Specification of the Unified Service Query Language (USQL)

A. Tsalgatidou, M. Pantazoglou, G. Athanasopoulos

Dept. of Informatics & Telecom. University of Athens (NKUA), Athens 15784, Greece
{atsalga, michaelp, gathanas}@di.uoa.gr

Abstract. This report presents the specification of the Unified Service Query Language (USQL). USQL is an XML-based language used for the unified discovery of heterogeneous services. It is used for uniformly describing service search criteria, which may be syntactic, semantic, as well as Quality-of-Service requirements. The resulting service request can then be executed against the various heterogeneous service registries, with the use of the appropriate query engine, which implements the USQL language and is thus capable of processing USQL request documents. The requested information of the services matching the submitted requirements is collected, appropriately formatted into a USQL response document, and finally returned back to the user, after the query process has completed.

1. Introduction

The *Unified Service Query Language (USQL)* enables users to express their service requirements in a service type-independent manner. These requirements can then be used by an appropriate query engine that implements the language, for the identification of existing, registered services. USQL is intended to be a cross-platform query language that enables the discovery of various types of services in a unified manner; however, for the time being, it will only provide support for Web, Peer-to-Peer (P2P), and Grid Services. Nevertheless, USQL has been designed to be as abstract, open and extensible as possible, so that it can easily accommodate other types of services as well.

Briefly, USQL provides formalism for the description of:

- unified service discovery requests
- unified service discovery responses

The ability to describe the request messages in a unified manner, as far as the heterogeneity of services and their corresponding registries are concerned, enables users to abstract from technical details regarding specific underlying protocols and standards and focus on the detailed description of service requirements, as they are determined by the needs of their applications.

1.1 Current Status in Service Discovery

The way in which service lookup is performed in the areas of Web, P2P, and Grid services depends primarily on their existing registry/repository infrastructure and supporting protocols. By now, Web services seem to be more mature in the field of service discovery, although there is no unified language and/or mechanism to enable searching against different kinds of registries, such as UDDI [1] and ebXML [2], to name a few. Rather, each one of these types of registries provides its own protocols and mechanisms for communicating with it.

Latest developments in the world of Grid signify a convergence among Grid and Web services. With the emergence of WSRF [4], a Grid service interface can be described with the use of WSDL, which in turn can be published in a UDDI registry.

On the other hand, P2P services do not directly support the discovery functionality in a straightforward way like the one followed by the Web services paradigm. The JXTA [3] protocols specify a discovery mechanism which is used to discover any kind of resources in a P2P network, i.e. peers, peer groups, bindings (pipes), services etc. through their respective advertisements. Nevertheless, there are no repositories upholding the advertisements of the various resources. Instead, a semi-decentralized infrastructure facilitates the discovery process with the use of indexes (rendezvous peers).

In general, the major shortcoming when looking for available P2P services in a P2P network is the lack of adequate service descriptions. Service interfaces are roughly described, while semantics and QoS properties are completely absent. This situation hinders the service discovery process, and is also apparent – although not to the same extent – in Web and Grid services as well.

Nevertheless, USQL is not affected by the lack or not of adequate information in service descriptions. Should the necessary service description extensions emerge and be adopted by the various bodies, organizations and communities, the language will enable users to formulate requests against them.

1.2 Need for Unification of Service Discovery

The Internet bursts with a large number of heterogeneous services which employ different/incompatible architectural models, protocols, and standards for service discovery, as well as for service description and composition. However, there is no infrastructure, in terms of tools and languages, available for facilitating the integration and interoperability of such services. It is undoubtedly expected in the near future, that the Service-Oriented Development (SOD) community will inevitably call for standards-based methodologies, languages and tools that will enable the development of applications composed of services of a variety of technologies, such as Web, P2P, and Grid services. USQL is the first step towards enabling the searching of heterogeneous services in a unified and standard-based manner. The results of such a search may then be used for the development of service compositions, regardless of the nature of the constituent components.

1.3 Challenges and Requirements

The various types of existing service registries enable requestors to perform queries against them with the use of specific APIs. However, since each type of registry introduces its own API and potentially supports different protocols/standards, heterogeneity is a major issue for users and/or applications that need to apply service discovery in a unified manner. Moreover, most of the offered APIs confine users, regarding the kind of information that is used for the formation of the queries. Most types of registries do not provide direct support for semantic-based or QoS-based service search, and this limitation has a serious impact on the accuracy of the query results.

USQL aims at providing a standard-based API for accessing and querying various heterogeneous registries, and in addition enables the incorporation of semantic and QoS search criteria in the queries. Besides these features, there are a number of challenging objectives that drive the specification of the language and motivate our work:

- **Simplicity.** The language should remain simple enough, so as to be understandable and usable by a wide range of users, from novice ones to IT professionals. Although described by an XML grammar, the USQL documents should be comprehensible by humans.
- **Expressiveness.** The language should cater for enabling its users to express their service requirements in a consistent and rich way.
- **Transparency.** The language should be standards-based, meaning that it should take into consideration and exploit the various existing registry and service description standards and protocols, without requiring from its users to have knowledge of their underlying structure. Users of the USQL should be able to compose their query documents, and process the corresponding responses, without having to worry about technical aspects and details.
- **Extensibility.** The language should be abstract enough, in order to accommodate the different kinds of services in a unified manner. It should also be open and extendable, so as to allow the definition of extra features deriving from other kinds of services that need to be supported, without affecting the already defined elements and supported types of services.

