SEMANTICALLY ENHANCED DISCOVERY OF HETEROGENEOUS SERVICES

A. Tsalgatidou, G. Athanasopoulos, M. Pantazoglou
University of Athens, Department of Informatics and Telecommunications

Abstract: Industrial application development approaches are striving for solutions that promote the rapid development of flexible and adaptable systems and the exploitation of legacy systems and resources. The Service-oriented Development (SOD) paradigm, a current trend in software development, could be beneficial to industrial application development approaches. However, the heterogeneity in existing standards and protocols for the discovery of the various service types is an obstacle for the use of SOD in industry. This paper addresses this issue by providing a solution that supports the unified discovery of heterogeneous services and thus supporting the use of SOD in industry. The proposed solution comprises a generic service model (GeSMO), which facilitates the specification of heterogeneous services, a query language called Unified Service Query Language (USQL), based on GeSMO, which facilitates the unified discovery of heterogeneous services within heterogeneous service registries and a query engine called USQL Engine, that enables the execution of queries described in terms of the USQL, upon heterogeneous service registries.

Key words: Service-oriented Development, Heterogeneous Services, Web Services, P2P Services, Grid Services, Generic Service Model, Semantically-enhanced Service Discovery.

1. INTRODUCTION

Software engineering is gradually shifting to Service-Oriented Architecture (SOA) [SOA] and related technologies, in order to address critical contemporary issues imposed by the emergence of the Web, such as low cost application development and application interoperability. Industry's

competitive environment needs technology solutions that will facilitate Rapid Application Development (RAD) and ensure application features such as flexibility and adaptability. Moreover, the exploitation and reuse of legacy systems constitutes a critical factor for the adoption and viability of these solutions. To satisfy the aforementioned requirements, industry seems to be embracing current trends of Service-Oriented Development, and it is expected that the emerging Semantic Web [SemWeb] will further accelerate the coalescence of the two worlds.

Nowadays, the Web bustles with services that are characterized by a high degree of diversity and heterogeneity. Web, Grid, and P2P services are continuously gaining momentum, yet, these are ruled by different and heterogeneous protocols and standards, making it difficult for them to interoperate. As a result, industry is intimidated in integrating and composing such diverse components for the utilization of service-oriented applications. Clearly, the full dynamics of these service technologies will be exposed and exploited by the industry, only when appropriate languages and tools emerge, which will render integration and interoperability among these technology areas feasible. Therewithal, services need to be discovered in order to be integrated in the context of an industrial application and, besides that, semantic annotations in service descriptions are required, in order to facilitate and automate the process of service discovery.

The provision of a framework that will encompass all previously mentioned requirements is expected to become the stepping stone to a new, service-oriented era in the world of industrial applications. SODIUM [SODIUM] forms an integrated solution for supporting and facilitating the comprehensive and unified visual composition, discovery, execution and monitoring of heterogeneous services. SODIUM platform comprises a set of languages as well as a set of individual, distributed and loosely-coupled components, which collaborate in order to support the aforementioned functionality.

In this paper, we focus on the service query language and its enacting search engine provided by SODIUM, which, combined, enable the unified and semantically enhanced discovery of diverse types of services over heterogeneous registries and/or networks. The results of such discovery can then be used for the development of service compositions in industrial environments.

The rest of this paper is structured as follows: A *motivating scenario* is presented to demonstrate how a real industrial application can be developed according to a Service-Oriented Architecture, by utilizing various heterogeneous services, as well as how it benefits from such an approach. Next, a *Generic Service Model* is described, offering a common point of reference for the various types of services. Following that, we introduce a

Unified Service Query Language catering for the discovery of heterogeneous services that are compliant with the model previously discussed, as well as its *enacting engine*. Based on the motivating scenario, examples on using the language and the engine are provided, in order to showcase the various assets of our framework. Consequently, we include a brief section with related work in service definition and discovery and finally the paper is closed with our conclusion statements.

