A Generic Query Model for the Unified Discovery of Heterogeneous Services

Michael Pantazoglou and Aphrodite Tsalgatidou

Abstract—In this paper, we propose Proteus, a generic query model for the discovery of operations offered by heterogeneous services. We demonstrate the need for such a model, and show how it unifies the task of service discovery through abstractions, which allow for the technology-independent formulation of service advertisements, queries, and query responses. On top of these generic elements, we build an intuitive, fuzzy-based query evaluation mechanism that supports the service matchmaking process by employing and appropriately combining existing similarity metrics. Thanks to the generality of Proteus, it is possible to seamlessly accommodate the discovery of operations provided by various types of services without the need of changing the existing service infrastructure. Thus, our approach is applicable to a variety of settings ranging from traditional Web services to service-oriented Grids, Peer-to-Peer networks, Geospatial Information Systems, etc. Overall, compared to the existing query models supported by standard service discovery technologies, our approach is marked by openness, flexibility, and improved performance in terms of precision and recall. The feasibility and efficiency of Proteus are verified by a series of experiments.

Index Terms—Services Discovery Process and Methodology, Web Services Interoperability, Metadata of Services Interfaces.

1 Introduction

Thanks to its high level of abstraction and the promise of interoperability among heterogeneous systems, service orientation appeals to designers and developers of distributed software applications. According to this computing paradigm, such applications are not necessarily built from scratch, but are rather composed by reuse of operations offered by existing software services, which can be discovered and appropriately orchestrated to deliver the desired functionality. Still, despite its importance, service discovery is widely regarded an intricate task, due to numerous deficiencies of the current standard query models and their associated mechanisms. In most cases, queries can only express search criteria that constrain trivial, service-level properties such as the service name, provider, or classification, while meaningful requirements on more important properties of the desired service operation cannot be directly expressed. As a result, the query response must often be further processed (in some cases manually) by the service requester, in order to check whether the included service operations suit their needs or not. Since the query evaluation mechanisms are mostly limited to keywords-based matchmaking, which yields poor results in terms of precision and recall, such processing becomes a considerably cumbersome procedure.

Service discovery becomes even more aggravating in the presence of operations that are implemented by different kinds of services. In our previous work [1], we found that, although generally abiding by the principles of the service-oriented model, such services are characterized by multi-dimensional heterogeneity. Being manifested by different, service model-specific properties, and the consequent use of divergent technologies for service description [2] [3] [4] [5] [6] [7], publication and discovery [8] [9] [10] [11] [12], such heterogeneity introduces additional burdens to service requesters. To better illustrate the problem, let us consider the following scenario, which has been inspired by one of the pilot applications developed in the SODIUM project [13].

• *The authors are with the Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, Panepistimiopolis, Ilissia, Athens 15784, Greece.*
E-mail: {michaelp,atsalga}@di.uoa.gr

1.1 Motivating Scenario

The IT department of a private company specializing in the domain of crisis management has adopted service orientation to develop the company's business processes. One of these processes supports the timely dispatching of assistance to an area where an explosion to a chemical factory has occurred, while also catering for evacuation planning. Each task in the process is implemented by an operation provided by a service. However, different types of services are needed, depending on the type of functionality. The process is activated upon receipt of an emergency call by the emergency unit call center, and its first task is to track the caller's position with the use of a Web service. Based on that information, the following sequences of tasks are executed in parallel:

- The nearest emergency unit is resolved, and an optimized route is calculated from that unit to the location of the accident, through the use of two Web services, respectively. In succession, the unit's ambulances are checked for availability, and the driving directions are dispatched to all available ones. Both these tasks are accomplished by making

use of multicasting *Peer-to-Peer (P2P) services* [10] allowing direct communication between the emergency center and each ambulance.
- Information about the wind direction in that area is retrieved through a *Sensor service* [14], and the retrieved observations are used by a high-performing *Grid service* [15], which generates a model for the dispersion of the chemical plume. The result is uploaded to the website of the area's civil protection agency through a secure *RESTful service* [16] [17], so that evacuation activities can be planned and executed accordingly.

Responding to such application scenarios, many service-oriented workflow tools [18] [19] [20] have addressed service heterogeneity through appropriate abstractions and plug-ins accommodating the different service invocation mechanisms. Thus it is feasible to uniformly invoke operations provided by any type of service from within a service-oriented process, regardless of the service's underlying peculiarities. Still, these frameworks are incomplete in the sense that they do not also cater for the discovery of heterogeneous services.

Going back to our scenario, the process designer/developer needs to express different kinds of requirements in order to discover each one of the desired services. For instance, in the case of operations offered by Web services, it is usually sufficient to express requirements on the desired *capability*, *inputs* and *outputs*; as for RESTful services, additional, operation-level requirements are useful to their discovery, such as the *HTTP method*; for P2P service operations, the requester needs in addition to be able to specify constraints on elements of the *peers* offering them, such as their *id* or belonging *group*; to find the appropriate Sensor service, certain details of the desired *observation offerings* need to be expressed in the query (e.g. semantics of the observed property, coordinates of covered area, etc.); as for the Grid service, it is more important that the underlying *resource properties* meet the user needs. Finally, for all service types, there is often a common need to express the desired QoS for each operation, such as *availability*, *performance*, or *cost*.

These properties of the above mentioned services are described by different languages, and the resulting documents are advertised in different brokers supporting heterogeneous query models. Thus the service requester is compelled to separately employ the different discovery mechanisms and express queries in many different ways, which still cannot fully capture requirements such as the aforementioned ones.

1.2 Contribution

In this paper, we tackle the problems that arise upon service discovery, when users need to search for operations delivered by services of different types. In particular, we are interested in answering the following challenging research questions:

- Q_1: *How can service discovery solutions attain independence from the conceptual peculiarities of the available service models?*
- Q_2: *How can service requesters express and effectively combine –within a single query– requirements towards an open set of service properties?*
- Q_3: *How can the overall degree of match for a service operation be calculated and quantified, given a query that combines different types of requirements having potentially different levels of importance?*
- Q_4: *Is it feasible for a query model to adapt to heterogeneous service environments without imposing changes on their existing technologies and infrastructure?*
- Q_5: *Is it possible for a query model to seamlessly accommodate additional kinds of requirements towards services without changing its existing structures?*

Motivated by the identified gap in solutions that could provide answers to all of these questions, we propose and evaluate a new, generic query model called Proteus, to support the *unified* discovery of heterogeneous services. The proposed approach is based on the premise that, service discovery should be orthogonal to the various service models and technologies. To this end, Proteus unifies the course of service discovery through the provision of abstractions for the construction of service advertisements, service discovery queries and responses.

Proteus supports the expression and evaluation of requirements towards an open set of functional and non-functional service properties. The underlying matchmaking mechanism works on top of appropriate abstractions for the description of service requirements in queries, and service properties in service advertisements. Also, query evaluation considers the different weights of the various requirements, thereby capturing the requester's priorities upon service discovery. The similarity of a service operation to the user requirements is conveniently quantified facilitating the service selection process. This way, our approach significantly improves the precision and recall of search results compared to the limited, keywords-based query mechanisms currently supported by existing service broker technologies.

Moreover, Proteus is generic, open, and flexible. For instance, introducing a new service property and its corresponding requirement in the model does not affect its structure or its evaluation mechanism. Similarly, thanks to the modularity characterizing the model, new features can be introduced to accommodate emerging types of services, without requiring changes to the existing elements. All in all, Proteus is intended to provide the foundations for the development of unified service discovery languages and tools that can be flexible, extensible and sustainable, thus immune to the heterogeneity and volatility of service-oriented technologies.

2 Related Work

Despite the considerable volume of research results in service discovery, the various proposed approaches only

TABLE 1
Evaluation of related work on the basis of requirements for heterogeneous services discovery

Query Model	Q_1	Q_2	Q_3	Q_4	Q_5
OWL-S/UDDI Matchmaker [21]	No	No	Partially	No	Yes
ODEN [22]	No	Partially	Partially	No	Partially
OWLS-MX Matchmaker [23]	No	Partially	Partially	No	Partially
WSMO-MX Matchmaker [24]	No	Partially	Partially	No	Partially
XAM4SWS [25]	No	Partially	Yes	Partially	No
USDL [26]	No	No	Partially	No	No
ASD Framework [27]	No	Partially	Yes	Partially	No
WS Query Algebra [28]	No	Partially	No	Yes	Partially
G-QoSM [29]	No	Partially	No	No	No
Woogle Search Engine [30]	No	No	Yes	No	No
WS Ranking & Clustering [31]	No	No	Yes	Yes	No
Service CatalogNet [32]	Yes	Partially	Partially	No	Yes
OSDA [33]	Yes	No	No	Yes	No

partially address the discovery of heterogeneous services, mainly because they were not fundamentally designed to do so. In this section, we evaluate and discuss the related work focusing on the proposed query models, while we assess their limitations on the basis of the five research questions (Q_1–Q_5) that were raised in the previous section. Serving as a point of reference throughout the discussion, Table 1 summarizes the evaluation results, and indicates the extent to which each question is answered by each one of the studied solutions.

