Preface

This volume contains the papers presented at SMR2 2010: Service Matchmaking and Resource Retrieval in the Semantic Web

The goal of SMR2 is to provide a forum for promoting, presenting, and discussing the latest scientific advances and industry best practices on Semantic Web service matchmaking and resource retrieval. Furthermore, through its association with S3: The Semantic Service Selection Contest, SMR2 acts as a yearly reality check of available tools for semantic (Web) service and resource retrieval, and the experimental evaluation of their performance in practice by means of an open contest. Thus, SMR2 aims at establishing and fostering cross-disciplinary relations between interested parties in research and/or industry, for the purpose of joint work on solutions to relevant problems in the domain. This year, we particularly emphasize the scientific relation of semantic service matchmaking to model-driven service engineering and discovery

October 2010

Abraham Bernstein,
Paul Grace,
Matthias Klusch,
Massimo Paolucci

Workshop Organization

Programme Chairs

Abraham Bernstein, Paul Grace, Matthias Klusch, Massimo Paolucci

Programme Committee

Liliana Cabral, Tommaso Di Noia, Eugenio Di Sciascio, Takahiro Kawamura, Freddy Lecue, Alain Leger, Tiziana Margaria, David Martin, Nils Masuch, Oliver Mueller, Pierluigi Plebani, Axel Polleres, Marco Sbodio, Stefan Schulte, Eran Toch, Roman Vaculin,

Table of Contents

Probabilistic Methods for Service Clustering

Gilbert Cassar, Payam Barnaghi, and Klaus Moessner

Centre for Communication Systems Research
University of Surrey
Guildford, GU2 7XH, UK
{g.cassar, p.barnaghi, k.moessner}@surrey.ac.uk

Abstract. This paper focuses on service clustering and uses service descriptions to construct probabilistic models for service clustering. We discuss how service descriptions can be enriched with machine-interpretable semantics and then we investigate how these service descriptions can be grouped in clusters in order to make discovery, ranking, and recommendation faster and more effective. We propose using Probabilistic Latent Semantic Analysis (PLSA) and Latent Dirichlet Allocation (LDA) (i.e. two machine learning techniques used in Information Retrieval) to learn latent factors from the corpus of service descriptions and group services according to their latent factors. By creating an intermediate layer of latent factors between the services and their descriptions, the dimensionality of the model is reduced and services can be searched and linked together based on probabilistic methods in latent space. The model can cluster any newly added service with a direct calculation without requiring to re-calculate the latent variables or re-train the model.

1 Introduction

The Service Oriented Architecture (SOA) is a model currently used to provide services on the Internet. SOA typically consists of three entities: a service provider, a client, and a broker. A service provider offers the service and advertises it by publishing a service description. A client looks for a service to fulfill a certain task or to consume service data. The service broker accepts queries from the client, refers to service descriptions supplied by the provider and returns Web Service Description Language (WSDL)[1] documents describing the best matches to the query. However the common technologies used to describe, publish, and discover services offer very limited freedom of expressiveness to the service designers. Service description frameworks such as WSDL provide descriptions of what a service is and what it can do; however with these kinds of descriptions machines still cannot easily interprete different concepts described in the service description. If more expressive technologies are used to describe a service, machines will be able to use semantics, reasoning, and linked data[2] to process the service descriptions and generate new knowledge from what is supplied in

[1] http://www.w3.org/TR/wsdl20/
[2] http://linkeddata.org/

the service description; so this knowledge can be refered to by software agents or search clients to offer better service search results and recommendations in different applications.

The service repositories on the Internet are being increasingly over-populated with more services created and published by users. Retrieving services which can match to a user query is becoming a challenging task. By organising the service data into clusters, services become easier and thus faster to be discovered and recommended [11]. Clustering is an approach that transforms a complex problem into a series of simpler sets which are easier to handle. Service Clustering aims to group together those services which are similar to each other. Service Clustering can be very helpful in terms of service recommendation and ranking since services that are similar to the one chosen by the user will be grouped in the close neighbourhood of that service. Methods for Service Composition can also benefit from clustering of services because compatible services can be found more easily if the services are clustered based on their functional attributes.

In this paper we investigate using probabilistic machine-learning methods to extract latent factors $z_f \epsilon Z = \{z_1, z_2, ..., z_k\}$ from semantically enriched service descriptions. By describing the services in terms of latent factors, the dimensionality of the system is reduced considerably. The latent factors can then also be used to efficiently cluster the services in a repository, making the model more scalable and also more efficient in terms of publishing new service descriptions to the repository. Figure 1 shows an example of assigning services to latent variables rather than conceptual words. In this work we elaborate the concept of latent variables and describe how those variables can be calculated for service descriptions.

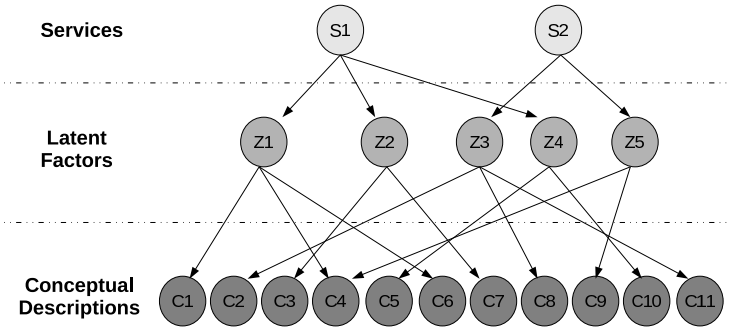

Fig. 1. Describing services in terms of latent factors $z_f \epsilon Z = \{z_1, z_2, ..., z_k\}$ rather than conceptual words

The rest of this paper is organised as follows. Section 2 provides an overview of related work. In Section 3 we propose a clustering mechanism based on Probabilistic Latent Semantic Analysis (PLSA) and Latent Dirichlet Allocation (LDA). Section 4 explains how we evaluate our clusters. Section 5 presents the results obtained in our experiments and discusses the experimental data. Section 6 concludes the paper and discusses the future work.

2 Related Work

In this section we briefly discuss some service description models and then discuss related concepts to service clustering. First we discuss the Onthology Web Language for Services (OWL-S)[3] as it is the fundamental building block of our approach and then we describe the clustering technologies related to our work.

2.1 Service Description Models

A detailed and machine-readable service description is the fundamental building block for providing an architecture in which advanced service discovery mechanisms can be performed [2]. Service descriptions are defined using a service description model. A service description model becomes a template that service providers will use in order to publish and advertise the services that they offer. Various approaches for creating service description models exist, including: WSDL, USDL[4], Web Service Modelling Language (WSML)[5] and Web Service Modelling Ontology (WSMO)[6], and OWL-S.

OWL-S is a Service Description Framework that provides both rich expressive descriptions and well-defined semantics. OWL-S describes the characteristics of a service by using three top-level concepts, namely ServiceProfile, ServiceGrounding, and ServiceModel. ServiceProfile provides the information needed to discover services. ServiceGrounding and ServiceModel provide information to deploy and use the service.

The Service Profile provides a concise but meaningful description of the service capabilities in order to advertise the service in a registry. However, once the service has been selected from the registry, the Profile is no more of use and the information contained in the ServiceModel is then used to interact with the service.

The ServiceGrounding describes how to access the services. In OWL-S, ServiceProfile and ServiceModel are abstract representations of a service and only the ServiceGrounding contains information about protocol and message formats, serialisation, transport, and addressing.

[3] http://www.w3.org/Submission/OWL-S/
[4] http://www.internet-of-services.com/index.php?id=24
[5] http://www.wsmo.org/wsml/
[6] http://www.wsmo.org/

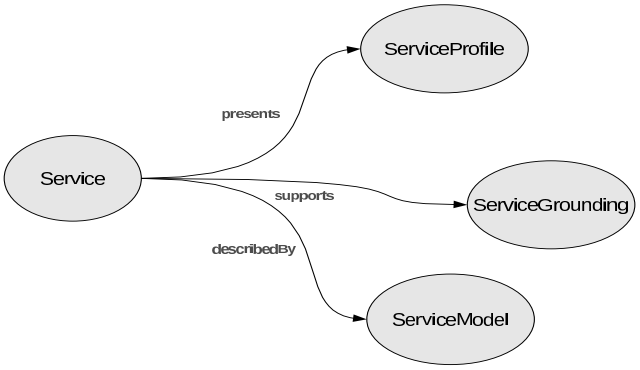

Fig. 2. The structure of OWL-S

The concepts of Input, Output, Preconditions, and Effects are all defined in OWL-S both in the ServiceProfile and in the ServiceModel. OWL-S provides the main attributes to describe services and their functional attributes.

Another effective aspect of OWL-S is that it still relies on existing standards for service invocation and discovery. Service invocations are still carried out using WSDL definitions and OWL-S is designed so that it can extend UDDI for service discovery making it easily integrable with the current SOA.

In this paper, we describe constructing a vector space model using OWL-S descriptions and in Section 3 we discuss how the vector space model is used for training probabilistic models. The probabilistic models are then used to cluster the services.

2.2 Vector Space Modelling and Feature Extraction

Vector Space Modelling (VSM) is a technique used to convert data to vector form in order to facilitate computational analysis of the data. In information retreival, a widely used method for converting a text document to Vector Space form is to use the Text Frequency and Inverse Text Frequency (TF/IDF) algorithm [13]. Ma *et al.* use this technique in [8] to represent a dataset of WSDL service descriptions in the form of a Service Transaction Matrix as shown in Table 1. In Ma *et al.*'s work each row represents a WSDL service description, each column represents a word from the whole text corpus, and each entry represents the TF/IDF weighting of that word in the respective WSDL document.

TF/IDF weight w_{ij} for a word j in service i is calculated as follows:

$$w_{ij} = tf_{ij} \cdot log\left(\frac{n}{n_j}\right) \tag{1}$$

where tf_{ij} is the word frequency of word j in service description i, n is the total number of service descriptions, and n_j is the number of services that con-

tain word j.

A variation of TF/IDF is proposed in [14] where a higher weight is given to the IDF value. The reason behind this approach is to normalise the bias of TF measure in short documents. The frequency of words in very short documents such as service descriptions tends to be incidental. Thus the proposed TF/IDF^2 equation in [14] is described as the following:

$$w_{ij} = tf_{ij} \cdot log\left(\frac{n}{n_j}\right)^2 \tag{2}$$

Wang *et al.* [16] prepare textual data for analysis using Part-of-Speech tagging to identify and remove stop words from the word corpus. The Stanford Log-Linear POS-tagger[7] is used in their work to POS-tag the text and only the nouns, verbs, and adjectives were kept for further analysis. The remaining words are then used to describe each document as a vector of text frequencies.Using vector space model, the proximity between two vectors will then also correspond to similarity of their data characteristics. The vectors which are very similar can be clustered together using clustering techniques.

Table 1. A simplified TF/IDF matrix.

	w1	w2	w3	w4	w5	w6	w7	w8
S1	0	0	1	0.0458	0	0	1	0
S2	0	0	0	0.0458	0.2218	0	0	0.3979
S3	0.0458	0	0.3979	0	1	0	0	0
S4	0	0.6990	0.5229	0	0	1	0	0
S5	0.3010	0	0	1	0.2218	0	0	1

When describing data in vector space, we represent different features of our data as different dimensions in a multi-dimensional form. The extraction of features to vector space is also called *Indexing*. Approaches like TF-IDF which parse the text in every document and then treat every word as a seperate dimension are referred to as Syntactic Indexes [11]. Syntactic Indexes have the advantage of being able to parse any textual document and convert it to a vector format which complies with the rest of the documents; however it results in a large number of dimensions in vector space that becomes very computationally expensive. In the context of service descriptions, it does not have the ability to focus on just some selected characteristics of a service. *Rich Indexes* [11] on the other hand consider only certain aspects of the information contained in the original data.

The service descriptions include different features of services as discussed in the following:

[7] http://nlp.stanford.edu/software/tagger.shtml

– Domain Information: Information about the service registration and other attributes such as domain, cluster group, back-links, and listings.
– Semantic Descriptions: Where service descriptions are enriched with semantic content such as those in WSMO and OWL-S.
– Functional Descriptions: Include interface information such as input and output parameters, preconditions and effects [7].
– QoS Descriptions: Include information about the performance of a service, specifying requirements such as bandwidth, response time and delay.

We propose using multiple indexes based on different features of a service description (i.e. profile, functional descriptions, etc.). This approach enables us to compare services over different domains; however in the context of service clustering Platzer *et al.* [11] recommend not to merge information from two seperate indexing strategies into one vector. In this case the components of the vector are likely to move apart from each other and result in reducing the effectiveness of the clustering algorithm.

Vector Space Modelling is a very useful method for analysing service descriptions. Once service descriptions are represented in vectors, vector algebra [11] and probabilistic methods [8] can be used to measure similarities between services and group them into clusters.

2.3 Proximity Measure

Measuring the proximity between a service and other services in a dataset is the basic step of most clustering techniques. If two vectors are close to each other in vector space, then they have similar service descriptions or functional attributes depending on characteristics used for constructing the model. Various techniques exist to measure the proximity of two vectors. The most commonly used proximity measures are described in the following.

Euclidean Distance The Euclidean Distance of two n-dimensional vectors corresponds to the actual distance between the absolute position of the two points in vector space described by the two vectors. The Euclidean distance can be calculated using the following formula.

$$dis(p,q) = \|p-q\|_2 = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \tag{3}$$

where p and q are the two vectors and $dis(p,q)$ is the Euclidean distance between them.

Jaccard Coefficient The Jaccard coefficient is a similarity measure that skips the components which give no information. Nayak and Lee [10] use the Jaccard coefficient to measure the similarity between two Web services based on the

terms that are present in both service descriptions. The Jaccard coefficient of two vectors p and q is given by the equation 4.

$$J(p,q) = \frac{T_{pq}}{T_p + T_q + T_{pq}} \tag{4}$$

where T_{pq} is the number of common terms used in describing p and q, T_p and T_q are the number of terms used in p only and q only respectively.

Multidimensional Angle Multidimensional Angle is an efficient measure of the proximity of two vectors. It is used in various clustering approaches [11, 8]. This proximity measure applies cosine of the angle between two vectors. It reaches from the origin rather than the distance between the absolute position of the two points in vector space. This method is more efficient because if a dimension is not present in both vectors it will automatically drop out of the equation. Thus it provides dimensional reduction and reduces required computations. The multidimensional angle between vectors p and q can be calculated using equation 5.

$$cos(p,q) = \frac{p \cdot q}{\|p\| \cdot \|q\|} = \frac{\sum_{i=1}^{n} p_i q_i}{\sqrt{\sum_{i=1}^{n} p_i^2} \sum_{i=1}^{n} q_i^2} \tag{5}$$

where n is the number of dimensions.

Weighted Similarity Measures This method is used by Nayak and Lee in [10] where the similarity of two services is based on different parts of their service descriptions. Each service description is enhanced with semantic components: *OWL-S Profile, Model,* and *Grounding*, and a *WSDL* document. The *Weighted Similarity* of two services is then calculated by summing together the *Jaccard coefficient* of each component where every Jaccard coefficient of the summation is multiplied by a weight which reflects how significant that component is. This proximity measure is given by:

$$\begin{aligned} Sim(p,q) = w_1 \cdot J_{Des}(p,q) + w_2 \cdot J_{SerP}(p,q) + w_3 \cdot J_{WSDL}(p,q) \\ + w_4 \cdot J_{PModel}(p,q) + w_5 \cdot J_{Ground}(p,q) \end{aligned} \tag{6}$$

where w_1 to w_5 are the assigned weights and $J_{Des}(p,q)$, $J_{SerP}(p,q)$, $J_{WSDL}(p,q)$, $J_{PModel}(p,q)$, and $J_{Ground}(p,q)$ are the *Jaccard coeffiecient* of the service description, service profile, *WSDL*, service model, and service grounding respectively.

2.4 Clustering Algorithms

Clustering algorithms generally deal with data described in vector space by using some form of vector algebra to measure the similarity between vectors and grouping together the vectors which are most similar to each other. The following describes some of the common approaches for clustering.

K-Means Algorithm The K-Means algorithm is a well know clustering algorithm based on *Squared Error* criterion [12]. Squared Error algorithms keep converging until a convergence criterion is reached. The steps for implementing the K-Means algorithm as given in [6] are:

1. Randomly generate K cluster centres within the vector space used.
2. Compute the proximity of each vector to each cluster centre and assign each vector to the nearest cluster centre.
3. Recompute the cluster centres by taking the mean of the member vectors in each cluster.
4. If the convergence criterion is not met, go back to step 2.

A typical convergence criterion would be a treshold value of a squared error equation [12] or converging until there is no or minimal change in cluster centres after each iteration [6].

Agglomerative Algorithm The Agglomerative algorithm is a bottom-up hierarchical clustering method. The algorithm starts by assigning each vector to its own cluster; then it starts merging together the most similar clusters at every iteration until a stopping criterion is met [6]. The steps for implementing the Agglomerative algorithm are:

1. Treat each vector as a cluster.
2. Compute a matrix with the proximity of each cluster to every other cluster.
3. Find the most two similar cluster in the matrix and merge these two clusters into one cluster.
4. Update the proximity matrix with the mean of the two merged clusters as the centre of the new cluster.
5. Stop if the proximity treshold is reached or if all the vectors are converged into one cluster. Otherwise go back to step 2.

A Web Service clustering approach based on this algorithm is proposed in [11]. This work uses a repository of 275 WSDL service descriptions. Each WSDL document is treated as a text document and the whole text is converted into vector space then clustered using the Agglomerative algorithm. The performance evaluation provided is based on data extrapolation to evaluate scalability of the method in larger repositories. The algorithm discusses efficiency of matching a client's query compared to a simple key word matching. However, the WSDL documents include serveral repetitive phrases and also specific terminology to describe technical aspects of services. Analysing WSDL as a text document and constructing the model based on the whole concept set extracted from the WSDL description could suffer from biasing and overfitting the results.

2.5 Probabilistic Latent Semantic Analysis (PLSA)

Probabilistic Latent Semantic Analysis is an unsupervised machine-learning technique used to map high-dimensional count vectors (such as the ones yielded by TF/IDF in vector space model) to a lower dimensional representation in *Latent Semantic Space* [4]. PLSA is based on the Aspect Model; a latent variable model which associates an unobserved class variable $z_f \epsilon Z = \{z_1, z_2, ..., z_k\}$ with each observation [5].

PLSA discovers the semantics behind the words in a text corpus, i.e. the topics which the words in the document belong to. Words are observable variables $w_j \epsilon W = \{w_1, w_2, ..., w_v\}$ which can be observed through the text corpus that consists of documents, topics on the other hand are latent variable which are not directly observable through the text.

In service descriptions, we use words taken from service descriptions and create a PLSA model. Once the latent variables $z_f \epsilon Z = \{z_1, z_2, ..., z_k\}$ are identified, services can be described as a multinomial probability distribution $P(z_f|d_i)$ where d_i is the service description (or document) describing service i. The representation of a service with these latent variables reflects the likelihood that the service belongs to certain concept groups [8].

To construct a PLSA model we first require to look at the joint proability of an observed pair. The joint probability of an observed pair $P(d_i, w_j)$ is given by:

$$P(d_i, w_j) = P(d_i) P(w_j|d_i) \tag{7}$$

where:

$$P(w_j|d_i) = P(d_i) P(w_j|d_i) \sum_{f=1}^{k} P(z_f|d_i) P(w_j|z_f) \tag{8}$$

assuming that a document and a word are conditionally independent given the latent factor.

This model indirectly associates the words $w_j \epsilon W = \{w_1, w_2, ..., w_v\}$ to their corresponding documents by introducing an intermediate layer of latent factors $z_f \epsilon Z = \{z_1, z_2, ..., z_k\}$. Thus we are getting dimensionality reduction by mapping a high word-document matrix $P(d, w)$ into a lower k-dimension latent semantic space [8]. By substituting equation 8 in equation 7, we obtain:

$$P(d, w) = \sum_{f=1}^{k} P(z_f) P(d|z_f) P(w|z_f) \tag{9}$$

The parameters $P(z)$, $P(d|z)$, and $P(w|z)$ can be found using a model fitting technique such as the Expectation Maximization (EM) algorithm as described in [4]. Once the algorithm is trained and the parameters are found, any new document can be folded into the model [8] using:

$$P(z_f|d_{new}) = \frac{P(d_{new}|z_f) \cdot P(z_f)}{\sum_{j=1}^{k} P(d_{new}|z_j)} \tag{10}$$

In our work, service descriptions are treated as documents and these textual and functional descriptions are used to construct the PLSA model. The details of the approach is discussed in Section 3.

2.6 Latent Dirichlet Allocation (LDA)

Latent Dirichlet Allocation is another machine-learning technique which uses a generative probabilistic model for collections of discrete data [1]. LDA introduces a Dirichlet prior on the document-topic distribution in order to simplify the problem of statistical inference [16]. The principal of LDA is the same as that of PLSA: mapping high-dimensional count vectors to a lower dimensional representation in latent semantic space.

Using the same notation described in PLSA, with LDA the probability of the ith word occuring in a given document is:

$$P(w_i) = \sum_{f=1}^{k} P(w_i|z_i = f) P(z_i = f) \tag{11}$$

where z_i is a latent factor (or topic) from which the ith word was drawn, $P(z_i = f)$ is the probability of topic f being the topic from which w_i was drawn, and $P(w_i|z_i = f)$ is the probability of having word w_i given the fth topic.

The generative model of LDA is obtained by letting:

$$\Phi^{(j)} = P(w|z = f) \tag{12}$$

and

$$\Theta^{(d)} = P(z) \tag{13}$$

Instead of estimating $P(d|z)$ and $P(w|z)$ as in PLSA, the LDA generative model estimates Φ, Θ, and z. Different methods can be used to train the algorithm and estimate these parameters Blei *et al.* [1] use variational inference with EM algorithm. Wang *et al.* [16] use a method based on Gibbs Sampling which was proposed in [3] and [15].

3 A Probabilistic Approach for Service Clustering

Classical service clustering algorithms use proximity measures to calculate the similarity between services and group similar services together. Our approach applies probabilistic machine-learning techniques to extract latent-factors from service descriptions and then uses the latent-factors to group the services into clusters. This approach gives us a number of advantages over classical clustering

algorithms. First, the dimensionality of the model is reduced as every service can be described in terms of a small number of latent factors rather than a large number of concepts. Consequently, searching for a service inside a cluster can be performed by searching for matching latent factors rather than matching the text describing the service to a set of key words extracted from the user query. Second, the algorithm is scalable and can be applied to large repositories because only a small portion of the data set is required to train the algorithm. The rest of the service descriptions and any other new service published to the repository can be folded-in and assigned to a cluster very easily without high computational requirements.

We use OWL-S as the basis of our model and all the service descriptions published to the repository are in OWL-S format. OWL-S provides rich machine-readable semantics which make it easier to define the different components of a service that facilitate the feature extraction stage for vector space modelling. The rich semantics of OWL-S make it also possible to implement logic based techniques for service matching and ranking; however the use of such techniques is beyond the scope of the current paper. In the current work, we focus on the textual descriptions and the functional descriptions contained in an OWL-S service representations. The two categories of features (i.e. service profile and service model) are extracted separately and stored in separate vector space models.

The OWL-S service profile contains a textual description of service operations. These textual descriptions together with the title of the service are extracted and used to build the vector space model. All punctuation marks are removed from the text and words merged together with a capital letter in between such as "TaxCalculator" are separated back to two different words. For example, "TaxCalculator" becomes "'tax calculator". The Stanford POS Tagger is then used to eliminate all the stop-words and only words tagged as nouns, verbs, and adjectives are retained. Once the text is processed, the vector space model describing each service as a vector of word frequencies is calculated using the TF/IDF give in equation 1.

The funtional descriptions in OWL-S consist of the properties *hasInput*, *hasOutput*, *hasParameter*, *hasPrecondition*, and *hasResult* found in the service model. The service model points to a process which in turn defines the WSDL message-map types needed to interact with the service. All these features are extracted from the OWL-S descriptions using a reasoner. In order to make the features fit in the vector space model, the property name and the type are appended together to produce a new term that is used as a 'word' in constructing the vector space model. For example "hasInput _Book" becomes "hasInput_Book". This way, a service in which _Book is defined as input will be distinguished from a service in which _Book is defined as output. TF/IDF is then used to analyse constructed 'words' to create the vector space model for functional attributes.

PLSA is implemented using the PennAspect[8] model which uses maximum likelihood to compute the three parameters: $P(w|z)$, $P(d|z)$, and $P(z)$. Half of the the dataset is used to train the algorithm and the other half is used for val-

[8] http://www.cis.upenn.edu/ ungar/Datamining/software_dist/PennAspect/index.html

idation in order to prevent overfitting [16]. Once the parameters are calculated, new services published to the repository can be folded-in as proposed in [8] by using the following.

$$P\left(z_f|d_{new}\right) = \frac{P\left(d_{new}|z_f\right) \cdot P\left(z\right)}{\sum_{j=1}^{k} P\left(d_{new}|z_j\right)} = \frac{P\left(d_{new}|w\right) \cdot P\left(w|z_f\right) \cdot P\left(z\right)}{\sum_{j=1}^{k} P\left(d_{new}|z_j\right)} \tag{14}$$

This equation can be computed for each latent factor $z_f \epsilon Z = \{z_1, z_2, ..., z_k\}$ with respect to every new service d_{new} thus obtaining the probability of d_{new} being described by each of the latent variables $P\left(z|d_{new}\right)$. The service d_{new} can be assigned to latent factor z_f having the highest probability given a service d_{new} [8] thus efficiently clustering each service.

We have implemented the LDA model using LingPipe[9] toolkit for processing text. This toolkit uses Gibbs sampling to train the algorithm and to obtain the parameter $P\left(d|z\right)$. After training the algorithm, new services can be folded in by using Gibbs sampling with fixed topic-word probabilities and sampling the assignments of words to topics in the new service [16].

4 Evaluation of Clusters

The Purity of clusters is used as a measure of evaluating the accuracy of a clustering technique [9], [8]. If the pool of services used to evaluate the algorithm were originally organised in a set of classes $c = \{c_1, c_2, ..., i_m\}$, then for clusters generated by the algorithm $z = \{z_1, z_2, ..., z_k\}$ the purity of a clustering algorithm can be computed as:

$$Purity = \frac{1}{n} \sum_{f=1}^{k} max_c \left\{n_f^c\right\} \tag{15}$$

where n is the total number of services and n_f^c is the number of services in cluster z_f belonging to class c while c varies from 1 to m.

It is easy to obtain a high value of cluster purity if the data set is clustered into a large number of clusters. If each cluster is very small, the likelihood of having a high percentage of the cluster belonging to one known class could be very high (especially in very short documents like service descriptions). Therefore in order for the purity measurement to give a more accurate result, the number of clusters generated should not be very big compared to the size of the dataset.

5 Results

In our experiment, we compared the accuracy of two probabilistic clustering algorithms (PLSA and LDA) to that of two proximity measure based algorithms.

[9] http://alias-i.com/lingpipe/

The dataset of service descriptions used in this experiment was obtained from the OWL-S service retrieval test collection, OWLS-TC[10]. This dataset consists of 1007 service descriptions defined in OWL-S form. Each service belongs to one out of seven service categories named as: communication, economy, education, food, medical, travel, and military. These categories are used as the base classes to evaluate Purity of clusters after the clustering algorithms are tested.

Two experiments were conducted: one using only the OWL-S attributes containing textual descriptions of the services, and another using only the OWL-S attributes describing functional attributes. Agglomerative and a K-Means clustering algorithm were also used to compare the performance of PLSA and LDA to proximity based clustering. The dataset was clustered with each algorithm starting with five clusters and increasing in steps of five up to fifty clusters. The Purity of clusters was computed for each algorithm at every step and the results were compared. The purity of the individual clusters for clustering the data set in seven clusters (same as the number of known classes) are shown in Table 2 and Table 3.

The results show that overall purity varies against number of clusters while using textual descriptions from service profile. The results of clustering based on profile descriptions are shown in Figure 3. The results for clustering based on functional attributes are also shown in Figure 4. The Agglomerative algorithm could not be used to cluster functional attributes because the dimensions of the vectors describing functional attributes are smaller than those describing textual data and consequently the algorithm could not converge to less than 65 clusters.

Table 2. Purity of clusters for 7 clusters based on Profile Descriptions.

	Agglomerative	K-Means	PLSA	LDA
Cluster 1	1.0000	0.8020	1.0000	0.8978
Cluster 2	1.0000	0.6585	0.5862	0.7574
Cluster 3	1.0000	0.6111	0.5556	0.6203
Cluster 4	0.7119	0.4324	0.5000	0.5971
Cluster 5	0.3974	0.3591	0.5000	0.5789
Cluster 6	0.3939	0.3509	0.3669	0.5244
Cluster 7	0.3600	0.3220	0.3385	0.4000

5.1 Discussion

The results show that LDA performs significantly better than PLSA. LDA also takes relatively less time than all the other algorithms to train and create the clusters. The Agglomerative and K-Means algorithm both perform better than PLSA. The K-means algorithm depends on the random factor of where the initial cluster centroids are generated and does not always converge in an optimal

[10] http://www.semwebcentral.org/projects/owls-tc/

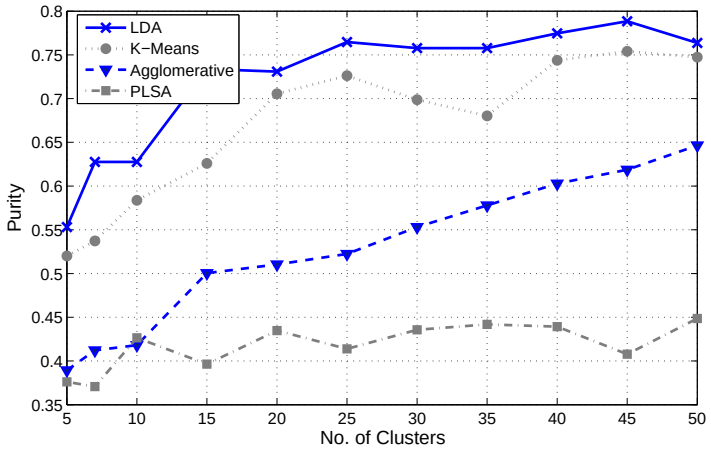

Fig. 3. Purity of clusters for clustering based on Profile Descriptions of the Services.

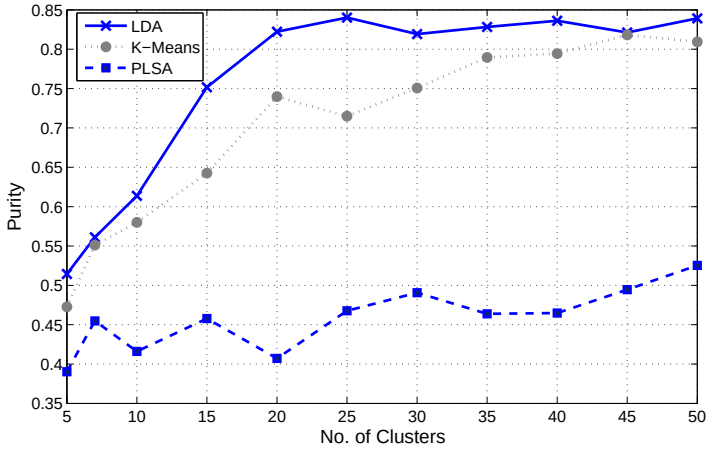

Fig. 4. Purity of clusters for clustering based on the Functional Attributes of the Services.

Table 3. Purity of clusters for 7 clusters based on Functional Attributes.

	K-Means	PLSA	LDA
Cluster 1	0.8883	1.0000	0.8397
Cluster 2	0.6299	0.6243	0.6552
Cluster 3	0.6051	0.5000	0.5677
Cluster 4	0.5455	0.4441	0.5603
Cluster 5	0.4321	0.4000	0.4804
Cluster 6	0.4108	0.3333	0.4237
Cluster 7	0.2976	0.2488	0.3624

way. K-means algorithm is also very slow and computationally expensive; this makes it unsuitable for large repositories. As it can be seen from Figure3 and Figure 4, LDA performs better than the other algorithms. This makes it an ideal solution for clustering services in large repositories. The low purity results for PLSA are due to limited number of concepts used for training the model. Service description are similar to short documents in this context. Thus PLSA is not able to converge to a high accuracy using these limited concepts. Extracting latent factors from a corpus of service descriptions also gives us the basis to construct more efficient service discovery and ranking mechanisms.

An important aspect of this work, also based on OWL-S design, are the semantics. Although in the current work we treat all the descriptions and attributes as text, semantic relations can be exploited to enhance the discovery and recommendation result based on the constructed clusters. Some of these aspects are described in future work in Section 6.

6 Conclusions and Future Work

This paper proposes a probabilistic method to create two seperate clustering schemes; one based on profile descriptions of the services and the other based on the functional attributes. This enables to search services based on classic text queries and/or using more specific functional queries. The latter can be very useful for service personalisation and service composition where the functional attributes of the services are of great importance.