Moving beyond these obvious objectives, the specification of the USQL language has heavily relied on a set of advanced requirements for service discovery, which are listed as follows:

- **Abstraction from technical details.** Service requesters want to focus only on describing their application-specific requirements, instead of acquiring technical details regarding the services they look for.
- **Service type-independent service requests.** In a Service-Oriented Development (SOD) environment, users do not care for the actual type of the service that fulfils a specific task. They only care for the service to meet to their functional and non-functional requirements.

- **Rich syntactic search criteria.** Existing service advertisements most usually contain syntactic information. Thus, a query language should provide requesters with adequate features that would enable expressing syntactic criteria in a rich fashion. Features such as case sensitivity or wildcards are required.
- **Semantically-enhanced service discovery.** Users need to be provided with the appropriate languages and tools, so as to be able to express their service requirements consistently. Use of semantics will greatly contribute in capturing the user needs in service discovery.
- **QoS-based service discovery.** Quality-of-Service (QoS) is imperative in real world service-oriented applications and determines the services to be used. Thus, users would like to be able to express QoS requirements, so that the discovered services meet specific QoS dimensions, such as availability, reliability, processing time, and price.
- **Flexibility in relaxing/constraining service requirements.** Naturally, requesters want to have full control over the formulation and stringency of their service requests. Thus, they want to be provided with the appropriate features that will enable them to relax or constrain their service requirements, according to their needs and/or the time when the service discovery process takes place (i.e. design-time or run-time).
- **Presentation of the results.** Like in the case of a service request, users want to control and to be able to pre-specify the contents of the respective response. Control involves the type of additional, human-intended information conveyed by a service discovery response (e.g. the provider name, or certain QoS aspects).
- **Sorting of results.** It's imperative to facilitate the users with control structures that enable them to arrange the order of the returned results according to specific information fields. The ordering of the returned results facilitates the selection of an appropriate service among a list of candidate services.
- **Service lookup in specific registries.** It is required that a service requester is provided with the ability to explicitly state the target registries for his/her service request, in cases where he/she is not interested in looking to the outside world for available services, but rather wants to search in a number of already known (and possibly trusted) registries.
- **Requirements at the service, operation and message level.** Given the fact that services usually offer functionality through a set of operations by exchanging well defined messages, it becomes natural for a service requester to express his/her requirements at the operation and message level, in addition to the service level. For instance, it is particularly useful for a requester to be able to search for services that provide specific a specific operation, compliant with a required operation signature.
- **Search criteria with different priority levels.** Naturally – and especially at early design time – not all requirements have the same priority/significance to the requester. He/she then must be able to explicitly state which requirements he/she considers to be of high importance and which not. For example, the operation signature requirement might not be as important as the service provider or availability requirement.

- **Smooth handling of null values.** Given the rather poor service descriptions of existing services nowadays, it is possible that service advertisements do not contain part of the information constrained by the search criteria of a service query. For example, though the desired price of the requested service has been explicitly specified in the query, the corresponding service advertisements might not provide price information. Thus, a mechanism must be provided to the user, so that services with advertisements having null values, which nevertheless match the rest of the search criteria, will also be accepted in the response.
- **Quantification of service matching / Service ranking.** For a requester to select the most appropriate service among a list of matching solutions, it is imperative that the degree of match for each candidate service is specified. Each service entity in a service discovery result, along with its offered operations, must be assigned with an appropriate value that indicates their degree of match, with respect to the initial service requirements.
- **Verification of matching services.** A strong criterion for selecting a candidate service in real world scenarios is whether that service can be trusted or not. It is important that resulting services convey information about their verification status (i.e. whether they have been verified or not).
- **Rich-content service results.** The data conveyed by a service entity in the service discovery result should not only comprise information necessary for the service to be invoked. Rather, it should be possible that human-intended information is also included (e.g. the service provider, textual description of the service, etc.).
- **Handling of errors during service discovery.** Users need to be notified of potential errors that prevented the successful execution of their service request.

In the next sections, we will describe how the present specification meets all the aforementioned requirements.

2. Language Overview

This section provides an overview of the USQL language, and discusses its structure and supported concepts.

2.1 Why USQL?

USQL is an XML-based language [6] and its grammar is defined with the use of the XML Schema [7]. As one might question the need for introducing a new language instead of using a standard query language, such as XQuery [5] or SPARQL [10], we would like to emphasize on their different scopes. USQL is used to *express the requirements set by a service requester as well as the information that is required for invoking the discovered services*. Thus, a USQL message can be considered as a description document of these requirements. On the other hand, XQuery is an expression language designed for processing and formatting XML data, which could be used in a complementary fashion by the implementation of the USQL specification

(i.e. the USQL Engine) in order to process XML-based service advertisements according to the requirements specified in a USQL query. Similarly, SPARQL is a language used for constructing queries against resources semantically described with the use of RDF 15. As such, it could also be used complementary by the implementation of USQL, in order to retrieve information from RDF-based service advertisements. All in all, USQL has a completely different scope with respect to these standard query languages, and – to the best of our knowledge – no language specification exists that addresses the same issue (i.e. the description of heterogeneous service requirements in a unified manner).

