# Behavior-aware, Unified Service Discovery

Michael Pantazoglou, Aphrodite Tsalgatidou
National & Kapodistrian University of Athens
Department of Informatics & Telecommunications
Panepistimiopolis, 15784 Athens, Greece
Phone: +30 210 727 5206
{michaelp, atsalga}@di.uoa.gr

George Spanoudakis
City University
Department of Computing
Northampton Square, London EC1V 0HB, UK
Phone: +44 20 7040 8346
gespan@soi.city.ac.uk

## ABSTRACT
Composite services commonly expose the choreography of message exchanges realized by their constituent services through appropriate descriptions and interfaces. Such information is very useful in deciding whether a composite service fully meets the behavioral requirements of a specific application or not. However, expressing behavioral requirements towards a service is currently a challenging task, often requiring the use of complex semantic annotations and the employment of related supporting technologies. Moreover, choreography descriptions and interfaces may be expressed in various formats and they are published in various types of registries, repositories, or networks. These issues hinder the applicability and effectiveness of behavior-aware service discovery. To overcome the aforementioned obstacles and to promote interoperability at the service discovery level, we propose in this paper a novel, unified solution towards expressing and evaluating behavioral requirements towards services. In our approach, queries are conveniently created in a visual manner with the use of UML; they are translated into a generic XML-based query language, namely USQL; they are uniformly executed in a wide variety of target registries, repositories, and networks, by means of a powerful service search engine; and they can be matched against different types of service choreography descriptions.

## Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.2.12 [**Software Engineering**]: Interoperability

## General Terms
Design, Human Factors, Languages.

## Keywords
Service Discovery, Behavioral Constraints, Visual Modeling, Service Choreography, Composite Services, Heterogeneous Registries.

## 1. INTRODUCTION
Service-Oriented Computing (SOC) is currently the leading paradigm in the development of distributed software applications. According to its principles, existing services, i.e. autonomous pieces of software which expose standards-based interfaces and communicate through well established internet protocols, can be discovered and composed to deliver a specific functionality. The result of a service composition may be further exposed as *composite* service, thereby rendering itself available for discovery and reuse. In terms of their exposed interfaces, both atomic and composite services describe their operations, input, output as well as the required binding and invocation details. However, unlike atomic services, composite services may also provide descriptions of their behavior, i.e. the choreography of message exchanges between the service itself and the constituent services.

To date, there are a number of languages which enable the description of composite service behavior, such as WS-BPEL [1], WS-CDL [5], WSMO [12], and OWL-S [8]. Once generated, behavior descriptions may be published to a variety of registries, such as UDDI (see http://www.uddi.org), uploaded to repositories such as ebXML (see http://www.ebxml.org), or even distributed within peer-to-peer networks [17]. Ideally, in order to search for services based on their behavior, requesters should be able to: (i) formulate behavioral queries in an easy and generic manner, independently of the various languages that services may use to describe their behavior; and (ii) uniformly execute their queries against heterogeneous targets. Alas, the current state-of-the-art makes behavior-aware service discovery a cumbersome task. Queries are formulated by means of specific choreography description protocols, which, despite their expressiveness, are not user-friendly, and are executed in a limited set of targets, which comply only with the language/protocol being used.

To address these limitations and facilitate the application of behavioral constraints in service discovery, in this paper we propose a novel framework for behavior-aware service discovery. Our framework comprises a visual query modeler, a generic XML-based query representation, and the corresponding middleware for query processing and execution. Thus, the contributions of the proposed framework are the following:

- User-friendly, intuitive formulation of behavior-based service queries by means of a visual query modeler
- Efficient query processing and execution in a unified and protocol-independent manner, by means of a powerful and flexible service search engine

It should be noted that, the proposed framework is the result of synergy and integration between two existing service discovery tools, namely the ASD platform [7] and the USQL Engine [10], which were developed in the contexts of two EU projects called SeCSE [13] and SODIUM [15], respectively.

Briefly, the rest of this paper is structured as follows. In Section 2 we present the proposed approach, describe its overall architecture and give details for the basic components; In Section

3 we discuss related work and compare our proposal to the various existing approaches; Finally, Section 4 concludes with a brief discussion on future work.