2. MOTIVATING SCENARIO

An appropriate domain for the application of service-oriented computing is the automobile industry. Car manufacturers and their suppliers face many significant challenges, including pressure to reduce cost and time to delivery in the supply chain. The dynamic nature of such supply chains and the heterogeneity among the systems of the respective stakeholders are some of the obstacles that a system developer has to face.

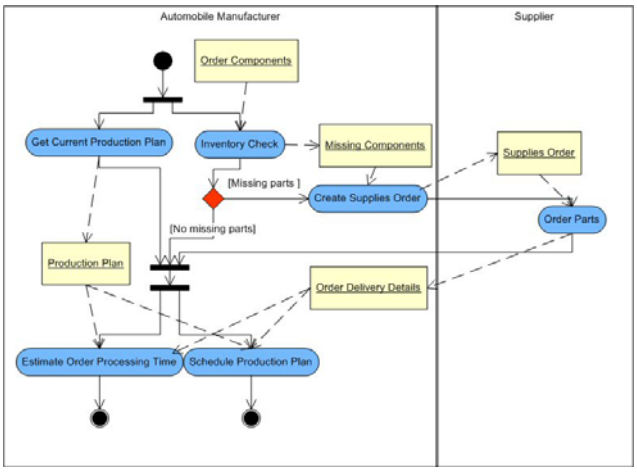

Figure 1. Order processing flow example

A crucial task that is usually met in an order processing workflow is the estimation of the processing time for a given order. In a car manufacturing industry there could be a plethora of requests for such calculations, which have to be answered promptly. However, the calculation of the order processing time is a computation intensive task that is depending on many factors such as the existence of all necessary components, the delivery time of non-existing components, the current factory production plans, order priority, etc.

A simplified workflow, which calculates the processing time of an order, is presented in *Figure 1*. According to this scenario, the workflow takes as input a list with the order's components that need to be provided. The workflow begins with the execution of two parallel tasks; one task provides the current production plans of the factory and the other checks whether the factory's warehouses have all the necessary materials. If some necessary materials are missing, the workflow continues with the preparation of an order for supplies and the submission of that order to a supplier which returns the necessary order details such as cost, delivery time and shipment method. Upon the completion of the aforementioned tasks, the workflow goes on with the execution of two parallel tasks, which estimate the order processing time and reschedule the production plan respectively. The outputs of the workflow are the estimation of the order completion time and the reformed production plan which takes into account the new order.

The tasks modeled in this scenario do not have to be developed from scratch. They could be performed by already existing services that may be available over the Internet. For example, the functionality required by the tasks *"Get Current Production Plan"*, *"Create Supplies Order"* and *"Order Parts"* may be provided by respective web services which are registered in a web service registry (e.g. UDDI) and are offered by various providers. . The *"Inventory Check"* task could be performed by a p2p service that is provided by a p2p network that exists between the manufacturer's warehouses. Last but not least, the functionality required by the *"Estimate Order Processing Time"* and *"Schedule Production Plan"* tasks could be provided by grid services which are utilizing the resources of a grid network where the car manufacturing organization is participating.

In order to be integrated in the aforementioned workflow, services have to be firstly discovered. However, service discovery is not an easy task, due to the heterogeneity and incompatibility between the existing description and discovery protocols and standards for web services, grid services and p2p services. In the following, we describe our solution to this problem that comprises a Generic Service Model, a Unified Query Language and a respective enacting engine.

3. GENERIC SERVICE MODEL

Although, service-oriented technologies (e.g. web, grid and p2p services) comply with the same paradigm, they adhere to different models, having different characteristics and different properties. Moreover, their heterogeneity spawns across other aspects such as architecture, supported protocols and standards, infrastructure, semantics and quality of service

(QoS). This diversity makes the integration of different services a strenuous task.