Further, the use of XML in specifying the language provides a set of advantages, such as flexibility and extensibility, and ease of development due to its proliferation and the existence of a handful of state-of-the-art tools for XML processing [13].
Another point we would like to stress out here is that the intention of the USQL language is not to substitute the existing and/or emerging service description protocols in the various service areas (e.g. WSDL [9], WSDL-S [11], OWL-S [12], etc.); rather, its main purpose is to provide a means that will enable unified service discovery, regardless of how service advertisements are formulated. In this regard, USQL can be seen as an abstract layer placed orthogonally to the existing service description protocols and the various service registry protocols (**Fig. 1**).

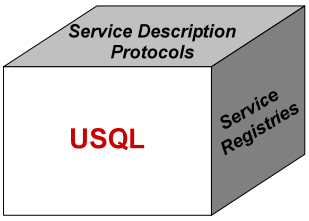

Fig. 1. USQL is orthogonal to service registries and service description protocols

2.2 Language Structure

USQL is used for the formulation of service discovery queries, and their corresponding results. The information sent to the various registries via a USQL request consists mainly of the search criteria, as they have been specified by the requestor. In addition, the users of USQL may specify what the resulting service entities should consist of, i.e. what kind of information should be returned for each matching service. They may also specify how the results should be displayed, i.e. how the service entities should be sorted. We describe the logical structures of the USQL request and response messages in the following paragraphs.

2.2.1 USQL Request

A USQL request message comprises two logical parts (**Fig. 2**):

- The **Projection part** consists of the fields of information that will be returned to the requestor. It also contains structures used for the presentation of the results, controlling the way these should be sorted.
- The **Criteria part** consists of the fields of information that constitute the search criteria to be applied during the service discovery process.

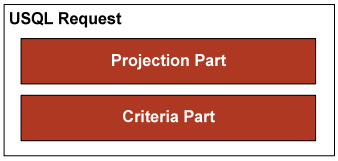

Fig. 2. USQL request message structure

2.2.2 USQL Response

A USQL response message contains all services that are found to meet the search criteria of the respective USQL request. For each service entity, there is a **mandatory** area containing the information that is necessary for the invocation of the service, and an **optional** area containing additional information about the service, such as textual description, service provider, QoS values etc. Alternatively, a USQL response message may contain information regarding an error that prevented the successful processing of the respective USQL request.

In either case, the structure of a response message in USQL is shown in the following figure (**Fig. 3**):

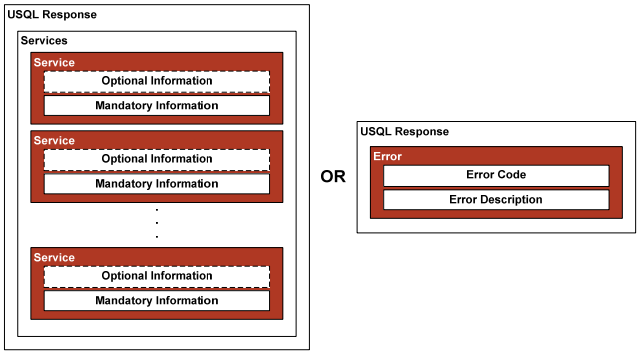

Fig. 3. USQL response message structure

2.3 Concepts

The following UML [14] conceptual diagram (**Fig. 4**) illustrates the Abstract Information Model of USQL. The language is used for the construction of request and response messages to be used for the process of service discovery. Therefore, the top-

level concept introduced by the language is the message. Both the request and response messages convey information which can be composed of syntactic, semantic, as well as QoS information elements.

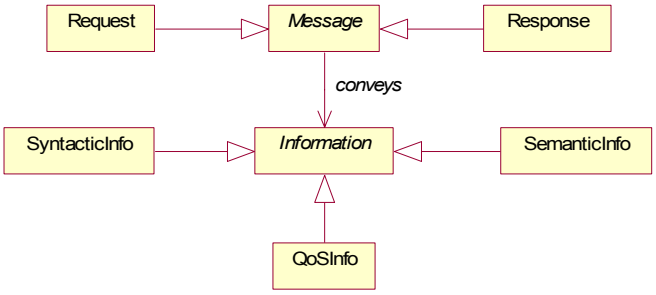

Fig. 4. USQL Abstract Information Model

The request message (**Fig. 5**) is used for the construction of the service discovery query; it incorporates the user requirements and organizes the conveyed information in the following parts:

- Projection part
- Criteria part

The *projection part* consists of a ***ViewAdditionalProperties*** section, which specifies the kind of additional information that the user wants to receive for each matching service, besides the information required for invoking it, and an ***OrderBy*** section, which specifies how the results should be sorted. The projection part is optional in a USQL request message; if omitted, the resulting service entities will only consist of the default fields, as defined by the USQL language, and in addition, they will be returned in arbitrary order.

The criteria part consists of an optional ***From*** section, where the user can explicitly identify the target service registries for the query, and a required ***Where*** section, where the rest of the search criteria are specified. In the case where the *From* section is omitted, it is assumed that the target service registries are identified implicitly by the implementation of the USQL language.