Future work will focus on developing a query mechanism based on latent factors rather than matching key words in a query to the service descriptions' text. This kind of service clustering also makes it easier to recommend services since similar services are grouped in the same cluster. It is important to note that although this work was focused on OWL-S service descriptions, our approach can be applied to other service description models such as WSMO and this will also be investigated in future work. The probabilistic methods used in this paper (i.e. LDA and PLSA) can be trained using a small percentage of the whole dataset, the rest of the service descriptions can be folded into the model as described in Section 3. This makes both algorithms very scalable to large service repositories. The semantics of service descriptions will be also used for further enhancement

of clustering results. The reasoning mechanisms will be also incorporated in the recommendation and discovery methods to provide more relevant results to service consumers.

Acknowledgment

This paper describes work undertaken in the context of the m:Ciudad project, m:Ciudad - A metropolis of ubiquitous services (http://www.mciudad-fp7.org/). m:Ciudad is a medium-scale focused research project supported by the European 7th Framework Programme, contract number: 215007. The authors would like to thank Dr. Wei Wang from the University of Nottingham Malaysia Campus for his comments and sugesstions regarding probablistic models.

References

1. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. J. Mach. Learn. Res. 3, 993–1022 (2003)
2. Charlton, P., Ribiere, M.: Rich service description for a smarter lifestyle. In: AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems. pp. 512–519. ACM, New York, NY, USA (2003)
3. Griffiths, T.L., Steyvers, M.: Finding scientific topics. Proceedings of the National Academy of Sciences 101(Suppl. 1), 5228–5235 (April 2004)
4. Hofmann, T.: Probabilistic latent semantic analysis. In: Proc. of Uncertainty in Artificial Intelligence, UAI99. pp. 289–296 (1999)
5. Hofmann, T., Puzicha, J., Jordan, M.I.: Learning from dyadic data. In: Proceedings of the 1998 conference on Advances in neural information processing systems II. pp. 466–472. MIT Press, Cambridge, MA, USA (1999)
6. Jain, A.K., Murty, M.N., Flynn, P.J.: Data clustering: a review. ACM Comput. Surv. 31(3), 264–323 (1999)
7. Liu, C., Peng, Y., Chen, J.: Web services description ontology-based service discovery model. In: WI '06: Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence. pp. 633–636. IEEE Computer Society, Washington, DC, USA (2006)
8. Ma, J., Zhang, Y., He, J.: Efficiently finding web services using a clustering semantic approach. In: CSSSIA '08: Proceedings of the 2008 international workshop on Context enabled source and service selection, integration and adaptation. pp. 1–8. ACM, New York, NY, USA (2008)
9. Mandhani, B., Joshi, S., Kummamuru, K.: A matrix density based algorithm to hierarchically co-cluster documents and words. In: WWW '03: Proceedings of the 12th international conference on World Wide Web. pp. 511–518. ACM, New York, NY, USA (2003)
10. Nayak, R., Lee, B.: Web service discovery with additional semantics and clustering. In: WI '07: Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence. pp. 555–558. IEEE Computer Society, Washington, DC, USA (2007)
11. Platzer, C., Rosenberg, F., Dustdar, S.: Web service clustering using multidimensional angles as proximity measures. ACM Trans. Internet Technol. 9(3), 1–26 (2009)

12. Rui, X., Wunsch, D.: Survey of clustering algorithms. IEEE Transactions on Neural Networks 16(3), 645–678 (2005)
13. Salton, G.: Automatic text processing: the transformation, analysis, and retrieval of information by computer. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1989)
14. Segev, A., Toch, E.: Context-based matching and ranking of web services for composition. IEEE Transactions on Services Computing 99(PrePrints), 210–222 (2009)
15. Steyvers, M., Griffiths, T.: Latent Semantic Analysis: A Road to Meaning, chap. Probabilistic topic models. Laurence Erlbaum (2007)
16. Wei, W., Barnaghi, P., Bargiela, A.: Probabilistic topic models for learning terminological ontologies. IEEE Transactions on Knowledge and Data Engineering 22, 1028–1040 (2010)

Personalization of Semantic Web Services*

Freddy Lécué

The University of Manchester
Booth Street East, Manchester, UK
{(firstname.lastname)@manchester.ac.uk}

Abstract. Nowadays web users have clearly expressed their wishes to receive and interact with personalized services directly. However, existing approaches, largely syntactic content-based, fail to provide robust, accurate and useful personalized services to its users. Towards such an issue, the semantic web provides enabling technologies to annotate and match services' descriptions with a users' features, interests and preferences, thus allowing for more efficient access to services and then information. The aim of our work, part of service personalization, is on automated instantiation of services which is crucial for advanced usability i.e., how to prepare and present services ready to be executed while limiting useless interactions with users? To this end, we exploit Description Logics reasoning through semantic matching to i) identify useful parts of a user profile that satisfy services requirements (i.e., input parameters) and ii) compute the description required by a service to be executed but not provided by the profile. Finally, the scalability of our approach has been evaluated through its integration in the service consumption of the EC-funded project SOA4All.

Key words: Semantic web, web service, Personalization, Automated reasoning.

1 Introduction

Personalization in web-based applications [1], as a global tendency nowadays, aims at alleviating the burden of information overload by tailoring the information presented based on an individual and immediate user's needs. Between the numerous examples that can be found all across the web, we can highlight the proliferation of personalized home sites, such as iGoogle (http://www.google.com/ig) or netvibes (http://www.netvibes.com), but also the fact that many other web applications of different kinds treat user configuration as one of their most prominent characteristics. From collaborative [2] and content [3] to hybrid-based [4], various personalization techniques have been introduced, depending on the data they manipulate and personalization levels they achieve. In most of these approaches, the user profile, as a collection of data modelling the user extended with its interests, its preferences and context, is a prominent element to ensure accurate and efficient personalized access to information.

In recent years, web service [5], as an emergent technology to consume information on the web, has benefited from research progress in web personalization. Indeed, many approaches addressing user-centric and preference-based consumption of services and

* Foundation Project: Supported by European Commission VII Framework IP Project Soa4All.

more specially their publication [6], discovery [7], selection [8] and execution [9] have emerged. The possibility to customize their results [10] even goes further by giving the users the chance to experience those services in a personalized fashion, which is prominent in order to permit the users to fulfil their desires more suitably.

However most of these approaches ensure personalization by collecting and analizing syntactic content of user profile and services description e.g., [11]. This under specification limits the accuracy of personalization and their automation. Towards this issue, the *semantic web* [12], where the semantic content of the information is tagged using machine-processable languages, provides many advantages over the current "formatting only" version of the web, its services and users. OWL [13], as one of its *Web Ontology Language* which is based on Description Logics (DLs) [14], aims at modelling knowledge on the web through ontologies i.e., formal conceptualization of a particular domain. Therefore, services with their functionalities (i.e., input and output parameters, preconditions, effects and invariants) and user profile with their interests, preferences can be both annotated and then enhanced using semantic descriptions. Such annotations are one important features to enable reasoning on services and user profile descriptions, hence automation of personalized tasks such as the consumption of services.

In this work, we address automated instantiation of services, part of personalization in service consumption, which is crucial for advanced usability. Service instantiation, which is between in selection and execution, aims at preparing and presenting preselected services ready to be executed while limiting useless interactions with users. To this end, execution-time constraints attached to services descriptions are required to be satisfy before their execution. Most of existing approaches [10] undervalue this issue by rarely considering suitable and efficient methods for such a personalization level. Contrary to the latter that consider several levels of interaction to manually collect information (from the users) required by input parameters of services to be executed, we consider automation of this process through semantic instantiation of services.

By addressing the latter, we thus aim at i) improving and easing the user interaction with services, beneficial for both parties and ii) better supporting the user by anticipating her needs. To reach the goal of automated and personalized consumption of services and more specially to suggest accurate and personalized presentations of services to users, we benefit from the semantic augmentation of service and user profile descriptions. In this direction, we define a framework, where potential matching between both descriptions is defined as a reasoning task to be solved for service instantiation (in the rest of the paper we refer to it as service personalization and adaptation). The semantic matching, core of our approach, exploits standard DL reasoning [15, 16] and abduction [17] to i) adapt services to the user by identifying useful parts of its profile that satisfy the service requirements (i.e., input parameters) and ii) compute the descriptions required by a service to be executed but not provided by the user profile.

The remainder of this paper is organised as follows. Section 2 briefly reviews i) DL reasoning and abduction, both required for automated personalization, ii) semantic web services and iii) semantic user profiles. Section 3 presents our approach to personalize semantic web services. Section 4 presents details about the prototype implementation and reports some experiment results. Section 5 briefly comments on related work. Finally Section 6 draws some conclusions and talks about possible future directions.

2 Background

In this section we review i) DL as a semantic language and abduction reasoning to perform personalization. Then we remain the definitions of ii) semantic web service and iii) semantic user profile, as core elements in our approach.

2.1 Description Logic and Abduction Reasoning

The model we considered to represent semantics of services and user profiles is provided by an ontology. In more detail, we focused on DL as a formal knowledge representation language to define ontologies since the latter offers good reasoning support for most of its expressive families and compatibility to current W3C standards e.g., OWL. Terminological Box $\mathcal{T}$ (or TBox i.e., intentional knowledge) and Assertional Box $\mathcal{A}$ (or ABox i.e., extensional knowledge) are core elements to represent knowledge in DL systems. In the following, we will focus on the TBox $\mathcal{T}$ (see Fig. 1 as an example) that supports different level of inference by means of DL reasoning. As a trade-off between expressivity and complexity, we use the expressiveness of the DL $\mathcal{ALC}$ [14] to perform service personalization, which is the standard DL $\mathcal{AL}$ (Attributive Language) extended with full existential qualification $\mathcal{E}$ and concept union $\mathcal{U}$.

$$
\begin{array}{l}
British \equiv Person \sqcap \exists hasSpokenLanguage.English \\
Name \equiv \exists hasFN.FirstName \sqcap \exists hasLN.LastName \\
BusinessAccount \equiv Account \sqcap \exists hasID.OpenID \sqcap \\
\qquad\qquad \exists hasSocialNetwork.SocialNetworkAccount \\
PersonalAccount \equiv Account \sqcap \exists hasID.ElectronicID \sqcap \\
\qquad\qquad \exists hasSocialNetwork.LinkedIn \\
SkypeAccount \sqsubseteq Account \qquad APIKey \sqsubseteq \top,\ BankID \sqsubseteq \top,\ Person \sqsubseteq \top \\
LinkedIn \sqsubseteq SocialNetworkAccount \qquad Latitude \sqsubseteq \top,\ Longitude \sqsubseteq \top,\ ID \sqsubseteq \top \\
SocialNetworkAccount \sqsubseteq Account \qquad BankAccount \sqsubseteq Account,\ English \sqsubseteq \top \\
OpenID \sqsubseteq ElectronicID \sqsubseteq Account \qquad GMail \sqsubseteq MaillingAddress \sqsubseteq OpenID \\
FirstName \sqsubseteq \top,\ LastName \sqsubseteq \top,\ British \sqsubseteq \top
\end{array}
$$

Fig. 1. Sample of an $\mathcal{ALC}$ Terminological Box $\mathcal{T}$.

Besides standard DL reasoning approaches such as satisfiability or subsumption to guarantee consistency of DL knowledge bases or build concepts hierarchy, authors of [17] suggest computing the *Abduction* (Definition 1) between concepts C and D, representing what is underspecified in D in order to completely satisfy C taking into account the information modelled in a $\mathcal{ALN}$ (so compliant with $\mathcal{ALC}$) TBox $\mathcal{T}$.

Definition 1 ***(Concept Abduction Problem)***
Let $\mathcal{L}$ be a DL, C, D be two concepts in $\mathcal{L}$, $\mathcal{T}$ be a set of axioms in $\mathcal{L}$ and $\mathcal{A}$ be a set of assertions. A Concept Abduction Problem (CAP), denoted as $\langle \mathcal{L}, D, C, \mathcal{T} \rangle$ (or shortly $C \backslash D$) consists in finding a concept $B \in \mathcal{L}$ such that $\mathcal{T} \models D \sqcap B \sqsubseteq C$.

Example 1 ***(Concept Abduction)***
Let C and D be two $\mathcal{ALC}$ descriptions respectively defined by $BusinessAccount \sqcap$

$\exists hasSkype.SkypeAccount$ *and* $PersonalAccount$ *(Fig.1). According to Definition 1, the description* B *required by* D *to satisfy (or more precisely* to be subsumed by*)* C *is denoted by* $C \backslash D$ *i.e.,*

$$C \backslash D \doteq \exists hasID.OpenID \sqcap \exists hasSkype.SkypeAccount \tag{1}$$

In other words, D *needs an OpenID and a SkypeAccount to satisfy* C*.*

2.2 Semantic Web Services

Semantics of web services can be expressed by means of different descriptions, from their process levels [18] (i.e., internal and complex behaviours) and causal levels [19] (i.e., preconditions and effects on the world) to their functional levels (i.e., simple interface). In this work we will focus on the latter and more generally their functional input and output parameters, which are prominent to personalize and execute any service. In the semantic web, these functional parameters are enhanced with DL concepts that determine the semantics of the operations they achieve. Therefore, semantic web services can be expressed as DL concepts in (2).

$$Service \doteq \exists requires.Input \sqcap \exists returns.Output \tag{2}$$

This definition confines a semantic web service to being anything that $requires$ input parameters $Input$ to be processed and $returns$ some output parameters $Output$. Both latter parameters are defined in $\mathcal{T}$ such that $\mathcal{T} \models Input \sqsubseteq \top$ and $\mathcal{T} \models Output \sqsubseteq \top$. According to this model, the OWL-S profile [20], WSMO capability [21] or SAWSDL [22] can be used to describe the functional level of semantic web services.

Example 2 ***(Semantic Web Service)***
Suppose a semantic web service S_1 *locating friends and professional colleagues of a specific person, as its main functionality. This service, starting from a* `FirstName`*,* `LastName`*,* `BusinessAccount`*,* `SkypeAccount` *and a* `GMail` *address of this person, returns the list of her nearby* `ContactPersons` *according to her* `Location`*. According to* (2)*, the semantic description of* S_1 *is defined by* (3)*.*

$$S_1 \doteq \exists requires.C^1_{S_1} \sqcap \exists requires.C^2_{S_1} \sqcap \exists requires.C^3_{S_1} \sqcap \exists requires.C^4_{S_1} \sqcap \exists returns.ContactPerson \tag{3}$$

where conjuncts $C^{i,1\leq i\leq 4}_{S_1}$*, described by means of TBox* $\mathcal{T}$ *in Fig.1, are defined by:*

$$C^1_{S_1} \doteq \exists hasFN.FirstName \sqcap \exists hasLN.LastName \tag{4}$$

$$C^2_{S_1} \doteq BusinessAccount \sqcap \exists hasSkype.SkypeAccount \tag{5}$$

$$C^3_{S_1} \doteq MaillingAddress \sqcap \exists hasMail.GMail \tag{6}$$

$$C^4_{S_1} \doteq Location \sqcap \exists hasLat.Latitude \sqcap \exists hasLong.Longitude \tag{7}$$

2.3 Semantic User Profile

Semantics of user profile can be expressed on different dimensions, mainly on information related to her i) identity and possessions, but could be also extended with her ii) long-term interests in topics [23] and preferences, iii) skills [24], iv) behaviour [25] and v) knowledge or beliefs in certain domains. Since the description of semantic user profiles are tailored, in this work, to access services in a personalized way trough their instantiation, we will focus solely on the former descriptions. Indeed, descriptions from (ii) to (v) such as the user's interests and preferences are mainly relevant information for prioritizing services in a discovery process rather than making services and their (input) parameters adapted to the user. Therefore, a DL expression of semantic user profiles is defined by the conjunction of different parts as expressed in (8).

$$Profile \doteq \bigsqcap_i \exists hasInfo.Info_i \bigsqcap_j \forall hasInfo.(\neg NonInfo_j) \tag{8}$$

This definition is based on a single role or property called $hasInfo$, which describes the user. While descriptions $Info_i$ cover a collection of data identifying users, descriptions expressed by $NonInfo_j$ refer to information users are not inclined to provide. The latter could be sensitive data such as medical record data. All initial information in (8) is collected from a short questionnaire [26] and could be also updated. Alternatively, the profile can be automatically generated from [27].

Example 3 ***(Semantic User Profile)***
Suppose a `British` *lady identified by her* `Name`, `ElectronicID` *and connected to the* `LinkedIn` *social network (http://www.linkedin.com/), but without any authorized access to her* `Bank Account` *from third parties. Its semantic profile* P_1 *is defined by:*

$$P_1 \doteq Female \sqcap \exists hasInfo.C^1_{P_1} \sqcap \exists hasInfo.C^2_{P_1} \sqcap \forall hasInfo.C^3_{P_1} \tag{9}$$

where conjuncts $C^{i,1\leq i\leq 3}_{P_1}$*, described by means of TBox* $\mathcal{T}$ *in Fig.1, are defined by:*

$$C^1_{P_1} \doteq Person \sqcap (\exists hasName.Name) \sqcap (\exists hasNationality.British) \tag{10}$$

$$C^2_{P_1} \doteq Account \sqcap (\exists hasID.ElectronicID) \sqcap (\exists hasSocialNetwork.LinkedIn) \tag{11}$$

$$C^3_{P_1} \doteq \neg BankAccount \sqcup \neg(\exists hasBank.BankID) \tag{12}$$

The model (8) we suggest for semantic user profile can be adapted with straightforward modifications and extensions (e.g., roles) depending on the application [24].

3 Semantic Web Service Personalization

The formalization of services (2) and user profiles (8) in DLs is required to compare their descriptions at a semantic level. The comparison is proceeded through a process of DL expressions matching, based on [15, 16] and [17]. This section, illustrated with examples, describes the personalization approach in details. First of all, matching through standard DL reasoning is employed to adapt services by means of user profiles. Then, we suggest an approach to compute relevant information which is missing in the user profile to improve personalization. Finally, we present two ways to extend the user profile with the latter information.

3.1 Personalized Adaptation of Services with Semantic User Profiles

Our approach, based on semantic matching, aims at discovering relevant information in the semantic user profile that could fit (i.e., be used by) the service (i.e., its functional input parameters) to be executed. By considering such a personalization, we suggest an approach which adapts any service to the user. To support this adaptation, the matching is performed over a service S and a user profile P with respect to a TBox $\mathcal{T}$. Therefore, a role hierarchy (13) between $requires$ and $hasInfo$ roles of S (2) and P (8) is required in $\mathcal{T}$, at least for satisfiable compatibility reasons [15] between S and P.

$$\mathcal{T} \models hasInfo \sqsubseteq requires \tag{13}$$

According to (13), some information ($hasInfo$) from the user profile could be used by services to adapt and then personalized its input parameters ($requires$).

In detail, our approach achieves such an personalized adaptation by following steps in Algorithm 1. From a logical point of view, this algorithm evaluates potential matching between conjuncts C_S and C_P of respectively S and P. By emphasizing only robust matching [28] i.e., Exact in line 7 and PlugIn in line 10, this algorithm focuses on the description required by S and provided by P. Therefore, other matching such as $\mathcal{T} \models C_P \sqsupseteq C_S$ (Subsume) or $\mathcal{T} \not\models C_P \sqcap C_S \sqsubseteq \bot$ (Intersection) are not valued since they do not fit our service adaptation purpose.

Algorithm 1: Matching-based Personalized Adaptation: $adapt(S, P, \mathcal{T})$.

```
1  Input: A user Service S, a Profile P, a Terminological Box T.
2  Result: match: a set of matching triples (C_S, C_P, type) if C_S could be adapted by C_P
           with a matching type, Incompatibility otherwise.
3  begin
4      match ← ∅;
5      foreach ∃requires.C_S ∈ Input(S) do
6          // Exact: The descriptions of user profile and service match perfectly.
7          if there exists ∃hasInfo.C_P ∈ Info(P) such that T ⊨ C_P ≡ C_S then
8              match ← match ∪_set (C_S, C_P, '' ≡'''');
9          // PlugIn: The user profile description is more specific than service description.
10         if there exists ∃hasInfo.C_P ∈ Info(P) such that T ⊨ C_P ⊑ C_S then
11             match ← match ∪_set (C_S, C_P, '' ⊑'');
12         // Incompatible: The User profile and service descriptions are incompatible.
13         if there exists ∀hasInfo.(¬C_P) ∈ NonInfo(P) such that T ⊨ C_P ⊑ C_S
           then
14             return Incompatibility;
15     return match;
16 end
```

In Algorithm 1 Exact matching are tested before the PlugIn matching, mainly because of the logical implication relation between these matching. Indeed, if $\mathcal{T} \models C_P \equiv C_S$ (Exact), then $\mathcal{T} \models C_P \sqsubseteq C_S$ (PlugIn). The algorithm is then based on structural algorithms for satisfiability and subsumption [29]. Since it is reasonable to assume that

users and service providers do not enter contradicting information, we assume that the services S and users profiles P descriptions are consistent in $\mathcal{T}$.

The result of this personalized adaptation step is a set of matching triples $(C_S, C_P, type)$ (referring to $match$) wherein the description C_P in P could be used to adapt the service parameter C_S in S. The latter descriptions are completed with their matching type, emphasizing the accuracy of the personalized adaptation step (from C_P to C_S). In case S violates some descriptions in P, S and P are returned as incompatible.

Example 4 ***(Semantic and Personalized Adaptation of Services)***
Suppose the service S_1 and user profile P_1 in Examples 2 and 3. According to Algorithm 1, the triple $(C^1_{S_1}, C^1_{P_1}, PlugIn)$ is returned. Indeed, according to the TBox $\mathcal{T}$ in Fig. 1, it seems possible to personalized S_1 by adapting the `FirstName` *and* `LastName` *input parameters of S_1 with information of $C^1_{P_1}$ in the user profile, and more specially with its* `Name`*.*

Even if our approach is able to personalize services with user profile descriptions using subsumption-based DL reasoning, the latter profile maybe not always as accurate as it should be hence limiting its benefits. Indeed, in some cases parts of services could only partially match or even mismatch the profile.

Example 5 ***(Limitation of Semantic and Personalized Adaptation of Services)***
Even if $C^2_{S_1}$ and $C^2_{P_1}$ in Examples 2 and 3 are both related to `Account` *description in $\mathcal{T}$, $C^2_{P_1}$ cannot be used to personalize S_1 and more specially $C^2_{S_1}$ because of missing description* (1) *in its profile P_1 i.e., neither* `BusinessAccount`*, nor* `SkypeAccount`*.*

3.2 Towards Incomplete Semantic User Profile

Since the personalization process may fail because of under specification or missing description in the user profile (e.g., Example 5), we suggest to extend Algorithm 1 with Algorithm 2 by exploiting results from non robust matching cases between conjuncts C_S and C_P i.e., Intersection and Subsume. Therefore, we aim at i) inferring further matching triples from the latter matching cases, and ii) discovering descriptions which are required by services but not provided by the user profile by applying abduction.

Further Matching Triples for Personalized Adaptation of Services: In both Intersection and Subsume matching cases, simple matching triple $(C_S, C_P, type)$ cannot be returned as in Algorithm 1 mainly because *only a part* of C_P is required to adapt *(again) a part* of C_S. Towards this issue, we identify descriptions B and A that need to be removed respectively from C_S and C_P to obtain a PlugIn matching between C_P and C_S i.e., $C_P \Delta C_S$ (Definition 2, adapted from [30]).

Definition 2 ***(Symmetric Difference)***
Let $\mathcal{L}$ be a DL, C, D be two concepts in $\mathcal{L}$, $\mathcal{T}$ be a set of axioms in $\mathcal{L}$ and $\mathcal{A}$ be a set of assertions. The symmetric difference between C, D, denoted as $C \Delta D$ consists in finding two concepts $A, B \in \mathcal{L}$ such that

$$\mathcal{T} \models C \backslash A \sqsubseteq D \backslash B \tag{14}$$

$C^* \Delta D$ and $C^* \Delta D$ refer respectively to A and B in (14)*.*

Example 6 ***(Symmetric Difference)***
Let $C^2_{P_1}$ and $C^2_{S_1}$ be two $\mathcal{ALC}$ descriptions respectively defined in Examples 2, 3 with respect to $\mathcal{T}$ in Figure 1. According to Definition 2, $C\Delta D$ is defined by A and B i.e.,

$$A \doteq \exists hasID.ElectronicID \tag{15}$$

$$B \doteq \exists hasID.OpenID \sqcap \exists hasSkype.SkypeAccount \tag{16}$$

Algorithm 2: (Refined) Personalized Adaptation: $refinedAdapt(S, P, \mathcal{T})$.

1 **Input**: A **S**ervice S, a user **P**rofile P, a **T**erminological Box $\mathcal{T}$.
2 **Result**: A pair $(match, miss)$ where $match$ is set of matching Triples $(C_S, C_P, type)$ if C_S could be adapted by C_P with a matching $type$, and $miss$ refers to the set of missing description in P in order to adapt S.
3 **begin**
4 $i \leftarrow 0$; $miss \leftarrow \emptyset$; $match \leftarrow \emptyset$;
5 **foreach** $\exists requires.C_S \in Input(S)$ **do**
6 *// Non Robust: Profile description partially covers service description.*
7 **if** *there exists* $\exists hasInfo.C_P \in Info(P)$ *such that* $\mathcal{T} \not\models C_P \sqcap C_S \sqsubseteq \bot$ *and* $\mathcal{T} \not\models C_P \sqsubseteq C_S$ **then**
8 $A \leftarrow C^*_P \Delta C_S$; *// Descriptions A in C_P and B in C_S...*
9 $B \leftarrow C_P \Delta C^*_S$; *// ...that make $\mathcal{T} \not\models C_P \sqsubset C_S$.*
10 $X \leftarrow C_S \backslash B$; *// Descriptions X in C_S and Y in C_P...*
11 $Y \leftarrow C_P \backslash A$; *// ...such that $\mathcal{T} \models Y \sqsubseteq X$.*
12 $match \leftarrow match \cup_{set} (X, Y, '' \sqcap'')$;
13 $miss_i \leftarrow C_S \backslash C_P$;
14 $i \leftarrow i + 1$;
15 $miss \leftarrow \inf_{\sqsubseteq}\{miss_j | 0 \leq j \leq i\} \backslash_{set} \{\top\}$;
16 **return** $(match, miss)$;
17 **end**

According to Definitions 1 and 2, it is straightforward to identify the parts $X \in \mathcal{P}(C_S)$ and $Y \in \mathcal{P}(C_P)$ respectively from C_S and C_P (where $\mathcal{P}(S)$ refers to power set of S) which are required to ensure that X can be adapted by Y (in the sense of Algorithm 1 i.e., $\mathcal{T} \models C_P \sqsubseteq C_S$). Both descriptions X and Y, parts of the new matching triples, are defined as $C_S \backslash (C_P \Delta C^*_S)$ and $C_P \backslash (C^*_P \Delta C_S)$. Algorithm 2 elaborates these further matching triples (line 12) from respectively lines 9, 10 and lines 8, 11. The first element of the matching triples represents the description in C_S which subsumes C_P, whereas the second element represents the part in C_P which is subsumed by C_S. In other words, these triples present the descriptions in C_P which will be used to further adapt the descriptions in C_S. Finally, as Algorithm 1, results are aggregated in $match$.

Example 7 ***(Matching Profile to Service with an Intersection Match)***
According to Example 2, 3 and $\mathcal{T}$ in Figure 1, $\mathcal{T} \not\models C^2_{P_1} \sqcap C^2_{S_1} \sqsubseteq \bot$. Therefore, only some parts Y of $C^2_{P_1}$ can be used to adapt some parts X of $C^2_{S_1}$. Applying Algorithm 2 (lines from 8 to 12) and using results from Example 6, Y and X are defined as:

$$Y \equiv Account \sqcap (\exists hasSocialNetwork.LinkedIn) \tag{17}$$

$$X \equiv Account \sqcap (\exists hasSocialNetwork.SocialNetworkAccount) \tag{18}$$

This simply means that the `SocialNetworkAccount` *of* $C^2_{P_1}$ *(i.e.,* `LinkedIn`*) can be used to instantiate the* `SocialNetworkAccount` *requirement (i.e., input parameter) of* $C^2_{S_1}$.

Computing Missing Description in a User Profile: In addition, Algorithm 2 (lines 13 and 15) computes descriptions which are required by C_S and not (or partially) provided by C_P. To this end, abduction is applied between the latter conjuncts (Definition 1) and then the result is aggregated in a set of missing description $miss$ (Definition 3).

Definition 3 ***(Set of Missing Description)***
The set of missing description $miss$ *of a service personalization problem* $(S, P, \mathcal{T})$ *is defined by:*

$$miss(S, P, \mathcal{T}) \doteq \inf_{\sqsubseteq}\{C_S \backslash C_P | \mathcal{T} \not\models C_P \sqcap C_S \sqsubseteq \bot\} \backslash_{set} \{\top\} \tag{19}$$

where C_S *and* C_P *are respectively conjuncts of* $\exists hasInfo.C_P$ *and* $\exists requires.C_S$.

According to Definition 3, $miss$ gathers the most specific descriptions of the set $\{C^i_S \backslash C^j_P\}$. Therefore a same description (i.e., the most specific in the service requirements) can be used to satisfy different abduction problems $C^i_S \backslash C^j_P$ and then could be exposed as a description not provided by P but required by some conjuncts (related by subsumption) of S. $miss$ does not only explain why services have not been adapted and personalized (regarding a user profile) but also suggest a solution to extend the personalization of semantic web services.

Property 1 ***(Empty Set of Missing Description)***
The set of missing description $miss$ *of a web service personalization problem* $(S, P, \mathcal{T})$ *with either i)* $\mathcal{T} \models C_P \equiv C_S$*; or ii)* $\mathcal{T} \models C_P \sqsubseteq C_S$ *is the empty set.*

Proof. By Definition 1, $C_S \backslash C_P$ is defined by $\mathcal{T} \models C_P \sqcap (C_S \backslash C_P) \sqsubseteq C_S$. Therefore, we obtain in both cases that $C_S \backslash C_P \equiv \top$ is a solution i.e., $miss$ is defined by the empty set according to Definition 3.

The property 1 justifies our choice of not computing missing descriptions in Algorithm 1. Indeed, such a computation would reach to the empty set.

Example 8 ***(Set of Missing Description)***
Since the description in P_1 *is not enough to totally adapt and personalize* S_1 *(Example 5), Algorithm 2 and Definition 3 are required to discover the missing description* $miss$ *in* P_1*. According to the latter definition,* $miss$ *is constituted by the union of results of the abduction problems* $C^2_{S_1} \backslash C^2_{P_1}$ *(Example 1),* $C^4_{S_1} \backslash \top$ *and* $C^3_{S_1} \backslash C^2_{P_1}$*:*

$$C^2_{S_1} \backslash C^2_{P_1} \equiv Account \sqcap \exists hasID.OpenID \sqcap \exists hasSkype.SkypeAccount \tag{20}$$

$$C^4_{S_1} \backslash \top \equiv Location \sqcap \exists hasLat.Latitude \sqcap \exists hasLong.Longitude \tag{21}$$

$$C^3_{S_1} \backslash C^2_{P_1} \equiv MaillingAddress \sqcap \exists hasMail.GMail \tag{22}$$

Since $GMail \sqsubseteq OpenID$, $hasMail \sqsubseteq hasID$ and $miss$ only considers most specific description (whether subsumption-based comparable), $miss$ is $\{A, C^4_{S_1}\}$ where:

$$A \equiv Account \sqcap \exists hasMail.GMail \sqcap \exists hasSkype.SkypeAccount \quad (23)$$

According to Property 1, conjuncts $C^{i,1\leq i\leq 4}_{S_1}$ and $C^{j,1\leq j\leq 2}_{P_1}$ such that $\mathcal{T} \models C^j_{P_1} \sqsubseteq C^i_{S_1}$ are not considered by Algorithm 2.

3.3 Extending Semantic User Profile with Further (Missing) Descriptions

Once the set of missing descriptions is retrieved through $miss$, two approaches are considered. First of all, an intuitive method consists in discovering [31] which new and appropriate services $S_{i,1\leq i\leq n}$ would be able to return the missing description. Following this approach, this description $miss$ could be identified and satisfied by the conjunction of some output parameters $Out\_S_{i,1\leq i\leq n}$ of these services $S_{i,1\leq i\leq n}$. Therefore, depending on the available description in the user profile P and the description of output parameters of $S_{i,1\leq i\leq n}$, we could proceed to the service S personalization. Indeed, the conjunction C of the output parameters $Out\_S_{i,1\leq i\leq n}$ and the user profile P i.e., $\prod_{i=1}^{n} Out\_S_i \sqcap P$ can be used to adapt and personalize the service S by applying Algorithm 1 as following: $adapt(S, C, \mathcal{T})$. However, each input parameter of these discovered services S_i has to be known at run time. To this end, we can imagine use the description available in P to adapt this new discovered services. In this direction, Algorithm 1 is applied on the n relevant services as following: $adapt_{i,1\leq i\leq n}(S_i, P, \mathcal{T})$. One constraint of this method is related to the number of services (and their input parameters to be satisfied by P) which are required to satisfy $miss$.

In the second approach, the set of missing description is simply suggested to the user. The user is then responsible of providing the description that the system needed to adapt and personalize the service. The requested information is also used to populate the semantic user profile, hence available for further personalization purposes.