Therefore, in order to remove this burden from a system developer a generic service model (GeSMO) incorporating features and properties of all service-oriented technologies needs to be provided. This model will facilitate the specification of any type of service and the mapping and/or association of service features of one technology to the other.

3.1 Service Model Structure

An assessment of the service models of the addressed service-oriented technologies brings up a set of common features and properties that may be regarded as the common denominator of the web, grid and p2p services, which are the service types being addressed in this paper. Nevertheless, apart from this set of common features there are a lot of discrepancies among the various types of services.

Thus, a layered structure seems to be appropriate for the specification of GeSMO comprising a core layer with common features of all service-oriented technologies and with appropriate extensions providing for the specific features and properties of each of the addressed service types. *Figure 2* illustrates the structure of GeSMO. Furthermore, crosscutting issues such as Semantics, Quality of Service, Trust, Security and Management are pertinent to all types of services and may be related to any element of the service model.

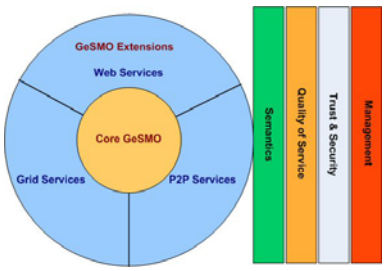

Figure 2. Generic Service Model Structure

In the following we present each of the identified layers and the interrelationships among their elements.

3.2 Core Service Model Concepts

After a thorough investigation of the current state of the art in service technology, we came up with a set of features that seem to be pertinent to all

types of services. As it is illustrated in *Figure 3* a service is regarded as a software system that exchanges messages, which are usually XML-formatted, it resides at a specific network address and it has a description that may be an XML- formatted document.

Figure 3. Service model

Service descriptions, which may be semantically and/or quality of service enhanced are published in service registries which are used by requestors for the discovery of appropriate services. A service description contains information that can be used for the identification and invocation of a service (*Figure 4*).

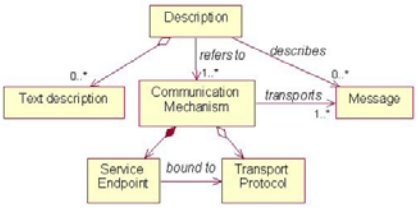

Figure 4. Service description structure

A service description conveys information, such as the specific endpoint that a service resides, the protocol that can be used for the message exchange and text descriptions providing human readable information about the service. In some cases, the specification of the message exchange mechanism may not be explicitly described, e.g. in P2P services an implied scheme is used. In these cases information related to the service endpoint or the protocol used is inferred by the underlying platform.

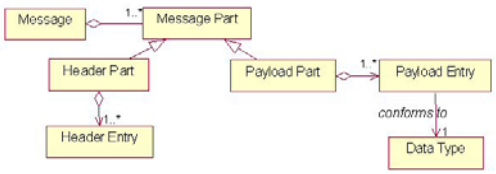

Figure 5. Message Structure

Exchanged messages are composed of two parts: header and payload information (*Figure 5*). The header part normally conveys information that is manipulated by the intermediate nodes/middleware transporting the messages. Such information may be routing information, security or transaction context information, etc. The payload part of a message conveys information that is consumed by the service or its client. This information is application specific and it normally abides by data types that are specified by the platform (e.g. Strings, Integers, etc) or the service provider (e.g. Addresses, Contacts, etc).

Service description documents contain additional information that facilitates the invocation of services. Services implement specific interfaces which describe the operations that are offered by a service (see *Figure 6*). These operations exchange messages, which convey information that abides by specific data types, with the service clients. These messages could be either incoming or outgoing with respect to the service. This information is also included in a service description document as it is necessary for the invocation of a service.

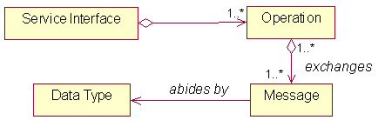

Figure 6. Service description elements

We have to note here that, service invocation information is not provided by all service type descriptions, e.g. by p2p service descriptions, as it can be either inferred by the underlying infrastructure or by the service implementation.