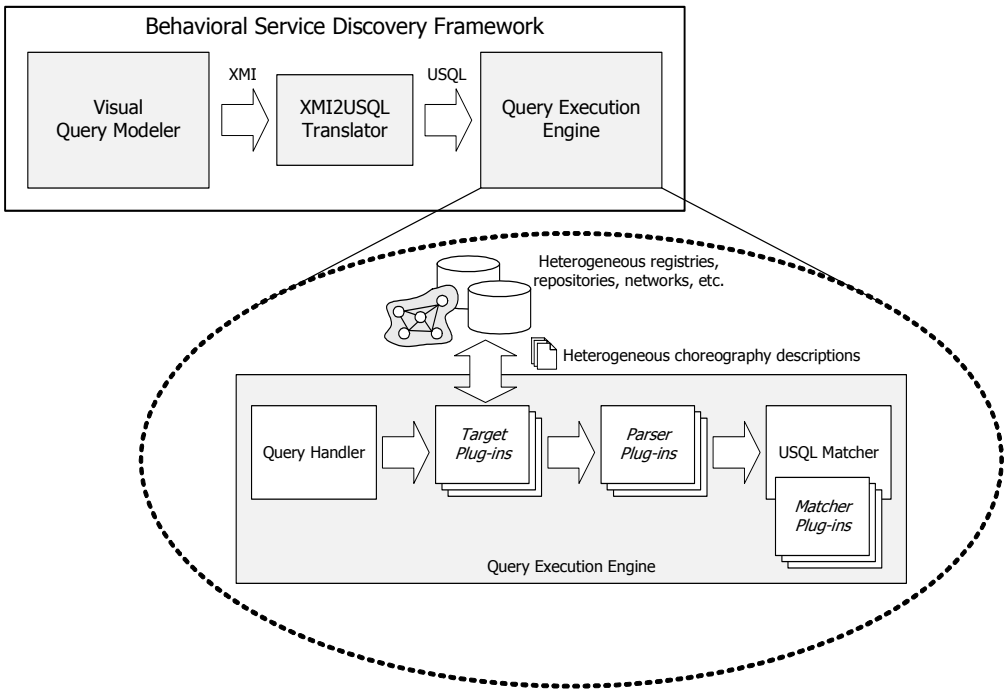


**Figure 1. Overall architecture of the proposed Behavior-aware Service Discovery Framework.**

## 2. THE PROPOSED APPROACH

As outlined in Section 1, our framework supports the discovery of services from heterogeneous service registries based on behavioral service queries which are expressed using an efficient, choreography protocol-independent formulation. The overall architecture of the proposed framework is displayed in Figure 1.

The proposed *Behavioral Service Discovery Framework (BSDF)* comprises three main components: (i) a visual query modeler which models behavioral service queries by means of UML behavioral diagrams; (ii) a translator which transforms the raw XMI output of the modeler to a generic XML-based query language, namely USQL; and (iii) a query engine capable of processing and executing USQL queries in various types of target registries, repositories, networks, etc. In the following paragraphs, we go through the details of each one of these components.

It should be noted that, although we have implemented a specific visual query modeler in the BSDF, the latter is generic enough and can also accommodate other front-ends, as long as the appropriate translator is provided. In this regard, the following paragraphs describe an instantiation of the BSDF front-end, which has been based on previous work and uses UML sequence diagrams to express behavior constraints towards services.

### 2.1 The Visual Query Modeler

The Visual Query Modeler (VQM) is the front end of our framework that enables the specification of service discovery queries. The VQM was originally developed to support architecture-time service discovery in the context of the SeCSE project. In our prototype implementation, we have used the UML-based graphical service query language that was defined by Kozlenkov et al. in [7]. This language enables the specification of service discovery query using a UML profile that was defined for this purpose. More specifically, the approach developed in [7] enables the specification of service discovery queries in reference to a UML model that includes interaction models specifying the behavior of systems that use services and class models that specify the classes and interfaces which are involved in this behavior. The user can specify queries in reference to a sequence diagram by identifying the messages that he/she expects to call operations provided by services and stereotype them as «asd_query_message». Following this annotation, the graphical modeling tool extracts from the UML model the classes that define the types of the parameters of the message and all the classes that can be reached by them via different types of relations and constructs the structural part of the query. It also extracts from the sequence diagram all the other messages which have the same receiver as the «asd_query_message» and uses them to construct a partial behavioral model for the required service.