Many service discovery frameworks such as OWL-S [3] and WSMO [34] have embraced the Semantic Web technologies in order to improve the matchmaking process. Among the first attempts to build upon such frameworks, the OWL-S/UDDI matchmaker [21] is based on the OWL-S language for semantically describing Web services, and provides extensions to the UDDI query model to accommodate the evaluation of semantically enhanced queries. Similarly, the OWL-S language has been applied to P2P service discovery in JXTA networks, through ODEN [22]. Being tightly coupled with the respective technologies and their conceptual models, these approaches are not applicable to the discovery of other types of services, while queries can only express requirements towards the closed set of service properties that are supported by OWL-S, namely the inputs, outputs, preconditions, and effects of the desired Web service. It should be noted though that, despite these limitations, these approaches could accommodate additional requirements, thanks to the extensibility of the UDDI and JXTA protocols, respectively. In any case, unlike Proteus, users are not supported in prioritizing their requirements, while matchmaking is performed on the basis of four predefined degrees of match, which might leave partial, yet potentially useful matches out of the query response.

Other approaches, such as the OWLS-MX [23] and WSMO-MX [24] matchmakers, provide a hybrid solution to the expression and evaluation of semantics-based queries, by effectively combining different matching mechanisms, in addition to the use of ontologies. This way, they extend the default query evaluation mechanisms of the OWL-S and WSMO frameworks, respectively, with additional support for syntactic, structural, and IR-based matching. Still, these matchmakers do not overcome the limitations of OWL-S and WSMO in heterogeneous service discovery: They are inherently technology-specific, support a closed set of service properties, and are only as flexible as the limited extensibility of these frameworks allows. On the contrary, Proteus combines different matching mechanisms for the evaluation of the different kinds of requirements that may be set in a query, while also addressing these limitations.

XAM4SWS [25] constitutes another semantics-based query model supporting service discovery based on requests that specify the desired functional properties of service operations. XAM4SWS puts emphasis on supporting the discovery of RESTful services, in addition to Web services, by exploiting the recently emerged hRESTS and MicroWSMO markups for service description [7]. Like Proteus, the proposed matchmaker quantifies the similarity of an offered service operation with respect to a given request, while weights are used to differentiate the search criteria priorities. However, the expressiveness of XAM4SWS is restricted to a limited set of requirements, which specify the desired semantics of service operations, along with their input and output messages. Thus, this approach cannot be directly applied to the discovery of other types of services, which often requires the expression of different kinds of requirements, including constraints on the non-functional service properties.

Departing from the widespread use of OWL-S and WSMO as the foundations of service discovery solutions, the USDL [26] approach employs WordNet [35] to semantically describe concepts that are then used to annotate the abstract part of WSDL documents. The universality of WordNet assists in tackling the diversity of semantic annotations that are otherwise produced by means of different domain ontologies. Further, USDL addresses service matchmaking by developing a theory

for service substitution that is based on the WordNet-supported semantic relations among concepts. Overall, although this approach addresses heterogeneity of service descriptions at the semantic level, it fails to fully accommodate the discovery of heterogeneous services.

WordNet has also been employed by the ASD framework [27] as a means to semantically describe the capability, inputs and outputs of Web service operations. Similarly to our approach, the ASD matchmaking process considers different weights in constraints and yields quantified degrees of match, by combining various similarity distance measures. Also, ASD as well as the formal query algebra for Web services that was proposed in [28] expands the set of supported search criteria during service discovery, allowing for the expression and evaluation of constraints towards functional, behavioral, and qualitative service properties. Besides, the G-QoSM framework [29] proposes an SLA-based query model to empower the discovery of Grid services based on their qualitative properties. Yet, unlike Proteus, all these approaches lack the generality and openness necessary for dealing with other types of services. Proteus supports the expression and evaluation of a much wider range of requirements, since its formal definition has been based on abstract constructs, instead of concrete, service property-specific elements.

In response to real-world situations, the Woogle [30] search engine implements a set of algorithms capable of operating on single WSDL documents with limited, keywords-based content. As in Proteus, the proposed matchmaker considers the different priorities among the search criteria in queries, and effectively uses this information upon quantification of the degree of match for a service operation. Woogle as well as the query model that was recently proposed in [31] employ clustering techniques to improve the assessment of service similarity. On the one hand, both of these approaches are not applicable to the discovery of heterogeneous services, mainly because of their limited set of supported search criteria. On the other hand, however, their contributed ideas revolving around services clustering could be further extended and complementary used by query models such as Proteus to further enhance the indexing and matchmaking processes.

The Unified Service Discovery Framework, presented in [32], provides appropriate models for service description and discovery and is applicable to a wide range of settings where service discovery is required. The proposed query model supports QoS and contextual information both in service description and discovery, and utilizes the XQuery language to construct the service discovery requests. This renders the resulting queries tightly coupled with the structure of the service advertisements, thereby limiting their expressiveness in terms of the requirements that can be supported. In our approach, queries are formulated independently of the heterogeneous service description languages, as the latter can be easily mapped into Proteus service advertisements. In another effort to unify the service discovery process with respect to the various heterogeneous service discovery domains, the authors in [33] propose an open, peer-to-peer architecture accompanied by a unified service description scheme. The latter is used to generate templates for describing services regardless of their type, although it relies on external specifications for the description of service properties and requirements. Thus, the overall approach acts more as middleware that achieves interoperability among heterogeneous service discovery mechanisms, rather than proposing a new query model.

In conclusion, compared to the investigated query models, we argue that our approach is more appropriate for the discovery of heterogeneous services, as it provides a unified interface to facilitate the work of service requesters operating in different environments. The inherent ability of Proteus to effectively address the five research questions through its *genericity* (Q_1), *expressiveness* (Q_2), *fuzziness* (Q_3), *flexibility* (Q_4), and *extensibility* (Q_5), is presented in the following sections.

3 The Proteus Query Model

We introduce the Proteus query model to enable the unified discovery of operations offered by heterogeneous services. The model supports three types of documents, namely the *service advertisement*, which is used to describe the properties of a service operation, the *query*, which captures the requirements of service requesters in regards to the desired service operation, and the *query response*, which is used to convey the service discovery results. On top of these documents, a mechanism is defined for *query evaluation*, which quantifies the similarity of a service advertisement to the requirements of a query.

The structure and contents of the three supported types of documents are defined in a generic, technology-agnostic manner by means of appropriate abstractions. A service advertisement is a collection of *service properties*, which in turn comprise one or more *features*. A query contains one or more *requirements* referring to particular service properties, with the former combining one or more *constraints* set on the features of the latter. Finally, a query response includes the *matches*, being the service advertisements that satisfy the query's requirements, accompanied by an indicative *degree of match*. All these concepts are formally defined and discussed in the following paragraphs.

3.1 Service Properties and Advertisements

At the core of our proposed approach lie appropriate abstractions that enable us to model service properties and service advertisements in a generic manner. These abstractions have been based on the observation that, regardless of their conceptual model, services are characterized by various functional and non-functional properties, which in turn comprise one or more features.

We conveniently consider an open set $\mathcal{F}$ containing all different types of features (denoted as t_f) that are met in service properties. Being open, this set allows for the seamless introduction of new service property features in our model, if needed. Thus, the concept of service property is defined by a generic structure as follows.

Definition 3.1. *Given the set of feature types, $\mathcal{F}$, a service property takes the form of a pair*

$$p = (n, F_p)$$

where n is a name uniquely identifying the property type, and F_p is the set of features, whose type, $t_f \in \mathcal{F}$, is associated with that property type. Furthermore, $\forall f_i, f_j \in F_p$, $1 \leq i < j \leq |F_p|$, we have $t_{f_i} \neq t_{f_j}$, which means that a service property is not allowed to define more than one features for each supported feature type.

Currently, $\mathcal{F}$ contains the following feature types, which can be combined according to Definition 3.1 to describe an open set of service properties:

- **String value**. Features of this type take the form of a string value, $s \in \Sigma^*$, with Σ being an alphabet, $|\Sigma| > 0$.
- **Numeric value**. This feature type is used to describe numeric properties. Thus a numeric value feature is simply given as $v \in \mathbb{R}$.
- **Semantics**. The semantics of a service property are defined by a semantics feature given as a pair $\sigma = (\pi, o)$, where π is a textual description, and o is a URI corresponding to some ontology concept.
- **Data type**. A service property may have a data type, $t = (n, ns)$, where n is the type name, and ns is its namespace.
- **Qualifiers**. A service property may contain an arbitrary number of *qualifiers* defined as an ordered set of service properties, P_Q. Each element in P_Q corresponds to a different qualifier type, which means that, for any given pair of properties $p_i = (n_i, F_{p_i}) \in P_Q$ and $p_j = (n_j, F_{p_j}) \in P_Q$, $1 \leq i < j \leq |P_Q|$, we have $n_i \neq n_j$.
- **Sub-properties group**. Besides qualifiers, a service property may contain grouped sub-properties of the same type. This kind of feature is defined as a pair $g = (n, P_g)$, where n is the group name, and P_g is an ordered set of service properties such that, for any given pair of properties $p_i = (n_i, F_{p_i}) \in P_g$ and $p_j = (n_j, F_{p_j}) \in P_g$, $1 \leq i < j \leq |P_g|$, we have $n_i = n_j$.