4 Validation

In this section, we discuss the prototype tool that we developed to provide personalized adaptation of semantic web services. Moreover we give a preliminary evaluation of the suggested approach by analyzing some results obtained with the prototype.

4.1 Architecture and Implementation

Figure 2 shows the high level prototype architecture wherein we implemented and tested our personalization approach. In detail, our approach, part of the core architecture of the EU project SOA4All[1] (Service Oriented Architectures for All), has been integrated with three main state-of-the-art modules, namely a *DL Reasoning*, a *Service Discovery* and a *SPARQL Query Engine* module. The main function of the former module is to check satisfiability, subsumption and infer on-line matching between user profile and service

[1] http://www.soa4all.eu/

description. The MAMAS-tng[2] reasoner has been used to compute standard reasoning and evaluate abduction. This reasoner has been extended to compute symmetric difference (Definition 2). The *SPARQL*[3] *Query Engine* module RDF2GO[4], is required to manipulate matching triples i.e., RDF-based C_P, C_S and $miss$ data returned by Algorithms 1 and 2. For instance, this module transforms RDF-based C_P in C_S using a CONSTRUCT query form of SPARQL in order to adapt C_S with C_P.

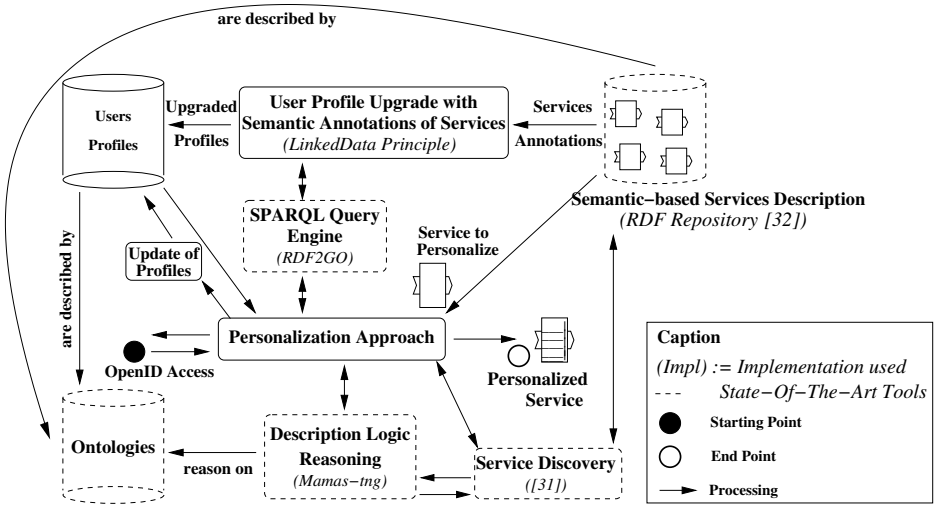

Fig. 2. Core Architecture for Service Personalization.

In addition, a pool of (SA-WSDL) *semantic-based services* (formalized in (2)), and *semantic user profile* (formalized in (8) and identified by OpenID[5] are stored in two different RDF[6] repositories, e.g., [32] for services and a Sesame-based[7] for profiles. Their descriptions are based on different $\mathcal{ALC}$ TBoxes, depending on *ontologies* used to annotate services. Therefore, the semantic descriptions used to annotate a service and a profile may differ, even if the annotated element is the same. Towards this issue of ontology matching [33], we integrated a simple component (*User Profile Upgrade* Component) which aims at semi-manually linking profiles and services descriptions, following the Linked Data principles [34] e.g., by further annotating profiles with relevant `owl:sameAs`, `rdfs:subClassOf`, `owl:equivalentProperty` or `rdfs:subPropertyOf` constructs. To this end, the *SPARQL Query Engine* module is used. Finally, once the service personalization process is achieved, the service is ready to be executed by any execution engine. Moreover, the semantic user profile is updated in case of missing description (Definition 3). To this end, either the user is able to provide it, or a service discovery process is performed (Section 3.3).

[2] http://dee227.poliba.it:8080/MAMAS-tng/DIG
[3] http://www.w3.org/TR/rdf-sparql-query/
[4] http://semanticweb.org/wiki/RDF2Go
[5] http://openid.net/
[6] http://www.w3.org/RDF/
[7] http://coconut.tie.nl:8080/storage/repositories/profiles

4.2 Experimental Results

Evaluation of personalization systems remains a challenge due to the lack of understanding of what factors affect user satisfaction with a personalization system [35]. Personalization systems are in general evaluated and compared on the accuracy of predictions. However, a comparison based on the accuracy of our approach and existing personalization methods is not appropriate. Indeed, our work i) features different expressivity (compared to syntactic-based approaches), and ii) does not only evaluate if a user profile and a service match but also explains how they could match and why they could not (compared to semantic-based approaches).

Therefore we analyze the performances of our approach by i) comparing abduction- with difference-based [36, 37] personalization using different expressivities of DL, and ii) studying the impact of the *User Profile Upgrade* component (Fig.2) on the personalization process. The experiments have been conducted on Intel(R) Core(TM)2 CPU, 2.4GHz and 2GB RAM.

Comparing Abduction- and Difference-based Service Personalization: Since other approaches based on DL difference operator such as [36] or [37] can be used to compute from a given description all the information different in another description, we suggest to compare them with abduction to achieve service personalization (Fig.3). The comparison is driven on three set of ontologies with different DLs used to annotate services and profiles i.e., $\mathcal{ALC}$, $\mathcal{ALN}$ and $\mathcal{ALE}$, from the most to the least expressive. In particular, the two former ontologies are based on the $\mathcal{ALE}$ TBox (formally defined by 1100 concepts and 390 properties) wherein only DL operators changed in descriptions. Personalization of up to 100 services have been considered in this experiment, especially for obtaining convincing results towards their applicability in real (industrial) scenarios.

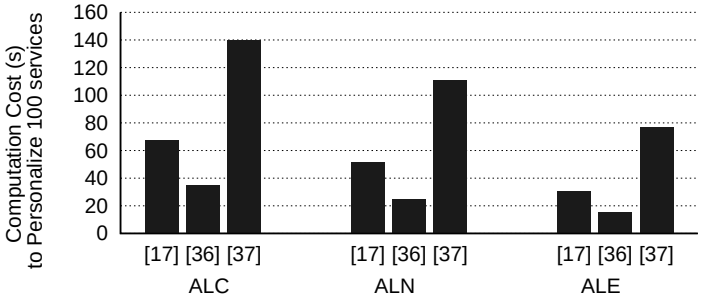

Fig. 3. Abduction vs. Difference.

The [37]'s difference consider a semantic maximum (ordering according to the subsumption operator) between (only) subsumption-based comparable descriptions. Even if they provides sufficient condition (i.e., structural subsumption relation) to characterize the uniqueness of difference, some TBoxes cannot be considered such as $\mathcal{ALN}$. In addition, this is the most time consuming approach regarding the three set of experimentation, mainly because they perform an equivalence between two concept descriptions ($\mathcal{T} \models D \sqcap B \equiv C$) whereas abduction computes (only) a subsumption of concept descriptions ($\mathcal{T} \models D \sqcap B \sqsubseteq C$). The difference operator of [36] is a refinement of

[37]'s difference that considers the syntactic minimum ($\preceq_d$) between incomparable descriptions. Such a consideration, limiting the relevance of its results using expressive DLs, explains its very good performance.

Even if deciding subsumption, computing abduction and difference in $\mathcal{ALE}$ is NP-complete, Figure 3 reports the feasibility and the scalability of the personalization process. However these results depend on size and structure of the used ontologies, size and complexity of user profile and service descriptions. The choice of abduction to personalize services is justified by its performance in the three different DLs studied. Indeed, our process of personalizing services with an expressive DL $\mathcal{ALC}$ over performs the time consuming [37]'s difference-based personalization using $\mathcal{ALN}$ or $\mathcal{ALE}$ DLs.

Impact of the User Profile Upgrade Component: Since our personalization approach aims at matching services to user profiles, it is required that their descriptions can be semantically compared, either using a same ontology, or by establishing subsumption-based relationships between some of them. This experiment studies the qualitative impact of the latter (i.e., *User Profile Upgrade* component in Fig.2) on personalization.

To this end, 55 different initial $\mathcal{ALN}$ TBoxes (i.e., average of 103 concepts and 61 properties) have been used to annotate 100 services $S_{i,1\leq i\leq 100}$ and one profile P. Then, progressively, some inter-connections have been established between them, favouring the matching hence the personalization. Roughly speaking, the progression of connections grows exponentially along 10 rounds $R_{i,1\leq i\leq 10}$ i.e., from adding 2^1 to 2^{10} connections. After each round, we run our personalization approach on these 100 services and evaluate the rate of i) input parameters q_S that can be adapted given P and ii) missing description q_{miss}, both regarding the number of description in S_i.

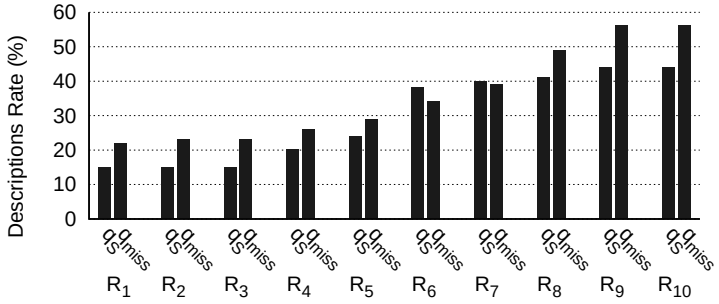

Fig. 4. Impact of the User Profile Upgrade Component on Service Personalization.

As shown in Fig.4, the more interconnections between ontologies, the better personalization (i.e., q_S). In the same way, the rate of description to be updated in the user profile (i.e., q_{miss}) is also improving. Both improvements follow a linear evolution even if an exponential generation of interconnections is applied. In more detail, 37% and 98% of service descriptions can be semantically compared (not matched) to profile description respectively in the first and last round of interconnections generation. The transition between round 5 and 6 (with 64 new connections) is the most significant with respectively 53% and 72% of comparable descriptions in services and profile. While our approach is able to adapt and personalize 44% of services in the last two rounds, the

personalization only reaches 15% of services in the first three rounds. This confirms the high impact of the user profile upgrade component in our personalization framework.

q_{miss} is in general higher than q_r since some input parameters of services cannot be captured by personalization e.g., variables such as flight destination, or sensitive data.

5 Related Work

Based on [38], [10] present *Personal Reader Framework*, an approach for RDF-based data extraction, combination, visualization and personalization. In particular, they generate personalized view of data by applying standard subsumption-based matching between data description, user profile and contextual information. To this end they adopt TRIPLE [39], a rule language which is designed for querying and transforming RDF models. Even if their approach is augmented with contextual information, their matching-based personalization approach do not consider automated update of user profile.

Contrary to our approach, [8, 23, 27] perform personalization to obtain more relevant services during the discovery process. Therefore they do not address services instantiation (through their personalized adaptation) but rather service selection. The selection process is based on the compatibility of users' interests, disinterests and service descriptions. Services that do not match a certain profile are discarded on the fly. Since the matching process between the latter descriptions is handled on mobile device, both approaches [23] sacrifice expressivity of DL and use standard DL inferences [15, 16]. Even more (semantically) limited, [27] consider only one-to-one syntactic matching of service and profile descriptions for personalization. [8] consider non-functional parameters, preferences and knowledge that is implicitly given by previous service to personalize the selection of services.

[24] present an approach for matching user profiles for applications such as job recruitment or dating system. The matching, which is performed on a demand profile P_d and a supply profile P_S, aims at evaluating their semantic similarity. In the same way as our work, abduction is used, but only for weighting, ranking purposes and not for extracting and reusing relevant parts of P_d and P_S. In addition they apply the non-standard inference contraction [40] to evaluate the effort (i.e., description) required to make $P_d \sqcap P_s$ satisfiable in $\mathcal{H}$. On the contrary, we assume the latter conjunct to be satisfiable in $\mathcal{H}$ since one goal of our personalization approach is to suggest more specific descriptions (and so satisfiable) to the user profile regarding the service descriptions.

6 Conclusion

In this work we studied service personalization or the way to tailor services to a particular user. In particular, we addressed automated instantiation of services (through personalized adaptation) which is crucial for advanced usability i.e., how to prepare and present services ready to be executed while limiting useless interactions with users? Towards this issue, we considered a semantic augmentation of services and extensible user profiles to infer potential matching between both descriptions. The semantic matching, core of our approach, exploits standard DL reasoning and abduction to i) identify useful parts of a user profile that satisfy the service requirements (i.e., input parameters) and

ii) compute the descriptions required by a service to be consumed but not provided by the user profile. Our approach, integrated in the service consumption of the EC-funded project SOA4All, has been augmented with a process of user profile upgrade in case heterogeneous ontologies are used to describe services. Such an augmentation, aiming at linking data description of services and user profile, has been validated by experimental results. In the same way, the latter results confirm our choice of preferring abduction rather than other difference operators for scalability and expressivity reasons.

In future work we will consider a more precise abduction operator, which is also easy-to-compute in expressive DLs in order to address more complex cases of personalization and user profile update. We will also focus on the context dimension for personalization. Another area of investigation is the policy-based control access [41] of the user profile by third parties during its update. Finally, as reported by experimental results, automating ontologies alignments is a key issue that need to be address.

References

1. Mobasher, B., Cooley, R., Srivastava, J.: Automatic personalization based on web usage mining. Commun. ACM **43**(8) (2000) 142–151
2. Goldberg, D., Nichols, D.A., Oki, B.M., Terry, D.B.: Using collaborative filtering to weave an information tapestry. Commun. ACM **35**(12) (1992) 61–70
3. Lieberman, H.: Letizia: An agent that assists web browsing. In: IJCAI (1). (1995) 924–929
4. Pazzani, M.J.: A framework for collaborative, content-based and demographic filtering. Artif. Intell. Rev. **13**(5-6) (1999) 393–408
5. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services: Concepts, Architectures and Applictions. Springer-Verlag (2004)
6. Sivashanmugam, K., Verma, K., Sheth, A.P., Miller, J.A.: Adding semantics to web services standards. In: ICWS. (2003) 395–401
7. García, J.M., Ruiz, D., Cortés, A.R.: A model of user preferences for semantic services discovery and ranking. In: ESWC (2). (2010) 1–14
8. Balke, W.T., Wagner, M.: Towards personalized selection of web services. In: WWW (Alternate Paper Tracks). (2003)
9. Mandell, D.J., McIlraith, S.A.: Adapting bpel4ws for the semantic web: The bottom-up approach to web service interoperation. In: ISWC. (2003) 227–241
10. Baumgartner, R., Henze, N., Herzog, M.: The personal publication reader: Illustrating web data extraction, personalization and reasoning for the semantic web. In: ESWC. (2005) 515–530
11. Blake, M.B., Nowlan, M.F.: A web service recommender system using enhanced syntactical matching. In: ICWS. (2007) 575–582
12. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American **284**(5) (2001) 34–43
13. Smith, M.K., Welty, C., McGuinness, D.L.: Owl web ontology language guide. W3c recommendation, W3C (2004)
14. Baader, F., Nutt, W. In: The Description Logic Handbook: Theory, Implementation, and Applications. (2003)
15. Li, L., Horrocks, I.: A software framework for matchmaking based on semantic web technology. In: WWW. (2003) 331–339
16. Paolucci, M., Kawamura, T., Payne, T., Sycara, K.: Semantic matching of web services capabilities. In: ISWC. (2002) 333–347

17. Noia, T.D., Sciascio, E.D., Donini, F.M., Mongiello, M.: Abductive matchmaking using description logics. In: IJCAI. (2003) 337–342
18. Pistore, M., Marconi, A., Bertoli, P., Traverso, P.: Automated composition of web services by planning at the knowledge level. In: IJCAI. (2005) 1252–1259
19. McIlraith, S.A., Son, T.C.: Adapting golog for composition of semantic web services. In: KR. (2002) 482–496
20. Ankolenkar, A., Paolucci, M., Srinivasan, N., Sycara, K.: The owl-s coalition, owl-s 1.1. Technical report (2004)
21. Fensel, D., Kifer, M., de Bruijn, J., Domingue, J.: Web service modeling ontology submission, w3c submission. (2005)
22. Kopecký, J., Vitvar, T., Bournez, C., Farrell, J.: Sawsdl: Semantic annotations for wsdl and xml schema. IEEE Internet Computing **11**(6) (2007) 60–67
23. Kleemann, T., Sinner, A.: User profiles and matchmaking on mobile phones. In: INAP. (2005) 135–147
24. Calì, A., Calvanese, D., Colucci, S., Noia, T.D., Donini, F.M.: A description logic based approach for matching user profiles. In: Description Logics. (2004)
25. Berendt, B., Spiliopoulou, M.: Analysis of navigation behaviour in web sites integrating multiple information systems. VLDB J. **9**(1) (2000) 56–75
26. Ghosh, R., Dekhil, M.: Discovering user profiles. In: WWW. (2009) 1233–1234
27. Weiß, D., Scheuerer, J., Wenleder, M., Erk, A., Gülbahar, M., Linnhoff-Popien, C.: A user profile-based personalization system for digital multimedia content. In: DIMEA. (2008) 281–288
28. Lécué, F., Delteil, A.: Making the difference in semantic web service composition. In: AAAI. (2007) 1383–1388
29. Borgida, A., Patel-Schneider, P.F.: A semantics and complete algorithm for subsumption in the classic description logic. J. Artif. Intell. Res. (JAIR) **1** (1994) 277–308
30. Herzig, A.: The pma revisited. In: KR. (1996) 40–50
31. Benatallah, B., Hacid, M.S., Léger, A., Rey, C., Toumani, F.: On automating web services discovery. VLDB J. **14**(1) (2005) 84–96
32. Pedrinaci, C., Lambert, D., Maleshkova, M., Liu, D., Domingue, J., Krummenacher, R.: Service Engineering: European Research Results. In: Adaptive Service Binding with Lightweight Semantic Web Services. (2010)
33. Euzenat, J., Shvaiko, P.: Ontology matching. Springer-Verlag, Heidelberg (DE) (2007)
34. Bizer, C., Heath, T., Berners-Lee, T.: Linked data - the story so far. Int. J. Semantic Web Inf. Syst. **5**(3) (2009) 1–22
35. Anand, S.S., Mobasher, B.: Introduction to intelligent techniques for web personalization. ACM Trans. Internet Technol. **7**(4) (2007) 18
36. Brandt, S., Kusters, R., Turhan, A.: Approximation and difference in description logics. In: KR. (2002) 203–214
37. Teege, G.: Making the difference: A subtraction operation for description logics. In: KR. (1994) 540–550
38. Antoniou, G., Baldoni, M., Baroglio, C., Baumgartner, R., Bry, F., Eiter, T., Henze, N., Herzog, M., May, W., Patti, V., Schaffert, S., Schindlauer, R., Tompits, H.: Reasoning methods for personalization on the semantic web. Annals of Mathematics, Computing & Telefinformatics **2(1)** (2004) 1–24
39. Sintek, M., Decker, S.: Triple - a query, inference, and transformation language for the semantic web. In: International Semantic Web Conference. (2002) 364–378
40. Colucci, S., Noia, T.D., Sciascio, E.D., Donini, F.M., Mongiello, M.: Concept abduction and contraction in description logics. In: DL. (2003)
41. Abel, F., Henze, N., Krause, D.: Context-aware ranking algorithms in folksonomies. In: WEBIST. (2009) 167–174

A purely logic-based approach to approximate matching of Semantic Web Services

Jörg Schönfisch[1,2], Willy Chen[1,2], and Heiner Stuckenschmidt[2]

[1] SysTec-CAx GmbH, München, Germany
{joerg.schoenfisch,willy.chen}@systec-cax.de
http://www.systec-cax.de
[2] KR & KM Research Group, University of Mannheim, Germany
heiner@informatik.uni-mannheim.de
http://ki.informatik.uni-mannheim.de

Abstract. Most current approaches to matchmaking of semantic Web services utilize hybrid strategies consisting of logic- and non-logic-based similarity measures (or even no logic-based similarity at all). This is mainly due to pure logic-based matchers achieving a good precision, but very low recall values. We present a purely logic-based matcher implementation based on approximate subsumption and extend this approach to take additional information about the taxonomy of the background ontology into account. Our aim is to provide a purely logic-based matchmaker implementation, which also achieves reasonable recall levels without large impact on precision.

Keywords: semantic web services, approximate matching, approximate subsumption, logic-based

1 Motivation

Web service discovery and matchmaking is a field of ongoing research. For semantic web services, logic-based reasoning offers the possibility of high precision. However, in general it achieves only poor recall levels. Due to this, most current matcher implementations utilize a combination of logic- and non-logic-based, or even only non-logic-based similarity measures.

The best performing matcher during 2009's Semantic Service Selection contest[3] S3, URBE [15], uses only non-logic-based matching. Most of the other matchers we found so far (SAWSDL-MX2 [8], DAML-S Matcher [14], COCOON Glue [1], SAWSDL-iMatcher3/1[3]) utilize a hybrid strategy, merging the results of a logic and non-logic matching step. However, they only support coarse degrees of a logic match, i.e., *exact*, *plug in*, *subsumes*, *subsumed-by*, *intersection* and *fail* [22].

LOG4SWS.KOM [18] improves this approach by defining adaptive degrees of match and assigning numerical values to them, which can easily be combined

[3] http://www-ags.dfki.uni-sb.de/~klusch/s3/s3-2009-summary.pdf

with other numerical similarity values. These numerical values are determined through the taxonomic distance of the concepts used to annotate the service offer and request. As a fallback strategy, LOG4SWS uses WordNet[4] to determine the similarity of interface, operation or parameter names, if no concepts are available or an error occurs during processing. It toop part non-competitively in 2009's S3 as a beta version and outperformed URBE and SAWSDL-MX2 on average precision [18].

iSeM [7] is based on concept abduction [2], which is a similar notion of approximate subsumption to the one we use in our implementation. Consequently, it also supports approximate and more fine-grained degrees of match. Through concept abduction iSeM determines which properties of a request or an offer prevent the subsumption from succeeding and ranks request-offer pairs according to the loss of information by omitting those properties. Additionally it implements non-logic-based similarity measures to further improve the ranking.

Another service matcher which directly takes the structure of the taxonomy into account when computing matches is F-Match [21]. Yau et al. compute semantic similarity of concepts by their distance and relationship in the taxonomy as proposed in a graph matching approach by Zhong et al. [23] and use this value, together with some others, for ranking the services. Their evaluation based on randomly generated service offers and requests shows a surprisingly high precision and recall of 100%.

Our matcher is based on the notion of approximate subsumption as proposed in [19]. We extend this approach in three ways: our first addition utilizes information about the taxonomy of the background ontologies to achieve a more fine grained ranking. The other two extensions are aimed at lowering query response times through a slight change in the definition of approximate subsumption and optimized query execution order. Our goal is to improve recall through approximation, but keeping high precision and a fast query response time. For our matcher we assume web services to be annotated with semantic information according to the SAWSDL standard [9].

This paper is structured as follows: In section 2 we define approximate subsumption and give an example of its usage. The concept of approximate subsumption as we applied it to semantic web services and our implementation is described in section 3. We evaluate our implementation in section 4 and conclude in section 5.

2 Approximate Subsumption

In this section we briefly describe approximate subsumption. In [19] S-Interpretations [16] are applied to description logic, assigning each concept not in the set S the top $\top$ or bottom $\bot$ concept, depending on which interpretation is used. Consequentially, concepts not in S will be ignored by a subsumption test or cause it to fail. The interpretation which maps concepts to the top concept is called

[4] `http://wordnet.princeton.edu/`

upper approximation $\mathcal{I}_S^+$ and the interpretation mapping concepts to the bottom concept is called lower approximation $\mathcal{I}_S^-$. By applying these approximations to the definition of standard subsumption $\forall \mathcal{I} : \mathcal{I} \models C \sqsubseteq D \Leftrightarrow (C \sqcap \neg D)^{\mathcal{I}} = \emptyset$ (with C and D being concept expressions), an approximate subsumption operator is defined: $\underset{S}{\sqsubseteq}$:

$$\forall \mathcal{I} : \mathcal{I} \models C \underset{S}{\sqsubseteq} D \Leftrightarrow (C \sqcap \neg D)^{\mathcal{I}_S^-} = \emptyset$$

It is shown that this approach has the property of generalized monotonicity, making it possible to generate weaker version of the subsumption operator with every approximation step by decreasing the size of S. This allows to rank a result list based on the degree the operator had to be weakened to receive a specific result. Possible strategies for altering S include contraction by removing concepts sequentially or permutation by creating every possible combination of concepts in S.

A great advantage of this approach is, that it can be implemented by syntactic modifications of concept expressions and then be evaluated by standard description logic reasoners [20]. The definition of the rewriting rules for theses modifications for the lower approximation $(.)^-$ are as follows (with A being an atomic concept):

$$(A)^- \rightarrow \bot \text{ if } A \in S \tag{1}$$
$$(\neg A)^- \rightarrow \bot \text{ if } A \in S \tag{2}$$
$$(\neg C)^- \rightarrow \neg (C)^+ \tag{3}$$
$$(C \sqcap D)^- \rightarrow (C)^- \sqcap (D)^- \tag{4}$$
$$(C \sqcup D)^- \rightarrow (C)^- \sqcup (D)^- \tag{5}$$

The definition of the upper approximation $(.)^+$ is analogous, only equations 1 and 2 are adapted:

$$(A)^+ \rightarrow \top \text{ if } A \in S \tag{6}$$
$$(\neg A)^+ \rightarrow \top \text{ if } A \in S \tag{7}$$

Applying these rewriting rules to the approximate subsumption operator we get the following alternative definition:

$$\forall \mathcal{I} : \mathcal{I} \models C \underset{S}{\sqsubseteq} D \Leftrightarrow (C)^- \cap \neg (D)^+ = \emptyset$$

So to check for approximate subsumption we have to create the lower approximation of a service offer and the upper approximation of the service request and test whether the intersection is equal to the empty set.

In our implementation we call this original definition of approximate subsumption Simple strategy. The following gives an example of how this strategy

is carried out[5]: Lets consider a search application for pizza delivery services. The user may specify different toppings he likes to have on his pizza, and the search then returns a number of delivery services offering pizzas with these ingredients. For example a user wants a vegetarian pizza with *Broccoli* and *Artichoke*. Unfortunately no delivery can satisfy this wish. Subsequently, the system approximates the request by creating two new request with *Broccoli* and *Artichoke* replaced with $\top$ respectively. Executing these two new requests, the system discovers pizza delivery services which offer *Broccoli* and *Mandarine* pizzas or *Artichoke* and *Mushroom* pizzas, which both might be relevant to the user. The next approximation step would approximate both *Broccoli* and *Artichoke* to $\top$ and subsequently return every pizza with two vegetarian toppings.

3 Applying Approximate Matching to Semantic Web Services

This chapter explains in more detail how we applied approximate subsumption to semantic web services and how we implemented partial matchmaking for service discovery and which extensions we made.

The implementation consists of two parts: The first part defines the web service ontology, processes the service offers and creates requests represented as concepts of the ontology. The second part, a generalized matchmaker, reasons on this ontology and computes matches to the request based on its subsumption relation to service offers.

The typical usage of our matchmaker is divided into an offline initialization and classification of service offers, and the online processing of requests. During the initialization semantic annotations are extracted from SAWSDL services and added to a service ontology. This ontology is then classified with the help of a description logic reasoner. After the initialization any number of request can be issued to the matchmaker without further reclassification.

3.1 Extracting Semantic Annotations

In order to build an ontology of all known web services the semantic annotations have to be extracted from the web service descriptions. A simplified version of a web service with SAWSDL annotation is shown in Fig. 1.

The concepts, which are extracted from the web service annotations, are added to a service ontology. This is a minimal ontology for describing a service, modelled in OWL [5]. It is quite similar to other upper ontologies for Web services, like OWL-S [12] or WSMO [10], but its only classes are *Service*, *Operation*, *Input* and *Output* and the object properties *hasOperation*, *hasInput* and *hasOutput* connecting them. Other aspects of a service, like how to connect to it, or

[5] To simplify this example we do not create the lower approximation of all offered pizzas, but only the upper approximation of the request. Nonetheless, the general procedure stays the same.

```
...

<interface name="pizzaDeliveryServiceSearch">
    <operation name="opSearchByTopping">
        <input messageLabel="In" element="searchDeliveryService"
            sawsdl:modelReference="Topping">
        <output messageLabel="Out" element="searchDeliveryServiceRepsonse"
            sawsdl:modelReference="Address">
    </operation>
</interface>

...
```

Fig. 1. Simplified excerpt from a SAWSDL-annotated Web service

information about availability or reliability, which are present in more complex ontologies, e.g. OWL-S or WSMO, are not modelled in our prototype. Figure 2 shows how this looks like for the Web service example from above.

```
pizzaDeliveryServiceSearch SubclassOf
   (hasOperation some opSearchByTopping)

opSearchByTopping SubclassOf
   (hasInput only Topping) and
   (hasOutput only Address) and
   (hasInput some Topping or hasOutput some
   Address)
```

Fig. 2. Excerpt from the service ontology showing a web service instance

While loading a service, the approximator also counts the occurrences of each concept used to annotate its parameters, which is important when determining the order in which they should be approximated (cf. approximation strategies). Furthermore, it maintains a list of cumulated occurrences, which are the sum of a concept's and all of its sub concepts' occurrences throughout the whole service ontology. This number is a better indicator for how restricting a specific constraint of the request is than the number of occurrences of a concept alone. For example, *OWLThing* (the super concept of all concepts) and *OWLNothing* (the sub concept of all concepts) will most likely never be used to annotate a Web Service, so they both have a number of occurrence of 0. However, the cumulated occurrence of *OWLThing* is the sum of all other concepts' occurrences, whereas the cumulated occurrence of *OWLNothing* is still 0. This means if a service parameter is enforced to be of type *OWLNothing*, this is much more restricting, than enforcing it to be of type *OWLThing*[6].

[6] Actually, a constraint for a parameter to be of type OWLNothing is unsatisfiable and a constraint for a parameter to be of type OWLThing is always fulfilled

3.2 Creating and Approximating Queries

Query Creation A request is represented in the same way as a normal service offer (cf. section 3.1). There are two possibilities of creating a new request:

- A *Query-by-Example* approach, by which an existing annotated service is used to create a request from it. This can be useful, if someone wants to find a replacement for an existing service, or services with duplicate abilities should be found.
- Input and Output parameters are specified directly and a request is built using these, for example when a user searches a service to fulfill a specific task.

Query Approximation The Web Service Approximator creates queries using permutations to change the S set. Two strategies define the approximation of concept expressions: The first strategy determines the order in which concepts are approximated, depending on their cumulated number of occurrences. This strategy supports three different variants, influencing the ranking of the results (cf. [17]):

- LESS: concepts are approximated in ascending order of their cumulated number of occurrences
- MORE: concepts are approximated in descending order of their cumulated number of occurrences
- RANDOM: concepts are approximated in a random order

The second strategy defines how fine grained the approximation should be. The SIMPLE variant approximates each concept of the request by replacing it with *OWLThing*, which is the approach in [19] described earlier. We extended this strategy with a variant that takes the taxonomy of the domain ontology into account:

Taxonomy Approach Instead of replacing a concept to be approximated with *OWLThing*, we use its direct super concept according to the taxonomy. We call this strategy TAXONOMY. Our definitions of lower and upper approximation for the TAXONOMY approach are the following:

$$(A)^- \rightarrow directsub(A) \text{ if } A \in S \tag{8}$$

$$(\neg A)^- \rightarrow directsub(A) \text{ if } A \in S \tag{9}$$

$$(A)^+ \rightarrow directsuper(A) \text{ if } A \in S \tag{10}$$

$$(\neg A)^+ \rightarrow directsuper(A) \text{ if } A \in S \tag{11}$$

The intention behind the TAXONOMY strategy is to create a much more detailed ranking of results and therefore achieve a higher precision than with the SIMPLE variant. This approach benefits especially if ontologies with a deep taxonomy are used for annotations and if concepts from every level of the taxonomy are used.

Revisiting the example of the pizza delivery service search, the first approximations according to the TAXONOMY strategy are the following: instead of replacing *Broccoli* and *Artichoke* with $\top$, we use their direct superconcept, which in this case is *Vegetable* for both. The search for pizzas with *Broccoli* and any other *Vegetable* or pizzas with *Artichoke* and any other *Vegetable* now only finds the delivery service offering *Artichoke* and *Mushroom* pizzas. The service delivering *Broccoli* and *Mandarine* pizzas is not found during the first approximation step, as *Mandarines* are a *Fruit* and no *Vegetable*. This pizza is found during the second step, when *Vegetable* is further approximated to *VegetarianFood*, of which *Fruits* are naturally are subconcept. Thus, the *Broccoli* and *Mandarine* pizza would be considered less relevant, as the request had to be further weakened to retrieve it.

Note that the approximator always generates all possible approximations of a request at once, avoiding duplicates by checking the parameters of each request. So for each super- or subconcept of a concept, respectively, a new request is generated. If the concept does not have a direct super- or subconcept, respectively, then no approximation is created. This is only the case for *OWLThing* and *OWLNothing*, as every other concept has *OWLThing* as superconcept and *OWLNothing* as subconcept.