In Figure 2 below, we show an example of a query specified using this approach. The UML sequence diagram shown in this figure specifies an interaction of a stock brokering system that enables its users to get daily market information including the rates of services of different stock brokers (see message *getRateInfo()* in the diagram), market headlines (see message *getMarketHeadlines()* in the diagram), and more detailed market news (see message *getMarketNews()* in the diagram), as well as information related to puchases of specific stocks (see message *getPurchaseInfo()* in the diagram). Assuming that a user wants to locate service operations that can provide the functionality expected by the messages *getMarketHeadlines()* and

*getMarketNews()*, all that he/she would need to do is to stereotype these two messages as «asd_query_message» as shown in Figure 2.

Users may also declare constraints for the services that they want to discover which are specified as OCL (Object Constraint Language) expressions [9].

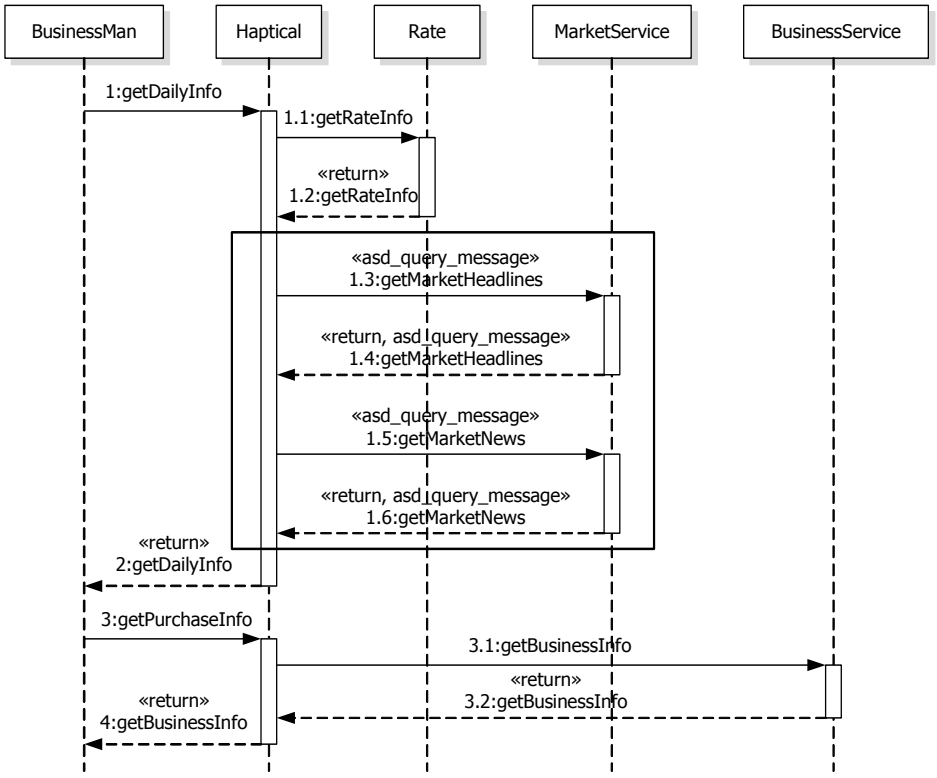


**Figure 2. An example behavioral query expressed in a UML sequence diagram.**

Based on the attachment of this stereotype to the messages, the graphical query modeling tool that has been developed in [7] can generate a query in XMI that includes a structural model and a behavioral model of the service that appears to provide the relevant operations in the diagram, i.e. the service represented by the interface *MarketService* in the model. The structural model in this case would include the classes that define the types of the parameters of the messages stereotyped as «asd_query_message» in the query. The behavioral model of the service *MarketService* in this case would represent the automaton shown in Figure 3.