With the use of Definition 3.1, we are able to introduce an appropriate abstraction for service advertisements, which are used to describe the functional and non-functional properties of service operations.

Definition 3.2. *A service advertisement is defined as a pair*

$$A = (id_A, P)$$

where id_A is a unique identifier, and P is the set of advertised service properties, such that, for any given pair of properties $p_i = (n_i, F_{p_i}) \in P$ and $p_j = (n_j, F_{p_j}) \in P$, $1 \leq i < j \leq |P|$, we have $n_i \neq n_j$. In other words, a service advertisement is not allowed to have more than one instances of the same service property.

In what follows, we formalize the query and query response documents by introducing similar abstractions.

3.2 Query and Response Documents

Proteus uses the concept of requirement to denote a search criterion that can be set by the service requester in regards to a given service property. Like service properties, requirements are modelled by a generic structure.

Definition 3.3. *Let $p = (n, F_p)$ be a service property according to Definition 3.1. Then, a requirement towards p is defined as*

$$r = (n, C)$$

where $C = \{(w, c(f)) : (w \in \mathbb{N}^*) \wedge (f \in F_p)\}$ is a set of constraints towards one or more of the features of p. More specifically, for each element in C, $c(f)$ denotes a constraint on feature $f \in F_p$, while w is a numeric weight denoting the significance of that constraint to the service requester (i.e., the more a constraint is weighted, the more important it is). Also, each constraint included in C corresponds to a different feature of p. This means that, $\forall (w_i, c(f_i)), (w_j, c(f_j)) \in C$, $1 \leq i < j \leq |C|$, we have $f_i \neq f_j$.

The concrete definition of a constraint, $c(f)$, depends on the type of its corresponding feature, f. For the currently supported feature types in $\mathcal{F}$, Proteus defines the following constraint types:

- **String value constraint**. In the case of a string value, $s \in \Sigma^*$, this constraint type defines a pair $c(s) = (S, \tau)$. In this pair, $S \equiv \cup_{i=1}^{n} S_i$, $n \geq 1$, where $S_i = op(s_i) \subseteq \Sigma^*$ are the implied sets produced by applying $op \in \{equals, startsWith, endsWith, contains\}$ to string values $s_i \in \Sigma^*$, while $\tau \in \{0, 1\}$ denotes whether these values are accepted ($\tau = 1$) or not ($\tau = 0$).
- **Numeric value constraint**. In the case of a numeric value, v, the constraint is defined as $c(v) = E$, where $E = [v_{min}, v_{max}]$, $v_{min} \in \mathbb{R}$ and $v_{max} \in \mathbb{R}$, $v_{min} \leq v_{max}$, is used to specify the desired value range.
- **Semantics constraint**. Allowing for the expression of the desired semantics of a service property, the constraint takes the form $c(\sigma) = (\pi_r, o_r)$, with π_r being a textual description, and o being a URI corresponding to some ontology concept.
- **Data type constraint**. A constraint on such feature of a service property specifies the name, n_r, and namespace, ns_r, of the desired data type, and is defined as $c(t) = (n_r, ns_r)$.
- **Qualifiers constraint**. The service requester may specify requirements on one or more of the qualifiers of a service property. In such case, the constraint specifies a set of requirements, $c(P_Q) = R_Q$, such that $\forall r = (n_r, C) \in R_Q \; \exists p = (n, F_p) \in P_Q$:

$n_r = n$. Furthermore, $\forall r_i = (n_i, C_i), r_j = (n_j, C_j) \in R_Q$ we have $n_i \neq n_j$.

- **Sub-properties group constraint**. A constraint on the sub-properties group, $g = (n, P_g)$, of a service property takes the form $c(g) = (n, R_g, \theta)$, where R_g is a set of requirements such that $\forall r = (n_r, C) \in R_g$ and $\forall p = (n_p, F_p) \in P_g$ we have $n_r = n_p$. Also, $\theta \in \{0, 1\}$ is an indicator of whether the cardinalities $|R_g|$ and $|P_g|$ should match ($\theta = 1$), or not ($\theta = 0$).

We are now in a position to formally define the structure of service discovery queries.

Definition 3.4. *Let $C_A = \{A : A = (id_A, P)\}$, $|C_A| \geq 1$, be a set of service advertisements. A service discovery query on the contents of C_A is defined as a tuple*

$$Q = (id, \epsilon, \mu, R)$$

where id is a string representing the query identifier; $\epsilon \in (0, 1]$ is the minimum acceptable degree of match for each advertisement $A \in C_A$; $\mu \in \mathbb{N}^*$ is the maximum allowed number of search results; $R = \{(w, r) : (w \in \mathbb{N}^*) \wedge (r = (n, C))\}$ is a set of service requirements that are formulated according to Definition 3.3, with each requirement accompanied by a numeric weight denoting its significance to the service requester (i.e., the more a requirement is weighted, the more important it is).

The response to a service discovery query Q contains a subset of corpus C_A, which consists of the service advertisements that adequately satisfy all specified requirements.

Definition 3.5. *Let $Q = (id, \epsilon, \mu, R)$ be a service discovery query and C_A be a set of service advertisements, according to Definition 3.4. The response to Q is defined as*

$$R_Q = (id, id_Q, M)$$

where id is a string representing the response identifier; id_Q is a reference to the query identifier ensuring correlation between Q and R_Q; $M = \{m_Q(A)/A : (A \in C_A) \wedge (m_Q(A) \geq \epsilon)\}$ is an ordered fuzzy set, with $m_Q(A) \in [0, 1]$ being the grade of membership *of service advertisement A in M, such that $\forall A_i, A_j \in M$, $1 \leq i < j \leq |M| \leq \mu$, we have $(id_{A_i} \neq id_{A_j})$ and $m_Q(A_i) \geq m_Q(A_j)$.*

We proceed in the following section with a formal definition of the grade of membership, $m_Q(A)$, which determines whether a service advertisement, A, should be included in the response, R_Q, based on its degree of match with the requirements expressed in a query, Q.

3.3 Query Evaluation

The grade of membership of a service advertisement in the response to a given query naturally depends on the extent, to which the specified requirements are satisfied. Thus, for a given query $Q = (id, \epsilon, \mu, R)$ and a given advertisement $A = (id_A, P)$, $m_Q(A)$ is calculated through proper evaluation of the contents of R against the contents of P.

The first step in the query evaluation process is to organize the requirements of R according to their weights. Specifically, if $|R| > 0$, we have $R \equiv \cup_{i=1}^{k} R_i$, where:

- k, $1 \leq k \leq |R|$, is the number of different weights, $w_i \in \mathbb{N}^*$, that have been assigned to the requirements in R
- $R_i \subseteq R$ contains all requirements $r \in R$ having weight w_i
- $\forall i, j : 1 \leq i < j \leq k$ we have $w_i < w_j$ and $R_i \cap R_j = \emptyset$, while $\sum_{i=1}^{k} |R_i| = |R|$

Next, we obtain the subset $P_R \subseteq P$ containing the service properties that have been constrained by the requirements in R, and organize it in a similar fashion. This yields $P_R \equiv \cup_{i=1}^{k} P_i$, where $P_i \simeq R_i$, which means that P_i and R_i can be put into *one-to-one correspondence*.

At this point, $m_Q(A)$ is calculated by applying the following linear combination:

$$m_Q(A) = \frac{1}{\sum_{i=1}^{k} w_i} \cdot \sum_{i=1}^{k} \left(w_i \cdot \prod_{j=i}^{k} d_{R_j}(R_j, P_j) \right) \quad (1)$$

In the above, $d_{R_j}(R_j, P_j) = (1/|R_j|) \cdot \sum_{l=1}^{|R_j|} d(r_l, p_l)$ is the average degree of match accrued by the individual degrees of match $d(r_l, p_l) \in [0, 1]$ between each requirement $r_l \in R_j$ and its corresponding service property $p_l \in P_j$, $1 \leq l \leq |R_j|$.

Equation 1 effectively captures the different requirement priorities, which are expressed by the service requester as numeric weights. The higher a requirement is prioritized within the query, the more its degree of match affects the grade of membership of service advertisements. Moreover, if a service advertisement A fails to adequately meet the requirements with the highest priority ($d_{R_k}(R_k, P_k) < \epsilon$), it is automatically excluded from the response, since we also obtain $m_Q(A) < \epsilon$.

