3.org/Submission/OWL-S), November 2004. W3C Member Submission.

[20] Nilo Mitra. SOAP version 1.2 Part 0: Primer W3C Recommendation, June 2003.

[21] W. Nejdl, B. Wolf, C. Qu, M. Sintek, A. Naeve, M. Nilsson, M. Palmer, and T. Risch. EDUTELLA: A P2P networking infrastructure based on RDF. In *WWW2002*, Honolulu, Hawaii, USA, May 2002.

[22] OASIS. WSRF, Web Services Resource Framework.

[23] Michael Pantazoglou. *Development of a language and its enacting engine for the unified discovery of heterogeneous services*. PhD thesis, Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, Hellas, 2009.

[24] Thomi Pilioura and Aphrodite Tsalgatidou. Unified publication and discovery of semantic web services. *ACM Transactions on the Web*, 3(3), June 2009.

[25] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. Technical report, W3C, October 2006.

[26] D. Roman, U. Keller, H. Lausen, R. Lara J. de Bruijn, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web Service Modeling Ontology. *Applied Ontology*, 1(1):77–106, 2005.

[27] R. S. Scowen. Extended BNF - a generic base standard. In *Proceedings of the 1993 Software Engineering Standards Symposium*. IEEE Computer Society Press, 1993.

[28] E. Stroulia and Y. Wang. Structural and semantic matching for assessing web-service similarity. *International Journal of Cooperative Information Systems*, 14(4):407–438, 2005.

[29] K. Sycara, M. Klusch, S. Widoff, and J. Lu. Dynamic service matchmaking among agents in open information environments. *ACM SIGMOD Record, Special Issue on Semantic Interoperability in Global Information Systems*, 28(1):47–53, 1999.

[30] M. Tian, A. Gramm, H. Ritter, and J. Schiller. Efficient selection and monitoring of qos-aware web services with the ws-qos framework. In *WI '04: Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 152–158. IEEE Computer Society, 2004.

[31] V. Tosic, K. Patel, and B. Pagurek. WSOL - Web Service Offerings Language. In *WES*, volume 2512 of *Lecture Notes in Computer Science*, pages 57–67. Springer, 2002.

[32] A. Tsalgatidou, G. Athanasopoulos, M. Pantazoglou, A. J. Berre, C. Pautasso, R. Gronmo, and H. Hoff. *Unified discovery and composition of heterogeneous services: The SODIUM approach*. Information Systems. MIT Press, 2008.