Additionally, the approximator tries to avoid term collapsing [4] by prohibiting queries whose terms are all approximated to OWLThing. Term collapsing is a negative effect of approximate subsumption, which occurs if the approximated concept consists of conjunctions and one term is changed to *OWLThing*, or if it consists of disjunctions and one concept is approximated to *OWLNothing*, effectively turning the whole concept into *OWLThing* or *OWLNothing*, respectively.

During the creation of approximations, we generate a graph which captures the relations between the original request and its approximations. The root node of the graph is the original request, its direct children are the approximations created during the first step, their children are the approximations generate from them during the second step, and so on. This graph is later used to optimize the execution time of the matchmaker, especially when approximating along the taxonomy, which produces a large amount of queries: $Q \in \mathcal{O}(N^P)$, with Q being the number of created queries, P the number of parameters the request has and N the maximum number of superclasses a concept has. So the number of queries, and consequentially the execution time of the matcher, grows exponentially with the number of parameters. However, web service have mostly only a couple of parameters, so we hope to still calculate results in a reasonable amount of time.

The object representing an approximated query also stores the step in which it was approximated and the sum of the cardinalities of all concepts which were approximated to create this query. These two numbers are important to deter-

mine the position of each query's results in the final aggregated result list, as the partial matchmaker may process queries out of approximation order, and thus their results can not simply be appended to the list.

Query-only Approximation A drawback of the definition of approximate subsumption in [19] is the fact, that the concepts on both sides of the operator, in our case service requests and service offers, have to be rewritten. This means the whole ontology containing the service offers has to be approximated and reclassified by the reasoner for each request. Obviously, given an arbitrarily large service ontology, this is not feasible in a reasonable amount of time. We therefore changed the definition of the approximate subsumption operator to only approximate the request, leaving the concepts of the service ontology untouched.

$$\forall \mathcal{I} : \mathcal{I} \models C \underset{S}{\sqsubseteq} D \Leftrightarrow C^{\mathcal{I}} \cap (D)^{\mathcal{I}_S^+} = \emptyset$$

According to this definition we only apply the upper approximation to the request and test for subsumption with the service offers.

Optimizations During Query Execution As mentioned before, the approximation along the taxonomy produces a large amount of queries, leading to long delays until the matchmaker returns the results. To reduce the number of queries which have to be executed, we take advantage of the monotonicity property of the approximate subsumption and the graph structure created when approximating a request. The approach is based on the observation, that if two different queries, which were created during different steps and are related as ancestor / descendant, produce the same result lists, then, due to the monotonicity, every query between these two queries must produce the exact same result list and subsequently need not be executed. Our matchmaker utilizes this property by first executing the root and leave queries of the graph, and compares their results pairwise. If the results of a pair are equal, every query between this pair are omitted, otherwise the graph is split in two parts, and again the top and the bottom queries of this subgraphs are executed and compared.

Intersection Query After having executed the original query and all of its approximations, we also add all service offers to the result list, which have at least one concept in common with the request, no matter if it is used as input or output parameter. Surprisingly, this has a quite large impact on the precision of the SIMPLE strategy, as we will show in the evaluation.

4 Evaluation

In this chapter we evaluate the performance of the implemented matcher. The following points are considered:

– Retrieval performance of the implemented matcher compared to others
– Response time to user requests

We conducted the performance tests on a Mac Pro 3,1[7] running Windows XP SP3 and Java 1.6.20.

4.1 SME²

The Semantic Web Service Matchmaker Evaluation Environment[8] (SME²) is a Java-based tool developed at the German Research Center for Artificial Intelligence (DFKI) and is used during the Semantic Service Selection (S3) contest to evaluate and compare the performance of matchmakers. It supports the addition of matchmakers and test collections as a plug-in; currently collections for OWL-S and SAWSDL and the matchmakers written by the DFKI are publicly available.

SME² evaluates the matchers' performance with standard performance measures of information retrieval [11]:

– Precision: Fraction of the retrieved results, which are relevant
– Recall: Fraction of all relevant documents, which are retrieved
– Fallout: Fraction of retrieved irrelevant documents.
– F-Measure: Weighted harmonic mean of Precision and Recall.

Furthermore, the average precision for each query is computed and other statistics like execution time, query response time and memory consumption are recorded.

We use SAWSDL-TC[9], which is supplied with SME², as test collection for our matchmaker. It consists of 894 service offers and 26 service requests. For each request the relevant service offers are specified, allowing the calculation of the service measures mentioned above. The services are annotated with concepts from several different ontologies from different domains, ranging from military over education to health care and food.

The ontology with the extracted annotations of 894 services from SAWSDL-TC contains 2149 named and 3261 anonymous classes (SubclassOf axioms). This ontology is then classified by the matcher using a description logic reasoner suitable for OWL.

4.2 Comparison of Retrieval Performance

This section compares the implemented matcher's retrieval performance to that of SAWSDL-MX2, which is, like SME², developed at the DFKI. It implements several possibilities for calculating matches [6]:

[7] Two 2.8 GHz Intel Xeon Quad-Core processors, 2GB RAM
[8] `http://www.semwebcentral.org/projects/sme2/`
[9] `http://projects.semwebcentral.org/projects/sawsdl-tc/`

- Logic-based: computes matches based on the semantic annotations of input and output parameters
- Text similarity: uses standard information retrieval methods for finding similar descriptions
- Structural similarity: compares the structure of services: interfaces, bindings, number of parameter, etc.

Additionally, SAWSDL-MX2 implements a machine learning feature to support the weighting of each of this properties, when it ranks the result list.

Table 1 gives an overview of the matchers' overall performance. The average precision is the average of the average precision for every query. The query response time is the time needed on average to execute a single query. The number of executed queries is the number of requests issued to the reasoner. This number is only available for our implementation.

	Simple	Taxonomy	SAWSDL-MX2
∅ precision	65.52%	73.97%	70.50%
∅ query response time	1.06sec	15.15sec	3.18sec
# executed queries	153	2487	n/a

Table 1. Overall performance of Simple, Taxonomy and SAWSDL-MX2

Overall Precision and Recall First, we will compare the overall precision and recall of our matcher utilizing the Simple and Taxonomy strategies and SAWSDL-MX2 (Fig. 3).

For every level of recall, Taxonomy achieves a higher precision than Simple. Because Taxonomy returns the same services as results as Simple does, and only refines the ranking produced by it, both strategies deliver an almost identical performance at lowest and highest recall levels. Between those, Taxonomy shows the advantage of the fine grained approximation of concepts, achieving a precision which is up to 30% higher than Simple's. The Taxonomy approach also has the highest average precision of all 3 matchers. However, its precision is worse than that of SAWSDL-MX2 at the first quarter of the result list, where SAWSDL-MX2 is able to reach a precision of almost 100%. Overall, the precision of both our implementations decreases slower than SAWSDL-MX2's, which drops below Simple's precision towards the end of the result list.

Overall Recall and Fallout The recall/fallout diagram (Fig. 4) shows the same tendencies as the overall recall and precision, and again emphasizes the good proportion between precision and recall of the Taxonomy strategy. It returns less false positives than SAWSDL-MX2, which produces almost twice as much at full recall.

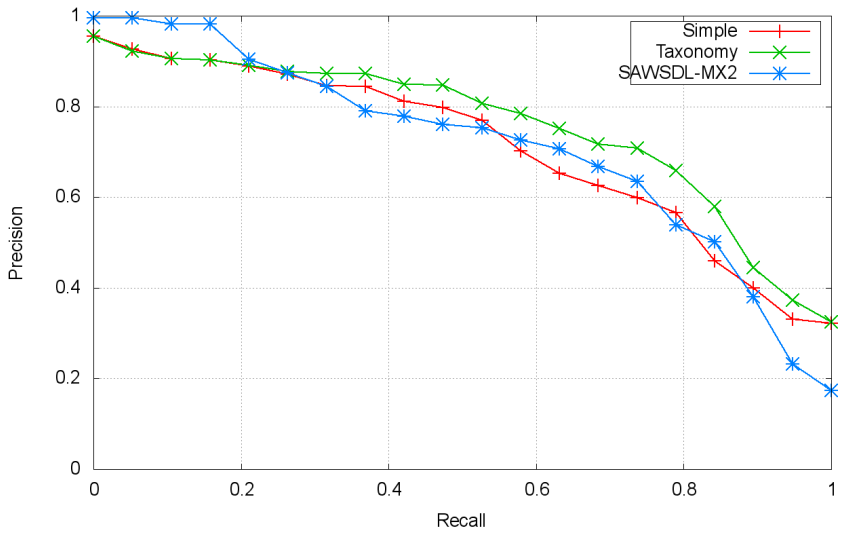

Fig. 3. Recall/Precision of SIMPLE, TAXONOMY and SAWSDL-MX2

Influence of the Intersection Query Figure 5 shows the influence of the intersection query on the performance of the SIMPLE and TAXONOMY approach. The recall/precision curves for TAXONOMY with and without the intersection query show only a slight difference. For SIMPLE, however, the intersection query increases the precision significantly; by up to 10% at full recall. The beginning of the curves is identical between the variants with and without intersection query, because it only adds offers to the bottom of the result list, as mentioned before, and does not change the ranking of previous results.

4.3 Response Time to User Requests

Besides the retrieval performance, for a real world usage of the matcher, it is also important to deliver results in a reasonable amount of time [13].

query response time	∅	median	std dev	cv
SIMPLE	1054ms	892ms	438ms	0.42
TAXONOMY	15204ms	7292ms	25863ms	1.70
SAWSDL-MX2	3178ms	2313ms	1541ms	0.48

Table 2. Average and median value, standard deviation (std dev) and coefficient of variation (cv) for the query response time of some variants.

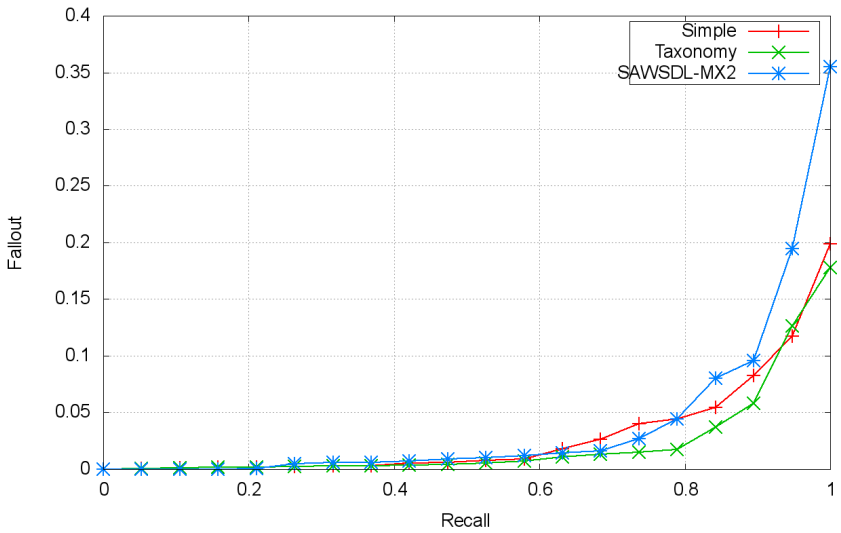

Fig. 4. Recall/Fallout of SIMPLE, TAXONOMY and SAWSDL-MX2

Table 2 shows the average value, median value, standard deviation and coefficient of variation of the response times the matchers need for evaluating a single request. These numbers show that a higher precision comes at the cost of longer wait time for the users. SIMPLE is quite fast with about one second on average and SAWSDL-MX2 is not much slower, taking around three seconds to deliver results; TAXONOMY uses a lot more time: 15 seconds on average.

Figure 6 and the median value, standard deviation and coefficient of variation show, that SIMPLE and SAWSDL-MX have quite steady query response times, which do not vary wildly. For TAXONOMY, however, the response times show huge differences. Fortunately, most times are lower than the average, as indicated by the median value only being half of the average precision. For some queries, TAXONOMY is faster than SAWSDL-MX, for others, there are some outliers for which execution took around two minutes.

In general, SIMPLE is faster than SAWSDL-MX, which is in turn faster than TAXONOMY, what corresponds directly with their average precision.

Optimization of Query Response Time The response times of SIMPLE and TAXONOMY are already improved through the optimization described in Section 3.2. Without it, TAXONOMY would need 20 seconds on average to answer a request (Table 3). The benefits for SIMPLE are minimal, as it only produces a small query graph. However, TAXONOMY issues almost one third less queries to the reasoner, saving as much time for a request.

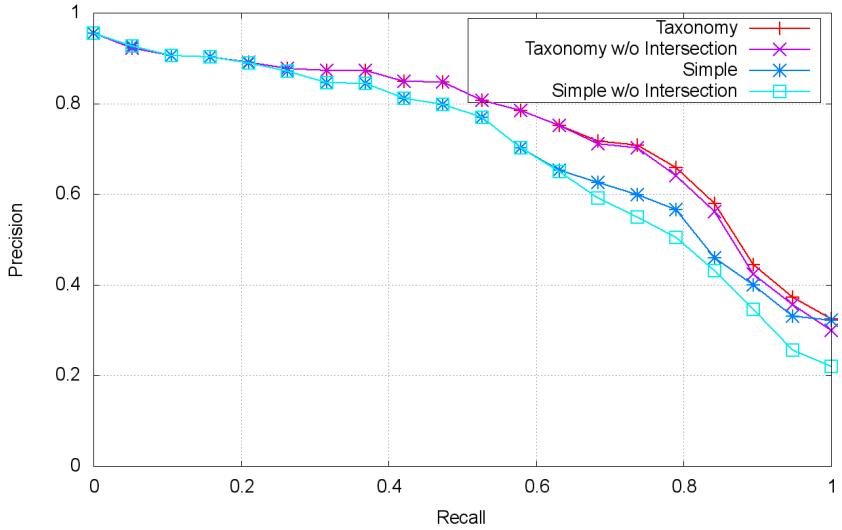

Fig. 5. Recall/Fallout of SIMPLE, TAXONOMY and SAWSDL-MX2

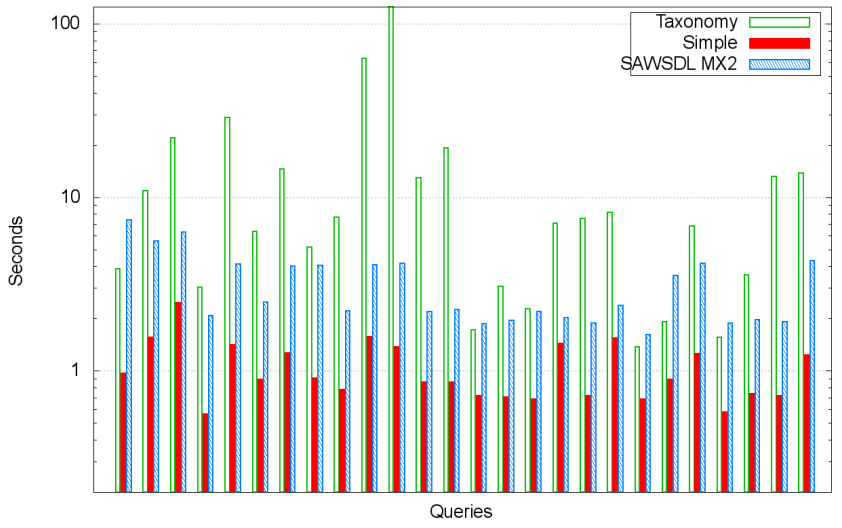

Fig. 6. Query response time of WSA Simple, WSA Taxonomy and SAWSDL-MX2

	Simple	Taxonomy
# created queries	153	3680
# executed queries	130	2487
∅ savings/request	15%	32%
∅ response time savings	3%	33%

Table 3. Number of created and executed queries, and savings through optimizations

5 Conclusion

We presented our implementation of a purely logic-based approximate web service matcher, which is based on approximate subsumption. The results of our evaluation are promising, especially for our goal to increase recall and still maintain a high precision. Considering the recall, our approach even performs better then all other matchers contending in the S3, and still achieves competitive precision. Our extensions aimed at decreasing query response time have generally helped to achieve a reasonable speed for our matcher, despite the additional executed queries. However, in some special cases our implementation still needs too much time for executing all approximations.

Future goals are to improve query performance, where we will consider several possibilities: As the approximated queries are independent from each other, they could be executed concurrently, but unfortunately parallelization is supported poorly by current reasoners. Another possibility to improve the perceived performance of the matcher is the implementation of an anytime behaviour, to show the user some first results and then refine or extend the list in the background.

Long term goals are the integration of user preferences in the approximation and ranking process, e.g., black or white lists defining which concepts should or should not be approximated. Furthermore, the matcher could create proposals for combining several services, which together can fulfill the users request. To improve the matchers performance in a heterogeneous environment, where service are annotated with concepts from different, not formally related ontologies, an ontology alignment [3] process could improve retrieval performance.

References

1. E. Della Valle and D. Cerizza. Cocoon glue: a prototype of "wsmo" discovery engine for the healthcare field. In *Proceedings of the WIW 2005 Workshop on WSMO Implementations, Innsbruck, Austria, June 6-7*, volume 134 of *CEUR-WS*, pages 1–12, 2005.
2. T. Di Noia, E. Di Sciascio, F.M. Donini, and M. Mongiello. Abductive matchmaking using description logics. In *INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE*, volume 18, pages 337–342. Citeseer, 2003.
3. J. Euzenat and P. Shvaiko. *Ontology matching*. Springer-Verlag New York Inc, 2007.

4. P. Groot, H. Stuckenschmidt, and H. Wache. Approximating description logic classification for semantic web reasoning. In *European Semantic Web Conference (ESWC), Heraklion, Greece, May 29 - June 1*, volume Volume 3532 of *Lecture Notes in Computer Science (LNCS)*, pages 318–332. Springer, 2005.
5. W3C OWL Working Group. OWL 2 web ontology language document overview. Technical report, W3C, October 2009. http://www.w3.org/TR/2009/REC-owl2-overview-20091027/.
6. M. Klusch and P. Kapahnke. Semantic web service selection with "sawsdl-mx". In Rubén Lara Hernandez, Tommaso Di Noia, and Ioan Toma, editors, *Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web (SMRR), Karlsruhe, Germany, October 27*, volume 416 of *CEUR Workshop Proceedings*, 2008.
7. M. Klusch and P. Kapahnke. isem: Approximated reasoning for adaptive hybrid selection of semantic services. In *Proceedings of 4th IEEE International Conference on Semantic Computing (ICSC)*, 2010.
8. M. Klusch, P. Kapahnke, and I. Zinnikus. "sawsdl-mx2": A machine-learning approach for integrating semantic web service matchmaking variants. In *ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services*, pages 335–342, Washington, DC, USA, 2009. IEEE Computer Society.
9. J. Kopecky, T. Vitvar, C. Bournez, and J. Farrell. "sawsdl: Semantic annotations for wsdl and xml schema". *IEEE Internet Computing*, November / December:60 – 67, 2007.
10. H. Lausen, A. Polleres, and D. Roman. Web service modeling ontology (wsmo). W3C member submission, W3C, June 2005. http://www.w3.org/Submission/2005/SUBM-WSMO-20050603/.
11. C.D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*. Cambridge Univ Pr, 2008.
12. D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. Owl-s: Semantic markup for web services. W3C member submission, W3C, November 2004. http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/.
13. R.B. Miller. Response time in man-computer conversational transactions. In *AFIPS Joint Computer Conferences 1968, San Francisco, CA, USA, December 9-11*, pages 267–277. ACM, 1968.
14. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Importing the semantic web in "uddi". In *Web services, E-Business, and the Semantic Web, (WES) CAiSE-Workshop, Toronto, Canada, May 27-28*, volume 2512 of *LNCS*, pages 815–821. Springer, 2002.
15. P. Plebani and B. Pernici. Urbe: Web service retrieval based on similarity evaluation. *IEEE Transactions on Knowledge and Data Engineering*, 21:1629 – 1642, 2009.
16. M. Schaerf and M. Cadoli. Tractable reasoning via approximation. *Artificial Intelligence*, 74(2):249–310, 1995.
17. S. Schlobach, E. Blaauw, M. El Kebir, A. Ten Teije, F. Van Harmelen, S. Bortoli, et al. Anytime classification by ontology approximation. In Ruzica Piskac, Frank van Harmelen, and Ning Zhong, editors, *New forms of reasoning for the Semantic Web*, volume 291 of *CEUR-WS*, pages 60–74, 2007.
18. S. Schulte, U. Lampe, J. Eckert, and R. Steinmetz. "log4sws.kom": Self-adapting semantic web service discovery for "sawsdl". In *IEEE Congress on Services, Miami, FL, USA, July 5-10*, pages 511–518. IEEE Computer Society, 2010.

19. H. Stuckenschmidt. Partial matchmaking using approximate subsumption. In *Proceedings of the 22nd Conference on Artificial Intelligence (AAAI-07)*, pages 1459–1464, Vancouver, British Columbia, Canada, July 2007. AAAI Press.
20. H. Stuckenschmidt and M. Kolb. Partial matchmaking for complex product- and service descriptions. In *Proceedings of Multikonferenz Wirtschaftsinformatik (MKWI 2008), Special Track on Semantic Web Technology in Business Information Systems, Munich, Germany, February 26-28*, 2008.
21. S.S. Yau and J. Liu. Functionality-based service matchmaking for service-oriented architecture. In *International Symposium on Autonomous Decentralized Systems (ISADS'07), Sedona, USA, March 21-23*. IEEE Computer society, 2007.
22. A.M. Zaremski and J.M. Wing. Signature matching: a tool for using software libraries. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 4(2):170, 1995.
23. J. Zhong, H. Zhu, J. Li, and Y. Yu. Conceptual graph matching for semantic search. In *International Conference on Conceptual Structures: Integration and Interfaces (ICCS), Borovets, Bulgaria, July 15-19*, volume 2393 of *LNCS*, pages 92–106, 2002.

SPARQL Endpoints as Front-end for Multimedia Processing Algorithms

Ruben Verborgh[1], Davy Van Deursen[1], Jos De Roo[2], Erik Mannens[1], and Rik Van de Walle[1]

[1]Ghent University – IBBT, ELIS – Multimedia Lab
Gaston Crommenlaan 8 bus 201, B-9050 Ledeberg-Ghent, Belgium
{ruben.verborgh,davy.vandeursen,erik.mannens,rik.vandewalle}@ugent.be
http://multimedialab.elis.ugent.be/
[2]Agfa Healthcare – ACA
Moutstraat 100, B-9000 Ghent, Belgium
jos.deroo@agfa.be

Abstract. Multimedia processing algorithms in various domains often communicate with different proprietary protocols and representation formats, lacking a rigorous description. Furthermore, their capabilities and requirements are usually described by an informal textual description. While sufficient for manual and batch execution, these descriptions lack the expressiveness to enable automated invocation. The discovery of relevant algorithms and automated information exchange between them is virtually impossible. This paper presents a mechanism for accessing algorithms as SPARQL endpoints, which provides a formal protocol and representation format. Additionally, we describe algorithms using OWL-S, enabling automated discovery and information exchange. As a result, these algorithms can be applied autonomously in varying contexts. We illustrate our approach by a use case in which algorithms are employed automatically to solve a complex multimedia annotation problem.

Keywords: multimedia processing, N3Logic, OWL-S, Semantic Web, SPARQL

1 Introduction

In the last decade, the world has witnessed an unprecedented growth of multimedia data production and consumption. In this context, metadata, which are generally defined as 'data about data', play a crucial role. Metadata enable the effective organization, access, and interpretation of multimedia content. Therefore, they play an increasingly important role in bringing order to the growing amount of available multimedia content. However, the lack of availability of many kinds of metadata forms the main obstacle in many multimedia applications. For professionals, metadata generation adds to the production cost because annotating requires tedious manual work. Amateur producers do not possess the necessary skills to provide metadata formally. Clearly, we need automated tools to assist with this cumbersome task.

While automated feature extraction algorithms exist, they are prone to errors and lack an intelligent view on the object under annotation. Currently, people select the appropriate algorithms and parameters for a specific problem and initiate the process. As a result, manual intervention is required when a proposed solution does not meet the requirements. This approach lacks actual cooperation between different algorithms, which are unaware of their own abilities and limitations.

Suppose we have a series of photographs that need to be provided with a textual description automatically. We dispose of the following multimedia processing algorithm implementations:

- *description generation:* creates a description based on image elements;
- *face detection:* detects face regions in an image;
- *face recognition:* recognizes a face in a region;
- *text recognition:* translates bitmap text into a string.

The above list encompasses all required algorithms to handle the task: we should detect faces, recognize them, translate text, and finally generate a description based on the found faces and text. While humans can find and execute the necessary steps for this task, an automated platform cannot, because:

(1) it does not know how to **select algorithms**;
(2) it cannot combine these algorithms into a **solution plan**;
(3) each algorithm has an **informally specified format** for input and output.

Clearly, the first problem arises because the algorithms lack a formal description of their capabilities and requirements. A textual explanation of what the algorithm performs, is insufficient to decide automatically whether it fits a certain purpose. Problems 1 and 2 are closely related, since the creation of a plan also involves a formal description of the desired solution. Based on this description and that of the processing algorithms, an automated planner can devise a solution plan.

The third problem occurs because algorithms usually have a proprietary interaction scheme. More specifically, different algorithms use different ways to specify input and output parameters. Further, these algorithms implement different (standardized or proprietary) multimedia content description schemes to provide their results. This makes it impossible to interact with these algorithms automatically. In addition, the required parameters and the effect of these parameters are often described informally.

In this paper, we address these shortcomings and aim to enable automated algorithm discovery and execution. We propose RDF as input and output representation format. We describe how to transform multimedia processing algorithms into SPARQL endpoints to formalize their interaction. Therefore, we introduce a query prototype suitable for algorithm invocations. We describe algorithm capabilities and requirements using OWL-S, adding support for SPARQL groundings. We zoom in on the expression of various relations between input and output, introducing N3 expressions into OWL-S. Finally, we discuss how to create concrete

algorithm implementations as SPARQL endpoints, providing a number of implementation details. These contributions pave the way for complex multimedia processing scenarios.

2 Algorithm Interaction

2.1 Representation format

The tasks performed by processing algorithms span a very wide range, so finding a comprehensive interaction model poses a challenge. Input and output parameters must be precisely specified, at the same time leaving room for new, previously unused parameters. The algorithm interaction should be:

- **interoperable:** enabling communication with various components, regardless of low-level details such as operating system and programming language;
- **flexible:** handling a variety of inputs and outputs;
- **formal:** able to communicate in a formal way with well-defined semantics so machines are able to interpret the information.

The *Resource Description Framework* (RDF, [13]) is a data format fitting the above requirements well. More specifically, RDF provides a means to represent knowledge in a formal, machine-understandable way [4]. Further, vocabularies and ontologies can be expressed with *RDF Schema* [5] and the *Web Ontology Language* (OWL, [17]), which are both built on top of RDF. We can define or reuse ontologies for the input and output vocabulary of a specific algorithm. For instance, formalized versions of multimedia content description standards (e.g., *COMM* representing a formal way to express *MPEG-7* [1]) constitute one possibility. Integrated semantics facilitate the automated interpretation and exchangeability of results, countering the issues with proprietary formats.

2.2 Communication protocol

An algorithm communication protocol should meet these design requirements:

- **flexible:** able to specify variations on input and outputs;
- **distributed:** provide access to algorithms located on different machines;
- **transparent:** exhibit identical behavior, regardless of varying properties such as physical location and technological differences.

The above requirements hint at a *Service-Oriented Architecture* (SOA, [19]). Therefore, we consider algorithms as Web services. A classic Web service communication protocol choice is the *Simple Object Access Protocol* (SOAP, [11]). However, this protocol is rather verbose and does not account for sufficient flexibility: input and output parameters are passed in a rigid structure that does not allow variations. Given the use of RDF and the definition of algorithms as information-generating entities, our approach is to implement algorithms as *SPARQL* endpoints.

The *SPARQL Protocol and RDF Query Language* [7, 20] is used to retrieve information from semantic data sources, traditionally static databases of RDF content. However, an algorithm is essentially a data source, which does not necessarily offer predefined information, but rather creates new information in a demand-driven way. By using SPARQL as Web service communication protocol, we obtain the following features:

- Web services can be invoked using formally defined input and output parameters;
- the input and output is represented directly in RDF;
- Web services wrapping multimedia algorithms can be seamlessly integrated with other endpoints in the Semantic Web.

There are clear advantages over using SPARQL endpoints instead of passing RDF literals through classic technologies such as SOAP:

- the overhead and verbosity of SOAP is absent;
- the endpoint can check the presence of necessary parameters, as it is able to parse and understand RDF. RDF literals, on the other hand, would be treated as plain text and syntactic or semantic errors would go unnoticed.

Additionally, the cost of maintaining an endpoint is outweighed by the possibility to do post-processing of the returned results directly in SPARQL:

- selection of relevant output values;
- structuring the output values to fit the application format.

In the next subsections, we present a number of querying techniques for on-demand data sources such as multimedia processing algorithms.

Classic SPARQL queries The concept behind SPARQL is similar to that of other query languages: to retrieve a specific view on a larger data collection. The query mechanism validates the data against the constraints in the `WHERE` clause, conditioning the output. This aligns with the way we think about the underlying RDF database: the `WHERE` clause is a *filter* that retains all triples matching the specified template. A first approach to query algorithms would be the exact same way we query data sources.

Suppose we have an algorithm mixing two colors. We begin by defining formal characteristics for the input and output parameters:

- **Input:** a number of `Color` entities with a `hasColorName` property;
- **Output:** a `Color` that has a `hasColorName` property, and a `isMixOf` property with the list of input colors as value.

If we want to ask the result of mixing red and yellow, we could invoke the algorithm using the query in Listing 1. Note that we can choose `SELECT` queries (to retrieve individual values) or `CONSTRUCT` queries (to retrieve an entire RDF graph).

```
PREFIX c: <http://example.org/ontologies/Colors#>
CONSTRUCT { ?color c:hasColorCode ?colorCode. }
WHERE {
  c:Red a c:Color;
         c:hasColorCode "#FF0000".
  c:Yellow a c:Color;
          c:hasColorCode "#FFFF00".
  ?color a c:Color;
        c:hasColorCode ?colorCode;
        c:isMixOf (c:Red c:Yellow).
}
```

Listing 1. Color mixer invocation with classic SPARQL

As opposed to querying data sources, invoking an algorithm is not as such about filtering, but about retrieving the outputs of a process on the inputs. Although it would be possible to build an algorithm this way, there are several drawbacks to this approach:

- **missing indication** that the query is an algorithm invocation;
- **unclear distinction between input and output** in the `WHERE` clause;
- **conceptual mismatch** by forcing the inputs into output conditions.

Clearly, we need a more advanced query which makes the invocation explicit, while strictly adhering to the SPARQL standard.

SPARQL queries with explicit invocation We consider the algorithm as a virtual data source and refer to each invocation by a dedicated `Request` entity. The input and output parameters of the invocation are are values of the `sr:input` and `sr:output` properties. The inputs are generally threated as known values; the outputs are usually unbound variables. Inside the SPARQL query, we can refer to this entity and its associated properties. This enables us to invoke an algorithm while keeping all the benefits of a SPARQL query and its declarativeness.

The query in Listing 2 shows how this technique overcomes previous weaknesses, clearly indicating the invocation, its parameters, and their direction.[1]

The algorithm maintains an entity corresponding to the `Request` in the `WHERE` clause, containing the same information as the entity in the query. The algorithm executes its task on the entity `input` values; its computed output gets bound to the entity `output` values. The resulting graph containing the `Request` entity is subsequently queried in its entirety. We highlight the fact that this `Request` entity is purely virtual: it is only accessible during the execution of the query and remains invisible to other clients accessing the endpoint at the same instant.

[1] The ontology can be found at `http://ninsuna.elis.ugent.be/ontologies/arseco/sparqlrequest#`

```
PREFIX c: <http://example.org/ontologies/Colors#>
PREFIX sr:
      <http://ninsuna.elis.ugent.be/ontologies/arseco/sparqlrequest#>
CONSTRUCT { :mixedColor c:hasColorCode ?colorCode. }
WHERE {
  [a sr:Request;
   sr:method "MixColor";
   sr:input [a c:Color;
             c:hasColorCode "#FF0000"],
            [a c:Color;
             c:hasColorCode "#FFFF00"];
   sr:output [a c:Color;
              c:hasColorCode ?colorCode]]
}
```

Listing 2. Algorithm invocation with a `Request` entity

SPARQL queries with named parameters To generalize the query prototype to a broader class of algorithms, it is necessary to provide parameter identification for input and output. This is done by setting the value of the `input` and `output` to a `ParameterBinding` entity, instead of solely the actual value. An example is applying a mask to an image (Listing 3).

For simplicity, the query can be abbreviated by removing parts of the `WHERE` clause that are known in advance. A request will always have type `Request`, and a parameter will always be of type `ParameterBinding`, so these statements can be omitted. The method name is mostly unnecessary, because algorithms usually have a single task that is consequently identified the moment they receives a query. However, if a query is viewed out of its usual context, these items clarify its intended meaning.