4. UNIFIED SERVICE QUERY LANGUAGE (USQL)

The *Unified Service Query Language (USQL)* is an XML-based [XML] language enabling requestors to formulate queries asking for available services. The language specification describes both requests and corresponding responses. The main contribution of USQL lies in that it follows a unified approach to expressing queries as regards the heterogeneous types of services. This is achieved with the language abiding by the core concepts introduced by GeSMO, as far as the abstract definition of a service is concerned. On the other hand, USQL responses may be easily extended so as to provide the concrete information for invoking the service, with respect to its type. Thanks to its flexible and extensible design, USQL

can consolidate virtually any GeSMO-compliant type of service, thus providing service-oriented industrial applications with a wide lookup range regarding candidate services that could be integrated and used for fulfilling a specific task.

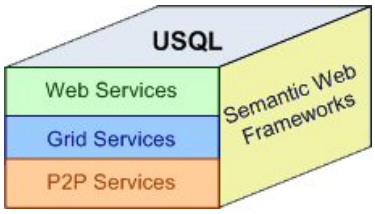

Figure 7. Orthogonal position of USQL with respect to services and semantic frameworks

USQL currently addresses - but is not limited to - Web, Grid, and P2P services, aiming at applying semantically enhanced queries for discovering them. As depicted in *Figure 7*, USQL is orthogonal with respect to these diverse service types and their description protocols; moreover, semantic concepts supported by the language are generic enough so as to map to most well known emerging semantic frameworks, such as OWL-S [OWL-S] and WSMO [WSMO], thus enabling the exploitation of their capabilities.

4.1 Semantics in USQL

Although syntactic information suffices for the invocation of a service, experience has shown that confining a service query to syntactic matching yields in most cases to scrappy results; the response to a query based on syntactic information either misses services, or contains services which are actually irrelevant to the initial request. Furthermore, the limited expressiveness of syntactic information is an obstacle when applying service discovery at runtime. To tackle such cut-backs, USQL enhances service requests with semantic information, in order to provide users with more expressive means. The supported semantics consist of domain-specific annotations which are bound to service operations and their respective input/output. In addition, USQL provides a set of elements and structures to allow for the application of QoS requirements in the search criteria, in order to refine service discovery and selection.

Briefly, USQL provides the following features for semantically annotating service requests:

- ***Domain –*** implemented by an element called *ServiceDomain*, this feature enables requestors to specify an application domain for the requested services and thus to semantically enhance the query and to confine the

search range. This is the first step towards overcoming scrappy and irrelevant results.

- ***Input/Output*** – the *Input/Output* elements enable requestors to apply semantic criteria regarding the expected input/output of an operation offered by a service.
- ***Capability*** – the *Capability* element enables requestors to apply semantic criteria regarding the expected capability (i.e. the abstract functionality) of an operation offered by a service.

USQL introduces a set of operators that can be applied to semantic elements during service discovery, determining the type of inference rules that should be employed for reasoning purposes. More specifically, the following types of inference are supported by the language:

- ***exact*** – indicates that the element's value must be an exact match of the value of the corresponding element in the service advertisement.
- ***abstraction*** – indicates that the element's value must be subsumed by that of the corresponding element in a service advertisement.
- ***extension*** – indicates that the element's value must subsume that of the corresponding element in a service advertisement, besides exact matching.

USQL defines a generic type for all supported semantic elements, which contains the following attributes:

- ***typeOfMatch*** – applies any combination of the aforementioned operators to the semantic element, indicating the type of inference that must be employed during the discovery process, in order to determine if a service satisfies the specific semantic requirement.
- ***nullAccepted*** – specifies whether services not including the corresponding element in their description should be further processed, and potentially included in the results, or not.
- ***ontologyURI*** – associates the value of the semantic element with an existing ontology, identified by a URI.