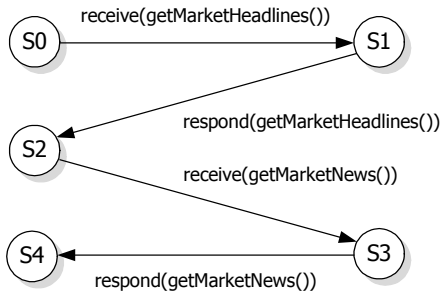


**Figure 3. Behavioral model of the market service.**

The current front end of the framework takes the query that is specified graphically and represents it in XMI. This front end has been implemented as a plug-in of IBM Rational Modeler.

## 2.2 USQL and the XMI2USQL Translator

The Unified Service Query Language (USQL) [16] is an XML-based language enabling the construction of unified, meaningful service discovery requests and responses. The primary goal of USQL is to provide a unified interface for the discovery of services, regardless of their type, description, and discovery protocols. To accomplish this objective, USQL has been designed to be as abstract and extensible as possible.

The specification of USQL provides the structures necessary to express syntactic, semantic, and qualitative criteria towards services, their operations, and the input/output messages. In order to support the expression of behavioral constraints towards a service operation, we extended the language by introducing an operation-level extension element called *Choreography*. Figure 4 displays the conceptual meta-model of the *Choreography* element.

With the use of the *Choreography* element, requesters may specify a desired sequence of service operations and, for each one of them, they may assign requirements for service-level properties (e.g. name, provider, and category of the service), operation-level properties (e.g. name, signature, semantics, and QoS properties), as well as input/output message-level properties (e.g. number of parts, part name, semantics, type, etc.). In this way, in addition to

the behavior-based evaluation performed for the whole composite service, it becomes possible to evaluate the constituent services against a rich set of criteria.

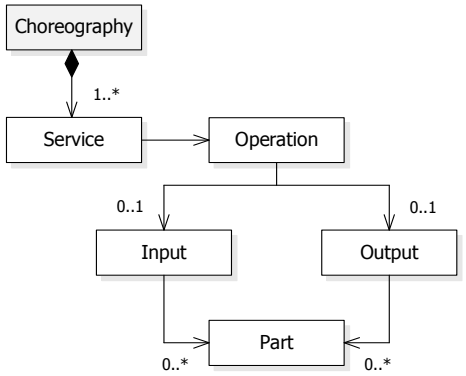


**Figure 4. Conceptual meta-model of the USQL Choreography extension element.**

Having explained the structure of the *Choreography* element in USQL, let us now proceed with a short description of the XMI2USQL translator component. The role of the translator is to transform UML sequence diagrams which are generated by the visual query modeler into USQL queries. In the case of sequential flows, the transformation is performed directly by adding a new *Service* sub-element to the *Choreography* element for each task, and appropriately setting the operation and input/output properties to reflect the desired data flows.

The XML snippet of Figure 5 illustrates the USQL request that results from the translation of our example UML sequence diagram (see Figure 2) discussed in Section 2.1. As it can be seen, a *Choreography* element has been appended to the requested service operation, in order to express requirements towards the desired behavior. The requested choreography involves two service operations, one for retrieving market headlines, and one for retrieving market news. Based on the information included in the XMI representation of the query, the XMI2USQL translator has mapped the exchanged messages to appropriate Input and Output parts in each service operation, and has also assigned their types. In general, the translator can also map QoS requirements expressed as OCL expressions in the UML query, to appropriate extension elements in a QoS container sub-element of each service operation. We have omitted the demonstration of such facility in our example due to limited space.

Thus, after completion of the XMI-to-USQL translation, the USQL-formatted query is ready to be processed and executed by the query execution engine of our framework, which we describe in the following.