Going back to the calculation of the degree of match between a requirement $r = (n, C)$ and its corresponding service property $p = (n, F_p)$, we see an analogy with the calculation of the grade of membership. Indeed, like the requirements in the query, the constraints within each requirement may have been given different weights. Hence we first split C in subsets, $C \equiv \cup_{i=1}^{k} C_i$, where k is the number of different weights, $w_i \in \mathbb{N}^*$, that have been assigned to the constraints in C, while $C_i \subseteq C$ contains all constraints $(w, c(f)) \in C$ having $w = w_i$. Also, $\forall i, j : 1 \leq i < j \leq k$ we have $w_i < w_j$ and $C_i \cap C_j = \emptyset$, while $\sum_{i=1}^{k} |C_i| = |C|$. Similarly, we split the subset $F_C \subseteq F_p$ of the features of p that have been constrained by r, $F_C \equiv \cup_{i=1}^{k} F_i$, so that we have $C_i \simeq F_i$.

Following these preparatory steps, the degree of match between r and p is calculated as follows:

$$d(r, p) = \frac{1}{\sum_{i=1}^{k} w_i} \cdot \sum_{i=1}^{k} \left(w_i \cdot \prod_{j=i}^{k} d_{C_j}(C_j, F_j) \right) \quad (2)$$

In the above equation, we have $d_{C_j}(C_j, F_j) = (1/|F_j|) \cdot \sum_{l=1}^{|F_j|} d(c(f_l), f_l)$, where $d(c(f_l), f_l) \in [0,1]$ is the degree of match between the constraint $c(f_l)$ and the feature f_l, which depends on the type of f_l.

Algorithmic Analysis: Let us consider a requirement, r, containing m constraints with $k \leq m$ weights. In the worst case, where $k = m$, Equation 2 takes the form $d(r,p) = \frac{1}{\sum_{i=1}^{m} w_i} \cdot \sum_{i=1}^{m} \left(w_i \cdot \prod_{j=i}^{m} d(c(f_j), f_j) \right)$. Thus, the calculation of $d(r,p)$ requires $(2m-2)$ additions and $\frac{1}{2}(m^2+m)$ multiplications. Now, let us consider a query, Q, with $n = |R|$ top-level requirements and $k \leq n$ weights. Similarly, in the worst case ($k = n$), Equation 1 becomes $m_Q(A) = \frac{1}{\sum_{i=1}^{n} w_i} \cdot \sum_{i=1}^{n} \left(w_i \cdot \prod_{j=i}^{n} d(r_j, p_j) \right)$, and accordingly requires $(2n-2)$ additions and $\frac{1}{2}(n^2+n)$ multiplications. Thus, the overall calculation of $m_Q(A)$ has a worst-case bound of $O(n^2 + n \cdot m^2)$, in terms of runtime complexity. Note though that, in practice, we usually have $n < 10$, whereas, in its current form, our query model mandates that $m \leq 6$.

Let us now describe the matchmaking mechanisms for each one of the feature types supported by Proteus.

Matching String Value Constraints: Processing of a constraint on a string value simply checks whether the string value belongs to the set of accepted, or not accepted values specified by the constraint. Thus the matchmaking process resembles the simple, keyword-based search mechanisms that are currently supported by most standard service brokers [8] [9]. More specifically, let s be a string value feature of a service property, and $c(s) = (S, \tau)$ be a constraint on s. Then, their degree of match is given as follows:

$$d(c(s), s) = 1 - (\tau + (-1)^{\tau} \cdot \chi_S(s)) \tag{3}$$

where χ_S is the *indicator function* of set S.

Matching Numeric Value Constraints: In order to evaluate a constraint on the numeric value of a service property, we employ the one-dimensional *Euclidean distance* metric in its normalized form. Thus, the degree of match between a numeric value v and a constraint $c(v) = E$ is calculated as:

$$d(c(v), v) = 1 - (1 - \chi_E(v)) \cdot m_{\alpha,\beta} \cdot M_{\alpha,\beta}^{\chi_E(v)-1} \tag{4}$$

where $m_{\alpha,\beta} = \min(\alpha, \beta)$ and $M_{\alpha,\beta} = \max(\alpha, \beta)$, with $\alpha = |v - v_{min}|$, $\beta = |v - v_{max}|$, while χ_E is the indicator function of the range of accepted values for v, $E = [v_{min}, v_{max}]$.

Matching Semantics Constraints: A semantics constraint can be expressed in Proteus through a textual value and/or with the use of an appropriate ontology concept. Thus, processing of semantics constraints involves the similarity assessment between textual values, as well as between ontology concepts. Overall, the degree of match between a semantics constraint $c(\sigma) = (\pi_r, o_r)$, and a semantic property, $\sigma = (\pi, o)$, is calculated as:

$$d(c(\sigma), \sigma) = \max\left(x_\pi \cdot d(\pi_r, \pi), x_o \cdot d(o_r, o)\right) \tag{5}$$

where $x_\pi + x_o > 0$, with:

$$x_\pi = \begin{cases} 0, & \pi_r = \text{null} \\ 1, & \pi_r \neq \text{null} \end{cases} \text{ and } x_o = \begin{cases} 0, & o_r = \text{null} \\ 1, & o_r \neq \text{null} \end{cases}$$

In Equation 5, $d(\pi_r, \pi) \in [0,1]$ quantifies the similarity between textual values π_r and π, whereas $d(o_r, o) \in [0,1]$ quantifies the similarity between ontology concepts o_r and o. In order to calculate $d(\pi_r, \pi)$, we have adopted the *Vector Space Model* [36], whereby textual values are represented as term documents. For each such document, which can be lexically processed so as to enhance its content according to [37], a corresponding vector $\vec{w}$ is created containing the weights of its constituent terms. Weights are calculated with the use of the *Term Frequency - Inverse Document Frequency (TF-IDF)* metric [38]. Hence, the degree of match between two given term documents having TF-IDF vectors $\vec{w}_{\pi_r}$ and $\vec{w}_{\pi}$, which correspond to the textual descriptions of a semantics constraint, π_r, and its respective semantics feature, π, is calculated with the use of the *cosine coefficient*:

$$d(\pi_r, \pi) = \cos(\vec{w}_{\pi_r}, \vec{w}_{\pi}) = \frac{\vec{w}_{\pi_r}^T \cdot \vec{w}_{\pi}}{\|\vec{w}_{\pi_r}\| \cdot \|\vec{w}_{\pi}\|}$$

For the assessment of similarity between two given concepts in an ontology, numerous methods have been proposed in the literature [39]. In our approach, assuming the existence of a common ontology for service requesters and providers, or the existence of established mappings between different ontologies, we consider the distance between the two ontology concepts in the ontology graph in order to calculate $d(o_r, o)$. Specifically, we employ *Dice's* coefficient of similarity [40], which is defined as $S_{dice}(X,Y) = 2 \cdot |X \cap Y| / (|X| + |Y|)$, where X and Y are the sets of *is-a* links between the root of the ontology and concepts o_r and o, respectively. Further, we capture the direction and hierarchy semantics between the advertised concept, o, and the requested one, o_r, by introducing a numeric variable, ρ, which quantifies the widely used semantic relations: *exact* ($\rho = 3$); *plug-in* ($\rho = 2$); *subsume* ($\rho = 1$); and *fail* ($\rho = 0$). Thus, the similarity assessment between the requested ontology concept, o_r, in a semantics constraint, and the corresponding one in the semantics feature, o, is performed by the following equation:

$$d(o_r, o) = \frac{1}{3} \cdot [\rho + (3 - \rho) \cdot S_{dice}] \tag{6}$$

Matching Data Type Constraints: As data type matching primarily depends on the type system, where the two given types are defined, the implementation of an evaluation mechanism goes beyond the scope of our query model. Nevertheless, to retain independence from such technology-specific details, appropriate plug-ins can be introduced to support type matching in the

context of different type systems, such as XML Schema[1] (e.g. see [41]), or JSON Schema[2].

Matching Qualifiers Constraints: To evaluate a constraint on the qualifiers of a service property, we first need to identify which of the qualifiers were constrained. More specifically, let $c(P_Q) = R_Q$ be a qualifiers constraint on P_Q. We extract subset $P_Q^{'} \subseteq P_Q$, such that, $\forall r_i = (n_r, C) \in R_Q$, $1 \leq i \leq |R_Q|$, $\exists p = (n, F_p) \in P_Q^{'}$ where $n_r = n$. Then, the degree of match between R_Q and P_Q is an average:

$$d(R_Q, P_Q) = d(R_Q, P_Q^{'}) = \frac{1}{|R_Q|} \cdot \sum_{i=1}^{|R_Q|} d(r_i, p_i) \quad (7)$$

where $r_i = (n, C) \in R_Q$, $p_i = (n, F_p) \in P_Q^{'}$, while $d(r_i, p_i)$ is calculated according to Equation 2.