By employing these relatively simple artefacts, USQL enriches service queries semantically, allowing requestors to express their requirements in a more explicit way, thus yielding to concrete and consistent results. Nevertheless, by keeping semantics support to this level of simplicity, the language retains its openness and orthogonal position as regards existing and emerging types of services and semantic frameworks. In the following paragraph, we demonstrate how USQL can be used to formulate semantically enhanced queries looking for appropriate services that would satisfy the requirements imposed by the previously presented motivating scenario.

4.2 Using USQL

As shown in the motivating scenario, the first step in order to calculate the processing time for an order in the domain of automobile is to retrieve the current production plan. Thus, a Web service with the specific output is needed to fulfil the task. The following USQL request (*Figure 8*) encompasses these details with the use of semantic annotations, in order to find the most appropriate service for the job:

```
<?xml version="1.0" encoding="UTF-8"?>
<USQL version="1.0">
  <find_servicesRequest>
    <Where>
      <Service serviceType="WebService">
        <ServiceDomain ontologyURI="http://http://nkua/sodium/usql/engine/ontology/dco"
          typeOfMatch="exact extension">
          AutoMobile</ServiceDomain>
        <Operation>
          <Capability ontologyURI="http://nkua/sodium/usql/engine/ontology/dco"
            typeOfMatch="exact">
            GetCurrentProductionPlan</Capability>
          <Output>
            <semantics ontologyURI="http://nkua/sodium/usql/engine/ontology/dco"
              typeOfMatch="extension">
              ProductionPlan</semantics>
          </Output>
        </Operation>
      </Service>
    </Where>
  </find_servicesRequest>
</USQL>
```


Figure 8. Example USQL request for Web services

```
<?xml version="1.0" encoding="UTF-8"?>
<USQL version="1.0">
  <find_servicesRequest>
    <Where>
      <Service serviceType="P2PService">
        <ServiceDomain ontologyURI="http://http://nkua/sodium/usql/engine/ontology/dco"
          typeOfMatch="exact extension">
          AutoMobile</ServiceDomain>
        <Operation>
          <Capability ontologyURI="http://nkua/sodium/usql/engine/ontology/dco"
            typeOfMatch="exact">
            InventoryCheck</Capability>
          <Input>
            <semantics ontologyURI="http://nkua/sodium/usql/engine/ontology/dco"
              typeOfMatch="exact extension">
              ListOfComponents</semantics></Input>
          <Output>
            <semantics ontologyURI="http://nkua/sodium/usql/engine/ontology/dco"
              typeOfMatch="exact extension">
              MissingComponents</semantics></Output>
        </Operation>
      </Service>
    </Where>
  </find_servicesRequest>
</USQL>
```


Figure 9. Example USQL request for P2P services

Given a set of required components for the production of a car, the workflow needs to access the established P2P network and look for a service that will enable checking against warehouses for missing components. The USQL message expressing a request for such a service is depicted in *Figure 9.*

The estimation of the time that is required for processing an order is a demanding operation in terms of processing power, due to its complex

calculations. Hence, a Grid service would be the perfect candidate for carrying out this task. *Figure 10* depicts the respective USQL request.

```
<?xml version="1.0" encoding="UTF-8"?>
<USQL version="1.0">
  <find_servicesRequest>
    <Where>
      <Service serviceType="GridService">
        <ServiceDomain ontologyURI="http://http://nkua/sodium/usql/engine/ontology/dco"
          typeOfMatch="exact extension">
          AutoMobile</ServiceDomain>
        <Operation>
          <Capability ontologyURI="http://nkua/sodium/usql/engine/ontology/dco"
            typeOfMatch="exact">
            EstimateOrderProcessingTime</Capability>
          <Input>
            <semantics ontologyURI="http://nkua/sodium/usql/engine/ontology/dco"
              typeOfMatch="exact extension">
              ProductionPlan</semantics></Input>
          <Input>
            <semantics ontologyURI="http://nkua/sodium/usql/engine/ontology/dco"
              typeOfMatch="exact extension">
              OrderDeliveryDetails</semantics></Input>
          <Output>
            <semantics ontologyURI="http://nkua/sodium/usql/engine/ontology/dco"
              typeOfMatch="exact">
              OrderProcessingTime</semantics></Output>
        </Operation>
      </Service>
    </Where>
  </find_servicesRequest>
</USQL>
```