SPARQL queries with complex parameters In case the input or output parameters are complex, it is possible to specify either of them as *Notation3* (N3, [2]) strings, which are serialized representations of RDF graphs. They may be required when the structure of the output RDF graph is unknown in advance, making it impossible to reserve sufficient variables for retrieval. Furthermore, complex graphs – such as those containing reified statements – are also better represented by N3 strings. The N3 specification has been extended with this functionality by means of the `log` namespace.

The query in Listing 4, segmenting an image, illustrates complex output. We do not know beforehand how deeply the segments and thus the result graph will be nested. Since SPARQL does not provide facilities to capture *all* descending nodes, we `SELECT` an N3 string to store the result, which must be parsed afterwards to obtain the corresponding RDF graph. For the sake of example, we also supply the input parameter as an N3 string. N3 strings also offer the freedom to return more expressive values. Consider an algorithm that recognizes words

```
PREFIX sr:
      <http://ninsuna.elis.ugent.be/ontologies/arseco/sparqlrequest#>
CONSTRUCT { <source.jpg> :hasMaskedImage ?maskedImage }
WHERE {
  [a sr:Request;
   sr:method "ApplyMask";
   sr:input [a sr:ParameterBinding;
             sr:bindsParameter "source";
             sr:boundTo <source.jpg>],
            [a sr:ParameterBinding;
             sr:bindsParameter "mask";
             sr:boundTo <mask.jpg>];
   sr:output [a sr:ParameterBinding;
              sr:bindsParameter "masked";
              sr:boundTo ?maskedImage]]
}
```

Listing 3. Algorithm invocation with named parameters

in an audio fragment. In the strict case, the return value is ill-defined when uncertainty between two alternatives arises. RDF enables us to express this uncertainty semantically as shown in Listing 5, where is stated that a certain audio fragment contains the word "summer" *or* the word "sombre".

2.3 Algorithm conversion

Having defined a formal model for invocations, we now direct our attention to a conversion method from algorithm to SPARQL endpoint. When designing an algorithm, an author arrives at a point where the output format must be decided. Often, a proprietary format is created, accompanied by an informal description. In this case, it would be convenient to choose RDF as underlying data model since this gives access to an existing formal model, compliant with other information in the Semantic Web. Should the author proceed with another model – proprietary or standardized – then an adapter needs to be written to convert the input and output into RDF.

As such, algorithms communicate entirely using RDF. A SPARQL query engine, surrounding the program, processes the virtual `Request` entity, turning the algorithm into a SPARQL endpoint. This process is visualized in Fig. 1. The RDF inputs are extracted from the query (1) and passed to the algorithm (2), which generates RDF output (3). Input and output form the completed `Request` entity (4). The query is executed on this entity (5), yielding the final result. The latter implies that SPARQL is not only used to simply pass input and output parameters, but can also be used to apply formalized restrictions on input and output parameters and the final result.

```
PREFIX sr:
      <http://ninsuna.elis.ugent.be/ontologies/arseco/sparqlrequest#>
PREFIX log: <http://www.w3.org/2000/10/swap/log#>
SELECT ?segmentsN3
WHERE {
  [a sr:Request;
   sr:method "SegmentImage";
   sr:input [a log:Formula;
             log:n3String "<image.jpg> a :Image."];
   sr:output [a log:Formula;
              log:n3String ?segmentsN3]]
}
```

Listing 4. Algorithm invocation with complex parameters

```
@prefix e: <http://eulersharp.sourceforge.net/2003/03swap/log-rules#>.
({<speech.wav#t=5.38,5.71> :contains "summer".}
 {<speech.wav#t=5.38,5.71> :contains "sombre".}) e:disjunction [a e:T].
```

Listing 5. Expressing uncertainty in algorithm output

3 Capability and Requirements Description

3.1 Description method

Since SPARQL endpoints are in fact Web services, we can describe them as such. A common RDF-compliant specification for semantic Web service descriptions is *OWL-S* [16], which consists of a three-part paradigm:

- **service profile:** a limited description of the service's capabilities and requirements;
- **service model:** service usage modalities and a more in-depth description of its capabilities and requirements, suitable for service composition;
- **service grounding:** technical details regarding communication with the service.

Other possibilities for service descriptions include WSMO [15]. We chose OWL-S because its definition of the process model is more mature [14], but the concepts are transferable to WSMO. In the next subsections, we elucidate these different parts through an example algorithm that recognizes a face in an image region:

- *input:* a region which depicts a face;
- *output:* the recognized face and the depicted person.

This informal description leaves room for interpretation and should be formalized. If the algorithm author already formalized the input and output parameter model in RDF, we can copy this into the service description. However, it will

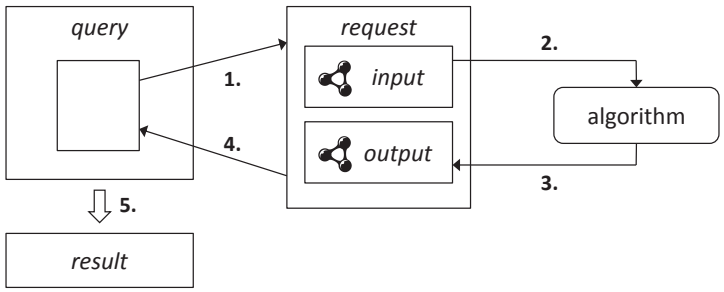

Fig. 1. Algorithm invocation process

```
@prefix Profile: <http://www.daml.org/services/owl-s/1.1/Profile.owl#>.
:FaceRecognitionProfile a Profile:Profile;
                        Profile:hasInput :RegionInput;
                        Profile:hasOutput :FaceOutput;
                        Profile:hasOutput :PersonOutput;
                        Profile:has_process :FaceRecognitionProcess.
```

Listing 6. Algorithm profile description

often be more convenient to create the formal service description first, deciding which RDF classes to use for input and output, and then use this description to implement the actual RDF parameters of the algorithm. This corresponds to the *design by contract* methodology in software engineering [18].

3.2 Service profile

The *profile* part is primarily meant for human reading or rendering purposes. Therefore, it contains some basic information (useful for human interpretation), reused from our model that we will define later on (Listing 6).

3.3 Service model

The model can be described as a process, containing detailed information about all parameters. We will start with a basic description (Listing 7), extending it as inadequacies come to light.

Although this profile seems correct on the surface, it does not convey all intended semantics for a reliable description of a face recognition activity. Consider an algorithm that, regardless of the input it receives, always returns the exact same predefined person in the `PersonOutput` parameter. This algorithm does not recognize faces, yet it fully complies with the description of the example above. Also, there is no guarantee whatsoever that the region in `RegionInput`

```
@prefix Process: <http://www.daml.org/services/owl-s/1.1/Process.owl#>.
:FaceRecognitionProcess a Process:AtomicProcess;
                        Process:hasInput :RegionInput;
                        Process:hasOutput :FaceOutput;
                        Process:hasOutput :PersonOutput.
:RegionInput a Process:Input;
             Process:parameterType "http://example.org/Images#Region".
:FaceOutput a Process:Output;
            Process:parameterType "http://example.org/Faces#Face".
:PersonOutput a Process:Output;
              Process:parameterType "http://example.org/Persons#Person".
```

Listing 7. Basic algorithm process description

actually contains a face; it could depict anything or nothing at all. This means that even an actual face detection algorithm could fail to return a correct result. To obtain a process description that fits the algorithm, we need to correct the following problems:

- the input is not guaranteed to contain a face
  ⇒ *the* ***input constraints*** *must be specified rigorously;*
- the face in the input is not necessarily that of the person in the output
  ⇒ *we require a* ***semantic relation*** *between input and output parameters.*

We can capture these semantics by using preconditions and postconditions.

Preconditions OWL-S supports the use of preconditions to enforce input constraints that go beyond RDF types. These conditions are typically expressed in languages such as *KIF* [9] or *SWRL* [12]. We have opted to use N3 expressions, which are very powerful due to the possibility of more sophisticated built-in functions [3] and the existence of advanced reasoners such as Euler [8].

Therefore, we needed to extend the `Expression` ontology to include support for N3 expressions and conditions, resulting in the `N3Expression` ontology.[2] Our example description is supplemented with preconditions in Listing 8.

Input and output parameters are referred to by *parameter variables* whose name is the last segment of the parameter URI, lowercasing the first letter. As a result, the parameter variable `?regionInput` refers to the parameter named `http://example.org/FaceDetection#RegionInput`. In case of ambiguity, the description document can provide parameter name aliases. This technique is similar to that of SWRL variables in OWL-S. However, a rigorous mechanism that links parameters to variable names should be created. This is left as future work.

Also note the introduction of custom variables (`face`) in the precondition. They are ordinary variables that remain bound in the postconditions, where they can be used together with input and output variables.

[2] `http://ninsuna.elis.ugent.be/ontologies/arseco/n3expression#`

```
@prefix Process: <http://www.daml.org/services/owl-s/1.1/Process.owl#>.
@prefix Expression:
      <http://www.daml.org/services/owl-s/1.1/generic/Expression.owl#>.
@prefix N3Expression:
      <http://ninsuna.elis.ugent.be/ontologies/arseco/n3expression#>.
:FaceRecognitionProcess Process:hasPrecondition :RegionPrecondition.
:RegionPrecondition a N3Expression:N3-Expression;
                    Expression:expressionBody
  "?regionInput <http://example.org/Images#regionDepicts> ?face.
   ?face a <http://example.org/Faces#Face>.".
```

Listing 8. Algorithm process description preconditions

```
@prefix Process: <http://www.daml.org/services/owl-s/1.1/Process.owl#>.
@prefix Expression:
      <http://www.daml.org/services/owl-s/1.1/generic/Expression.owl#>.
@prefix N3Expression:
      <http://ninsuna.elis.ugent.be/ontologies/arseco/n3expression#>.
:FaceRecognitionProcess Process:hasResult :FaceRecognitionResult.
:FaceRecognitionResult a Process:Result;
                       Process:hasEffect :FaceRecognitionEffect.
:FaceRecognitionEffect a N3Expression:N3-Expression;
                       Expression:expressionBody
  "?regionInput <http://example.org/Images#regionDepicts> ?faceOutput.
   ?faceOutput <http://example.org/Faces#isFaceOf> ?personOutput.".
```

Listing 9. Algorithm process description postconditions

Postconditions In a similar fashion, relations between input and output parameters can be expressed in the N3 format. OWL-S terminology defines postconditions as effects of a service result. It is possible to specify multiple results that account for different cases, such as normal and erroneous execution. For simplicity, we will limit the face recognition example to a single result and effect. In Listing 9, we express that the region of the input contains the face of the person in the output.

3.4 Service grounding

The remaining part of the algorithm description details its SPARQL endpoint properties. To access a SPARQL endpoint, we need at least the following details:

- the **URL** of the endpoint;
- the **SPARQL versions** supported;
- the supported **query forms** (e.g., `CONSTRUCT`, `SELECT`, `ASK`, or `DESCRIBE`).

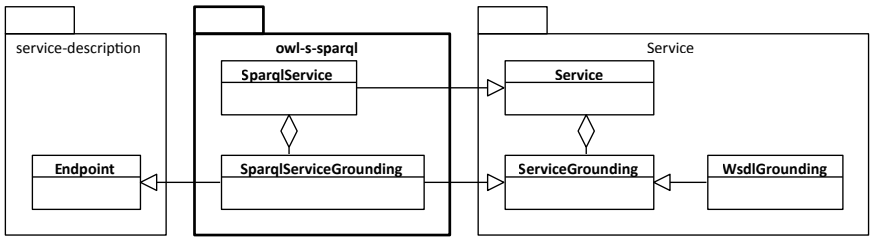

Fig. 2. `owl-s-sparql` ontology overview

```
@prefix sd: <http://www.w3.org/2009/sparql/service-description#>.
@prefix owl-s-sparql:
      <http://ninsuna.elis.ugent.be/ontologies/arseco/owlssparql#>.
:FaceRecognitionGrounding a owl-s-sparql:SparqlServiceGrounding;
  owl-s-sparql:supportsQueryForm owl-s-sparql:SparqlQueryFormConstruct,
                                 owl-s-sparql:SparqlQueryFormSelect;
  owl-s-sparql:supportsSparqlVersion owl-s-sparql:SparqlVersionQuery1_0;
  sd:url <http://example.org/FaceRecognition/sparql>.
```

Listing 10. Algorithm SPARQL grounding description

Unfortunately, OWL-S only provides built-in support for *Web Services Description Language* (WSDL, [6]) service groundings. At the moment of writing, a draft by the W3C SPARQL Working Group on endpoint descriptions exists, but it is not linked to the OWL-S ontologies [23, 24].

An interesting approach is to create a service grounding, compatible with OWL-S, linked to the SPARQL endpoint description. Our `owl-s-sparql` ontology[3] offers this functionality by uniting the definitions of service grounding and SPARQL endpoint. Common grounding properties are provided by OWL-S. SPARQL endpoint specific properties are imported from the service description ontology of the W3C draft. Properties that are currently missing, such as supported SPARQL versions and query forms, are described in `owl-s-sparql`. As depicted in Fig. 2, `SparqlServiceGrounding` is both a `ServiceGrounding` (OWL-S) and an `Endpoint` (service description). A `SparqlService` is a `Service` that has at least one `SparqlServiceGrounding`. Listing 10 shows a possible grounding for the face detection algorithm.

3.5 Service

Finally, the three service parts need to be stitched together in an OWL-S service construct (Listing 11). The user can choose to complement this description with properties that facilitate human interpretation, such as textual descriptions.

[3] `http://ninsuna.elis.ugent.be/ontologies/arseco/owlssparql#`

```
@prefix Service: <http://www.daml.org/services/owl-s/1.1/Service.owl#>.
:FaceRecognitionService a Service:Service;
                        Service:describedBy :FaceRecognitionProcess;
                        Service:presents :FaceRecognitionProfile;
                        Service:supports :FaceRecognitionGrounding.
```

Listing 11. Algorithm service description

4 Implementation

While algorithms can be developed independently, it is convenient to have a software library to abstract common tasks. Two approaches are possible:

- the *wrapper* approach, in which the algorithm is a standalone program, invoked by a configurable wrapper;
- the *toolkit* approach, in which the algorithm directly uses the library for tasks such as RDF parsing and SPARQL querying.

The advantages of the former are that the algorithm does not need to be altered, at the cost of customizing the wrapper sufficiently. More importantly, creating a configuration mechanism that accounts for all possible algorithm interaction schemes is impossible, which is of course the main reason why we introduced SPARQL endpoints. The toolkit approach requires adaptations to the algorithm, improving performance but depending on the existance of an interface between the employed programming language and the toolkit. In general, the wrapper approach – where possible – proves best for existing algorithms and the toolkit approach for algorithms in development, eliminating the need to provide an intermediary communication format.

We implemented toolkits in C++ and C#/.Net, as well as a standalone command line version, to enable interaction with a great variety of programming languages. As an example, we transformed two multimedia processing algorithms into SPARQL endpoints: a face detection and a face recognition algorithm. The face detection algorithm was built from scratch and is based on an implementation of the Viola-Jones face detection algorithm [22]. Hence, in this case, we used the toolkit approach to enable SPARQL communication. For the face recognition algorithm, we used an existing implementation developed by Verstockt et al. [21], which recognizes a face in a well-delineated region. Therefore, we applied the wrapper approach for the face recognition algorithm in order to make the algorithm accessible through SPARQL. Note that the face recognition algorithm adheres to the example description of Section 3. Also note that both algorithms can be deployed for the use case lined out in the introduction of this paper.

Example output for the face detection and recognition algorithms is shown in Listing 12 and Listing 13 respectively. Invoking the face detection algorithm results in one found region depicting a face. The latter is used as input for the face recognition algorithm, which subsequently results in the recognition of the

```
@prefix sr:
      <http://ninsuna.elis.ugent.be/ontologies/arseco/sparqlrequest#>.
@prefix img: <http://example.org/Images#>.
@prefix face: <http://example.org/Faces#>.
_:regionInput sr:bindsParameter "regionInput";
              sr:boundTo <pict.jpg#xywh=45,121,51,51>.
<pict.jpg#xywh=45,121,51,51> img:regionDepicts [a face:Face>].
```

Listing 12. Face detection algorithm example output

```
@prefix sr:
      <http://ninsuna.elis.ugent.be/ontologies/arseco/sparqlrequest#>.
@prefix dbpedia: <http://dbpedia.org/resource/>.
@prefix face: <http://example.org/Faces#>.
_:faceOutput sr:bindsParameter "faceOutput";
             sr:boundTo [face:isFaceOf dbpedia:Johnny_Depp].
_:personOutput sr:bindsParameter "personOutput";
               sr:boundTo dbpedia:Johnny_Depp.
```

Listing 13. Face recognition example output

face of the actor Johnny Depp. As one can see, multimedia processing algorithms are not only accessed in a transparent and formalized way, they can also contain pointers to other information available in the Semantic Web. This way, software agents using these kind of endpoints can transparently query both static linked open data sets and multimedia processing algorithms.

5 Conclusions and Future Work

The use of RDF as input and output representation format for algorithms adds expressiveness with well-defined semantics. By approaching algorithms as SPARQL endpoints, we have a standardized protocol to access them, combined with a flexible query language to demand specific information. This way, we can employ algorithms transparently, regardless of low-level system properties. Algorithm queries consist of classic SPARQL, yet indicating the fact that they are invocations. OWL-S enables us to describe the capabilities and requirements of algorithms rigorously, N3 expressions therein can relate input and output parameters in various ways. An additional ontology lets us describe the SPARQL grounding in OWL-S. We illustrated our approach with two example algorithms.

An interesting direction for future research, is the composition of a plan to solve complex solutions using algorithms. We also require a framework which executes the plan and maintains state between invocations. Additionally, we must elaborate some details such as variable identification in OWL-S descriptions.

It is important that we will apply our approach to several larger multimedia use cases. As outlined in the introduction, information-generating tasks such as metadata annotation prove interesting. Applications such as multimedia adaptation could benefit from service-based algorithms, as suggested in [10]. Eventually, we could extend the approach to general problem solving.

Acknowledgments

The research activities as described in this paper were funded by Ghent University, the Interdisciplinary Institute for Broadband Technology (IBBT), the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT), the Fund for Scientific Research Flanders (FWO-Flanders), and the European Union.

References

1. Arndt, R., Troncy, R., Staab, S., Hardman, L., Vacura, M.: COMM: Designing a Well-Founded Multimedia Ontology for the Web. In: 6th International Semantic Web Conference (ISWC 2007). Busan, Korea (November 2007)
2. Berners-Lee, T., Connolly, D.: Notation3 (N3): A readable RDF syntax. W3C Recommendation (Jan 2009), http://www.w3.org/TeamSubmission/n3/
3. Berners-Lee, T., Connolly, D., Kagal, L., Scharf, Y., Hendler, J.: N3Logic: A logical framework for the World Wide Web. Theory and Practice of Logic Programming 8(3), 249–269 (2008)
4. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American 284(5), 34 (2001)
5. Brickley, D., Guha, R.V.: RDF vocabulary description language 1.0: RDF Schema. W3C Recommendation (Feb 2004), http://www.w3.org/TR/2004/REC-rdf-schema-20040210/
6. Christensen, E., Curbera, F., Greg, M., Weerawarana, S.: Web Services Description Language (WSDL) 1.1. W3C Member Submission (Mar 2001), http://www.w3.org/TR/wsdl
7. Clark, K.G., Feigenbaum, L., Torres, E.: SPARQL protocol for RDF. W3C Recommendation (Jan 2008), http://www.w3.org/TR/rdf-sparql-protocol/
8. De Roo, J.: Euler proof mechanism, http://eulersharp.sourceforge.net/
9. Genereth, M.R.: Knowledge Interchange Format. Draft Proposed American National Standard, http://logic.stanford.edu/kif/dpans.html
10. Geyter, M.D., Soetens, P.: A planning approach to media adaptation within the Semantic Web. In: Distributed Multimedia Systems. pp. 129–134 (2005)
11. Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.J.: SOAP version 1.2 part 1: Messaging framework (second edition). W3C Recommendation (Apr 2007), http://www.w3.org/TR/2007/REC-soap12-part1-20070427/
12. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S.: SWRL: A Semantic Web Rule Language combining OWL and RuleML. W3C Member Submission (May 2004), http://www.w3.org/Submission/SWRL/
13. Klyne, G., Carrol, J.J.: Resource Description Framework (RDF): Concepts and abstract syntax. W3C Recommendation (Feb 2004), http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

14. Lara, R., Roman, D., Polleres, A., Fensel, D.: A conceptual comparison of WSMO and OWL-S. In: ECOWS 2004. LNCS, vol. 3250, pp. 254–269. Springer (2004), http://dx.doi.org/10.1007/978-3-540-30209-4_19
15. Lausen, H., Polleres, A., Roman, D.: Web Service Modeling Ontology (WSMO), howpublished = W3C Member Submission, note = http://www.w3.org/ Submission/WSMO/, day = 3, month = jun, year = 2005
16. Martin, D., Burstein, M., Hobbs, J., Lassila, O.: OWL-S: Semantic markup for web services. W3C Member Submission (Nov 2004), http://www.w3.org/ Submission/OWL-S/
17. McGuinness, D.L., van Harmelen, F.: OWL Web Ontology Language overview. W3C Recommendation (Feb 2004), http://www.w3.org/TR/2004/REC-owl-features-20040210/
18. Meyer, B.: Applying "Design by Contract". IEEE Computer 25(10), 40–51 (1992)
19. Perrey, R., Lycett, M.: Service-Oriented Architecture. IEEE/IPSJ International Symposium on Applications and the Internet Workshops p. 116 (2003)
20. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF. W3C Recommendation (Jan 2008), http://www.w3.org/TR/rdf-sparql-query/
21. Verstockt, S., Van Leuven, S., Van de Walle, R., Dermaut, E., Torelle, S., Gevaert, W.: Actor recognition for interactive querying and automatic annotation in digital video. In: IASTED International conference on Internet and Multimedia Systems and Applications, 13th, Proceedings. pp. 149–155. ACTA Press, Honolulu, HI, USA (2009)
22. Viola, P., Jones, M.J.: Robust real-time face detection. International Journal of Computer Vision 57(2), 137–154 (May 2004)
23. Williams, T.G.: SPARQL 1.1 service description. SPARQL Working Draft (Oct 2009), http://www.w3.org/TR/2009/WD-sparql11-service-description-20091022/
24. Williams, T.G., Mikhailov, I.: SPARQL service description. SPARQL Working Group (2009), http://www.w3.org/2009/sparql/wiki/Feature:ServiceDescriptions

Integrating Semantic Web Services and Matchmaking into ebXML Registry

Stefan Schulte, Melanie Siebenhaar, and Ralf Steinmetz

Multimedia Communications Lab (KOM)
Technische Universität Darmstadt, Germany
schulte@kom.tu-darmstadt.de

Abstract. While the "Universal Description, Discovery and Integration" (UDDI) service registry standard has drawn great attention by the research community, it has not been widely adopted by the software industry. Objections towards UDDI include technical as well as conceptional arguments. Being an official ISO standard and providing a number of features UDDI is missing, "Electronic Business using Extensible Markup Language" (ebXML) Registry could be an adequate alternative for the implementation of service registries and/or repositories. However, little work has been done regarding the integration of Semantic Web Services (SWS) into ebXML Registry.
In this paper, we present a solution extending the ebXML Registry by capabilities to handle and provide SWS. This includes a concept for the integration of SWS into ebXML Registry as well as a prototypical implementation using SAWSDL and the open source framework freebXML.

1 Motivation

One of the primary application areas of SWS is service discovery, which has been a major topic from the very beginning of SWS research [19], [22]. In most scenarios, services are registered at some kind of service catalogue, which can be searched by (potential) service consumers. Apart from proprietary solutions, the SWS research community has mostly deployed UDDI as service registry standard and a multitude of solutions to include SWS into such registries have been proposed (cp. Section 2).

Even though UDDI is still deemed to be one of the key building blocks of service-oriented computing, it suffers from some major drawbacks. While conducting research on query formulation, we had to learn that it is difficult to use UDDI as a starting point for an advanced query formalism applied in SWS discovery [25].

A comparison of some major features provided by ebXML Registry and UDDI, which are important for the selection of a registry standard, are shown in Table 1: First, UDDI provides by default only a registry, where metadata about artifacts is stored. The actual artifacts (e.g., service descriptions making use of the "Web Service Description Language" (WSDL)) are not stored in UDDI. Instead, references to these artifacts are published in the registry. This

Table 1. Comparison of Registry Standards ebXML and UDDI (cp. [3], [28])

Category/Feature	ebXML Registry 3.0	UDDI 3.0
Service description standards	WSDL 1.1	WSDL 1.1
Registry	YES	YES
Repository	YES	NO
Object-oriented information model and API	YES	NO
Extensible information model	YES	NO
User-defined queries	YES	NO
SQL query syntax	YES	NO
XML query syntax	YES	YES
JAXR API	YES	YES

aspect of UDDI has been deemed as a major drawback by the software industry, as it makes it difficult to establish service life cycle management for Service-oriented Architectures[1] (SOA) and therefore, service and SOA governance [28]. In contrast to UDDI, the ebXML Registry provides both, a registry and a corresponding repository. Hence, besides the metadata, also the artifacts themselves are published in ebXML Registry.

Second, UDDI makes use of a relatively flat data model, which cannot be extended, whereas ebXML Registry offers an object-oriented and extensible information model and Application Programming Interface (API). Finally, while both registry standards can be used by utilizing the "Java API for XML Registries" (JAXR), which provides a uniform way for communicating with a registry, search facilities differ as ebXML offers enhanced querying capabilities by providing SQL support and user-defined queries in comparison to UDDI, which is only able to process XML-based queries.

Regarding the integration of an advanced query formalism into a service registry (cp. [25]), this is a major point and has been our main motivation to restrain from making use of UDDI. As there has been little work on the application of ebXML Registry in a generic SWS discovery framework, we have developed our own solution which is presented in this paper.

The remainder of this paper is structured as follows: In Section 2, we will give an overview of SWS integration approaches into registries. Afterwards, we give an overview on ebXML Registry. In Section 4, we introduce a solution to integrate SWS into ebXML Registry and a first implementation including an interface for matchmakers. The paper closes with a conclusion and an outlook on our future work.

[1] `http://www.zdnet.com/blog/service-oriented/ibm-acknowledges-bypassing-uddi-calls-for-new-soa-registry-standard/864`, `http://www.computing.co.uk/vnunet/news/2188598/ibm-calls-soa-discovery`, accessed at 2010-09-04

2 Related Work

The integration of SWS descriptions in service registries has been examined in a multitude of approaches, mostly making use of UDDI as service registry standard. In their seminal work on SWS, Paolucci et al. present the integration of "DARPA agent markup language for services" (DAML-S) profiles in UDDI [22]. The authors propose the mapping of a service profile to UDDI records. Besides the DAML-S/UDDI mapping, an external matchmaker architecture is suggested by the authors, which uses DAML ontologies publicly available on the Web for semantic capability matching. In this approach, it is possible to search for services using UDDI keyword-based search and a capability matching engine, if requests are specified in the DAML-S format. Several authors have proposed enhancements of the work by Paolucci et al., e.g., regarding the usage of UDDI-internal matchmakers [1], the application of the "Web Ontology Language for Web Services" (OWL-S) or semantically enhanced WSDL instead of DAML-S [26], [27], or the integration of functionalities enabling the usage of semantic search in UDDI on the client-side instead of altering the UDDI implementation [17].

There are further approaches to integrate SWS into service registries in general SWS frameworks, with METEOR-S [29] and the "Web Service Modeling eXecution Environment" (WSMX) being prominent examples. However, in both approaches the actual registry is more a means to an end than in the focus of the work. In an early "Web Service Modeling Ontology" (WSMO) Registry Working Draft, UDDI was intended to provide registry functionalities [12]. However, for WSMX, which is the reference implementation of WSMO, no further information is given if a particular registry standard has been applied or not. In fact, WSMX's *Resource Manager* is an internal registry [11]; furthermore, it is stated that an ebXML- or UDDI-based registry *could* be used for WSMX data persistence [5]. More recently, Kourtesis et al. have proposed a combination of SAWSDL, OWL DL, and UDDI (Version 2.0) for semantically enhanced Web service discovery in the FUSION Semantic Registry [16]. While this framework does not rely on any specific SWS standard, the reference implementation presented is based on SAWSDL. Neither the UDDI server nor its specification API are altered, but are wrapped in the semantic registry.

A different approach to the integration of semantic information in service registries has been implemented in PYRAMID-S [23]. Actually, PYRAMID-S is an overlay to service registries which uses a hybrid peer-to-peer topology to manage heterogeneous service registries. The aim of the framework is to allow unified Web service publication and discovery, which does not adhere to a particular service registry standard. As PYRAMID-S facilitates the usage of different service registry standards, it is necessary to define mediators for the designated standards. Mediators for UDDI (based on [6]) and ebXML (based on [4]) have already been defined. There are several differences between PYRAMID-S and the work at hand: Most importantly, in PYRAMID-S, only those matchmakers provided by ebXML Registry are explicitly regarded. The authors give no information on how to extend a registry's matchmaking capabilities. As ebXML

Registry does not provide any semantic matchmaker, matching is limited to syntax-based query statements and "non-fuzzy" semantic matchmaking, i.e., a semantic annotation in a service advertisement has to be *exactly* the same as specified in a service request. Furthermore, Pilioura and Tsalgatidou make use of their own WSDL variant, namely PS-WSDL and do not regard, e.g., SAWSDL or OWL-S.

Dogac et al. introduce another approach, which incorporates the integration of "Web Ontology Language" (OWL) ontologies into ebXML registries in order to enhance service discovery [8]. The work by Dogac et al. has been committed as an OASIS Committee Draft for an *ebXML Registry Profile for Web Ontology Language* [7], which could be used to integrate OWL-S services into ebXML. Notably, this work is limited to OWL Lite, while OWL-S ontologies are written in OWL DL [2]; there is no information given on how this contradiction is handled. The authors define a mapping of OWL elements to ebXML class hierarchies, which can be performed automatically from a given OWL ontology. Concerning the suggested mapping, OWL classes are represented through classification nodes in ebXML, while RDF properties are modeled using ebXML associations. This allows to represent whole OWL class hierarchies through ebXML elements. Finally, stored procedures are defined in order to handle the OWL semantics, e.g., to obtain all the super- or subclasses of a given class. These stored procedures can then be utilized by users in order to retrieve appropriate services that are classified using the OWL classification nodes from the ebXML registry [8]. This way, this solution is very inflexible, as it does not account for inferred semantic relationships and relies on querying predefined semantic hierarchies.

In their work on making use of SPARQL as means to define preconditions and effects in SWS descriptions, Iqbal et al. also use ebXML Registry [13]. The registry is used to store SAWSDL-based service descriptions, while SPARQL-based conditions are stored separately in the repository infrastructure. The authors state that their ebXML-based service repository does not (yet) allow to query for the integrated semantic metadata. The authors suggest to store the semantically enhanced service descriptions within the ebXML infrastructure and indicate a mechanism to reference additional semantic information in form of SPARQL-based conditions. Unfortunately, the details of this approach are not stated explicitly.

As it can be seen, existing solutions to integrate SWS into ebXML Registry are either constricted to the elements needed in a particular matchmaking approach [8], [13] and/or rely on the existing matchmakers provided by ebXML Registry [23]. In contrast, the solution at hand has been designed in order to provide a generic framework for SWS discovery using ebXML Registry.

3 ebXML Registry – Overview

In 1999, ebXML[2] has been initiated by OASIS and the United Nations/ECE agency CEFACT. In general, it provides a modular suite of specifications for en-

[2] `http://www.ebxml.org/geninfo.htm`, access at 2010-09-04

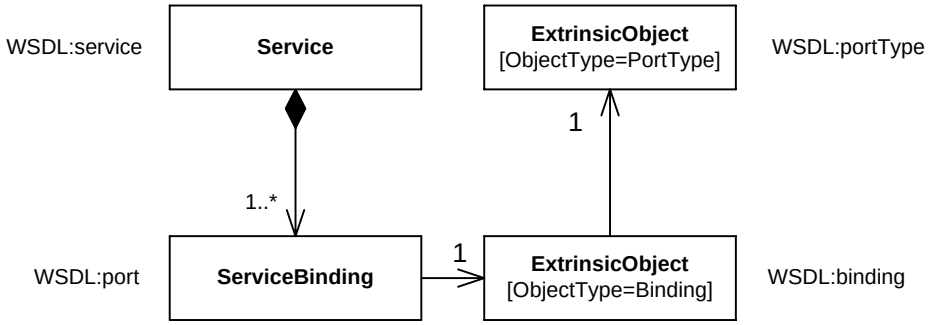

Fig. 1. Mapping WSDL Information to the ebXML RIM

terprises to perform business over the Internet (e.g., message exchange, registration of services), from which the specifications for registries and repositories are relevant within the work at hand. In this context, two documents are currently available as approved OASIS standards: the *ebXML Registry Information Model* (RIM) [9] and the *ebXML Registry Services and Protocols* (RS) [10]. ebXML RIM and ebXML RS have been standardized as ISO 15000, Parts 3 and 4, respectively. The former specifies the underlying data model of the registry, i.e., the metadata classes, whose instances are used to describe the objects stored in the repository, and the latter describes the functionalities provided by the registry and the protocols used for interacting with the registry.