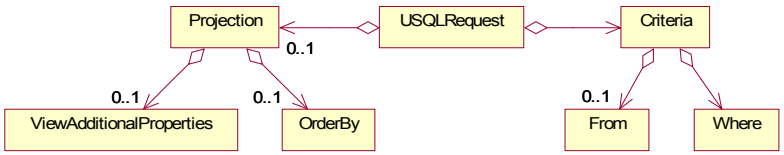

Fig. 5. Concepts of the USQL request message

The response message (**Fig. 6**) contains all ***Services*** that were found to meet the search criteria specified by the requester. Besides the default conveyed information

for each type of service, each service entity must also include the information fields specified in the *ViewAdditionalProperties* section of the request message. In the case where no services have been found, the response message will be empty. Alternatively, the response message may contain an ***Error*** section, indicating that the service discovery process did not complete successfully. As the following figure illustrates, a response message will either contain a (maybe empty) set of discovered services, or an error section with the appropriate information.

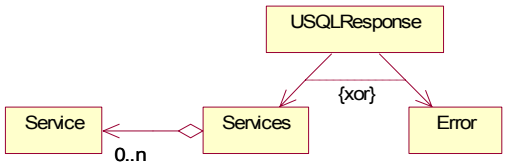

Fig. 6. Concepts of the USQL response message

2.4 Operators

The USQL language supports a set of operators that can be applied to the search criteria in the request message. Like the information elements, operators are categorized according to their domain of application. Hence, we define the syntactic, semantic, and QoS operator groups, each containing a number of operators that alter the default evaluation of the syntactic, semantic, and QoS search criteria elements, respectively.

The following table (**Table 1**) summarizes the operators which are defined by the language.

USQL OPERATORS		
Type	**Operator**	**Description**
Syntactic	equal	The value of the target element must be syntactically equal to the one specified.
	contain	The value of the target element must contain the one specified (equivalent to wildcards)
	notEqual	The value of the target element must not be syntactically equal to the one specified.
	notContain	The value of the target element must not contain the one specified.
Semantic	exact	The value of the element in the request (or one of its equivalent synonyms) must match exactly the value of the corresponding element in the advertisement.
	abstraction	The value of the element in the request (or one of its equivalent synonyms) must be subsumed the value of the corresponding element in the advertisement.

	extension	The value of the element in the request (or one of its equivalent synonyms) must subsume the value of the corresponding element in the advertisement.
	sibling	The value of the element in the request (or one of its equivalent synonyms) must have the same parent as the value of the corresponding element in the advertisement.
QoS	equal	The value of the target element must be equal to the one specified.
	notEqual	The value of the target element must not be equal to the one specified.
	greater	The value of the target element must be greater than the one specified.
	less	The value of the target element must be less than the one specified.
	equalOrGreater	The value of the target element must be either equal to or greater than the specified one.
	equalOrLess	The value of the target element must be either equal to or less than the one specified.

Table 1. USQL-supported operators

The operators introduced by USQL are applied to the target elements with the use of attributes, as we will see next.

3. Language Elements

In this chapter, we will describe in detail each one of the elements and structures comprising the USQL language. We will also describe the structure of the messages supported by the language and show how the various elements are combined for the formation of a service discovery request, as well as for the construction of the respective response message.

3.1 Notational Conventions

The definition of each USQL element and/or supported type is given in an informal, XML-like grammar, using the `monospace` typeface. The definition of an element is shown with the element name in **bold** typeface, enclosed in angle brackets.

Required attributes also appear in **`bold`** typeface. Where the attribute type has an enumerated type definition, the values are shown as separated by vertical bars. Where the attribute type is given by a simple type definition, the type definition name from either XSD or the USQL Schema is used. Where the attribute is optional and has a default value, it is shown following a colon.

Support for extension attributes is shown by {extension attribute}. Where used in the grammar, it indicates support for any number of attributes defined in a namespace other than the USQL namespace.

The allowed content of USQL elements is shown using a simple grammar:

- An element name is used for any content part that must be an element of that type.
- A name enclosed in curly braces and appearing in italic typeface refers to content parts of some other type.
- Support for extension elements is shown by {extension element}. Where used in the grammar, the content part may be any element defined in a namespace other than the USQL namespace.
- Support for mixed content is shown by {mixed}. Where used in the grammar, the allowed content is a mix of character data and of elements defined in any namespace.
- The cardinality of any content part is specified using the operators "?" (zero or one), "*" (zero or more) and "+" (one or more). If no operator is used, the content part must appear exactly once. Cardinality that cannot be expressed using either operator is shown using curly braces, with the minimum and maximum values separated by comma, e.g. "{2, *}" denotes two or more.
- Content parts may be grouped together using parentheses "()" to form a new content part. A choice group consists of all consecutive content parts, separated by a vertical line "|". A sequence group consists of all consecutive content parts that are separated by a comma ",".

For the formal definition of the USQL grammar, an XML schema is provided.

3.1.1 Prefixes and Namespaces Used In This Specification

xs	"http://www.w3.org/2001/XMLSchema" Defined in the W3C XML Schema specification
srv	"urn:sodium:USQL:services" Defined by this specification
tns	"urn:sodium:USQL" Defined by this specification

3.2 USQL Elements

In this section, we thoroughly present the elements defined in the current version of USQL. Their attributes, content and semantics are described and discussed in order to make clear their intended purpose.

3.2.1 ServiceName

```
<ServiceName nullAccepted=xs:boolean:false
  minDegreeOfMatch=tns:tPercentage:1.0
  priorityLevel=tns:tPriorityLevel:low
  valueIs=tns:tSyntacticOperator:equal
  caseSensitive=xs:boolean:false>
  Content: {xs:string}
</ServiceName>
```

The **ServiceName** element is used for expressing the requirement regarding the name of the requested service(s) and takes a string value. A set of attributes are used to customize the requirement regarding the service name.