Figure 10. Example USQL request for Grid services

5. USQL ENGINE

The *USQL Engine* is a service search engine, based on the USQL language, which provides the means for accessing and querying heterogeneous service registries and/or networks in a unified and standards-based manner. The functionality offered by the engine is exposed as a Web service; thus, abiding by the SOA principles, the USQL Engine itself may be integrated in the context of a service-oriented industrial application allowing for automated service discovery.

The main concept underlying the USQL Engine framework is the abstraction regarding registry details, from the requestor's perspective. This is achieved with the adoption of a domain-centric categorization of the various supported registries, depending on the service advertisements they host. Domain information provided by the requestor is exploited by the engine so as to identify, access and query the appropriate registries in a transparent manner.

The USQL Engine follows an architecture distinguished by its high degree of openness and extensibility, which is achieved by applying plug-in mechanisms in order to accommodate virtually any type of service, registry, as well as their governing protocols and standards. The plug-ins used for this purpose can be integrated in a flexible manner, so as to enable different configurations and to broaden the range of supported registries.

Many of the tasks accomplished by the engine during service discovery are facilitated by an *Upper Ontology*, which forms a constituent part of the overall framework and reflects the domain-driven aspect of our approach. The ontology classes and properties mirror the semantic concepts supported by USQL and thus, the ontology is directly used for the population of semantic elements within USQL requests. Upon submission of a USQL request to the engine, the implicit identification of the registries and/or networks where the query will be forwarded is carried out by navigating in the ontology, making use of the domains specified by the requestor, and finding the registries that have been registered therein as belonging to these domains. Finally, reasoning during the matchmaking process is performed based on the structure and rules imposed by the upper ontology. *Figure 11* depicts the structure of the *USQL Engine Upper Ontology*:

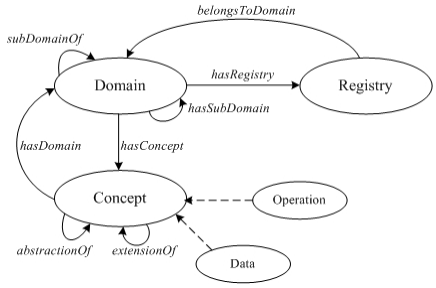

Figure 11. The USQL Engine Upper Ontology

The upper ontology consists of the following classes:

- ***Domain***: represents the domain where a service belongs to.
- ***Registry***: represents a registry/repository/network holding service advertisements.
- ***Concept***: represents Domain-specific concepts that may be used for describing services. A concept may be either an *Operation* or a *Data* description, related to a specific domain. Therefore, two subclasses of the *Concept* class are defined:
  - ***Operation***: represents an abstract functionality that is specific to a domain.
  - ***Data***: represents a piece of information that is specific to a domain.

It is worth noting that the *Concept* class is never instantiated. Instead, it serves as an abstraction to hold properties that are common to both operations and data.

The *Domain* class has the following properties:

- ***hasRegistry***: Takes as value a *Registry* instance. A domain may have zero or more associated registries.

- ***hasConcept***: Takes as value either a *Data* or an *Operation* instance.
- ***subDomainOf***: Takes as value a *Domain* instance. A domain may be the sub-domain of at most one parent domain.
- ***hasSubDomain***: Takes as value a *Domain* instance. A domain may have zero or more sub-domains.

Hence, with the use of the *subDomainOf* and *hasSubDomain* properties we can build a bi-directional tree, i.e. a domain hierarchy, which is easy to navigate.