An ebXML registry may further implement different profiles, each defining a processing standard as well as extensions and restrictions of the core ebXML features for a specific type of content. The *ebXML Registry Profile for Web Services* (WS) defines the publication, management and discovery of Web service artifacts [21]. The RIM classes which are relevant for the registration of Web services in an ebXML registry implementing the Web Service profile are depicted in Figure 1.

A service is represented by an instance of the `Service` class, which itself contains one or more instances of the `ServiceBinding` class providing technical information (e.g., the access URI) on how to access a concrete service instance in a specific way. The class `ExtrinsicObject` represents the primary metadata class for items stored within the repository [9]. To specify the type of content submitted to the repository represented by an instance of the `ExtrinsicObject` class, classification schemes are used. In standard ebXML implementations, WSDL files are stored as `ExtrinsicObject` instances and classified with the WSDL classification node. When submitting a WSDL document to the registry, a corresponding *Cataloging Service* is invoked which performs a mapping of the WSDL components to the ebXML RIM [21]. This is due to the fact, that the default service information model as part of the RIM also supports the registration of other types of services than Web services [9], i.e., represents a generic service model. Consequently, the components which are specific to a certain kind of service have to be stored as extrinsic objects and classified using custom-built classification

schemes. So far, ebXML WS has only been defined for WSDL 1.1-based service descriptions.

4 Solution Approach and Prototypical Implementation

In order to integrate SWS into ebXML and provide appropriate service discovery facilities, it is necessary to provide solutions for the following issues:

- Integration of SWS descriptions
- Integration of query formulations
- Integration of matchmaking capabilities

Regarding the integration of query formulations, we refer to our former work presented in [25]. In the following, we will focus on the integration of SWS and the provision of a matchmaking interface in ebXML; query formulation is only regarded if necessary to complete a particular consideration. Afterwards, we present a prototypical implementation using *freebXML 3.1*[3].

4.1 Integration of SWS Descriptions

In order to integrate SWS descriptions into ebXML Registry, it is necessary to enhance the ebXML RIM by a new classification node, e.g., called `SWS`. Using the newly created object type `SWS`, it is possible to classify SWS objects and to distinguish between non-semantic and semantic Web services. The handling of these new objects has to be implemented in the corresponding ebXML Registry realization (cp. Section 4.4). We determine SWS descriptions to be published using a subclass of the new classification node `SWS`; the corresponding WSDL information is published by the standard publication mechanism. In doing so, we assume SWS descriptions to make use of a WSDL grounding, as provided by, e.g., WSMO and OWL-S [14], [15], [18].

SAWSDL services can be published without any modifications using the standard WSDL cataloging service of ebXML registrations. However, it is still necessary to publish the semantic information described in SAWSDL using a separate node in order to make a differentiation for syntax- and semantic-based service discovery.

4.2 Integration of Matchmaking Capabilities

Per se, ebXML Registry offers syntax-based matchmaking capabilities based on service queries defined using SQL or so-called ebXML filters [10]. In order to enable semantic-based service discovery, it is necessary to provide new matchmaking facilities.

Therefore, within the work at hand, an exemplary matchmaker is directly integrated into a service registry as proof-of-concept. A direct integration mechanism for matchmakers into registries demands a generic concept, i.e., the creation

[3] `http://ebxmlrr.sourceforge.net/`

and provision of interfaces. In doing so, further matchmakers can be introduced without changing existing classes, but only through the implementation of additional classes.

Since the registry should also provide semantic matching capabilities, the management of ontologies has to be addressed. Regarding service (information) life cycle management as well as service and SOA governance, (information about) ontologies need to be managed by a service registry itself. Ontologies needed in matchmaking depend on the services published in a registry. Thus, a flexible mechanism for the management of ontologies is required, so that new ontologies can be added at any time. For this, a semantic reasoning engine can be integrated into a service registry in conjunction with an ontology knowledge base, where arbitrary ontologies can be registered. For example, a service provider could register the necessary ontologies together with the service offers at publication time. If a query is enhanced with semantic information, the syntax-based part can be directed to the standard search facilities provided by a registry and the additional semantic information can be directed to semantic matchmakers to allow for real reasoning support.

4.3 freebXML – An Open Source Reference Implementation of ebXML Registry

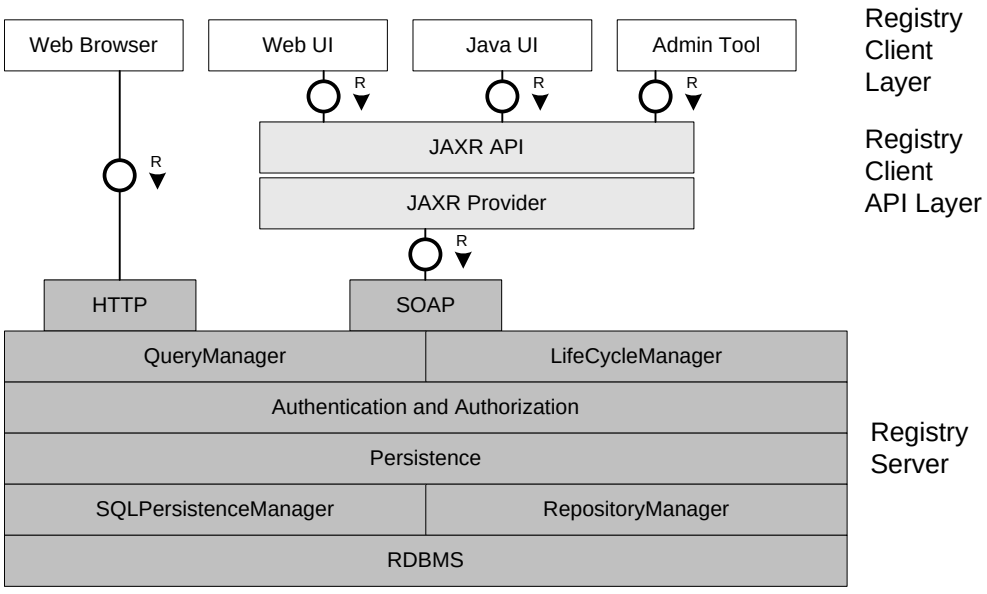

Fig. 2. The freebXML Architecture

For our prototypical implementation, freebXML 3.1 was used and enhanced. freebXML is an open source reference implementation of the OASIS ebXML Registry standards [9], [10]. freebXML is made up from a registry, where metadata about artifacts can be published, and a repository, where the actual artifacts are stored. In general, an ebXML registry may implement different profiles, i.e., provide functional enhancements for a specific type of content. Concerning profiles, freebXML implements, among others, ebXML WS [21]. Basically, the freebXML architecture comprises the three parts depicted in Figure 2: the *Registry Client Layer*, the *Registry Client API Layer* and the *Registry Server*.

The Registry Client Layer depicts possible registry client types, which may be represented by a Web browser with direct HTTP access for querying purposes only, a thin client Web User Interface (UI) running within some Web container, that can be accessed via a Web browser, a fat client Java UI running on a client machine and a command line interface, e.g., in the form of an administration tool. Instances of the last three client types are provided by the freebXML registry.

The Registry Client API Layer is the subsequent layer in the architecture. It is represented by the *JAXR API*, which provides standard Java interfaces to access the registry. The JAXR API requires a JAXR provider, which represents an implementation of the JAXR API. The freebXML registry contains its own *JAXR Provider*, which represents an advanced implementation of the JAXR API.

Concerning the freebXML Registry Server, a HTTP and a SOAP interface are provided. For browsing and discovery capabilities, the HTTP and the SOAP interface offer a binding to the *QueryManager* interface, while the SOAP interface also provides a binding to the *LifeCycleManager* interface for publishing content. Furthermore, modifying access to the registry requires user authentication and authorization. For this, a security layer is part of the registry server. In order to store content and metadata, an abstract persistence layer is defined by the registry. For the actual storage of content and metadata, a relational database management system (*RDBMS*) is used. By default, freebXML makes use of Apache Derby, which runs in the same Java Virtual Machine as the registry server. To manage the persistence of the registry server and the repository items, an SQL persistence manager interface and an repository manager interface are provided, respectively.

A comprehensive presentation of the freebXML registry architecture is given at the project's Web page[4].

4.4 Prototypical Implementation

Based on the defined requirements, we have prototypically enhanced freebXML as a generic SWS discovery framework. As SWS formalism, we make use of SAWSDL and WSDL 1.1. Furthermore, we make use of a "query by example"-approach, i.e., a SAWSDL description needs to be provided by the service requester. In the following, the integration of SAWSDL into ebXML Registry, an

[4] `http://ebxmlrr.sourceforge.net/`

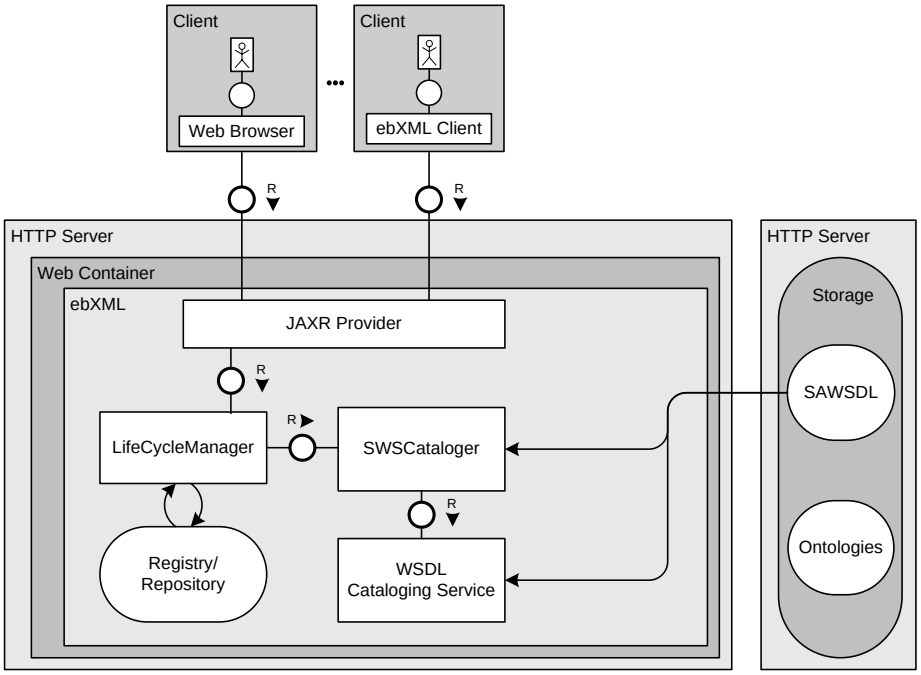

Fig. 3. Enhanced ebXML Architecture (Publishing)

interface for arbitrary matchmakers, and the utilization of an example matchmaker are presented.

Apart from freebXML, the following major software components have been used in our prototypical implementation:

Apache Tomcat 5.0.28 is the Web container freebXML is deployed to,
Java JDK 5.0 Update 22 is deployed for enhancements of freebXML and the development of a registry client, and
Apache HTTP Server 2.2 is used to store Web service descriptions and ontologies.

Integration of SAWSDL into ebXML Registry: For SAWSDL-based service descriptions, a classification node `SAWSDL` has to be created in the registry; as presented in Section 4.1, `SAWSDL` is derived from `SWS`. Generally, depending on the type of content, an associated cataloging service, which extracts the required information from submitted services, is invoked on publication time. The information is then mapped to instances of the ebXML RIM representing the content's metadata, while the actual content is stored within the repository,

In detail, by submitting SAWSDL documents to the registry, they are classified using `SAWSDL`. The *LifeCycleManager* (cp. Figure 3) of the registry makes a call to the appropriate cataloging service associated with the `SAWSDL` object

type. For this, a new cataloging service *SWSCataloger* has been developed. On invocation, the SWSCataloger first makes a call to the standard *WSDL Cataloging Service* of freebXML, which performs a normal publication of the WSDL information associated with the SWS document.

Finally, the published WSDL information has to be associated with the SAWSDL representation of the service. For this, ebXML provides the ability to relate any two objects in the registry using arbitrary relationship types. The resulting registry and repository objects are then passed to the *LifeCycleManager*, which submits the contents to the *Storage* database of freebXML.

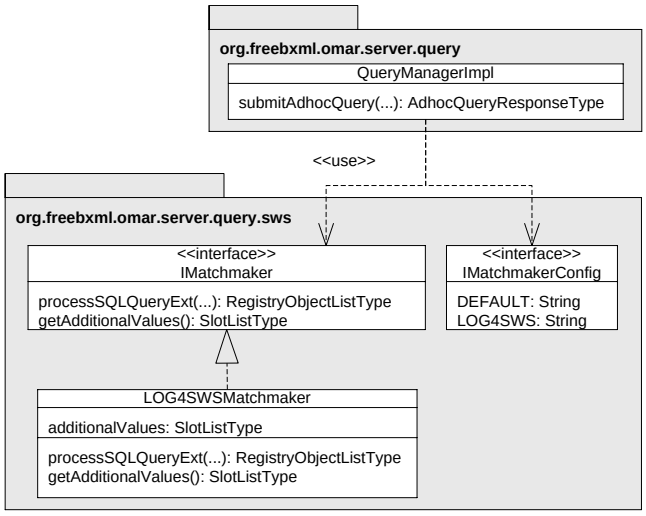

Fig. 4. Registry Enhancements for the Integration of Matchmakers

Matchmaker Interface: For the integration of different matchmakers, the QueryManager implementation of freebXML has been modified and additional classes have been created. An overview of the classes is depicted in Figure 4. In detail, each SQL query that is submitted to the registry is scanned for an SQL extension, i.e., an enhanced SQL query string indicating that a semantic "query by example" has been submitted as service request. If no extension is found, the SQL query is processed in the usual way. Else, the enhanced query string is forwarded to the semantic matchmaker.

Possible matchmakers can be registered within the `IMatchmakerConfig` class in the form of a string-based constant that is associated with the fully qualified name of the main class of a matchmaker. Using the Java Reflection API[5], the

[5] `http://java.sun.com/docs/books/tutorial/reflect/index.html`

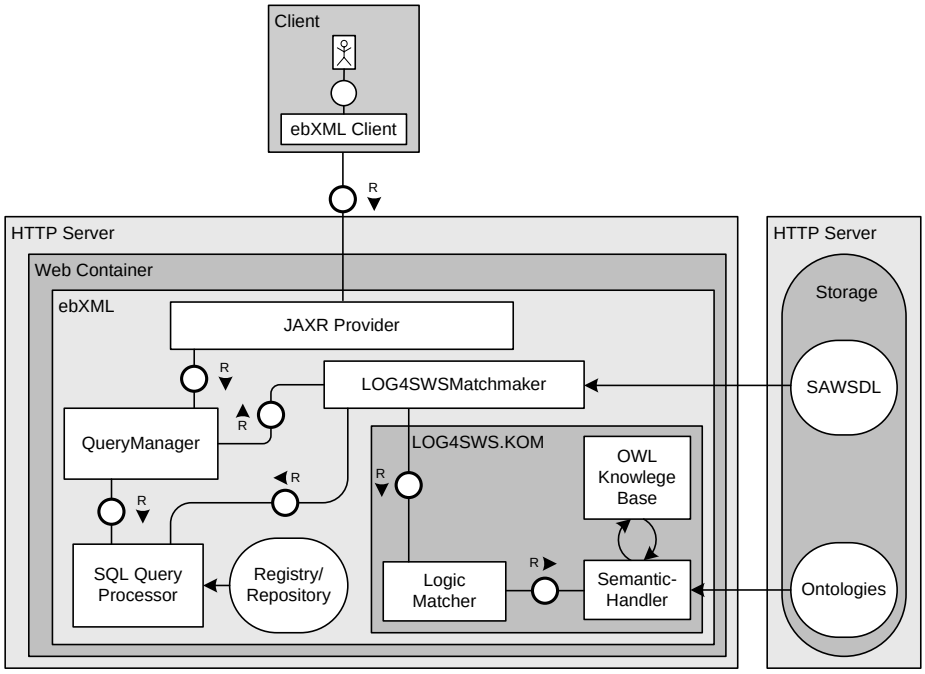

Fig. 5. Enhanced ebXML Architecture (Discovery)

QueryManager implementation is able to redirect a query to the desired matchmaker. For the integration of a new matchmaker, the `IMatchmaker` interface can be implemented by the respective class. In doing so, two methods have to be realized by the new matchmaker; one for processing the SQL query extensions and a second method to return an optional response slot list, which can be used to send back additional information (e.g., a similarity value) of a matching service to the client. The optional response slot list is provided by default by the freebXML registry implementation as part of the response to an SQL query. For the proof of concept implementation, the `LOG4SWSMatchmaker` class has been created (see below), which establishes the connection to the matching facilities provided by our matchmaker LOG4SWS.KOM [24]. In order to retrieve the set of available service descriptions from the registry, the LOG4SWSMatchmaker class makes use of the querying facilities provided by the *SQL Query Processor* of freebXML. Each service object that matches the query is added to the final result set, which is then sent back to the respective client.

The process for the retrieval of SWS is illustrated in Figure 5. First of all, a "query by example"-enhanced SQL query string is created by a service requester using the ebXML client. Afterwards, the matchmaker processes the SAWSDL service descriptions according to its specific matchmaking algorithm. Finally, the result is sent back to the ebXML client.

Example Matchmaker: For the prototypical implementation, an exemplary matchmaker for SAWSDL, namely LOG4SWS.KOM, is utilized, which has been presented in [24] and is also participating in this year's "Annual International Contest S3 on Semantic Service Selection Retrieval Performance Evaluation of Matchmakers for Semantic Web Services" (S3 Contest)[6]. LOG4SWS.KOM takes a "query by example" as service request and is provided with a set of service offers by the ebXML Registry.

By default, LOG4SWS.KOM aims at performing a logical subsumption matching, which accounts for the semantic annotations on the different component levels of a SAWSDL-based service description. These components comprise interfaces, operations and parameters (i.e., inputs and outputs). For each component level, an individual similarity value is computed during the matching process, which is then aggregated to a global similarity value for the whole service based on predefined weights for each level. In doing so, the resulting, individual similarity values representing classical Degrees of Match [22] are transformed into numerical representations between 0 and 1 for their combination. Since semantic annotations may not be present on all service component levels or required ontologies may be missing within the OWL knowledge base of a registry or may even fail to load, LOG4SWS.KOM also provides a fallback strategy, which is applied in these situations. The fallback strategy then processes the remaining information, i.e., the names of the components in the first case or the names of the semantic concepts in the other cases. The similarity measure is then determined using the WordNet ontology [20], which represents a semantic net of words for the English language. Based on the distance of a pair of words in WordNet, the similarity value is computed.

In order to be able to determine subsumption relationships between the concepts specified within a service query and the concepts specified within a service offer, a reasoning engine has to be integrated into the registry. For this purpose, a *SemanticHandler* is part of LOG4SWS.KOM, which provides an interface to access a semantic reasoner (here: Pellet 2.0) and an OWL knowledge base. Since the initialization of the semantic reasoner and the OWL knowledge is a very time consuming task, it cannot be performed once again on every incoming service query. Therefore, an instance of the SemanticHandler has been integrated into the *RepositoryManagerFactory*, which represents a permanent and unique instance within the freebXML registry, so that the SemanticHandler and the corresponding OWL knowledge base can be initialized once upon registry startup. Within the work at hand, it is further assumed, that the required ontologies for discovery already exist within the OWL knowledge base, when a query is submitted to the registry. For this purpose, a list with the required ontologies is provided to the SemanticHandler, so that the necessary ontologies are loaded into the OWL knowledge base at registry startup.

[6] `http://www-ags.dfki.uni-sb.de/~klusch/s3/html/2010.html`

5 Conclusion

Even though ebXML Registry provides a valid alternative to the usually applied UDDI-based service registries, surprisingly little work has been done regarding the application of ebXML Registry in SWS discovery frameworks. In the paper at hand, we have presented a corresponding approach, which has been prototypically implemented deploying SAWSDL as SWS formalism and freebXML as ebXML implementation.

Also our prototypical implementation is fully operational, we consider it primarily as a foundation for more sophisticated solutions. Among other thing, we want to address the following questions in our future work:

In this paper, we have proposed a quite lightweight interface for matchmakers. The interface is based on the assumption that service requests are formulated using a "query by example"-approach. However, as we have stated in our previous work [25], it might be helpful to provide more sophisticated and fine-grained query formalisms for SWS. Thus, the matchmaker interface might be replaced by a more heavyweight one; the features such an interface should provide are subject to a discussion in the research community.

Second, the prototypical implementation is restricted to SAWSDL-based service descriptions. In our opinion, the heterogeneity of SWS formalisms is a major obstacle especially regarding service discovery as, e.g., it is not possible to find an OWL-S service profile based on a SAWSDL-based service request. In our future work, we want to address this issue by providing a sophisticated, unified query language in ebXML Registry.

Finally, as presented in Section 2, matchmakers can either be integrated into registries or service queries can be intercepted to allow for a redirect to specific external matchmakers. Both approaches require a modification of the registry's source code. However, using the first approach, a repetitive modification of the source code is necessary, when new matchmakers are to be integrated. In the second approach, a generic redirection mechanism can be implemented, so that a modification of the registry's source code has to be only performed once. Although, a generic redirection mechanism is preferable, it is far more complex. Thus, we used the first-mentioned approach in the work at hand. In our future work, we will examine the integration of a generic redirection mechanism for service requests to external matchmakers.

Acknowledgements. This work is supported in part by the E-Finance Lab e. V., Frankfurt am Main, Germany (`www.efinancelab.de`).

References

1. Akkiraju, R., Goodwin, R., Doshi, P., Roeder, S.: A Method for Semantically Enhancing the Service Discovery Capabilities of UDDI. In: Workshop on Information Integration on the Web (IIWeb-03) at Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03). pp. 87–92 (2003)

2. Ankolekar, A., Martin, D., McGuinness, D., McIlraith, S., Paolucci, M., Parsia, B.: OWL-S' Relationship to Selected Other Technologies. W3C Member Submission (November 2004), http://www.w3.org/Submission/OWL-S-related/, access at 2010-08-12
3. Châtel, P.: Service Registries Study. Thales Study (June 2006), http://www.chatelp.org/work/LUCAS_registry_study.pdf, last access at 2010-04-13
4. Chiusano, J.M., Najmi, F.: Registering Web Services in an ebXML Registry, Version 1.0. OASIS Technical Note (March 2003), http://www.oasis-open.org/committees/download.php/11907/regrep-webservices-tn-10.pdf, access at 2010-01-24
5. Cimpian, E., Zaremba, M. (eds.): Web Service Execution Environment (WSMX). W3C Member Submission (June 2005), http://www.w3.org/Submission/WSMX/, access at 2010-04-03
6. Colgrave, J., Januszewski, K.: Using WSDL in a UDDI Registry, Version 2.0.2. OASIS Technical Note (June 2004), http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm, access at 2010-01-24
7. Dogac, A. (ed.): ebXML Registry Profile for Web Ontology Language (OWL). OASIS Committee Draft (September 2006), http://docs.oasis-open.org/regrep/v3.0/profiles/owl/regrep-owl-profile-v1.5.pdf, access at 2010-09-05
8. Dogac, A., Kabak, Y., Laleci, G., Mattocks, C., Najmi, F., Pollock, J.: Enhancing ebXML Registries to Make them OWL Aware. Distributed and Parallel Databases 18(1), 9–36 (2005)
9. Fuger, S., Najmi, F., Stojanovic, N. (eds.): ebXML Registry Information Model Version 3.0. OASIS Standard (May 2005), http://docs.oasis-open.org/regrep/v3.0/specs/regrep-rim-3.0-os.pdf, access at 2010-02-09
10. Fuger, S., Najmi, F., Stojanovic, N. (eds.): ebXML Registry Services and Protocols Version 3.0. OASIS Standard (May 2005), http://docs.oasis-open.org/regrep/v3.0/specs/regrep-rs-3.0-os.pdf, access at 2010-02-09
11. Haller, A., Cimpian, E., Mocan, A., Oren, E., Bussler, C.: WSMX – A Semantic Service-Oriented Architecture. In: 2005 IEEE International Conference on Web Services (ICWS 2005). pp. 321–328. IEEE Computer Society, Washington, DC, USA (2005)
12. Herzog, R., Lausen, H., Roman, D., Zugmann, P. (eds.): D10 v0.1 WSMO Registry. WSMO Working Draft (April 2004), http://www.wsmo.org/2004/d10/v0.1/20040426/, access at 2010-04-03
13. Iqbal, K., Sbodio, M.L., Peristeras, V., Giuliani, G.: Semantic Service Discovery using SAWSDL and SPARQL. In: Fourth International Conference on Semantics, Knowledge and Grid (SKG 2008). pp. 205–212. IEEE Computer Society (2008)
14. Kopecký, J., Moran, M., Vitvar, T., Roman, D., Mocan, A. (eds.): D24.2 v0.1 WSMO Grounding. WSMO Working Draft (April 2007), http://www.wsmo.org/TR/d24/d24.2/v0.1/#grounding_wsdl, access at 2010-09-03
15. Kopecký, J., Roman, D., Moran, M., Fensel, D.: Semantic Web Services Grounding. In: Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT-ICIW 2006). IEEE Computer Society, Washington, DC, USA (2006)
16. Kourtesis, D., Paraskakis, I.: Combining SAWSDL, OWL-DL and UDDI for Semantically Enhanced Web Service Discovery. In: 5th European Semantic Web Conference (ESWC 2008). LNCS, vol. 5021, pp. 614–628. Springer (2008)
17. Luo, J., Montrose, B., Kim, A., Khashnobish, A., Kang, M.: Adding OWL-S Support to the Existing UDDI Infrastructure. In: 2006 IEEE International Conference

on Web Services (ICWS 2006). pp. 153–162. IEEE Computer Society, Washington, DC, USA (2006)
18. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: OWL-S: Semantic Markup for Web Services. W3C Member Submission (November 2004), http://www.w3.org/Submission/OWL-S/, access at 2009-06-12
19. McIlraith, S.A., Son, T.C., Zeng, H.: Semantic Web Services. IEEE Intelligent Systems 16(2), 46–53 (2001)
20. Miller, G.A.: WordNet: a lexical database for English. Communications of the ACM 38(11), 39–41 (1995)
21. Najmi, F., Chiusano, J. (eds.): ebXML Registry profile for Web Services. Version 1.0 Draft 3. Draft OASIS Profile (September 2005), http://www.oasis-open.org/committees/download.php/14756/regrep-ws-profile-1.0-draft3.pdf, access at 2010-03-05
22. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.P.: Importing the Semantic Web in UDDI. In: International Workshop on Web Services, E-Business, and the Semantic Web (WES 2002) in connection with The 14th Conference on Advanced Information Systems Engineering (CAiSE 2002). Lecture Notes in Computer Science, vol. 2512, pp. 225–236. Springer, Berlin Heidelberg (2002)
23. Pilioura, T., Tsalgatidou, A.: Unified publication and discovery of semantic Web services. ACM Transactions on The Web 3(3), 1–44 (2009)
24. Schulte, S., Lampe, U., Eckert, J., Steinmetz, R.: LOG4SWS.KOM: Self-Adapting Semantic Web Service Discovery for SAWSDL. In: IEEE 2010 Fourth International Workshop of Software Engineering for Adaptive Service-Oriented Systems (SEASS '10) at 2010 IEEE 6th World Congress on Services (SERVICES 2010). pp. 511–518. IEEE Computer Society, Washington, DC, USA (2010)
25. Schulte, S., Siebenhaar, M., Eckert, J., Steinmetz, R.: Query languages for semantic web services. In: Informatik 2010 (FORTHCOMING). Gesellschaft für Informatik (2010)
26. Sivashanmugam, K., Verma, K., Sheth, A.P., Miller, J.A.: Adding Semantics to Web Services Standards. In: International Conference on Web Services (ICWS 2003). pp. 395–401. CSREA Press (2003)
27. Srinivasan, N., Paolucci, M., Sycara, K.P.: An Efficient Algorithm for OWL-S Based Semantic Search in UDDI. In: First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004), Revised Selected Papers. Lecture Notes in Computer Science, vol. 3387, pp. 96–110. Springer, Berlin Heidelberg (2004)
28. Sun Microsystems, Inc.: Effective SOA Deployment using an SOA Registry Repository. A Practical Guide (September 2005), http://www.sun.com/products/soa/registry/soa_registry_wp.pdf, access at 2010-04-14
29. Verma, K., Sivashanmugam, K., Sheth, A.P., Patil, A., Oundhakar, S., Miller, J.: METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services. Journal of Information Technology and Management 6(1), 17–39 (2005)

Comprehensive service semantics and light-weight Linked Services: towards an integrated approach

Stefan Dietze, Neil Benn, Hong Qing Yu, Carlos Pedrinaci,
Bassem Makni, Dong Liu, Dave Lambert, John Domingue

Knowledge Media Institute, The Open University, MK7 6AA, Milton Keynes, UK
{s.dietze, n.j.l.benn, h.q.yu, c.pedrinaci, b.makni, d.liu, d.j.lambert, j.b.domingue}@open.ac.uk

Abstract. Semantics are used to mark up a wide variety of data-centric Web resources but, are not used in significant numbers to annotate online services—that is despite considerable research dedicated to Semantic Web Services (SWS). This is partially due to the complexity of comprehensive SWS models aiming at automation of service-oriented tasks such as discovery, composition, and execution. This has led to the emergence of a new approach dubbed *Linked Services* which is based on simplified service models that are easier to populate and interpret and accessible even to non-experts. However, such *Minimal Service Models* so far do not cover all execution-related aspects of service automation and merely aim at enabling more comprehensive service search and clustering. Thus, in this paper, we describe our approach of combining the strengths of both distinct approaches to modeling Semantic Web Services – "lightweight" Linked Services and "heavyweight" SWS automation – into a coherent SWS framework. In addition, an implementation of our approach based on existing SWS tools together with a proof-of-concept prototype used within the EU project NoTube is presented.

Keywords: Semantic Web Services, Linked Services, Linked Data, IPTV.

1 Introduction

The past decade has seen a wide range of research efforts in the area of Semantic Web Services (SWS), mainly aiming at the automation of Web service-related tasks such as discovery, orchestration or mediation via broker-based approaches. Building on formal service semantics, several conceptual models, such as OWL-S [14] and WSMO [9], and also standards such as SAWSDL [18] have been proposed which aim at formalizing semantic service descriptions usually covering aspects such as service capabilities, interfaces and non-functional properties. Besides, a considerable research community evolved around these SWS frameworks, providing, for instance, related annotation and execution tools [7].

While semantics are used to mark up a wide variety of data-centric resources on the Web, that does not apply to online services in significant numbers. The reasons for this are two-fold. Firstly, SWS research has for the most part targeted WSDL [22] or SOAP-based [21] Web services, which are not prevalent on the Web [4]. Secondly,

due to the inherent complexity required to fully capture computational functionality, creating SWS descriptions has represented an important knowledge acquisition bottleneck and has required the use of rich knowledge representation languages and complex reasoners. There exists an inherent conflict between the need to capture comprehensive and meaningful service semantics – to allow reasoning-based automation of any sort – and the requirement to keep the costs for providing services descriptions low in order to simplify the modeling process and to ensure that efficient and scalable solutions can be implemented [17]. Hence, despite considerable amount of research dedicated to the SWS vision and the existence of a range of SWS-related projects, tools and specifications, so far there has been little take up of SWS technology within non-academic environments.

The prevalent lack of impact of SWS technology is particularly concerning since Web services – nowadays including a range of often more light-weight technologies beyond the WSDL/SOAP approach, such as RESTful services, HTTP GET-style requests or XML-feeds – are in widespread use throughout the Web where applications use distributed requests to combine and mash-up data from a variety of open data sources. Hence, the challenges SWS attempted to tackle are of even more crucial importance for today's highly distributed Web applications. These issues have led to the emergence of more simplified SWS approaches to which we shall refer here as "lightweight", such as WSMO-Lite [19] or the Micro-WSMO/hRESTs [10] approach which replace "heavyweight" service semantics with less comprehensive and less costly to produce service models that are represented in RDF(S), and hence, comply with the infrastructure of the growing *Semantic Web* [2]. Analogous to the *Linked (Open) Data (LOD)* term [3], this approach was recently dubbed as the *Linked Service* approach [17]. Due to the fact that such service annotations are much easier to produce and can be populated with references to widely established LOD vocabularies, they address a much wider audience and allow even non-SWS experts to describe and annotate services. However, while those models are easier to produce [4], they merely aim at enabling structured, semantics-enabled search by humans or automated service clustering. More expressive solutions are required to achieve greater levels of automation, for instance, dealing with matching service requests with extensive capability representations of available services, or with handling of data-level mismatches when executing a set of services in an orchestrated manner.

Therefore, here, we aim to combine the strengths of both distinctive SWS approaches – lightweight Linked Services and more heavyweight broker-based SWS automation – into a coherent SWS framework. By integrating collaborative and user-driven Web-scale service annotations with comprehensive SWS specifications, we benefit from both low cost for providing annotations and a high level of automation. This also has the benefit of enabling a range of matchmaking scenarios (from user-driven keyword matching to automated capability matchmaking).