The **nullAccepted** attribute is optional and determines the behavior of the matchmaking process, in cases where the corresponding element in the advertisement has not been specified. The `nullAccepted` attribute takes a `boolean` value, which specifies whether null values for the given element in service advertisements are accepted or not. The default value is `false`.

The **minDegreeOfMatch** attribute is optional and allows the explicit declaration of the minimum desired degree of match (threshold) for the given element. It is of type `tPercentage`, which restricts floats between 0 and 1. The default value is 1.

The **priorityLevel** attribute is optional and is used for explicitly stating whether the given requirement element is considered as important or not. The importance or not of a requirement element directly affects the calculation of the overall degree of match for a service. The attribute is of type `tPriorityLevel`, which enumerates `low` and `high` as allowed values. By default, the `priorityLevel` attribute is set to `low`.

The **valueIs** attribute is optional and determines the syntactic operator to be applied, i.e. how the value of the element in the request must be evaluated, with respect to the value of the corresponding element in service advertisements. It is of type `tSyntacticOperator` and in case it has not been specified, the `equal` value is assumed for the evaluation of the element. The `tSyntacticOperator` allowed values are listed in **Table 1**, referred to as syntactic operators.

The **caseSensitive** attribute is optional and specifies whether the matchmaking process will be case sensitive or case insensitive. It is of type `boolean` and, by default, this attribute takes `false` as its value, meaning that if it has not been explicitly specified for the element, the corresponding matchmaking will not be case sensitive.

3.2.2 ServiceDescription

```
<ServiceDescription nullAccepted=xs:boolean:false
  minDegreeOfMatch=tns:tPercentage:1.0
  priorityLevel=tns:tPriorityLevel:low
  valueIs=tns:tSyntacticOperator:equal
  caseSensitive=xs:boolean:false>
  Content: {xs:string}
</ServiceDescription>
```

The `ServiceDescription` element is used for expressing the requirement regarding the description of the requested service(s) and takes a string value. For a detailed description of the `nullAccepted`, `minDegreeOfMatch`, `priorityLevel`, `valueIs`, and `caseSensitive` attributes, see 3.2.1.

3.2.3 ServiceTaxonomy

The **`ServiceTaxonomy`** element allows the requester to specify the desired taxonomies for the requested service(s). This element is a bag containing one or more `code` elements.

```
<ServiceTaxonomy>
  Content: code+
</ServiceTaxonomy>

<code scheme=tns:tTaxonomyScheme>
  Content: {xs:string}
</code>
```

The **`code`** element specifies the taxonomy code for the taxonomy scheme identified by the value of the required `scheme` attribute. The current version of USQL allows the use of the following taxonomy schemes[1]:

ntis-gov:naics:1997	North American Industry Classification System (NAICS) 1997 Release
ntis-gov:naics:2002	North American Industry Classification System (NAICS) 2002 Release
unspsc-org:unspsc	ECCMA Product and Service Code System: UNSPSC Version 7.3
unspsc-org:unspsc:v6.0501	United Nations Standard Products and Services Code System (UNSPSC) Version 6.0501
unspsc-org:unspsc:3-1	United Nations Standard Products and Services Code System (UNSPSC) Version 3.1

Table 2. USQL-supported taxonomy schemes

3.2.4 ServiceProvider

[1] It should be noted that the USQL specification does not provide guarantees regarding support of these taxonomy schemes by the queried registries and networks.

```
<ServiceProvider>
   Content: (name|desc)
</ServiceProvider>

<name nullAccepted=xs:boolean:false
   minDegreeOfMatch=tns:tPercentage:1.0
   priorityLevel=tns:tPriorityLevel:low
   valueIs=tns:tSyntacticOperator:equal
   caseSensitive=xs:boolean:false>
   Content: {xs:string}
</name>

<desc nullAccepted=xs:boolean:false
   minDegreeOfMatch=tns:tPercentage:1.0
   priorityLevel=tns:tPriorityLevel:low
   valueIs=tns:tSyntacticOperator:equal
   caseSensitive=xs:boolean:false>
   Content: {xs:string}
</desc>
```

The **ServiceProvider** element allows requesters to search for services by specifying requirements regarding their provider. A `ServiceProvider` element may contain either a `name` element, for expressing requirements regarding the name of the requested provider, or a `desc` element, for expressing requirements regarding the textual description of the requested provider.

For a detailed description of the `nullAccepted`, `minDegreeOfMatch`, `priorityLevel`, `valueIs`, and `caseSensitive` attributes, see 3.2.1.

3.2.5 ServiceDomain

```
<ServiceDomain nullAccepted=xs:boolean:false
   minDegreeOfMatch=tns:tPercentage:1.0
   priorityLevel=tns:tPriorityLevel:low
   typeOfMatch=tns:tSemanticOperatorList
   ontologyURI=xs:anyURI>
   Content: {xs:string}
</ServiceDomain>
```

The **ServiceDomain** element enables requesters to explicitly specify the domain of the service(s) they are looking for. The values for this element should be concepts that are defined in an ontology identified by the URI value of the required **ontologyURI** attribute.

The **typeOfMatch** optional attribute signifies the type of reasoning that will be applied in matching the given value with the corresponding one in service advertisements. It is of type `tSemanticOperatorList`, and takes as value any combination of the semantic operators which are given in **Table 1**.