The *Registry* class has the following property:

- ***belongsToDomain***: Takes as value a domain instance. A registry may belong to one or more domains, depending on the kind of service advertisements it holds.

The *Concept* class has the following properties:

- ***hasDomain***: Takes as value a *Domain* instance. A concept must belong to at least one domain.
- ***abstractionOf***: Takes as value either a *Data* or an *Operation* instance. A concept may be the abstraction of zero or more others concepts. More specifically:
  Concept A is an abstraction of concept B, if A subsumes B
- ***extensionOf***: Takes as value either a *Data* or an *Operation* instance concept. A concept may be an extension of at most one other concept. More specifically:
  Concept A is an extension of concept B, if A is subsumed by B.

The *abstractionOf* and *extensionOf* properties allow for the construction of data and operation concept hierarchies. Hence, the upper ontology provides a tree structure for both domains and their concepts, which is useful when applying reasoning and inference during service discovery.

6. RELATED WORK

Related work with respect to this paper can be classified into work related to the provision of a service model and work related to service discovery.

As far as work related to service models is concerned, one major approach is that of W3C. W3C's Architecture Working group in [w3c2004] has established a model for the specification of web services. The core concepts of the generic service model presented in this paper have a lot of similarities with the concepts of the W3C model. However, our model remains abstract enough allowing thus for extensions that are able to support p2p and grid services, whereas the W3C's model is confined to web services and lately to grid services complying with the WSRF specification [WSRF].

OASIS has recently announced the formation of a task group working on the specification of a service-oriented architecture reference model. This group has produced a working draft version of the reference model [SOARfMo]. However, the provided document is in draft version and no useful results can come out of it.

Service discovery on the other hand, is currently performed in the areas of Web, Grid, and P2P services with the use of custom APIs and discovery mechanisms offered by registries and networks.

UDDI [UDDI] has become the registry model of choice for publishing and discovering Web services. The framework provides for keyword-based search, allowing requestors to look for services according to their provider, classification, name, description etc. UDDI does not take into consideration semantic, as well as QoS descriptions and properties of services, although it provides a structure allowing the incorporation of arbitrary service descriptions within the registry. To exploit this feature, many efforts have been made towards integrating semantic annotations in UDDI; Paolucci et al. have proposed a way to map the OWL-S profile and process model in UDDI [Paolucci].

JXTA [JXTA] comprises a set of open, generic and implementation-independent p2p protocols allowing any device to communicate and collaborate as a peer over a network. One of the most important contributions of the JXTA framework is the explicit definition of p2p services, with the use of XML-based JXTA advertisements. This enhancement allows for the application of service discovery within JXTA networks, with the use of the standard discovery service provided by the platform. Still, JXTA protocols and advertisements are generic and very limited with respect to syntactic information, and moreover they do not detail crucial aspects of a p2p service like semantics and QoS, which could be exploited during service discovery.

JAXR [JAXR] provides a uniform and standard API for accessing different kinds of XML registries. On the other hand, the evolution of frameworks such as OWL-S and WSMO enables formulation of semantically-enhanced service requirements that can be checked against service offerings also described with the use of these frameworks.

Currently, a number of search engines have been proposed and/or implemented, all of which are activated in the area of Web services, without taking into account other existing types of services.

Woogle [Woogle], a search engine for Web services, enables similarity search by employing a set of matching and clustering algorithms with promising experimental measures and results. However, Woogle does not cater for the discovery of other types of services, while, in the context of Web services, matchmaking relies on the information provided in the WSDL

[WSDL] file and the UDDI entry only, without taking into account and exploiting semantics.

Like Woogle, other existing Web service search engines also focus on UDDI and WSDL descriptions of Web services, thus confining their queries to syntactic-based matchmaking only. SalCentral [SalCentral], a WSDL aggregator and analysis engine, allows for WSDL and XSD [XSD] based service lookups, while BindingPoint [BindingPoint] categorizes and provides access to a large number of Web services.