## 2.3 The Query Execution Engine

The Query Execution Engine included in the BSDF is an extensible service search engine supporting service discovery in an open set of registry types, and is capable of processing and matching various heterogeneous types of service advertisements. This kind of flexibility is accomplished by the plug-in mechanism that the engine implements. The query engine is an extended version of the USQL Engine, a tool developed for the needs of the SODIUM platform [15], and as such it is fully compatible with the USQL specification, using USQL queries and responses to communicate with its users. For the rest of this paper, we will refer to the query execution engine as USQL Engine.

```
<?xml version="1.0" encoding="UTF-8"?>
<USQL version="" xmlns="urn:usql">
 <USQLRequest id="1184582426765">
  <SearchCriteria>
   <Service>
    <Operation>
     <Choreography xmlns="urn:usql:ext">
      <Service>
       <Operation>
        <Name><Value>get market headlines</Value></Name>
        <Input>
         <Part>
          <Type namespace="…"><Value>date</Value></Type>
         </Part>
        </Input>
        <Output>
         <Part>
          <Type namespace="…"><Value>Headlines</Value></Type>
         </Part>
        </Output>
       </Operation>
      </Service>
      <Service>
       <Operation>
        <Name><Value>get market news</Value></Name>
        <Input>
         <Part>
          <Type namespace="…"><Value>date</Value></Type>
         </Part>
        </Input>
        <Output>
         <Part>
          <Type namespace="…"><Value>News</Value></Type>
         </Part>
        </Output>
       </Operation>
      </Service>
     </Choreography>
    </Operation>
   </Service>
  </SearchCriteria>
 </USQLRequest>
</USQL>
```

**Figure 5. The USQL query that corresponds to our example UML query.**

Figure 1 displays the overall architecture of the USQL Engine, and the main components involved in a service discovery process. Upon receiving a USQL request, the engine employs a query handler which will be responsible for its processing. The query handler validates the USQL request against the USQL schema, configures the target registries and invokes the appropriate target plug-ins to execute the query. Search in each registry yields a number of service choreography advertisements which may be generally expressed in various different formats, such as WS-BPEL, OWL-S, or WS-CDL. Before being passed to the USQL matcher, the advertisements are processed by appropriate parser plug-ins. There is one parser plug-in for each supported choreography description language, which is responsible for mapping the advertisement to a USQL format. Subsequently, all

USQL-formatted advertisements along with the USQL request are submitted to the USQL matcher, which applies matching and ranks the results according to their similarity.

The USQL matcher supports the matching of structural and behavioral descriptions of services and any QoS constraints that may have been defined for them in service discovery queries, with models of services retrieved from service registries, repositories or networks. Matcher plug-ins may be employed during the matching process to cater for USQL extension elements, such as the *Choreography* element which we introduced to express behavioral constraints. The matching is based on the computation of a *structural*, *behavioral* and *QoS distance* between the service operations which are required by a query and service operations which exist in service registries.

```
<?xml version="1.0" encoding="UTF-8"?>
<USQL version="" xmlns="urn:usql">
 <USQLResponse queryId="1184582426765">
  <Services>
   <Entry rank=”0.817”>
    <results xmlns=”urn:asd”>
     <pair>
      <query_operation>getMarketHeadlines</query_operation>
      <service_operation wsdl_idref="…" service="MarketService"
          provider="...">getMarketHeadlines</service_operation>
      <distance>
       <overall>0.194</overall>
       <structural>0.205</structural>
       <behavioral>0.00</behavioral>
      </distance>
     </pair>
     <pair>
      <query_operation>getMarketNews</query_operation>
      <service_operation wsdl_idref="…" service="MarketService"
          provider="...">getMarketNews</service_operation>
      <distance>
       <overall>0.183</overall>
       <structural>0.192</structural>
       <behavioral>0.00</behavioral>
      </distance>
     </pair>
    </results>
    <wsdls>
     <wsdl wsdl_id="…"  service="MarketService">http://...</wsdl>
     <wsdl wsdl_id="…" service="MarketService">http://...</wsdl>
    </wsdls>
   </Entry>
  </Services>
 </USQLResponse>
</USQL>
```

**Figure 6. An example USQL response to our USQL request.**

The structural distance between a service operation in a query and a service operation in a registry is calculated by finding the best possible morphism between two graphs that represent the data types of these two operations, respectively. These graphs are generated automatically from the description of service operations in the query and the registries (the framework supports, for example, the generation of such graphs from USQL and WSDL descriptions of service operation signatures). The computation of this morphism is formulated as an instance of the “assignment problem” and is solved by an algorithm implementing the Hungarian method as defined in [7].