Section 2 introduces work related to our research. Section 3 gives an overview of our approach and describes the approach and tools that were developed to support our two-stage approach, while Section 4 describes the deployment and evaluation of our work within in an EU research project.

2 Related work & background

The landscape of SWS is characterized by a number of conceptual models that, despite a number of common characteristics, remain essentially incompatible due to the different representation languages and expressivity utilized as well as because of conceptual differences. The main conceptual frameworks and specifications devised thus far include for instance WSMO [20], OWL-S [14]. SAWSDL [18], which in turn derives from WSDL-S [18]. The vast majority of the SWS initiatives were built upon the enrichment of WSDL Web services with semantics. It is only recently that researchers have started focusing on Web APIs and RESTful services. The main outputs of this recent research are SA-REST [18] and MicroWSMO [12]

Over the last few years, a significant portion of research on the SW has been devoted to creating what is referred to as LOD [3] which is based upon a set of principles, including the usage of HTTP URIs to provide information and allow access based on RDF and SPARQL. Since these principles were outlined, there has been a large uptake, most notably through DBpedia [1] and others that have produced a vast amount of linked datasets. While the great potential of this massive data space still remains largely unexploited, service-oriented computing has been argued to be a suitable approach to supporting the construction of advanced applications based on linked data [16].

2.1. Lightweight service annotation: the iServe Linked Services approach

In order to support annotation of a variety of services, such as WSDL services as well as REST APIs, the EC-funded project SOA4ALL[1], has developed iServe[2] a novel and open platform for publishing semantic annotations of services based on a direct application of linked data principles [17]. iServe supports publishing service annotations as linked data—Linked Services—expressed in terms of a simple conceptual model that is suitable for both human and machine consumption and abstracts from existing heterogeneity around service kinds and annotation formalisms. In particular iServe provides:

- Import of service annotations in a range of formalisms (e.g., SAWSDL, WSMO-Lite, MicroWSMO, OWL-S) covering both WSDL services and Web APIs;
- Means for publishing semantic annotations of services which are automatically assigned a resolvable HTTP URI;
- Support for content negotiation so that service annotations can be returned in plain HTML or in RDF for direct machine consumption;
- SPARQL endpoint allowing querying over the services annotations;
- REST API to allow remote applications to consume and provide annotations.
- Support for linking service annotations to existing vocabularies on the Web.

In order to cater for interoperability, iServe uses what can be considered the maximum common denominator between existing SWS formalisms which we refer to

[1] http://www.soa4all.eu/
[2] http://iserve.kmi.open.ac.uk

as the Minimal Service Model (MSM). The MSM, first introduced together with WSMO-Lite and hRESTS [19], is thus a simple RDF(S) ontology able to capture (part of) the semantics of both Web services and Web APIs in a common model. MSM is extensible to benefit from the added expressivity of other formalisms. The MSM, denoted with the 'msm' namespace in Fig. 1, defines *Services* as having a number of *Operations* each of which have an *Input*, *Output MessageContent*, and *Faults*. In turn, a MessageContent may be composed of *MessageParts* which may be *mandatory* or *optional*. iServe additionally uses the SAWSDL, WSMO-Lite and hRESTS vocabularies. The SAWSDL vocabulary captures in RDF the three main kinds of annotations over WSDL and XML Schema, including *modelReference*, *liftingSchemaMapping* and *loweringSchemaMapping* that SAWSDL supports. WSMO-Lite builds upon SAWSDL by extending it with a model specifying the semantics of the particular service annotations. It provides a simple RDFS ontology together with a methodology for expressing functional and non-functional semantics, and an information model for WSDL services based on SAWSDL's modelReference hooks. The hRESTS vocabulary extends the MSM with specific attributes for operations so as to allow modeling additional details necessary for Web APIs.

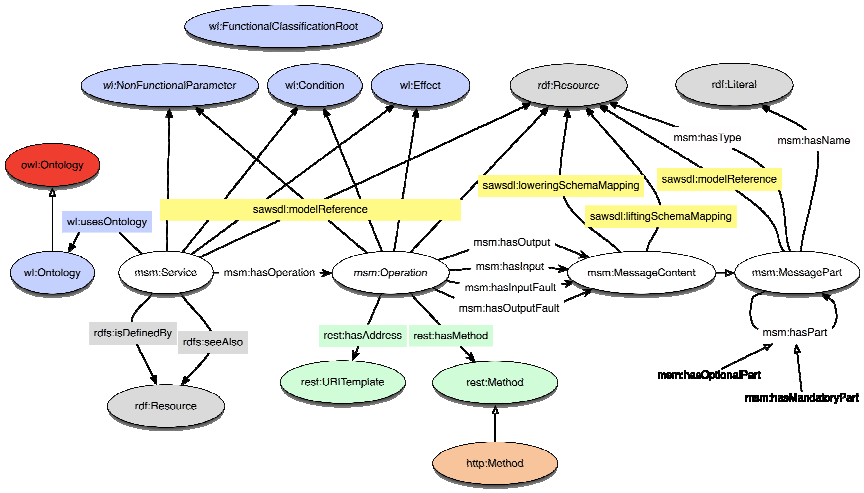

Fig. 1. iServe conceptual model for services – The Minimal Service Model and WSMO-Lite.

In order to support users in creating semantic annotations for services two editors have been developed: SWEET [12] (SemanticWeb sErvices Editing Tool) and SOWER (SWEET is nOt a Wsdl EditoR), which support users in annotating Web APIs and WSDL services respectively. However, SWEET and SOWER build on the assumption that either HTML documentation of services/APIs (SWEET) or WSDL files (SOWER) are available as starting point for annotation. In addition, while the iServe approach enables uptake of SWS technology by a wider audience, the automation and matchmaking scenarios which it facilitates are actually limited. The reason for that being that the MSM deliberately excludes execution aspects for the

sake of simplicity and the lack of a commonly prescribed capability representation model.

2.2. Automated services brokerage: the IRS-III framework

IRS-III[3] [7] is a SWS execution environment which acts as a service broker – mediating between the goals of a client and relevant services that are deployed on the Web – striving for a high level of service automation. IRS-III adopts the WSMO conceptual model of services. The ultimate aim of the WSMO model of Web services is to be able to provide a well-defined semantics, which can then be interpreted by a reasoner to enable automatic discovery, selection, composition, mediation, execution, and monitoring of services [10]. As opposed to MSM, IRS-III directly covers execution-related aspects.

The WSMO conceptual model of services consists of the following core elements: *goal*, *mediator*, and *Web service*. These are described in a formal representation language, for instance, OCML [15] in the case of IRS-III. The functionality offered by a Web Service is captured by its *capability* description, which defines necessary *pre-* and *postconditions* as well as underlying *assumptions* and *effects* of the service. These are usually formalized as logical expressions. The means to interact with the Web service is captured by its *interface* definition.

Given that IRS-III directly aims at automating service execution related aspects, the interface covers *choreography* and *orchestration* descriptions. Choreography addresses the communication between the IRS-III broker and a Web service, and is described as so-called *grounding*. The IRS-III grounding mechanism supports REST-based, SOAP-based, and XML-RPC based services [11]. Grounding involves two processes referred to as *lifting* and *lowering*. Lowering involves transforming input parameters at the semantic level to data input to the service at the syntactic level. Lifting involves the opposite transformation, i.e. transforming the data output from the service at the syntactic level into an ontological object at the semantic level.

Orchestration addresses the problem of how to model functionality that is composed of several Web services. At the semantic level the orchestration is represented by a workflow model expressed in OCML, that describes the flow of control between the Web services. The IRS-III orchestration model supports the main control-flow primitives of sequence, selection, and repetition.

At runtime, IRS-III automatically discovers and invokes Web services suitable for a given client request, formulated as a goal instance, by selecting suitable services and executing these whilst adhering to any data, control flow and Web service invocation constraints. In principle, selection is based on comparing the capability descriptions of the request with the ones of the relevant SWS. Such matchmaking is currently supported, for instance, via (a) comparison and evaluation of logical expressions used in the capability descriptions, or (b) a hybrid approach [6] which combines similarity-computation via vector representations of SWS instances with (a). The IRS-III functionalities are exposed through a Java API[4] (details in [7]), and an HTTP-based

[3] http://technologies.kmi.open.ac.uk/irs - IRS: Internet Reasoning Service
[4] http://technologies.kmi.open.ac.uk/irs/irs3docs/api/index.html

REST API, which applications use to interact with IRS-III.

3 Two-stage service annotation and reasoning

In order to tackle the challenges introduced in Section 1, we aim at combining the two distinct SWS representation approaches

(R1) lightweight Linked Services (as facilitated by MSM), and
(R2) heavyweight SWS automation (as facilitated by WSMO).

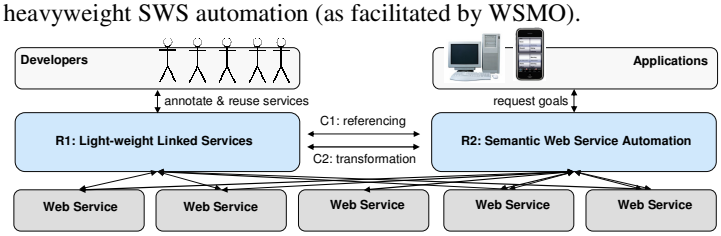

Fig. 2. From lightweight service annotations to heavyweight SWS automation - overall approach.

While these approaches currently co-exist without a well-defined relationship, we propose two different bi-directional correlations, which are under investigation:

(C1) service model cross-referencing,
(C2) service model transformation and augmentation.

Under (C1), we subsume all kinds of references between models across (R1) and (R2) as depicted in Fig. 2. For instance, a lightweight service annotation (MSM) could point to a heavyweight WSMO description that models the same service more exhaustively or vice versa. That would allow semantics to be exploited in (R1) as well as (R2) for reasoning of different sorts, for instance, to perform some clustering or filtering based on (R1) to reduce the amount of potentially interesting services for a given query in (R2). In addition, (C2) considers the transformation between models across (R1) and (R2), either manually or (semi-)automatically. Our current implementation builds upon existing SWS research namely WSMO and WSMO-Lite/MSM by integrating iServe and IRS-III. The remainder of this section describes the two approaches - (C1) and (C2) - in more detail.

3.1. Services model cross-referencing

Service model cross-referencing involves the formal definition of relationships between service models. The two main types of relationship are depicted in Figure 3.

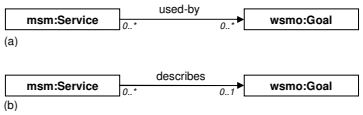

Fig. 3. Supported service model cross-referencing relationships.

(a) MSM instances referring to WSMO goal instances:
This involves specifying a link between an MSM instance and a corresponding WSMO goal description. Links of this kind define that the respective goal (*wsmo:Goal*) makes use of the service described by the respective *msm:Service*, i.e. for instance, the goal is linked to the service and potentially allows its discovery and execution as part of a more complex orchestration. Following that reference, developers are able to query the iServe repository via SPARQL or its API to (i) discover suitable services described via MSM, and then (ii) use a corresponding goal invocation URI to execute the service via IRS-III execution facilities. However, one assumption for such use cases is the existence of service models in both, iServe as well as IRS-III, which describe the same underlying service.

(b) MSM instances describing WSMO goal instances:
An additional link between MSM (iServe) and WSMO (IRS-III) is established by annotating the interface for achieving a particular goal (*wsmo:Goal* within IRS-III) itself as a minimal service description (*msm:Service*) within iServe. This is feasible and useful since WSMO goals within IRS-III are exposed via a REST-API and hence, each goal constitutes a particular service itself, which makes use of one or more actual Web services/APIs to provide a specific functionality. This has the benefit of allowing developers to query the MSM knowledge base in order to keep track of and discover WSMO goals. In that, complex functionalities – which might make use of a number of services – can be exposed via IRS-III and then be annotated within iServe as (higher level) services themselves.

3.2. Service model transformation and augmentation

Here, we consider the transformation and augmention of models across (R1) and (R2), either manually or (semi-)automatically. This involves transforming service descriptions based on one conceptual model of services (e.g. the MSM) into the other, e.g., WSMO and vice versa.

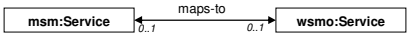

Fig. 4.Transformation between service representations across both conceptual models.

As can be seen from the previous sections, there is some overlap between the elements of a service description according to the MSM and the elements of a service description according to the WSMO conceptual model. This applies in particular to the service entity within both models. Here, we investigated the overlap between both schemas in order to establish potential mapping rules. The following figure depicts the core entities of WSMO and MSM, their relationships and their potential cross-model mapping. Please note, that for the sake of simplification, we left aside the WSMO elements goal and mediator, which have no expression in the MSM whatsoever.

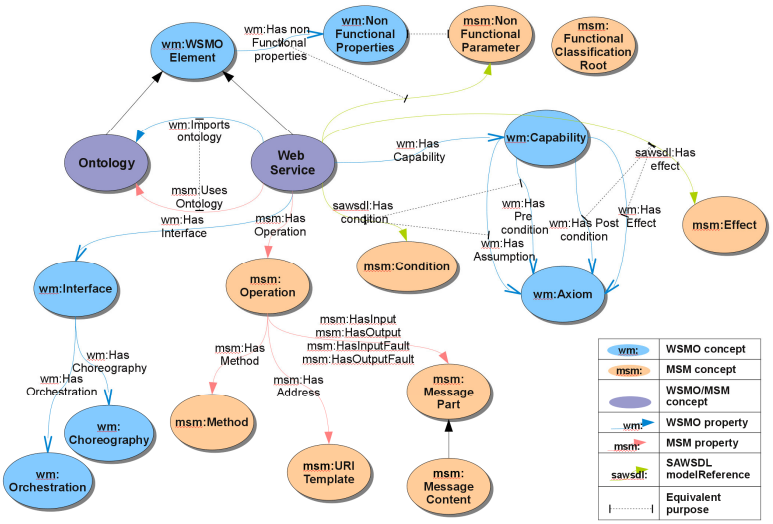

Fig. 5.MSM vs WSMO entities: relationships and mappings.

As depicted above, both models share a certain overlap, mainly relating to the core concepts such as *Ontology*, *Web Service* and *Non-Functional Parameter (Property)* and a number of properties which are equivalent. We foresee a bi-directional semi-automated transformation strategy between WSMO and MSM consisting of the following steps:

S1. Generating raw target model from source model.
S2. Semi-automatic augmentation of target model.

This transformation is making use of the mentioned model schema overlap and aims at generating raw target models (e.g. a WSMO service instance) from a given source model (e.g. a MSM service instance) as part of *S1*. *S2* then aims at semi-automatically enriching the generated service instance in order to create a fully target schema compliant service instance.

3.3. Implementation: service annotation and integration via SmartLink

In order to tackle some of the issues mentioned above and to approach integration of service models, a new services annotation and search tool was created, *SmartLink* ("SeMantic Annotation enviRonmenT for Linked services"). SmartLink allows annotation of REST-ful services based on the MSM from scratch, that is, without any pre-existing services documentation such as WSDL or HTML files, as assumed by existing iServe annotation tools (Section 2.1). Besides, SmartLink exploits an extension of the MSM schema including a number of additional non-functional properties. MSM-schema properties are directly stored in iServe, while additional

properties are captured in a complementary RDF store based on OpenRDF Sesame[5]. Due to these extensions, we refer in the following to our service RDF store as *iServe+*. These non-functional properties are, for instance, *contact person, developer name, Quality of Service* (*QoS), development status, service license,* and *WSMO goal reference*. The latter property directly contributes to facilitate our vision of allowing MSM models to refer to existing WSMO goals which utilize the same service entity; that is, it facilitates our model referencing vision (Section 3.1) between MSM and WSMO models. In addition, by allowing developers to directly annotate existing REST-ful services and APIs, SmartLink directly provides another contribution to enable our service model integration vision (Section 3.1) based on allowing the annotation of WSMO goal requests – which in fact are REST-ful services themselves – as MSM service instances.

SmartLink currently provides mechanisms which enable the export of particular (MSM) service instances as RDF or human-readable HTML. In order to facilitate service model transformation and augmentation between MSM and WSMO as proposed in Section 3.2, current research deals with the establishment of an export mechanism of MSM service models as WSMO instances. While current implementation work is concerned with adding corresponding export facilities to SmartLink, model transformation is just enabled on a manual basis at the moment.

4 Case study: two-fold service annotation within NoTube

This section describes a first application of our approach in the context of the NoTube project[6] where the ultimate goal is to develop a network of services, connected through the use of semantics, to personalize consumption of digital (IP)TV content.

4.1. NoTube challenges

In order to illustrate the challenges with respect to service-related tasks, we describe one of the main use cases driven by the TV broadcast industry partners within the NoTube project – namely the requirement to provide personalized content and metadata delivery to users. Here, the basic feature is the matching of heterogeneous users' profiles, e.g. including interests, preferences and activity data, and user contexts (e.g. current location and viewing device) to filter and deliver TV content from a variety of sources. Addressing this particular use case in a service-oriented manner involves selecting and orchestrating between numerous services that provide various functionality, for instance, to aggregate users' topic interests based on their social networking activities, retrieve electronic program guide (EPG) data from various sources, and provide recommendations based on a dedicated algorithm. To support the highly service-oriented nature of the project, two major goals need to be supported: (1) support of distributed developers with lightweight service annotations,

[5] http://www.openrdf.org/
[6] http://www.notube.tv

and (2) support of application automation with Semantic Web Service brokerage. To support these goals, we deploy and adapt iServe and IRS-III as SWS frameworks.

4.2. Two-fold service semantics: implementation and integration within NoTube

Supporting lightweight NoTube service annotations via SmartLink and iServe
While the NoTube development takes place in a highly distributed setting and follows service-oriented principles, NoTube developers need to be provided with the means to document and search for appropriate services and data sources in order to build applications and higher-level services.

```
<rdf:RDF xmlns:so="http://www.purl.org/vocabularies/service-ontology#"
xmlns:msm="http://cms-wg.sti2.org/ns/minimal-service-model"
xmlns:saw="http://www.w3.org/ns/sawsdl#"...>

<rdf:Description
rdf:about="http://lupedia.ontotext.com/lookup#text2rdfa">
   <so:hasContactPerson>Stefan Dietze</so:hasContactPerson>
   <so:hasGoal>GET-LUPEDIA-ENTITIES-GOAL</so:hasGoal>
   <msm:hasInput
   rdf:resource="http://lupedia.ontotext.com/lookup/input#lookupText"/>
   <msm:hasOutput
   rdf:resource="http://lupedia.ontotext.com/lookup/output#lookupResult"/>
  …
   <so:hasOneLiner>Lookup of free text in DBPedia based on entity recognition and
   DBPedia lookup.</so:hasOneLiner>
   <msm:hasOperation
   rdf:resource="http://lupedia.ontotext.com/lookup/#text2rdfa"/>
   <sa:modelReference rdf:resource="http://www.service-
   finder.eu/ontologies/ServiceCategories#Multimedia"/>
   <sa:modelReference rdf:resource="http://www.service-
   finder.eu/ontologies/ServiceCategories#Content"/>
  …
</rdf:Description>
…
<rdf:Description
rdf:about="http://lupedia.ontotext.com/lookup/output#lookupResult">
   <sa:modelReference rdf:resource="http://dbpedia.org/data/Entity"/>
</rdf:Description
…
</rdf:RDF>
```

Listing 1. RDF-excerpt of LUPedia service description based on MSM.

Hence, as an initial step, lightweight service semantics need to be generated, stored and exposed in an explorable way to support the NoTube developers in finding and reusing appropriate services. NoTube adopts the iServe environment by utilising the iServe+ and SmartLink tools which cater for additional NoTube-specific requirements (Section 3.3) which operates on top of the iServe RDF store. In addition, the general-purpose service taxonomy used by iServe (ServiceFinder ontology[7]) was extended with a service classification specific to the NoTube domain.

Listing 1 depicts an extract of the RDF description of a particular NoTube service (LUPEDIA[8]) which performs a lookup of free text in DBPedia in order to allow enrichment of EPG metadata with additional DBPedia entities. Besides the utilisation

[7] http://www.service-finder.eu/ontologies/ServiceCategories
[8] http://lupedia.ontotext.com/

of model references to external vocabularies – please note the highlighted reference (`<sa:modelReference>`) at the bottom – the listing also highlights some of the integrative elements which had been utilized within NoTube. For instance, the `<so:hasGoal>`-property refers to a particular WSMO goal instance within IRS-III to cater for our model referencing approach (Section 3.1).

The following screenshot depicts the query interface of SmartLink, which allows to query for services. Service matchmaking is being achieved by matching a set of core properties (input, output, keywords) or submitting more comprehensive SPARQL queries.

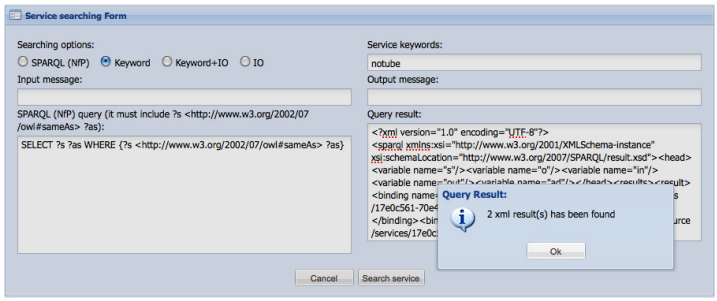

Fig. 6. SmartLink service query interface as utilized in NoTube.

Support of service automation with Semantic Web Service brokerage

The IRS-III acts a middleware component for the NoTube project with the purpose of automatically finding, combining and invoking relevant Web Services based on goals specified by NoTube application developers. By annotating existing services via WSMO, we abstract from the Web service implementations, ensuring a high level of autonomy and flexibility. That is, service consumers treat goals as black boxes which provide abstract functionalities achievable by IRS-III in terms of reasoning on WSMO service instances to discover and orchestrate suitable services. Goals are requested via the IRS-III REST API (Section 2.2), and, as such, each individual goal achievement request constitutes a service itself.

The following code excerpt shows the WSMO description (in OCML) of the same NoTube service (LUPedia) and a corresponding WSMO goal (`GET-LUPEDIA-ENTITIES-GOAL`). This code has been obtained by manually applying our transformation strategy from Section 3.2. Besides the I/O definitions ("`has-text`" and "`has-lupedia-entities`") the listing also shows the grounding definitions that determine how the WSMO goal invocation instance is grounded to the underlying Web service. The grounding consists of three key definitions (highlighted in the Listing):

- The definition of the service listener (`GET-LUPEDIA-ENTITIES-WS-PUBLISHER-INFORMATION`);
- The lowering definition defining the lowering from the semantic level (WSMO/OCML instances) into the input parameters of the Web service (`LOWER-FOR-GET-LUPEDIA-ENTITIES-GOAL`, not shown in full detail);
- The lifting definition which describes the lifting of service execution results into WSMO/OCML instances (`LIFT-FOR-GET-LUPEDIA-ENTITIES-GOAL`, not shown in full detail). The lifting defines a rule for parsing and handling the XML result of

the LUPedia service (see also [11])

```
(DEF-CLASS GET-LUPEDIA-ENTITIES-GOAL (GOAL)
             ((HAS-INPUT-ROLE :VALUE has-text)
              (HAS-OUTPUT-ROLE :VALUE has-lupedia-entities)
              (has-text :TYPE String)
              (has-lupedia-entities :TYPE List)))
  …
(DEF-CLASS GET-LUPEDIA-ENTITIES-WS-PUBLISHER-INFORMATION (PUBLISHER-INFORMATION)
             ((HAS-WEB-SERVICE-HOST :VALUE "lupedia.ontotext.com")
    (HAS-WEB-SERVICE-LOCATION :VALUE "/lookup/text2xml")))

(DEF-RULE LOWER-FOR-GET-LUPEDIA-ENTITIES-GOAL
    …)
(DEF-RULE LIFT-FOR-GET-LUPEDIA-ENTITIES-GOAL
     …
     (extract-lupedia-entities-from-xml ?xml ?list-of-lupedia-entities)
     if
     (= ?list-of-lupedia-entities
        (setofall ?lupedia-entity
                (and
                (#_xml:rootElement ?xml ?rootEl)
                (#_xml:contents ?rootEl ?rootContents)
                (member ?lookupsEl ?rootContents)
                (#_xml:tag ?lookupsEl "lookups")
                (#_xml:contents ?lookupsEl ?lookupsContents)
                (member ?instanceURIEl ?lookupsContents)
                (#_xml:tag ?instanceURIEl "instanceUri")
                (#_xml:contents ?instanceURIEl (?instanceURIContents))
                (#_xml:value ?instanceURIContents ?instance-uri)
                (member ?classURIEl ?lookupsContents)
                (#_xml:tag ?classURIEl "instanceClass")
                (#_xml:contents ?classURIEl (?classURIContents))
                (#_xml:value ?classURIContents ?class-uri)
                (= ?lupedia-entity (#_LUPediaEntity
                                    ?instance-uri
                                    ?class-uri)))))))
```

Listing 2. WSMO/OCML-code of LUPedia service.

Integration aspects between MSM and WSMO within NoTube

Section 3 introduced two methods for integrating the MSM and WSMO approaches: (a) Service model cross-referencing, and (b) Service model transformation. Within NoTube, the service model cross-referencing approach as described in Section 3.1 was implemented in two ways: by including a property in the extended MSM schema that provides a link to a corresponding WSMO goal description in the IRS-III execution environment (as illustrated by Listing 1). Furthermore, each WSMO goal invocation URI, that is the REST API goal achievement request which itself represents a REST-ful service for invoking some particular functionality, is also represented as a service following the extended MSM. That allows to expose higher level functionalities – achieved by orchestrating a number of heterogeneous services – as services themselves. Due to a lack of automated model transformation mechanisms so far and the lack of use cases requiring models being used in both representations, service instances had so far been transformed manually between WSMO and MSM. For instance, the service description in Listing 1 was generated by following the transformation procedure introduced in Section 3.2 to generate the service instance illustrated by Listing 2.

4.3. Lessons learned

From our initial use case, a few observations have been made which will shape our future efforts related to our two-fold services annotation and reasoning approach. While it was fairly easy to gather lightweight service semantics within NoTube by encouraging developers in the project to directly annotate their services via SmartLink, the lack of service automation and execution support provided by our extended MSM models, and, more importantly, the current tool support, made it necessary to transform and augment these models to expose them via IRS-III, i.e. as WSMO models within IRS-III, in order to perform more execution-oriented tasks. While transformation currently was achieved manually, future work will be dedicated to minimize this effort by striving for (semi-)automated transformation as sketched in Section 3.2.

The recommendation of LOD model references via open APIs – SmartLink currently uses WATSON[9] – proved very useful to aid the population of our iServe+ store. However, due to the increasing number of LOD datasets – strongly differing in terms of quality and usefulness – it might be necessary in the future to select recommendations only based on a controlled subset of the LOD cloud in order to reduce available choices.

With respect to service automation and brokerage, WSMO and IRS-III provide certain facilities to define service orchestrations or to achieve automated service selection [5]. However, while SWS frameworks strive for fully automated service brokerage, current tools and technologies do not facilitate that vision and allow only a very limited degree of actual automation. Still, the execution-oriented nature of WSMO/IRS-III provided a number of benefits when dealing with highly heterogeneous services. For instance, NoTube benefited from applying our rule-based definition of lifting- and lowering mechanisms [11] to map between heterogeneous service input and output schemas – e.g., based on JSON, RDF or XML – and the knowledge-level representations of services, to allow some further reasoning-based processing of data.

However, while our integrative approach proved useful in the sense that it supported required services discovery and automation scenarios within NoTube, maintaining services models following two distinct representation approaches turned out to be a costly task triggering the need for further investigation.

5 Conclusions

We have described a two-stage approach to semantic service representation and reasoning, aiming at a combination of the strengths of two distinctive methods – lightweight Linked Services and more heavyweight broker-based SWS automation – into a coherent SWS framework. The paper argued that by integrating collaborative and user-driven Web-scale service annotations with comprehensive SWS specifications, application developers benefit from both low cost for providing annotations and a high level of automation. In that, while taking advantage of service

[9] http://watson.kmi.open.ac.uk/WatsonWUI/

models produced by a large non-expert audience, both structured search for service instances by humans as well as automation of service tasks is supported to some extent.

In our vision, integration between lightweight service annotations and comprehensive SWS specifications is achieved by different means of (a) model cross-referencing and (b) model transformation and augmentation. Based on this vision we proposed a consistent approach of integrating a set of SWS-related tools and service models aiming at interoperability between lightweight service annotations and heavyweight service specifications. Besides, an application of our approach within the EU research project NoTube was presented as a proof-of-concept prototype.

While the current solution provides an overall framework for integrated service models which support different levels of automation, future work needs to investigate automated model transformation mechanisms in order to support the seamless integration of instances across distinct service models schemas. However, it might be argued, that there exists only an insufficient overlap between MSM and WSMO which does not support a more automated means of transformation as such. Besides, as mentioned above, maintaining services models following two distinct representation paradigms leads to additional effort. As additional downside, we like to point out that existing SWS brokerage technologies, such as IRS-III, support automation only to a certain extent.

In these respects, our future work will also investigate on (a) different levels of services automation, ranging from simple I/O matchmaking to capability matchmaking and execution handling, (b) their feasibility and usefulness and (c) possibilities to extend light-weight approaches, such as MSM, in order to support higher levels of automation.

6 Acknowledgments

This work is partly funded by the European project NoTube. The authors would like to thank the European Commission as well as all partners of the NoTube project for their support.

7 References

[1] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. (2008). Dbpedia: A nucleus for a web of open data. In Proceedings of 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference (ISWC+ASWC 2007), pages 722–735.

[2] Berners-Lee, T., Hendler, J., Lassila, O (2001). "The Semantic Web". Scientific American Magazine. retrieved March 29, 2009.

[3] Bizer, C., T. Heath, et al. (2009). "Linked data - The Story So Far." Special Issue on Linked data, International Journal on Semantic Web and Information Systems (IJSWIS).

[4] Davies, J., Domingue, J., Pedrinaci, C., Fensel, D., Gonzalez-Cabero, R., Potter, M., Richardson, M., and Stincic, S. (2009). Towards the open service web. BT Technology Journal, 26(2).

[5] Dietze, S., Benn, N., Domingue, J., Conconi, A., and Cattaneo, F.: "Interoperable Multimedia Metadata through Similarity-based Semantic Web Service Discovery". In Proceedings of 4th International Conference on Semantic and Digital Media Technologies (SAMT '09), 2--4 December 2009, Graz, Austria.

[6] Dietze, S., Benn, N., Domingue, J., Conconi, A., and Cattaneo, F. (2009) Two-Fold Semantic Web Service Matchmaking – Applying Ontology Mapping for Service Discovery, 4th Asian Semantic Web Conference, Shanghai, China.

[7] Domingue, J., Cabral, L., Galizia, S., Tanasescu, V., Gugliotta, A., Norton, B., Pedrinaci, C.: "IRS-III: A broker-based approach to Semantic Web Services", Jounal of Web Semant, pp. 109-132. Elsevier Science Publishers B. V, 2008.

[8] Farrell, J., and Lausen, H. 2007. Semantic Annotations for WSDL and XML Schema. http://www.w3.org/TR/sawsdl/. W3C Candidate Recommendation 26 January 2007.

[9] Fensel, D.; Lausen, H.; Polleres, A.; de Bruijn, J.; Stollberg,, M.; Roman, D.; and Domingue, J. 2007. Enabling Semantic Web Services: The Web Service Modeling Ontology. Springer.

[10] Kopecky, J.; Vitvar, T.; and Gomadam, K. 2008. MicroWSMO. Deliverable, Conceptual Models for Services Working Group, URL: http://cms-wg.sti2.org/TR/d12/v0.1/20090310/d12v01_20090310.pdf.

[11] Lambert, D., and Domingue, J. (2008) Grounding semantic web services with rules, Workshop: Semantic Web Applications and Perspectives, Rome, Italy

[12] Maleshkova, M., Pedrinaci, C., and Domingue, J. (2009). Supporting the creation of semantic restful service descriptions. In Workshop: Service Matchmaking and Resource Retrieval in the Semantic Web (SMR2) at 8th International Semantic Web Conference.

[13] Maleshkova, M., Kopecky, J., and Pedrinaci, C. (2009). Adapting SAWSDL for semantic annotations of restful services. In Workshop: Beyond SAWSDL at OnTheMove Federated Conferences & Workshops.

[14] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., and Sycara, K. (2004). OWL-S: Semantic Markup for Web Services. Member submission, W3C. W3C Member Submission 22 November 2004.

[15] Motta, E., Reusable Components For Knowledge Modelling: Case Studies in Parametric Design Problem Solving. IOS Press, ISBN I 58603 003 5, 1999.

[16] Pedrinaci, C., Domingue, J., and Reto Krummenacher (2010) Services and the Web of Data: An Unexploited Symbiosis, Workshop: Linked AI: AAAI Spring Symposium "Linked data Meets Artificial Intelligence".