Nevertheless, it is clear that service-oriented development lacks a query language that would enable accessing and querying heterogeneous registries in a unified, standards-based manner. Moreover, exploitation of semantics and QoS within service descriptions proves to be a crucial part of service discovery. USQL and its enacting engine address these issues and constitute a stepping stone to the unification of the various heterogeneous service areas.

7. SUMMARY AND CONCLUSIONS

Industrial applications impose many requirements that can be met by following the SOA paradigm. Moreover, as shown in the scenario presented in this paper, a service-oriented industrial application will most probably consist of various heterogeneous services, which in turn may be described with the use of different semantic frameworks. Currently, most of the emerging semantic frameworks apply to the Web Service paradigm, without supporting directly other types of services. Yet, discovery of P2P as well as Grid services could be greatly facilitated by the accommodation of semantics, as it has been argued in this paper. This heterogeneity in existing service-oriented frameworks, protocols and standards, particularly in the area of service discovery constitutes a major obstacle towards the use of SOA paradigm in the development of industrial applications.

The solution presented in this paper, comprising a generic service model (GeSMO), a compliant query language (USQL) and its supporting engine provides for a unified way of discovering such diverse kinds of services, facilitating their interoperability and enabling their integration in industrial environments. More specifically, GeSMO facilitates the specification of heterogeneous services, while the USQL and its supporting engine enable the unified service discovery over heterogeneous registries and/or networks. The language inherits from the service model features such as abstraction, generality, openness, and extensibility, so as to allow for the seamless unification of the various types of services with respect to discovery. Moreover, we showed how the application of a generic set of domain-centric

semantics enhances service requests and flavors the task of service discovery with transparency, regarding the nature of the registries and networks that are being looked up.

8. ACKNOWLEDGEMENT

This work is partially supported by the European Commission under contract IST-FP6-004559 [SODIUM].

9. REFERENCES

[w3c2004] D. Booth, et al. Web Services Architecture, W3C Working Group Note, Feb 2004, http://www.w3c.org/ws-arch/

[AlCa2004] G. Alonso, et al, Web Service: Concepts, Architectures and Applications, Springer-Verlag, 2004

[SOA] SOA, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm

[SemWeb] T. Berners-Lee et al, *"The Semantic Web"*, Scientific American, May 2001.

[SODIUM] SODIUM http://www.atc.gr/sodium

[XML] XML, Extensible Markup Language, http://www.w3.org/XML/

[OWL-S] OWL-S http://www.w3.org/Submission/OWL-S/

[WSMO] WSMO, Web Service Modeling Ontology, http://www.wsmo.org/

[WSRF] K. Czajkowski, et al, The WS-Resource Framework, ver 1.0, http://www.globus.org/wsrf/specs/ws-wsrf.pdf

[SOARfMo] C. M. MacKenzie, et al Service-oriented Architecture Reference Model, working draft 07, OASIS, May 2005

[UDDI] UDDI, Universal Description, Discovery and Integration, http://www.uddi.org

[Paolucci] Massimo Paolucci, et al, "Importing the Semantic Web in UDDI", Proceedings of Web Services, E-business and Semantic Web Workshop, 2002

[JXTA] JXTA, Juxtapose Technology, http://www.jxta.org

[JAXR] JAXR, Java API for XML Registries, http://java.sun.com/xml/jaxr/index.jsp

[Woogle] Xin Dong et al, *Similarity Search for Web Services*, Procs of VLDB 2004, Canada

[WSDL] WSDL 1.1, W3C Note 15 March 2001, http://www.w3.org/TR/wsdl

[SalCentral] Salcentral, http://www.salcentral.com

[XSD] XSD, XML Schema Definition, http://www.w3.org/XML/Schema

[BindingPoint] Binding point, http://www.bindingpoint.com/