Matching Sub-properties Group Constraints: The evaluation of a constraint $c(g) = (n, R_g, \theta)$ on a sub-properties group $g = (n, P_g)$ involves the evaluation of the requirements $r \in R_g$ against the properties $p \in P_g$. The calculation of the degree of match between R_g and P_g can be defined as an instantiation of the *Assignment Problem* [42]. Indeed, in our case, we need to find the optimum assignment of the properties in P_g to the requirements in R_g, i.e. the one that yields the maximum overall degree of match between the two sets. We solve the Assignment Problem in polynomial time by appropriately adapting and utilizing the well-known *Hungarian Method* [43]. Taking as input the two sets, R_g and P_g, the algorithm produces as output a set of triples $S_H = \{(r, p, d_o) \mid (r \in R_g) \wedge (p \in P_g) \wedge (d_o \in [0, 1])\}$, where $S_H \simeq R_g$. Each entry in S_H informs that, the optimum degree of match d_o for requirement $r \in R_g$ is obtained if we match r with property $p \in P_g$. The calculation of $d_o = d(r, p)$ is performed according to Equation 2.

Having resolved the optimum degrees of match for each requirement $r \in R_g$, and considering the value of θ, the degree of match between $c(g)$ and g is calculated as:

$$d(c(g), g) = \frac{1}{3} \cdot d_c(R_g, P_g) \cdot [3 - \theta \cdot (1 - d_s(R_g, P_g))] \quad (8)$$

where $d_c(R_g, P_g) = \frac{1}{|R_g|} \cdot \sum_{i=1}^{|R_g|} d_{o_i}$, with d_{o_i} being the optimum degree of match of requirement $r_i \in R_g$, and $d_s(R_g, P_g) = 1 - \frac{||R_g| - |P_g||}{\max(|R_g|, |P_g|)}$.

Thus, the degree of match between a sub-properties group feature and its corresponding constraint is determined by the average of the optimum degrees of match of the requirements in R_g, while, depending on the requester's preference, it may also optionally reflect the potentially different sizes of R_g and P_g.

1. http://www.w3.org/XML/Schema
2. http://json-schema.org/

4 The Proteus Engine

Proteus is supported by a query engine, which was particularly designed to enable the unified discovery of heterogeneous services. Among the most essential features of the engine is its openness, thanks to which it effectively addresses the diversity of existing service description and discovery technologies. Thus, the engine allows for the seamless application of the Proteus query model in many diverse service-oriented environments, without imposing any changes to their existing infrastructure. Although a detailed presentation of the Proteus engine goes beyond the scope of this paper, we herein briefly report on its reference architecture, so as to better explain how our proposed approach to unified service discovery is implemented.

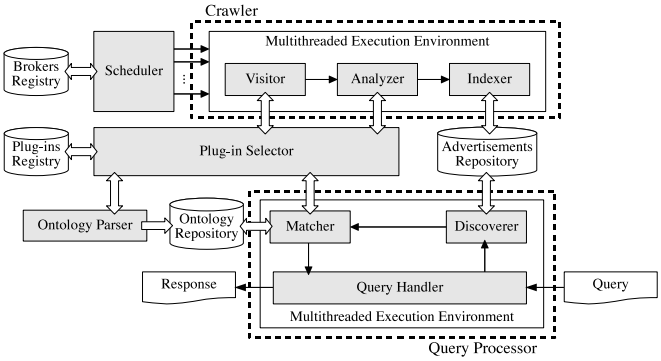

Fig. 1. Architecture of the Proteus engine.

As it is depicted in Figure 1, the engine is strategically split into two main subsystems, namely the *Crawler* and the *Query Processor*, which serve complemental purposes and operate independently of each other. Besides, at the core of the engine lies a powerful extensibility mechanism, which allows for seamless accommodation of additional functionality in the form of *plug-ins*. This mechanism comprises a registry, where meta-information on each contributed plug-in is maintained, and a *Plug-in Selector*, which is commonly used by both subsystems to find and dynamically instantiate pluggable components.

The Crawler subsystem is used by the engine in order to import service descriptions from external sources (service brokers) where services are originally published. This procedure, which can be either manually triggered or be periodically performed for each known source by the engine's *Scheduler*, involves three main steps. First, the Crawler instantiates the *Visitor* component, which makes use of an appropriate, technology-specific plug-in to connect to the external source. The Visitor retrieves the service descriptions published in that source, and forwards them to the *Analyzer*, which is responsible for their further processing. In turn, the Analyzer uses a language-specific plug-in to transform the service descriptions into Proteus service advertisements, before applying text analysis on their contents. The produced term documents along with the service advertisements are

finally passed to the *Indexer* component, which appropriately stores them in the engine's internal repository.

Being completely decoupled from the time-consuming task of service retrieval, the Query Processor subsystem is responsible for the execution of incoming queries, which becomes a time-effective procedure. Each query is handled by an instance of the *Query Handler* running in a dedicated thread. This component employs the *Discoverer* in order to retrieve from the internal repository the corpus of service advertisements, against which the query will be executed. That corpus is then passed along with the query to the *Matcher*, which implements the matchmaking mechanisms supported by the Proteus model, and is thus responsible for the query evaluation process. Depending on the requirements specified in the query, the Matcher may employ one or more plug-ins to accommodate the technology-specific task of data type constraint matching. Also, depending on the way semantics constraints are expressed in the query, the Matcher may need to use the engine's *Ontology Repository*, in order to retrieve the concept subsumption hierarchies that are required by the ontology concept matchmaking mechanism. The Ontology Repository is populated offline by means of the *Ontology Parser* component, which can be extended with plug-ins to support parsing of ontologies written in different languages. This way, the demanding task of ontology parsing and reasoning takes place ahead of the query execution. Nevertheless, the query evaluation results (i.e. the matched service advertisements) are sent back to the Query Handler, which consolidates them into a query response and finally forwards that document to the requester.

5 Experimental Evaluation

We conducted an experimental evaluation of our approach, so as to assess its *efficiency*, in terms of the accomplished recall-precision of the query results, while also investigating its *feasibility*, by measuring the various computational costs. Within our goals was also to verify that, Proteus is capable in operating in a variety of service settings. For the purposes of the evaluation, we implemented a prototype of the Proteus engine in Java™ 6. All measurements were taken by deploying and executing the engine on a 2.4 GHz Intel Core 2 Duo processor, with 2 GB of RAM, running Mac OS X 10.5, whereas a Java Hotspot™ 64-bit Server VM was used by the Java Runtime Environment (JRE).

The experiments were conducted in two separate phases. The first phase involved the preparation of an adequate set of heterogeneous environments by means of the Proteus Crawler, while the second one involved the generation and execution of the queries by means of the Proteus Query Processor. Both phases are presented in detail in the following paragraphs.

5.1 Setup

In order to ensure objectivity to the maximum extent possible, and eliminate any skewness from the results, we decided to conduct our experiments on the basis of pre-existing service retrieval test collections, which have been developed by third parties and are publicly available. Due to the limited number of such public collections for the time being, we managed to establish three heterogeneous service environments, thus being prevented from experimenting with an even wider range of services. Nonetheless, the established settings were deemed adequate for the purposes of our evaluation.

Specifically, we tested Proteus in an environment of Web services, by using the SAWSDL-TC3 service retrieval test collection[3], which comprises 1.080 WSDL 1.1 [2] service descriptions spanning nine application domains. In our experiments, we did not consider the existing semantic annotations in these descriptions, since we wanted to test Proteus in a more pragmatic environment, where the available WSDL descriptions are most usually of limited semantic content. SAWSDL-TC3 also provides a total of 42 queries with pre-configured binary relevance sets, which we used in the experiments.

Proteus was also evaluated in an alternative, semantically enhanced Web services environment. We specifically utilized the OWLS-TC3 service retrieval test collection[4] that comprises 1.007 individual service operations. The latter are described with the use of the OWL-S 1.1 [3] profile and span seven different application domains. The collection also includes 25 domain ontologies written in OWL, the concepts of which are referenced by the various elements of the OWL-S profile documents, and 29 queries with pre-configured binary relevance sets.

The third experimental environment was shaped by means of the hRESTS-TC1 service retrieval test collection[5]. hRESTS-TC1 contains 894 XHTML documents, which describe an equal number of RESTful services with the use of the hRESTS [7] microformat. In all descriptions, the service inputs and outputs are semantically annotated by MicroWSMO extensions, which are populated by concepts taken from 38 domain ontologies in OWL, also provided by the test collection. Similar to the other two environments, hRESTS-TC1 includes 26 queries in XHTML, along with their binary relevance sets. With the use of an XSLT script, we converted the service descriptions and queries into RDF, before using them in our experiments.

In preparation of the three environments, we first employed the Proteus Crawler to generate the experimental corpora of service advertisements. That process involved (i) the proper translation of the heterogeneous service descriptions into Proteus service advertisements, and (ii) the lexical analysis of the results so as to produce the text documents required by the text matchmaker. The first task was performed by means of plug-in translators implementing the algorithms and mapping rules that are fully described in the **Appendix**, while for the second

3. http://semwebcentral.org/projects/sawsdl-tc
4. http://semwebcentral.org/projects/owls-tc/
5. http://www.semwebcentral.org/projects/hrests-tc/

TABLE 2
Corpus generation and ontology pre-processing results.