The computation of the behavioral distance between a query and a service operation is based on an algorithm that checks if an automaton that is generated from the behavioral model of the service of an operation in the query can be admitted to an automaton that is generated to describe the behavior of a service that provides an operation in a registry. The algorithm that is deployed to check the admissibility computes a minimum distance morphism between the transitions of the two automata and can be configured to increase flexibility by leaving up to K adjacent transitions without a match in the two automata during the search process (see [7] for more details). The framework supports the generation of service behavior automata from the choreography model of USQL and standardized service behavior specification languages such as WS-BPEL. Finally, in evaluating queries the framework checks whether the QoS constraints that may have been specified are satisfied by the descriptions of services in registries. These constraints may be hard or soft – the former are constraints that must be satisfied by all the service operations which are discovered in a query whilst the latter contribute to the overall distance between an operation in a query and an operation in a service registry depending on whether they are satisfied or not (unsatisfied constraints increase the distance between two operations whilst satisfied constraints reduce it).

An example USQL response that corresponds to the USQL request of Figure 5 is shown in Figure 6. The response contains one service entry with two matching operation pairs, one for the *getMarketHeadlines* message exchange, and one for the *getMarketNews* message exchange. For each matching pair, their overall, structural, and behavioral distances are displayed, as they were calculated by the matcher component of our search engine. Moreover, a rank has been assigned to the overall service entry, based on the overall distances of the constituent operations. Finally, for each matching operation, the URL of the corresponding WSDL description document is provided.

# 3. RELATED WORK

The framework proposed in this paper is the first one which, to the best of our knowledge, remains independent from the various service description and discovery languages and protocols, and combines structural, behavioral, QoS and semantic criteria in service discovery. Other approaches which support only subsets of such criteria are overviewed in the following.

The application of behavioral constraints in service discovery has received a lot of attention over the last years, and is generally seen as a means to automate the development lifecycle of service-oriented applications. In [11], a framework called MoSCoE is presented, which supports choreography-based service discovery and composition, and further provides facilities for failure cause detection. The main difference between MoSCoE and our approach is the assumptions made on the description of constituent services and service choreographies. In MoSCoE, it is expected that constituent services provide semantic descriptions using common semantic concepts, whereas service choreography advertisements are modeled with the use of a notation called STS (Symbolic Transition System). Instead, our approach retains a high degree of flexibility and independency from choreography description protocols, as different parser plug-ins can be employed to process the various heterogeneous types of advertisements. In [14], the authors propose a behavioral model

for web services using automata and logic formalisms. They also propose a query language which is based on first-order logic and is used to express requirements towards the behavior of service operations. In general, the proposed model and query language adopt the Input-Output-Precondition-Effect (IOPE) model supported by OWL-S. Although expressive, the proposed query language is not very user-friendly, and relies on the use of semantics to capture the requested operation behavior. On the other hand, our proposed framework provides a convenient visual query modeler which allows for easy formulation of behavioral queries, which are not necessarily bound to the use of semantics.

The METEOR-S [2] system adopts a constraint driven service discovery approach in which queries are integrated into the composition process of a Service-Centric System (SCS) and represented as collections of tuples of *features*, *weight*, and *constraints*. In our approach, the queries contain information about features, weights, constraints, and parts of the design models of the SCS being developed. Klein and Bernstein [6] support service discovery queries based on matching operation signatures using string matching. Their approach is limited as it does not support behavioral constraints and cannot account for changes in the order or names of operation parameters. Horrocks et al. [3] have developed an approach in which the discovery of services is addressed as a problem of matching queries specified as a variant of Description Logic (DL) but do not support the flexible matching of service behavioral models.

# 4. CONCLUSIONS

We presented a novel approach in service discovery which considers the behavioral aspects of composite services. The proposed framework enables modeling queries with behavioral constraints in a visual manner, and supports their execution in an open set of registries, repositories, and networks. In addition, the framework is able to match the query against various types of service choreography advertisements, independently from their format and protocols. Thanks to its flexibility, the framework can cooperate with various front-ends supporting the formulation of behavioral service queries.