For a detailed description of the `nullAccepted`, `minDegreeOfMatch`, and `priorityLevel` attributes, we refer to 3.2.1.

3.2.6 Price

```
<Price
   valueIs=tns:tQoSOperator:equal
   nullAccepted=xs:boolean:false
   minDegreeOfMatch=tns:tPercentage:1.0
   priorityLevel=tns:tPriorityLevel:low
   currency=tns:tCurrency
   context=tns:tPriceContext:perCall>
   Content: {xs:float}
</Price>
```

The **`Price`** element enables requesters to specify requirements regarding the desired price of a service operation. The value range for this element is the set of float numbers. Although negative values seem unreasonable for a price requirement, the language does not impose the use of positive float numbers exclusively as values.

The **`valueIs`** optional attribute determines the operator that will be applied to the matchmaking of the requested value with the corresponding one in advertisement advertisements. The default operator is `equal`. For a list of available values for the attribute, we refer to the QoS operators listed in **Table 1**.

The **`currency`** required attribute is of type `tCurrency`, which specifies a 3-character currency code to represent the currency of the requested price. It is expected that `currency` attribute values will be formatted in 3-letter codes, such as the currency abbreviations specified by ISO 4217 [8]. Though, USQL does not enforce the usage of any specific standard body regarding currency codes.

The **`context`** attribute is optional and declares the temporal context of the service price. If omitted, the default value is assumed to be `perCall`, meaning that the requestor establishes a price-per-call requirement regarding the service. The type of this attribute (`tPriceContext`) restricts allowed values to one of the following: perCall, perDay, perWeek, perMonth, perYear.

For a detailed description of the `nullAccepted`, `minDegreeOfMatch`, and `priorityLevel` attributes, we refer to 3.2.1.

3.2.7 Availability

```
<Availability
   valueIs=tns:tQoSOperator:equal
   nullAccepted=xs:boolean:false
   minDegreeOfMatch=tns:tPercentage:1.0
   priorityLevel=tns:tPriorityLevel:low>
   Content: {xs:float}
</Availability>
```

Availability of services or operations refers to whether a service or an operation is present or ready for immediate use. The **`Availability`** element enables the requester to search for services based on this QoS aspect. Usually, the availability of services or operations is expressed as a percentage in a 24-hour context. For example,

a service being available 8 hours per day, the availability can be quantified as 33.33%. As regards the range of values for the `Availability` element, these are restricted between 0 and 1, both edges included.

For a detailed description of the `valueIs` attribute see 3.2.6.

For a detailed description of the `nullAccepted`, `minDegreeOfMatch`, and `priorityLevel` attributes, we refer to 3.2.1.

3.2.8 Reliability

```
<Reliability
   valueIs=tns:tQoSOperator:equal
   nullAccepted=xs:boolean:false
   minDegreeOfMatch=tns:tPercentage:1.0
   priorityLevel=tns:tPriorityLevel:low>
   Content: {xs:float}
</Reliability>
```

Reliability is the quality aspect of a service that represents the degree of being capable of maintaining the service and service quality. The **`Reliability`** element enables the requester to express this kind of QoS requirement. The same value principles and logics as in the `Availability` element apply here.

For a detailed description of the `valueIs` attribute see 3.2.6.

For a detailed description of the `nullAccepted`, `minDegreeOfMatch`, and `priorityLevel` attributes, we refer to 3.2.1.

3.2.9 ProcessingTime

```
<ProcessingTime
   valueIs=tns:tQoSOperator:equal
   nullAccepted=xs:boolean:false
   minDegreeOfMatch=tns:tPercentage:1.0
   priorityLevel=tns:tPriorityLevel:low
   unit=tns:tProcessingTimeUnit:millis>
   Content: {xs:float}
</ProcessingTime>
```

The **`ProcessingTime`** element is used for declaring the desired processing time of a requested operation – or of all operations – offered by a service. The **`unit`** attribute is optional and determines the measure for the specified float value of the element. Allowed values are: millis, seconds, minutes. The default value is millis.

For a detailed description of the `valueIs` attribute see 3.2.6.

For a detailed description of the `nullAccepted`, `minDegreeOfMatch`, and `priorityLevel` attributes, we refer to 3.2.1.

3.2.10 QoS

```
<QoS>
  Content: (Price?,Availability?,Reliability?,ProcessingTime?),
  {extension element}
</QoS>
```

The **`QoS`** element is actually a container for all USQL-supported QoS requirement elements. QoS requirements in USQL may be applied both at the service and operation level. When applied at the service level, requesters implicitly express that these requirements must be fulfilled by all operations offered by a service. However, it is always implied that the usage of the `QoS` container at the operation level overrides the `QoS` container at the service level.

The `QoS` element contains an extension point, thus allowing requesters to make use of their own QoS dimensions or use elements defined in another specification, should they need to do so.

3.2.11 Operation

The **`Operation`** element is used for the formulation of search requirements at the operation level. More specifically, requesters use this element in order to discover services based on the operations they provide. This enhancement is already a significant departure from the existing service discovery mechanisms supported by well known registry protocols, such as UDDI, ebXML, etc.

The **`Name`** element is used for the inclusion of the syntactic name of the operation as a requirement in the request.

The **`Capability`** element may be used for semantically expressing the desired capability (i.e. abstract functionality) of the operation.