	WSDL	OWL-S	hRESTS
Corpus Generation			
Number of original service descriptions	1.080	1.007	894
Number of generated service advertisements	1.134	1.007	894
Number of generated text documents	3.016	980	1.411
Total translation time for all service descriptions	141,00 ms	815,00 ms	381,00 ms
Average translation time per service advertisement	0,12 ms	0,81 ms	0,43 ms
Total text analysis time for all service advertisements	271,63 s	533,54 s	124,10 s
Average text analysis time per service advertisement	239,54 ms	529,83 ms	138,82 ms
Computational overhead introduced by translation	0,05%	0,15%	0,31%
Ontology Pre-processing			
Number of parsed ontologies	n/a	25	38
Minimum number of concepts in a single ontology	n/a	1	2
Maximum number of concepts in a single ontology	n/a	2.155	7.596
Average number of concepts per ontology	n/a	184	374
Minimum ontology concepts hierarchy depth	n/a	1	1
Maximum ontology concepts hierarchy depth	n/a	18	18
Average ontology concepts hierarchy depth	n/a	8	7
Total parsing time	n/a	13,60 s	22,11 s
Minimum parsing time for a single ontology	n/a	47,00 ms	5,00 ms
Maximum parsing time for a single ontology	n/a	3.494,00 ms	5.396,00 ms
Average parsing time per ontology	n/a	544,00 ms	582,00 ms

task we used WordNet [35] to run the necessary Natural Language Processing (NLP) steps, as described in [37].

Besides, for the OWL-S and hRESTS environments, the Proteus Ontology Parser was used to pre-process the corresponding ontologies. We specifically implemented an OWL ontology parser plug-in, powered by the HermiT reasoner[6], which we used to extract the ontology concept subsumption hierarchies, and store them in the Proteus Ontology Repository for later use by the Proteus Query Processor.

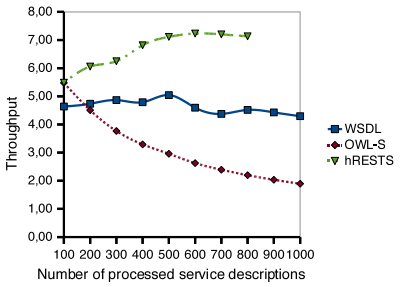

Fig. 2. Crawler throughput measurements.

The results of the aforementioned preparatory activities are summarized in Table 2, while Figure 2 displays the observed throughput of the Proteus Crawler, being calculated as the rate of generated service advertisements per second, after having processed a certain amount of service descriptions. In all cases, the download times were deliberately not measured, since they were independent of Proteus. Hence, in accordance to the tasks involved in the corpus generation process, we distinguished two computational costs, namely the time taken to translate the service descriptions, and the time taken to apply lexical analysis on the textual contents of the produced service advertisements.

6. http://hermit-reasoner.com/

The translation overhead introduced by our approach was trivial in all cases, taking less than 1% of the overall processing time of the Proteus Crawler's Analyzer component. On the other hand, there was a non-trivial latency inflicted by the text analysis step. It should be noted though that, such cost is inevitable for any service discovery approach moving beyond the limits of keywords-based querying to support elaborate, text-based matchmaking.

The measurements also gave evidence that, the pre-processing of the OWL ontologies by the Proteus Ontology Parser is computationally affordable, and scales well even for large ontologies containing more than 7.500 concepts. In conclusion, the verification of the relatively low computational costs induced by the preparatory phase and the fact that, thanks to the architecture of the Proteus engine, both corpus generation and ontology pre-processing are totally decoupled from query execution, attested to the feasibility of our unified approach to heterogeneous services discovery.

5.2 Results

Having prepared the experimental environments, we proceeded with the generation and execution of their respective queries by the Proteus Query Processor. Since all queries were considered equal within each environment, we assessed the efficiency of our matchmaking algorithm by measuring (i) the average query processing time, and (ii) the *macro-averaged recall-precision*. This metric calculates the precision p_i at each one of λ predefined, equidistant recall levels, $1 \leq i \leq \lambda$, as

$$p_i = \frac{1}{|S_Q|} \cdot \sum_{Q \in S_Q} \max\{p : (r \geq r_i) \wedge ((r,p) \in R_Q)\}$$

where S_Q is the set of executed queries, while R_Q is the set of recall-precision value pairs of all relevant documents for query $Q \in S_Q$ in the test collection.

TABLE 3
Details of the Proteus queries used in the experiments.

			Constraints in Requirement					
Environment	**# Queries**	**Requirements in Query**	$c(s)$	$c(v)$	$c(\sigma)$	$c(t)$	$c(P_Q)$	$c(g)$
WSDL	42	Service			✓			
		Capability			✓			
		Input			✓			✓
		Output			✓			✓
		(Input/Output) Parameter			✓	✓		
OWL-S	29	Capability			✓			
		Input						✓
		Output						✓
		(Input/Output) Parameter			✓			
hRESTS	26	Service			✓			
		Capability			✓			
		Input						✓
		Output						✓
		(Input/Output) Parameter			✓			
		HTTPMethod	✓					

Further, we benchmarked our recall-precision measurements against the ones taken by selected third-party tools, which were also deployed and run in the same environments, in order to assess the comparative benefits and potential pitfalls of our unified approach. To this end, we first compared our approach against UDDI [8], which is the predominant standard for WSDL-based Web services discovery. We particularly used jUDDI[7], an open source implementation of UDDI v3, to publish the WSDL documents of the SAWSDL-TC3 collection. Then, from the WSDL queries of the collection, we generated an equal number of keywords-based, find_service UDDI requests, which were subsequently executed in jUDDI.

In the OWL-S environment, we compared Proteus to the OWLS-MX matchmaker[8], driven by the fact that, like Proteus, this tool also adopts a hybrid approach in semantics matching, supporting the evaluation of semantics constraints that are expressed in text and/or by means of ontology concepts. In our benchmark, we used the second version of the OWLS-M4 variant, which achieves the best performance [23]. Finally, in the context of the hRESTS environment, the efficiency of Proteus was benchmarked against the implementation of the XAM4SWS matchmaker[9], which is one of the very few available tools currently supporting the discovery of RESTful services.

The Proteus queries that were used in the experiments were automatically produced by proper translation of the original queries provided by the three service retrieval test collections. Further, they were relaxed by setting $\epsilon = 0.01$ and $\mu = |C|$, where C is the corpus in each environment. As it can be seen in Table 3, the translation process mostly focused on capturing the semantics constraints, while keyword-based, string value constraints were deliberately omitted, with the exception of the HTTPMethod requirement that was used in the queries of the hRESTS environment. Besides, for the queries that were used in the WSDL environment, we retained the data type constraints, and accordingly equipped the Proteus engine with an appropriate plug-in supporting the matching of built-in XML Schema types. Nevertheless, despite the diversity of the service description languages in the three service retrieval test collections, the produced Proteus queries exhibited remarkable homogeneity in terms of the expressed requirements and constraints. This gave more evidence of the genericity of Proteus, and its ability to support service discovery in a service model- and technology-agnostic way, through queries that can be uniformly evaluated against heterogeneous service descriptions.

The benchmarking results are shown in Figure 3. As it was expected, the limited, keywords-based query model of UDDI was easily outperformed by Proteus, for a number of reasons: First of all, Proteus supports the expression and evaluation of semantics constraints, even in the absence of rich, ontology-based semantic annotations in service descriptions. Second, the Proteus query model effectively captures the structure of the Web services model, thereby providing for the specification of requirements not only at the service level, but also at operation- and input/output-level properties. Finally, the fuzziness of the Proteus query evaluation mechanism allows for the relaxation of constraints upon querying, and subsequently yields the inclusion of partial, yet positive matches to the query results. All these features of our approach are currently missing from the UDDI standard.