As it was previously mentioned, the proposed framework is the result of integrating two existing service discovery tools, and it is an ongoing effort. At a first stage, we intend to finalize the XMI2USQL translator component, and develop the plug-ins necessary for searching in UDDI and ebXML registries, as well as for processing WS-BPEL choreography advertisements. At a later stage, we aim at expanding the capabilities of our search engine by adding plug-ins that will allow processing of other types of advertisements, specifically targeting for WS-CDL and OWL-S Process Model descriptions. Hence, we will be able to verify the efficiency of our framework in addressing the heterogeneity of the various service behavior description languages.

Finally, we would like to note that, we are evaluating the expressiveness of our generated queries. Currently, our framework supports behavior-based service discovery by taking into account simple sequential behavior patterns during matchmaking. In the future, we will investigate the option of expressing and evaluating more complex behavioral patterns in queries, such as conditions, loops, parallel flows, etc., which are supported by most choreography protocols.

# 5. ACKNOWLEDGMENTS

This work has been partially funded by the European Commission under contracts IST-FP6-004559 for the SODIUM project and FP6-IST-511680 for the SeCSE project.

# 6. REFERENCES

[1] Alves, A., Arkin, A., et al. 2007. Web Services Business Process Execution Language Version 2.0. OASIS Standard.

[2] Aggarwal R., Verma K., et al. Constraint Driven Web Service Composition in METEOR-S, *IEEE Int. Conf. on Services Computing*, 2004.

[3] Horrocks, I., Patel-Schneider, P.F., and van Harmelen, F. From SHIQ and RDF to OWL: The making of a Web ontology language. *Journal of Web Semantics*, 1(1), 7-26, 2003.

[4] Hoschek W 2002. The Web Service Discovery Architecture. *IEEE/ACM Supercomputing Conf.*, Baltimore, USA, 2002.

[5] Kavantzas, N., Burdett, et al. 2004. Web Services Choreography Description Language Version 1.0. W3C Working Draft, October 2004.

[6] Klein M., and Bernstein A. Toward High-Precision Service Retrieval. *IEEE Internet Computing*, 30-36, January 2004.

[7] Kozlenkov, A., Spanoudakis, G., et al. 2007. Architecture-driven Service Discovery for Service Centric Systems, *International Journal of Web Services Research*, 4(2), 81-112, 2007.

[8] Martin, D., Burstein, N., et al., 2004. OWL-S: Semantic Markup for Web Services. W3C Member Submission, 22 November 2004.

[9] OCL, Object Constraint Language, OMG specification available at http://www.omg.org/docs/ptc/03-10-14.pdf (accessed July 2007).

[10] Pantazoglou, M., Tsalgatidou, A., and Athanasopoulos, G. 2006. Discovering web services and JXTA peer-to-peer services in a unified manner. In *Proc. of ICSOC'06*, 104-115.

[11] Pathak, J., Basu, S., et al. 2006. Parallel Web Service Composition in MoSCoE: A Choreography-Based Approach. In *Proceedings of ECOWS 2006*, 2006, 3-12.

[12] Roman, D., Keller, U., et al. 2005. Web Service Modeling Ontology. *Applied Ontology*, 1 (1), 2005, IOS Press, 77-106.

[13] SeCSE, Service-Centric System Engineering, FP6-IST-511680, http://secse.eng.it/.

[14] Shen, Z., and Su, J. 2005. Web Service Discovery Based on Behavior Signatures. In *Proc. of SCC 2005*, 279 - 286.

[15] SODIUM, Service-Oriented Development In a Unified fraMework, IST-FP6-004559, http://www.atc.gr/sodium.

[16] Tsalgatidou, A., Pantazoglou, M., and Athanasopoulos, G. 2006. Specification of the Unified Service Query Language (USQL). Technical Report, June 2006, available at http://www.di.uoa.gr/~s3lab/TR/2006/usql-1.0-spec.pdf.

[17] Younas, M., Awan, I., et al. 2007. A P2P Network Protocol for Efficient Choreography of Web Services. In *Proc. of AINA 2007*, 839-846.