The **`input/output`** elements are used for expressing requirements regarding the operation inputs and/or outputs. The order in which the requested inputs/outputs of an operation are expressed is not taken into consideration when applying matchmaking.
The semantics of the **`Inputs/Outputs`** container elements is as follows: In case they are specified but contain no `input/output` elements respectively, then it is implied that the requested operation must not have any inputs/outputs. In case they are not included in the operation element, then input/output matchmaking should not take place at all (i.e. the requester has no specific requirements regarding the input/output of the operation).

```
<Operation minDegreeOfMatch=tns:tPercentage:1.0
  priorityLevel=tns:tPriorityLevel:low>
  Content: {extension element}*,Name?,Capability?,Inputs?,
       Outputs?,QoS?
</Operation>

<Name nullAccepted=xs:boolean:false
  minDegreeOfMatch=tns:tPercentage:1.0
  priorityLevel=tns:tPriorityLevel:low
  valueIs=tns:tSyntacticOperator:equal
  caseSensitive=xs:boolean:false>
  Content: {xs:string}
</Name>

<Capability nullAccepted=xs:boolean:false
  minDegreeOfMatch=tns:tPercentage:1.0
  priorityLevel=tns:tPriorityLevel:low
  typeOfMatch=tns:tSemanticOperatorList ontologyURI=xs:anyURI>
  Content: {xs:string}
</Capability>

<Inputs minDegreeOfMatch=tns:tPercentage:1.0
  priorityLevel=tns:tPriorityLevel:low
  nullAccepted:xs:boolean:false>
  Content: input?
</Inputs>

<input minDegreeOfMatch=tns:tPercentage:1.0
  priorityLevel=tns:tPriorityLevel:low>
  Content: name?,type?,semantics?
</input>

<Outputs minDegreeOfMatch=tns:tPercentage:1.0
  priorityLevel=tns:tPriorityLevel:low
  nullAccepted:xs:boolean:false>
  Content: output?
</Outputs>

<output minDegreeOfMatch=tns:tPercentage:1.0
  priorityLevel=tns:tPriorityLevel:low>
  Content: name?,type?,semantics?
</output>

<name nullAccepted=xs:boolean:false
  minDegreeOfMatch=tns:tPercentage:1.0
  priorityLevel=tns:tPriorityLevel:low
  valueIs=tns:tSyntacticOperator:equal
  caseSensitive=xs:boolean:false>
  Content: {xs:string}
</name>

<type nullAccepted=xs:boolean:false
  minDegreeOfMatch=tns:tPercentage:1.0
  priorityLevel=tns:tPriorityLevel:low namespace:xs:string>
  Content:{xs:string}
</type>

<semantics nullAccepted=xs:boolean:false
  minDegreeOfMatch=tns:tPercentage:1.0
  priorityLevel=tns:tPriorityLevel:low
  typeOfMatch=tns:tSemanticOperatorList:exact ontologyURI=xs:anyURI>
  Content: {xs:string}
</semantics>
```

3.2.12 Service

```
<Service minDegreeOfMatch=tns:tPercentage:1.0
  serviceType=srv:tServiceTypes>
  Content: {extension element}*,ServiceName?,ServiceDescription?,
     ServiceProvider?,ServiceTaxonomy?,
     ServiceDomain?,QoS?,Operation*
</Service>
```

The **`Service`** element is used for the description of the requested services. It is a container for all service-level requirement elements previously described.

The **`serviceType`** attribute is optional and enables the requestor to specify the desired type of services as an extra requirement. If the attribute is omitted, it is assumed that any service of any supported type that matches the rest of the requirements must be returned. Currently, USQL supports the following service types:

- WebService
- GridService
- P2PService

Any combination of these types constitutes a valid value for the attribute. Nevertheless, since service types are declared in a separate namespace which is then imported in the USQL schema, it is always possible to expand the service type list with other service types without affecting the definition of the `Service` element.

Multiple `Service` elements may be used within a USQL request to express alternative requirements.

3.2.13 USQLRequest

The **`USQLRequest`** message is used for the formulation of the query with requirements that will be sent to heterogeneous registries for service discovery.

```
<USQLRequest>
  Content: ViewAdditionalProperties?,From?,Where,OrderBy?
</USQLRequest>
```

The `USQLRequest` message consists of the following content parts:

- **`ViewAdditionalProperties`** - This element enables requestors to define the additional information that will be included in the service entities of the query response. Its usage within the message is optional.
- **`From`** - This element enables requestors to explicitly specify the target registries for the request. Its usage within the message is also optional.
- **`Where`** - This element encapsulates all service requirements that requestors specify. Its usage is required within the message.

- **`OrderBy`** - This element enables requestors to define the sorting order for the resulting services. Its usage is optional.

```
<ViewAdditionalProperties>
  Content: property+
</ViewAdditionalProperties>

<property>
  Content: (ServiceProvider|ServiceDescription|Price|
       Availability|Reliability|ProcessingTime)
</property>

<From>
  Content: Registry+
</From>

<Registry>
  Content: {xs:string}
</Registry>

<Where>
  Content: Service+
</Where>

<OrderBy direction=tns:tOrderDirection:ascending>
   Content: (ServiceDegreeOfMatch|OperationDegreeOfMatch|
      ServiceName|ServiceProvider|Price|Availability|Reliability|
      ProcessingTime)
</OrderBy>
```

3.2.14 USQLResponse

```
<USQLResponse>
  Content: (Services|Error)
</USQLResponse>

<Services>
  Content: (srv:WebService|srv:GridService|srv:JXTAService)*
</Services>

<Error>
  Content: (code,desc)
</Error>

<code> Content: {xs:string} </code>
<desc> Content: {xs:string} </desc>
```

The **`USQLResponse`** message contains either the services that where found to meet the requirements expressed in the respective USQL request, or an error that prevented the successful application of service discovery. Each service entity contains the information that was initially requested. In addition, services are sorted according to the value of the `OrderBy` element in the respective USQL request.