The comparison with OWLS-MX showed that, Proteus improves the precision of the query results for recall values between 0 and 0.8, while for higher recall levels it is outperformed by the OWLS-M4 matchmaker. This observation can be explained by the different implementations of the degree of match concept by the two approaches. Indeed, in OWLS-MX, the degree of match is shaped according to concrete, predefined semantic levels of similarity (namely *exact*, *plug-in*, *subsumes*, *subsumed-by*, *logic-based fail*, *nearest-neighbor*, *fail*), while in Proteus

7. http://juddi.apache.org/
8. http://www.semwebcentral.org/projects/owls-mx/
9. http://www.semwebcentral.org/projects/xam4sws/

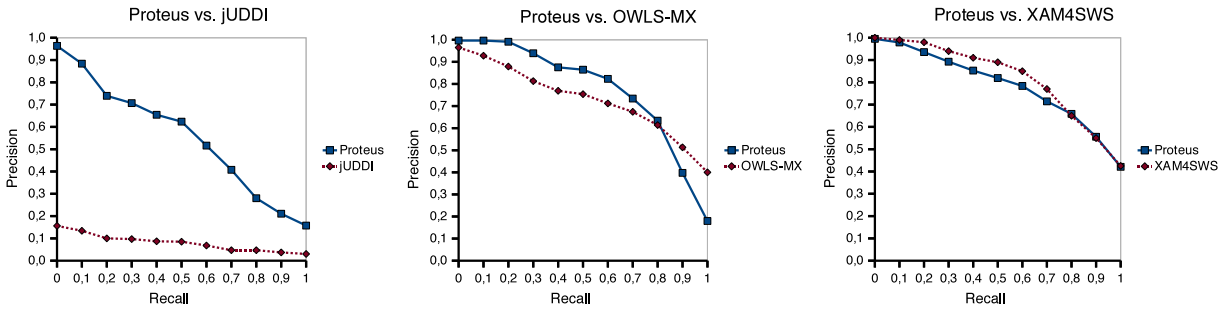

Fig. 3. Macro-averaged recall-precision benchmarks against third-party service discovery tools.

the degree of match is fuzzy and continuous, ranging between 0 and 1. Consequently, the relaxation of the minimum degree of match, ϵ, in Proteus queries unavoidably yields the inclusion of more false positives in the results, the presence of which apparently had an affect on the accomplished precision at high recall levels. However, we believe that this trade-off is acceptable, as it allows Proteus to be applicable to an open set of heterogeneous service-oriented environments, where semantics similarity cannot always be determined and categorized according to predefined levels.

The benchmark with XAM4SWS suggested that, Proteus achieves a similar level of precision with the best performing variant of XAM4SWS, as reported in [25]. The slightly lower precision of Proteus for recall levels 0.2–0.7 is explained by certain differences between the two matchmakers. In contrast to our approach, the benchmarked variant of XAM4SWS requires a pre-query, training phase in order to estimate the semantic degrees of match, which is corpus-specific and thus manages to more accurately quantify the various semantic similarities. Further, the tokenization step upon lexical analysis is recursively applied by XAM4SWS, a technique that we retrospectively realized is needed to more efficiently process the contents of the service descriptions in the hRESTS-TC collection, while in Proteus it is only performed once. Finally, in Proteus, the translation of hRESTS service descriptions into service advertisements retains the, rather limited, semantics information that is conveyed in the RDF resource elements. This content, which yielded a decrease in the calculated degrees of match, is not utilized by the XAM4SWS matchmaker, which solely relies on the MicroWSMO semantic annotations.

Apart from the benchmark experiments, we studied the behavior of Proteus in the presence of multiple requirements within queries. For each one of the three experimental environments, we incrementally populated the respective queries with additional requirements (see Table 4), measuring in each case (i) the achieved precision of the query results, and (ii) the average query matchmaking time. The results of both measurements are plotted in the diagrams of Figures 4 and 5, respectively. It should be noted that, in the diagrams we did not display the cost associated with the query context initialization, which refers to the time consumed for loading in-memory the service advertisements, text documents, and ontology concept subsumption hierarchies. This cost averaged to 3.333 ms in the WSDL environment, 2.665 ms in the OWL-S environment, and 3.649 ms in the hRESTS environment.

TABLE 4
Formulation of queries with different content.

Case	Included Requirements
WSDL Environment	
Case 1	Service
Case 2	Service, Capability
Case 3	Service, Capability, Input, Output
Case 4	Service, Capability, Input, Output, Parameter
OWL-S Environment	
Case 1	Capability
Case 2	Capability, Input, Output, Parameter
hRESTS Environment	
Case 1	Service
Case 2	Service, Capability, HTTPMethod
Case 3	Service, Capability, HTTPMethod, Input, Output, Parameter

In all the investigated environments the results verified the intuitive assumption that, the more requirements are expressed in a Proteus query, the more precise its results are expected to be. It was also shown that, the additional computational cost was insignificant, notably taking less than 100 ms for the evaluation of up to 1.000 service advertisements in the fourth case of the WSDL environment, which was proved to be the most resource-demanding.

6 Discussion

We proposed a generic query model called Proteus to support the unified discovery of operations provided by heterogeneous services. Proteus defines appropriate abstractions that can be used for the description of an open set of services and for setting requirements towards their various properties. The query evaluation mechanism underpinning the proposed model effectively combines the different requirements, while also capturing their potentially different priorities in the produced degree of match.

We validated Proteus in three heterogeneous environments, by assessing and benchmarking the effectiveness

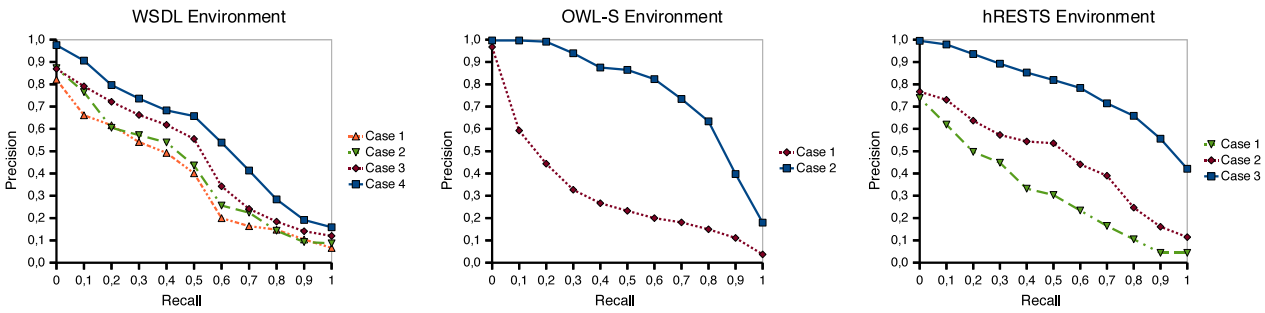

Fig. 4. Macro-averaged recall-precision measurements in the presence of multiple, combined query requirements.

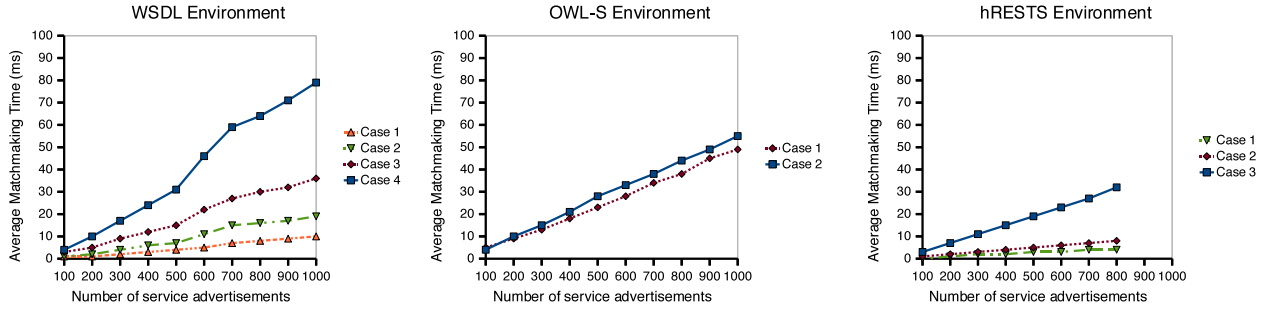

Fig. 5. Average matchmaking time measurements in the presence of multiple, combined query requirements.

of its query evaluation mechanism. The results suggest that, unification in the discovery of heterogeneous services by Proteus is feasible, whereas the inclusion of meaningful requirements in queries leads to enhanced precision. Moreover, it was shown that, Proteus scales well with the number of service advertisements, while the computational overhead induced by the necessary preprocessing of ontologies and the translation of the heterogeneous service descriptions is not restrictive for its application in real-world settings.

In future work, we aim to further investigate the usefulness of supporting more complex queries in service discovery, for instance by including and combining logical operators set on requirement groups. We expect that, such study will also allow us to perform an additional validation of the proposed query model in terms of expressiveness. We would also like to explore the potential integration of Proteus with another line of research that is currently going on in our laboratory, and deals with the data-driven adaptation of service-oriented processes [44]. We are particularly interested in the use of Proteus by existing workflow systems at *run-time*, i.e. upon execution of service compositions that dynamically adapt to data-driven events, so as to enable the dynamic discovery and late-binding of heterogeneous service operations that meet both functional and non-functional, execution context-specific requirements.

REFERENCES

[1] G. Athanasopoulos, A. Tsalgatidou, and M. Pantazoglou, "Interoperability among heterogeneous services," in *Proceedings of the 2006 IEEE International Conference on Services Computing, SCC 2006*. IEEE Computer Society, 2006, pp. 174–181.

[2] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Service Description Language (WSDL) 1.1," http://www.w3.org/TR/wsdl, March 2001, W3C Note, World Wide Web Consortium (W3C).

[3] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIllraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara, "OWL-S: Semantic markup for web services," http://www.w3.org/Submission/OWL-S/, 2004, W3C Member Submission, 22 November 2004.

[4] J. Farrell and H. Lausen, "Semantic Annotations for WSDL and XML Schema," http://www.w3.org/TR/sawsdl/, August 2007.

[5] Open Geospatial Consortium Inc., "Sensor Observation Service 1.0.0."

[6] M. Hadley, "Web Application Description Language," http://www.w3.org/Submission/wadl/, 2009, W3C Member Submission, 31 August 2009.

[7] J. Kopecký, K. Gomadam, and T. Vitvar, "hRESTS: An HTML Microformat for Describing RESTful Web Services," in *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*. IEEE Computer Society, 2008, pp. 619–625.

[8] L. Clement, A. Hately, C. von Riegen, and T. Rogers, "UDDI version 3.0.2," http://uddi.org/pubs/uddi_v3.htm, October 2004, UDDI Spec Technical Committee.

[9] International Organization for Standardization, "Electronic business Extensible Markup Language (ebXML) — Part 3: Registry Information Model Specification (ebRIM)," ISO/TS 15000-3:2004, May 2004.

[10] L. Gong, "JXTA: A network programming environment," *IEEE Internet Computing*, vol. 5, no. 3, pp. 88–95, 2001.

[11] E. Laure, S. Fisher, A. Frohner, C. Grandi, P. Kunszt, A. Krenek, O. Mulmo, F. Pacini, F. Prelz, J. White, M. Barroso, P. Buncic, F. Hemmer, A. D. Meglio, and A. Edlund, "Programming the Grid with gLite." *Computational Methods in Science and Technology*, vol. 12, no. 1, pp. 33–45, 2006.

[12] I. Foster, "Globus toolkit version 4: Software for service-oriented systems," in *IFIP International Conference on Network and Parallel Computing*, ser. Lecture Notes in Computer Science, no. 3779. Springer-Verlag, 2005, pp. 2–13.

[13] A. Tsalgatidou, G. Athanasopoulos, M. Pantazoglou, A. J. Berre,

C. Pautasso, R. Gronmo, and H. Hoff, *Unified discovery and composition of heterogeneous services: The SODIUM approach*, ser. Information Systems. MIT Press, 2008, ch. 4, pp. 67–100.

[14] M. Botts, G. Percivall, C. Reed, and J. Davidson, *OGC Sensor Web Enablement: Overview and High Level Architecture*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, vol. 4540, pp. 175–190.

[15] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," 2002. [Online]. Available: http://www.globus.org/alliance/publications/papers/ogsa.pdf

[16] R. T. Fielding, "REST: architectural styles and the design of network-based software architectures," Doctoral dissertation, University of California, Irvine, 2000. [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[17] S. Vinoski, "Serendipitous reuse," *IEEE Internet Computing*, vol. 12, no. 1, pp. 84–87, 2008.

[18] I. J. Taylor, M. S. Shields, I. Wang, and O. F. Rana, "Triana applications within grid computing and peer to peer environments," *Journal of Grid Computing*, vol. 1, no. 2, pp. 199–217, 2003.

[19] C. Pautasso and G. Alonso, "From web service composition to megaprogramming." in *Proceedings of the 5th VLDB Workshop on Technologies for E-Services*, ser. Lecture Notes in Computer Science, M.-C. Shan, U. Dayal, and M. Hsu, Eds., vol. 3324. Springer, 2004, pp. 39–53.

[20] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the Kepler system," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[21] N. Srinivasan, M. Paolucci, and K. Sycara, "An efficient algorithm for OWL-S based semantic search in UDDI," in *Semantic Web Services and Web Process Composition*, ser. Lecture Notes in Computer Science, vol. 3387. Springer, 2005, pp. 96–110.

[22] D. Elenius and M. Ingmarsson, "Ontology-based service discovery in p2p networks," in *Proceedings of the First International Workshop on Peer-to-Peer Knowledge Management, P2PKM'04*, 2004.

[23] M. Klusch, B. Fries, and K. Sycara, "Automated semantic web service discovery with OWLS-MX," in *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2006*. ACM Press, 2006, pp. 915–922.

[24] M. Klusch and F. Kaufer, "WSMO-MX: A hybrid semantic web service matchmaker," *Web Intelligence and Agent Systems*, vol. 7, no. 1, pp. 23–42, 2009.

[25] U. Lampe, S. Schulte, M. Siebenhaar, D. Schuller, and R. Steinmetz, "Adaptive matchmaking for RESTful services based on hRESTS and MicroWSMO," in *Proceedings of the 5th International Workshop on Enhanced Web Service Technologies*. ACM, 2010, pp. 10–17.

[26] S. Kona, A. Bansal, L. Simon, A. Mallya, G. Gupta, and T. D. Hite, "USDL: A service-semantics description language for automatic service discovery and composition." *International Journal of Web Services Research*, vol. 6, no. 1, pp. 20–48, 2009.

[27] A. Kozlenkov, G. Spanoudakis, A. Zisman, V. Fasoulas, and F. Sanchez, "Architecture-driven service discovery for service-centric systems," *International Journal of Web Services Research*, vol. 4, no. 2, pp. 82–113, 2007.

[28] Q. Yu and A. Bouguettaya, "Framework for web service query algebra and optimization," *ACM Transactions on the Web*, vol. 2, no. 1, pp. 1–35, 2008.

[29] R. Al-Ali, O. Rana, D. Walker, S. Jha, and S. Sohail, "G-QoSM: Grid service discovery using QoS properties," *Computing and Informatics Journal, Special Issue on Grid Computing*, vol. 21, no. 4, pp. 363–382, 2002.

[30] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang, "Similarity search for web services," in *Proceedings of the Thirtieth international conference on Very large data bases*, 2004, pp. 372–383.

[31] D. Skoutas, D. Sacharidis, A. Simitsis, and T. Sellis, "Ranking and Clustering Web Services Using Multicriteria Dominance Relationships," *IEEE Transactions on Services Computing*, vol. 3, no. 3, pp. 163–177, 2010.

[32] B. Jin, L. Zhang, and Z. Zang, "A unified service discovery framework," in *Sixth International Conference on Grid and Cooperative Computing (GCC 2007)*. IEEE Computer Society, 2007, pp. 203–209.

[33] N. Limam, J. Ziembicki, R. Ahmed, Y. Iraqi, D. T. Li, R. Boutaba, and F. Cuervo, "OSDA: Open service discovery architecture for efficient cross-domain service provisioning," *Computer Communications*, vol. 30, no. 3, pp. 546–563, 2007.

[34] D. Roman, U. Keller, H. Lausen, R. L. J. D. Bruijn, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel, "Web Service Modeling Ontology," *Applied Ontology*, vol. 1, no. 1, pp. 77–106, 2005.

[35] C. Fellbaum, Ed., *WordNet: An Electronic Lexical Database*. Cambridge, MA: MIT Press, 1998.

[36] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol. 8, no. 11, pp. 613–620, 1975.

[37] N. Kokash, "A comparison of web service interface similarity measures." in *STAIRS*, ser. Frontiers in Artificial Intelligence and Applications, vol. 142. IOS Press, 2006, pp. 220–231.

[38] T. Roelleke and J. Wang, "TF-IDF uncovered: A study of theories and probabilities," in *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, 2008, pp. 435–442.

[39] V. Cross, "Fuzzy semantic distance measures between ontological concepts," in *Proceedings of the 2004 IEEE Annual Meeting of the Fuzzy Information Processing, NAFIPS '04*, vol. 2, June 2004, pp. 635–640.

[40] C. J. van Rijsbergen, *Information Retrieval*. London: Butterworths, 1979.

[41] J. Madhavan, P. A. Bernstein, and E. Rahm, "Generic Schema Matching with Cupid," in *Proceedings of the 27th International Conference on Very Large Data Bases*, 2001, pp. 49–58.

[42] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, 1962.

[43] H. W. Kuhn, "The Hungarian Method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, 1955.

[44] G. Athanasopoulos and A. Tsalgatidou, "An approach to data-driven adaptable service processes," in *Proceedings of the 5th International Conference on Software and Data Technologies*, 2010.

Michael Pantazoglou is a post-doctoral researcher at the Department of Informatics and Telecommunications of the National and Kapodistrian University of Athens. He received his Diploma (*Ptychio*) and PhD from the same institute in 2005 and 2009, respectively. He is currently a member of the S^3Lab (http://s3lab.di.uoa.gr) laboratory, where he is – or has been– involved in a number of research projects. His current research interests span service-oriented computing, with a focus on service discovery and composition, P2P computing, and semantic web technologies.

Aphrodite Tsalgatidou is a permanent member of the academic staff at the Department of Informatics and Telecommunications of the National and Kapodistrian University of Athens since 1999. She holds a diploma degree in chemistry from the same institution and an MSc and a PhD in computer science from the University of Manchester, UK. She is the director of the S^3Lab group (http://s3lab.di.uoa.gr), which pursues research in service engineering, software engineering and software development. She has participated in and/or managed a large number of projects in these areas, and has published relevant papers. Her current research focuses on service interoperability, service discovery and composition, P2P systems, and business process management.