The current version of USQL supports the following types of response services:

- WebService
- GridService
- JXTAService

The content of each supported service type is given as follows:

WebService

```
<srv:WebService degreeOfMatch:xs:float verified:xs:boolean:false>
   Content: ((tns:ServiceProvider?,tns:ServiceDescription?,
        tns:Price?,tns:Availability?,tns:Reliability?),
        {extension element}*,name,wsdl,portType+)
</srv:WebService>

<portType name=xs:string>
   Content: srv:Operation+
</portType>

<srv:Operation degreeOfMatch=xs:float>
   Content: ((tns:Price?,tns:Availability?,tns:Reliability?,
        tns:ProcessingTime?),{extension element}*,name)
</srv:Operation>

<name> Content: {xs:string} </name>
<wsdl> Content: {xs:anyURI} </wsdl>
```

GridService

```
<srv:GridService degreeOfMatch:xs:float verified:xs:boolean:false>
    Content: ((tns:ServiceProvider?,tns:ServiceDescription?,
       tns:Price?,tns:Availability?,tns:Reliability?),
       {extension element}*,name,wsdl,portType+,resourceInstance?)
</srv:GridService>

<portType name=xs:string>
   Content: srv:Operation+
</portType>

<srv:Operation degreeOfMatch=xs:float>
   Content: ((tns:Price?,tns:Availability?,tns:Reliability?,
        tns:ProcessingTime?),{extension element}*,name)
</srv:Operation>

<name> Content: {xs:string} </name>
<wsdl> Content: {xs:anyURI} </wsdl>
```

JXTAService

```
<srv:JXTAService degreeOfMatch:xs:float verified:xs:boolean:false>
   Content: ((tns:ServiceProvider?,tns:ServiceDescription?,
      tns:Price?,tns:Availability?,tns:Reliability?),
      {extension element}*,jxtasdl,name,interface+)
</srv:JXTAService>

<interface name=xs:string>
  Content: srv:Operation+
</interface>

<srv:Operation degreeOfMatch=xs:float>
  Content: ((tns:Price?,tns:Availability?,tns:Reliability?,
       tns:ProcessingTime?),{extension element}*,name)
</srv:Operation>

<name> Content: {xs:string} </name>
<jxtasdl> Content: {xs:anyURI} </jxtasdl>
```

The *verified* optional attribute appears in all services regardless their type. It takes a true/false value and indicates whether a specific service entity has been somehow verified or not. By default, services in a USQL Response are not verified. It must be noted that, as far as the language is concerned, the mechanism employed for the actual verification of services is agnostic. It is the responsibility of the implementation of the language to provide (or not provide) an appropriate service verification mechanism.

3.2.15 USQL

USQL defines an element named **USQL** to be the root element for all supported documents.

```
<USQL version=1.0>
  Content: (USQLRequest|USQLResponse)
</USQL>
```

The root element contains a required attribute that holds versioning info. Its value is fixed and currently set to "1.0", which is the present version of the USQL language.

3.3 Summary

In this section, we introduced and described the elements comprising the USQL language. For each element, an XML-based notation was used to detail its attributes and content. Having presented all USQL elements and structures, we proceed in the next chapter with discussing the underlying mathematical model of the language. The mathematical model reveals how the various elements and their attributes are treated by the matchmaking algorithm, in order to calculate the degree of match for each candidate service and its operations.

4. References

[1] Universal Description, Discovery & Integration (UDDI), http://www.uddi.org
[2] OASIS and UN/CEFACT, Electronic Business XML (ebXML), http://www.ebxml.org
[3] Juxtapose (JXTA™) Technology, http://www.jxta.org
[4] The WS-Resource Framework (WSRF), http://www.globus.org/wsrf/
[5] XQuery 1.0: An XML Query Language, http://www.w3.org/TR/xquery/
[6] Extensible Markup Language (XML) 1.0 (Second Edition), http://www.w3.org/TR/2000/REC-xml-20001006
[7] XML Schema Definition, http://www.w3.org/XML/Schema
[8] Codes for the representation of currencies and funds, http://www.iso.org/iso/en/prods-services/popstds/currencycodes.html
[9] Web Services Description Language, http://www.w3.org/TR/2004/WD-wsdl20-primer-20041221/
[10] SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query/
[11] Web Service Semantics, WSDL-S, W3C Member Submission, http://www.w3.org/Submission/WSDL-S/
[12] OWL-S: Semantic Markup for Web Services, W3C submission, Nov. 2004 http://www.w3.org/Submission/OWL-S/
[13] XML Beans, http://xmlbeans.apache.org/
[14] UML, Unified Modeling Language, http://www.uml.org/
[15] Resource Description Framework (RDF), http://www.w3.org/RDF/