[17] Pedrinaci, C., Liu, D., Maleshkova, M., Lambert, D., Kopecky, J., and Domingue, J. (2010) iServe: a Linked Services Publishing Platform, Workshop: Ontology Repositories and Editors for the Semantic Web at 7th Extended Semantic Web Conference.

[18] Sheth, A. P., Gomadam, K., and Ranabahu, A. (2008). Semantics enhanced services: Meteor-s, SAWSDL and SA-REST. IEEE Data Eng. Bul l., 31(3):8–12.

[19] Vitvar, T.; Kopecky, J.; Viskova, J.; and Fensel, D. 2008. Wsmo-lite annotations for web services. In Hauswirth, M.; Koubarakis, M.; and Bechhofer, S., eds., Proceedings of the 5th European SemanticWeb Conference, LNCS. Berlin, Heidelberg: Springer Verlag.

[20] WSMO Working Group (2004), D2v1.0: Web service Modeling Ontology (WSMO). WSMO Working Draft, (2004). (http://www.wsmo.org/2004/d2/v1.0/).

[21] World Wide Web Consortium, W3C: Simple Object Access Protocol, SOAP, Version 1.2 Part 0: Primer, (2003). (http://www.w3.org/TR/soap12-part0/).

[22] World Wide Web Consortium, W3C: WSDL: Web services Description Language (WSDL) 1.1, (2001). (http://www.w3.org/TR/2001/NOTE-wsdl20010315)

An Interest-based Offer Evaluation System for Semantic Matchmakers

Samira Sadaoui and Wei Jiang

Computer Science Department,
University of Regina, Regina, SK, Canada S4S 0A2

{sadaouis, jiang20w}@uregina.ca

Abstract. Matchmaking systems failed to provide the best matched results to individuals. Semantic matchmaking can help the buyer find the requested offers but it is not good enough to find the best offer. In this work, we propose a system that evaluates and sorts the request-matched offers according to the buyers' interests and tastes. To evaluate the offers, we modify the MultiNomial Logit model to produce an interest model that analyzes individual's interests and favors. Our system captures the buyer's interests, builds his interest model, and then returns the best offer. The best offer denotes the highest interest value to the buyer. Through a case study, we present in detail the phases of our offer evaluation process.

Keywords: Interest model, multiple offer attributes, best matched offer, Self Organizing Map (SOM), MultiNomial Logit (MNL).

1 Introduction

Matchmaking is the online process through which buyers and sellers trade goods or services. Most of the matchmaking systems are semantic-based. Research on ontology lead the early semantic matchmaking systems to understand and process the purchasing requests much better [1, 2, 3]. Nevertheless, with the blooming of e-commerce and e-services, buyers can obtain more and more request-matched offers. It is really time consuming for buyers to browse, evaluate and sort all the candidate offers in order to find the best offer. Today determining the best offer is more important than before for any matchmaker. Recent matchmakers are trying to determine the best offer by using the semantic ranking [4, 5, 6, 7, 8]. Most semantic ranking algorithms examine the similarity of inputs, outputs, preconditions and effects. Yet all existing matchmakers failed to bring the best matched results to individuals. Indeed, they do no guarantee that the best offer will be purchased by the buyer since his specific interests and tastes are ignored. Without studying human interests and only relaying on the semantic matching and ranking, matchmakers cannot recognize the differences between buyers' favors and needs.

Researchers realized that a better matchmaking system "*could quicken the trend toward personalization*" [9]. Matchmaking based on semantic can help the buyer find

the requested offers but it is not good enough to find the best offer. Consequently, we develop a system that evaluates and sorts the request-matched offers according to the buyers' specific interests and tastes. In our system, the best offer denotes the highest interest value to the buyer. Analyzing individual interests along with the semantic matching brings better results to each individual buyer. As illustrated in Figure 1, first the buyer submits to our system a purchasing request which is then sent to a connected semantic matchmaker. The latter returns a list of request-matched offers. To sort these offers according to the buyer's interests, our system performs the following tasks: cluster the values of each offer attribute, interact with the buyer to take into account his interests and tastes, calculate the attribute's interest weight and interest rate, build the interest model based on interest weights and rates, evaluate and sort the offers according to the buyer's interest model.

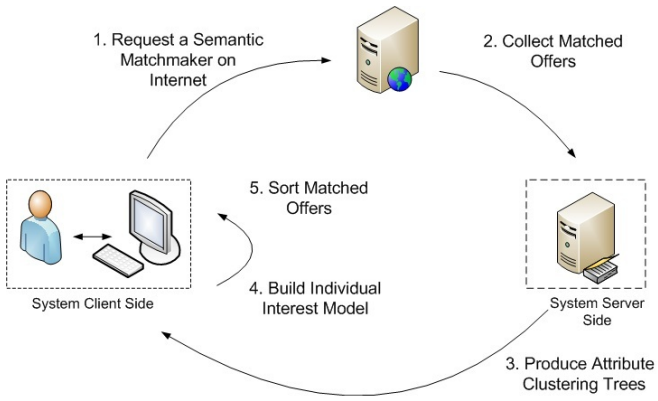

Fig. 1. System Process Overview

In order to take into account individual's interests, we need to utilize the clustering technology called Self Organizing Map (SOM) [10, 11]. As a neural-network approach, SOM is employed to cluster high-dimensional inputs onto lower-dimensional outputs. The reason of using SOM in our work is that the offer attributes may be complex or contain high-dimensional data, such as the attributes of our case study. Furthermore, to define the interest model specifically for each buyer, we modify the MultiNomial Logit (MNL) model [12]. MNL is widely used in commerce to study human shopping behaviors [13]. Nevertheless, MNL suggests a model for a group of people and needs an appropriate sample data. The good thing is that we are able to modify MNL to analyze individual's needs alone and without considering a sample data.

2 Related Work

In the early time of e-commerce, matchmakers focused on mapping the offer attribute values [14]. Requests and offers can be expressed in different schemas and words

even when representing the same semantic meaning. This causes matchmakers to be blind to some potential offers. To solve this problem several semantic matching models, based on ontological technologies, have been proposed [1, 2, 3]. In [1], matching the offers is based on the similarity of the request and services. [2] uses logical relationships to map the offers with the request. [3] focuses on the semantic matching using a platform-independent framework called UDDI. These matchmakers return an unranked list of offers.

To find the best offer, recent matchmakers evaluate service description [4, 5], service constraints [6], service process [7] or both [8]. The best offer denotes the highest semantic matching degree. These matchmakers map the functional properties of offers with the request's functional description. However, these matchmakers cannot explain why sometimes a buyer prefers an offer different from the returned best offer. To address this issue, non-functional matching methods [15, 16] have been introduced by following a matching standard like Qos [17]. These papers argue that the buyer's choice is caused by other criteria often referred to as non-functional properties.

All these matchmakers are based on matching the query words but not on matching individual's interests. Our goal is to build the interest model specifically for the buyer and then find the best matched offer which is the closest to the buyer's real needs.

3 An Example

Our case study consists of purchasing computers based on 2-dimensional attributes: CPU and Price. We have here a company which submits the following inventory query:

> *Request computers with CPU > 1.5 GHz, Price < $3500 for the first ten purchased computers, and Price < $3000 for the next ten.*

To formulate the requests and offers, matchmakers utilize well known web service languages, such as WSDL, SWSL and WSML, which offer a high degree of flexibility and expressiveness. Thus, we translate the purchasing request to for example WSDL following the structure defined in [18] (cf. Figure 2). We assume the connected semantic matchmaker returns the candidate offers given in Table 1 where CPU contains high dimensional values and Price has two ranges.

In the next phases, our system will evaluate all the candidate offers of Table 1 in order to help the company find the best supplier which might become its long-term business partner.

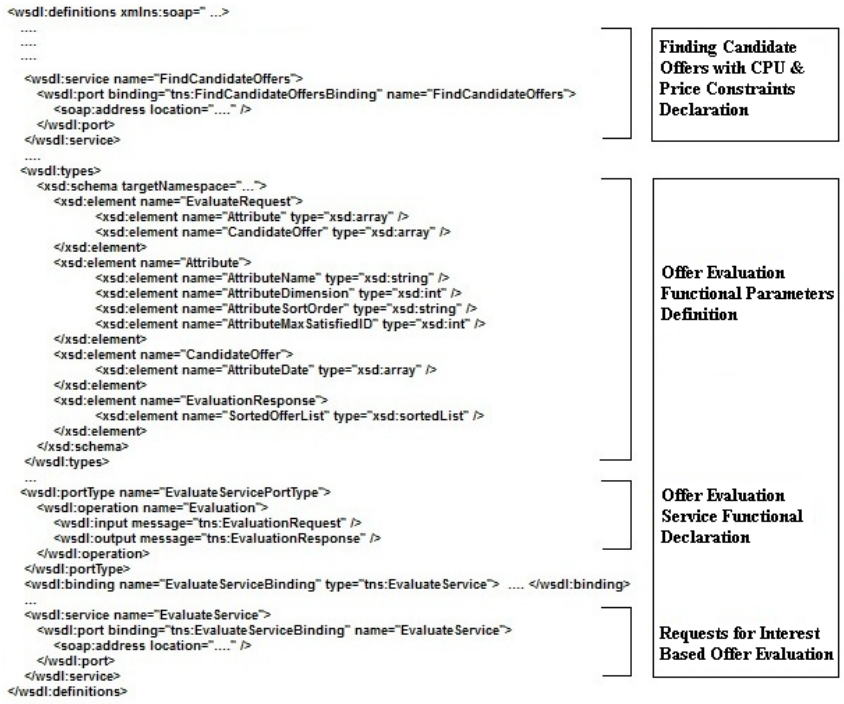

Fig. 2. Formatting the Purchasing Request

Table 1. Candidate Offers

Supplier ID	CPU (GHz)	PriceRange1 ($) For the First 10 Items	PriceRange2 ($) For Item 11 to 20
1	2.5	1100	799
2	2.2	470	370
3	2.5	600	500
4	2.33	1100	800
5	2.4	999	799
6	2.5	1030	830
7	2.66	2500	2200
8	2.3	1000	880
9	2.2	420	400
10	2.4	950	900
11	2.8	1200	1150
12	(1.9, 1.9)	800	700
13	(3.0, 3.0)	2900	2600
14	(1.8, 1.8)	680	680
15	(3.2, 3.2)	3200	2500

4 Attribute Data Clustering

Our system first determines all the attributes from the purchasing request. For each attribute, it extracts all its values from the candidate offers and stores them in a single table. Our system can now cluster the values of each attribute. The purpose of this clustering is to be able to take into account the buyer's interests. So the buyer can select one of the clustering to represent his most interested area. In Figure 3, we define the clustering function called *ClusterAttributeData()* which is based on the algorithm Self Organizing Map (SOM) [11] given in Figure 4.

We apply SOM to recursively divide a large clustering into three sub-clustering until there are less data in the clustering. During the learning time, a set of Learning Vector Quantizations (LVQs) is tuned towards the input attribute data. SOM applies competitive-learning given in the steps 3.1.1 and 3.1.2 of Figure 4. In SOM function: t is the learning time; *CreateLVQ()* is a function that generates three random LVQ m_i in the range of *DataAttribute*; *alpha (t)* controls the learning loop and is the learning rate function which is decreased by learning time t; m_c is the closest LVQ to the selected data x; $h_{ci}()$ is the "neighborhood" function that updates LVQ in the learning time [11].

```
void ClusterAttributeData(DataAttribute: Array)
{1. SOM(DataAttribute, A1, A2, A3);
    //Cluster all data into 3 groups
 2. ArrangeClustering(A1, A2, A3);
    //Arrange clustering in ascending order
 3. for i = 1 to 3
    if  (IsLargeEnough(Ai))
         ClusterAttributeData(Ai);
         //Cluster each sub-group}
```

Fig. 3. Clustering Algorithm of Attribute Data

```
void SOM(DataAttribute: Array, A1: Array, A2: Array,
A3:   Array)
{1. var   t = 1;    //Initialize  learning time
 2. CreateLVQ(DataAttribute, m1,m2,m3);//Create LVQ mi
 3. while (alpha(t) is not too small)//Decrease by time
    {3.1 while (PickUpValue (DataAttribute, x))
                //Select x in the data set
      {  3.1.1  ||x - mc||  =   min{||x - mi||};
                //Find closest LVQ mc
         3.1.2  mi(t+1) = mi(t) + hci(t)[x(t)-mi(t)];
                //Update mi during learning
      }
     3.2 t = t+1;//Update time
    }}
```

Fig. 4. SOM Algorithm

Example. Figures 5 and 6 illustrate respectively the CPU and Price clustering (represented as a tree structure).

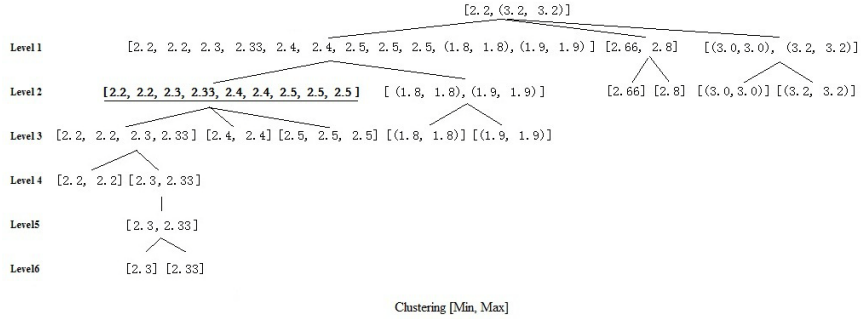

Fig. 5. CPU Data Clustering

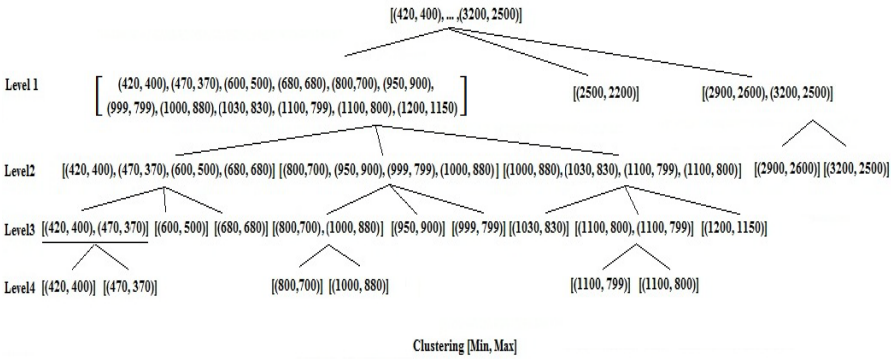

Fig. 6. Price Data Clustering

Our system displays the CPU and Price clustering to the company. The latter can now select the most interested clustering for each attribute. These selections are performed through the GUI of Figure 11. With these selections, our system knows the range and depth of the company's needs. The selection information, representing the company's purchasing interests, will help the system to create the interest model in the following phases.

4 Interest Weight Computation

An interest weight denotes the degree of importance of an attribute in a matching. It is related to the depth and range of the buyer's selection. The smallest and deepest clustering shows the best interest of the buyer. In order to produce the attribute's weight, we use the interest-weight coefficient which contains the buyer's interest in one attribute. We create Formula **(1)** to compute the interest-weight coefficient of an attribute called *k*: *K* is the attribute set, *DataAttribute* is the data range of k, *SelectedClustering* is the buyer's selected clustering for *k*, *SelectedLevelTree* is the selected clustering level, and *TotalLevelTree* is the number of levels of the clustering tree.

$$IW\_coe_k = \frac{DataAttribute_k}{SelectedClustering_k} \cdot \frac{SelectedLevelTree_k}{TotalLevelTree_k} \quad k \in K \tag{1}$$

This coefficient has to be compared with the other attributes' coefficients (cf. Formula **(2)**). An attribute with a larger interest weight coefficient gets a larger interest weight compared to the other attributes. This means we can finally link the interest weights with the buyer's selection.

$$IW_k = \frac{IW\_coe_k}{\sum_{k=1}^{K} IW\_coe_k} \qquad k \in K \tag{2}$$

Example. We suppose the company chooses the CPU clustering [2.2, 2.5]. This selection is at level 2 of the 5-level CPU tree and all CPU data are in the range of [2.2, (3.2, 3.2)]. So, the CPU interest weight coefficient is calculated as 4.4667 with Formula **(1)**. We perform the same calculation for the Price attribute with a coefficient of 44.8125. According to Formula **(2)**, we can get all the attribute interest weights (as shown in Figure 13). For example, CPU interest weight is the following:

$$IW_{CPU} = \frac{4.4667}{4.4667 + 44.8125} \approx 0.0906$$

Based on the interest weights of Figure 11, we can see that Price is much more important than CPU. Such interest feature will help the company to select its best supplier.

5 Interest Rate Function Generation

The goal here is to produce the interest rate of each attribute. To do this, we need to define an interest rate function. A linear function is usually used to measure the attributes' rates [1, 13, 14]. In some cases, linear utility functions cannot assign weights to attributes in order to make an offer as the best one.

In order to solve this problem, we use the un-linear sigmoid function $\varsigma_k(x) = \frac{1}{1+\exp^{-x}}$. We believe the sigmoid function is the closest function to human natural interest change. In our work, we use the sigmoid function to simulate each

attribute interest rate. Here x denotes the value of an attribute and y its interest value. In Figure 7, we show that x of the sigmoid function has less changes in the two intervals [-∞, -2] and [2, +∞]. The Sigmoid function in these two intervals can be considered as a linear function with an acceptable standard error. Meanwhile the interval [-2, 2] is a quickly changeable area. A buyer's selected attribute clustering contains a specific interest for this attribute. In order to represent such interest in our interest rate function, we need to bind the buyer's selected clustering into the quickly changeable interval [-2, 2].

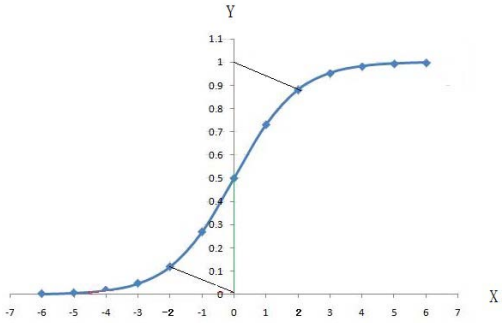

Fig. 7. Sigmoid Function and Value Change

We employ LVQ as the center of our interest rate function since LVQ can be considered as the density center of the attribute data. Attribute data [AL, AR] can be distributed accordingly, and the selected clustering [L, R] is bound into the interval [-2, 2]. LVQ point can be either in the selected clustering [L, R] or outside. If LVQ is inside the clustering, we divide the interest rate function into two functions (cf. Formula **(3)**) where α_L is generated when binding [L, LVQ] into the interval [-2, 0], α_R is generated when binding [LVQ, R] into [2, 0], *Sign* is +1 or -1 w.r.t ascending or descending order of the clustering.

$$\varsigma_{Attribute}(x) = \begin{cases} \dfrac{1}{1+\exp^{[\alpha_L \cdot Sign \cdot (x-\mathrm{LVQ})]}} & x \in [AL, \mathrm{LVQ}] \\ \dfrac{1}{1+\exp^{[\alpha_R \cdot Sign \cdot (x-\mathrm{LVQ})]}} & x \in [\mathrm{LVQ}, AR] \end{cases} \tag{3}$$

$$\alpha L \cdot (x - \mathrm{LVQ}) = -1 \cdot (-2 - 0) \quad \text{when } x = L$$
$$\alpha L = 2 / (L - \mathrm{LVQ})$$
$$\alpha R \,.\, (x - \mathrm{LVQ}) = -1 \cdot (2 - 0) \quad \text{when } x = R$$
$$\alpha R = -2 / (R - \mathrm{LVQ})$$

Example. We can now generate the interest rate functions for CPU and Price by binding all the selected clustering into the quickly changeable area. According to the CPU attribute tree, the selection [2.2, 2.5] has the LVQ of (2.4767, 0.3189). To be

able to process high dimensional values, we use the distance between attribute data and the best attribute data. Here, we believe CPU clustering [3.2, 3.2] is the best data.

Table 2. Distances of High-Dimensional Data

x	BestAttributeData	Distance(x, BestAttributeData)
2.2 ***	(3.2, 3.2)	3.3526
2.5 **	(3.2, 3.2)	3.2757
(2.4767,0.3189) *	(3.2, 3.2)	2.9705

***Min selection value (L), **Max selection value (R), *LVQ

According to the distance calculation in Table 2, LVQ is closer to the best data than any other data in the selected clustering. Consequently, the following interest rate function for attribute CPU has only α_L. Figure 8 displays the company's interest rate for CPU.

$$\varsigma_{CPU}(x) = \frac{1}{1+\exp^{[\alpha_L \cdot Sign \cdot (Distance(x, BestAttributeData) - Distance(\mathrm{LVQ}, \mathrm{BestAttributeData}))]}}$$

$$= \frac{1}{1+\exp^{5.2342 \cdot [Distance(x,(3.2,3.2)) - 2.9705]}} \quad x \in [2.2, (3.2, 3.2)]$$

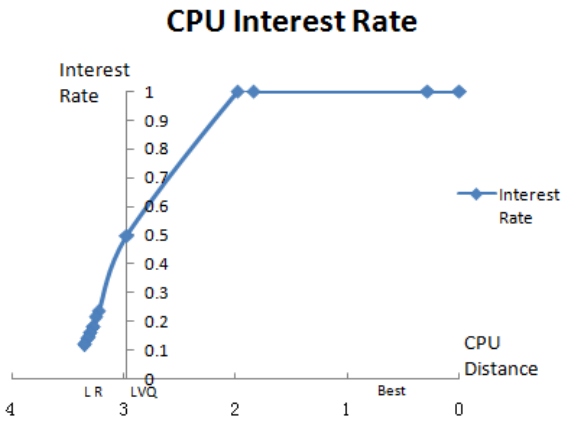

Fig. 8. Interest Rate Function for CPU

We suppose that the company selected the Price clustering [(420, 400), (470, 370)] which is then bound into the area [-2, 2]. After data binding, we produce the following interest rate function where LVQ of 39.88 is its center point. Based on this function, we can easily get the company's interest rate for Price attribute as shown in Figure 9.

$$\varsigma_{Price}(x)=\begin{cases}\dfrac{1}{1+\exp^{[\alpha_L \cdot Sign\cdot(Distance(x,BestAttributeData)-Distance(LVQ,BestAttributeData))]}}\\ \dfrac{1}{1+\exp^{[\alpha_R \cdot Sign\cdot(Distance(x,BestAttributeData)-Distance(LVQ,BestAttributeData))]}}\end{cases}$$

$$=\begin{cases}\dfrac{1}{1+\exp^{0.0502\cdot[Distance(x,(420,400))-39.88]}}\\ \dfrac{1}{1+\exp^{0.1085\cdot[Distance(x,(420,400))-39.88]}}\end{cases} \quad x\in[(420,400),(3200,2500)]$$

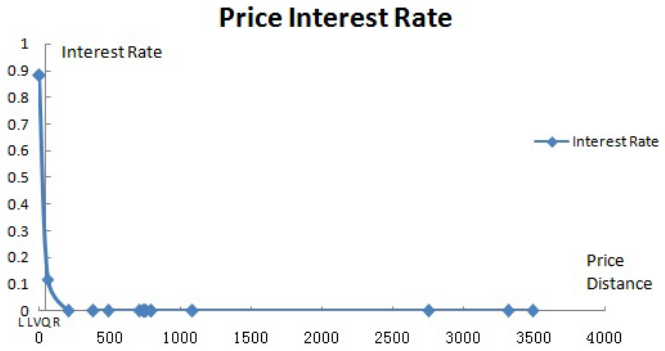

Fig. 9. Interest Rate Function for Price

6 Offer Evaluation with the Interest Model

MNL model expresses the utility for a group of people choosing an item. The utility function for an individual in a population includes the deterministic and random components as follows [13]:

$$U_{nj}=\sum_{k=1}^{K} b_k \cdot x_{njk}+\varepsilon_{nj}, \quad j\in C \tag{4}$$

where

- U_{nj} is the utility for buyer n selecting item j.
- $(b_k \cdot x_{njk})$ *is* the "representative" taste of the population. This component consists of K observed deterministic features x_{njk}; b_k is the weight for each feature x_{njk}.
- ε_{nj} is the individual taste for the item j.
- C is the set of items.

We now adapt the MNL formula to define the interest model for each individual. First we consider the deterministic features as the offer attributes. Furthermore, ε_{nj} can be decomposed into K attributes since ε_{nj} is the total alternative with K features. This means the evaluation of the attributes may be different according to the interests of each individual (cf. Formula **(5)**). At the market level, the individual taste ε is brought into the population model as a random utility. Since it comes from individuals and its value is random, ε is usually removed when users generate the MNL model. However, in our system individual taste becomes important.

$$U_{nj} = \sum_{k=1}^{K} b_k \cdot (x_{njk} + \varepsilon_{njk}) \quad j \in C \tag{5}$$

In order to produce the interest model, our system calculates the interest weights, IW, and simulates the interest rates, IR. IW denotes the interest weight b_k and IR the interest rate value $x_{njk} + \varepsilon_{nj}$. We propose Formula **(6)** to build the interest model IM for each individual. IM is the degree of interest of the buyer purchasing an offer j with K attributes. Based on the consumer theory, an individual seeks to maximize his utility in each purchasing behavior.

$$IM_{(j)} = \sum_{k=1}^{K} (IW_{jk} \cdot IR_{jk}) \quad j \in C \tag{6}$$

Example. After our system gets the interest weights and interest rate functions for the two attributes, it generates the interest model (IM) for the company as follows.

$$IM_{Company}(Supplier) = 0.0906 \cdot \varsigma_{CPU}(CPU) + 0.9094 \cdot \varsigma_{Price}(Price)$$

For example, we show below how the interest model calculates the interests for the first two suppliers:

$$\begin{aligned} IM_{Company}(Supplier1) &= 0.0906 \cdot \varsigma_{CPU}(2.5) + 0.9094 \cdot \varsigma_{Price}[(1100,799)] \\ &= 0.0906 \cdot 0.1684 + 0.9094 \cdot 6.2583E\text{-}36 \\ &= \mathbf{0.0189} \\ IM_{Company}(Supplier2) &= 0.0906 \cdot \varsigma_{CPU}(2.2) + 0.9094 \cdot \varsigma_{Price}[(470,370)] \\ &= 0.0906 \cdot 0.1192 + 0.9094 \cdot 0.1192 \\ &= \mathbf{0.1192} \end{aligned}$$

So, we got an interest rate of 0.0189 for Supplier1's offer and 0.1192 for Supplier2's offer. Based on these values, we can conclude that Supplier2 has a higher chance than Supplier1 to be the company's partner. With our interest model, we can evaluate all the candidate offers of Table 1. Table 3 shows that Supplier9 is the best supplier for the company.

Table 3. Sorted Offers for the Company

Supplier ID	CPU (GHz)	PriceRange1	PriceRange2	Interest
9*	2.2	420	400	0.8118
2	2.2	470	370	0.1192
12	(1.9, 1.9) *	800	700	0.0152
13	(3.0, 3.0) *	2900	2600	0.0127
14	(1.8, 1.8)*	680	680	0.0137
15	(3.2, 3.2)*	3200	2500	0.0152
11	2.8	1200	1150	0.0173
7	2.66	2500	2200	0.0122
3	2.5	600	500	0.0152
6	2.5	1030	830	0.0137
1	2.5	1100	799	0.0189
5	2.4	999	799	0.0903
10	2.4	950	900	0.0906
4	2.33	1100	800	0.0900
8	2.3	1000	880	0.0906

**: best offer with the max interest degree*

7 Design and Implementation

We developed our system with a distributed architecture as illustrated in Figure 10. The buyer interacts with the client side via the *GUI*. After the buyer submits his request, the *GUI* passes it to the connected semantic matchmaker. On the server side, the *OfferManager* component: (1) collects the request-matched offers and stores them in the *ContentOffer* database, (2) analyzes the request and offers to extract the attributes and their values, and (3) stores them in the *OfferAttribute* database. For each attribute, the *AttributeDataClustering* component clusters its values and sends its clustering tree to the client side. *GUI* helps the buyer to select the most interested clustering (cf. Figure 11). Once the buyer's selection is completed, the *InterestModelCreator* component is called to build the buyer's interest model. It first passes all the selected clustering and the whole attribute clustering trees to *InterestWeightCalculator* and *InterestRateFunctionCreator*. For each attribute, *InterestWeightCalculator* returns the interest weight coefficient and interest weight (cf. Figure 12), and *InterestRateFunctionCreator* the interest rate function. The *OfferEvaluator* component applies the generated interest model on the candidate offers, and returns to the buyer the list of offers sorted by interests. In Figure 13, we show for instance the offer evaluation process on the client side.

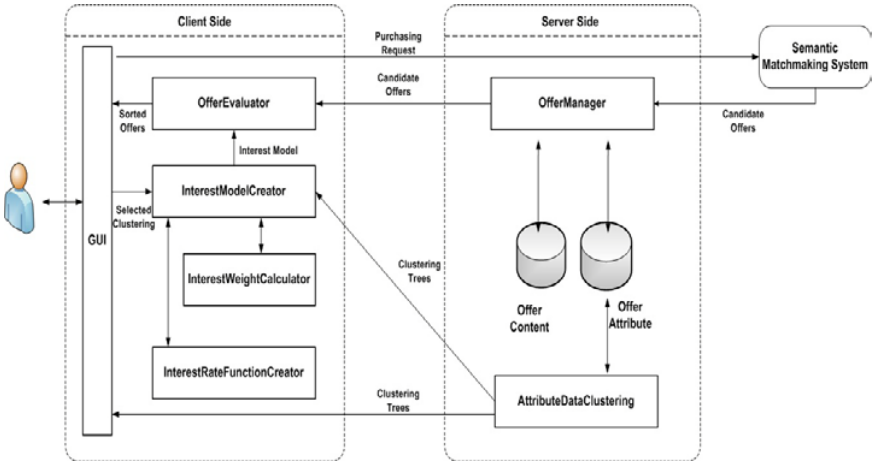

Fig. 10. System Top-Level Architecture

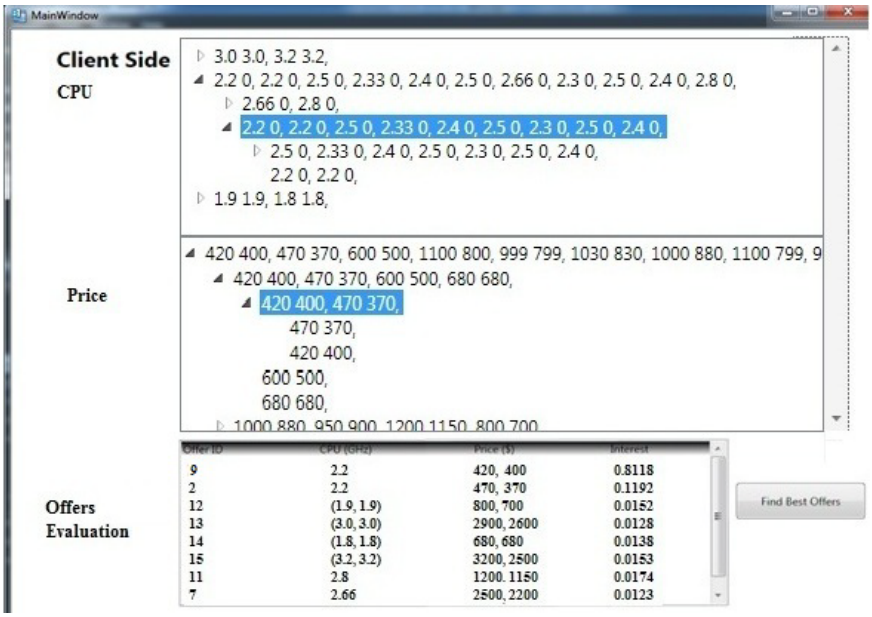

Fig. 11. Selecting CPU and Price clustering

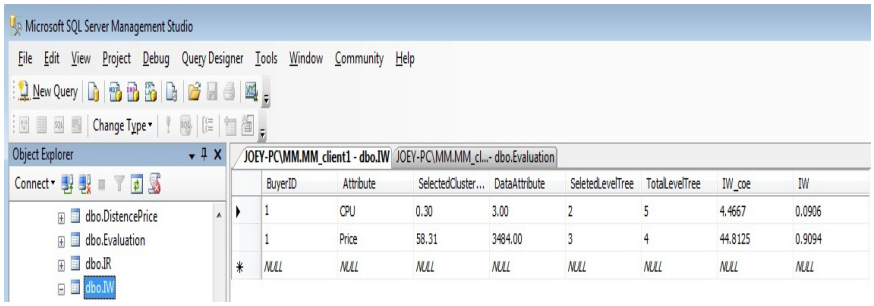

Fig. 12. Calculating the Interest Weights for CPU and Price

We implemented our system using Visual Studio C# (on the .net 3.5 framework) and the SQL Server 2008. Figures 14 and 15 show the class diagrams of the client and server side programs. We created two separate databases sources, called *serverDB* and *clientDB*, to support server and client side programs. These two classes contain all the necessary functions about database processing and data binding. The other classes are created with a window interface by using software MS Blend3, the interface developing tool for Windows form application. The classes *ServerWindow* and *ClientWindow* contain multi-thread and network communication functions.

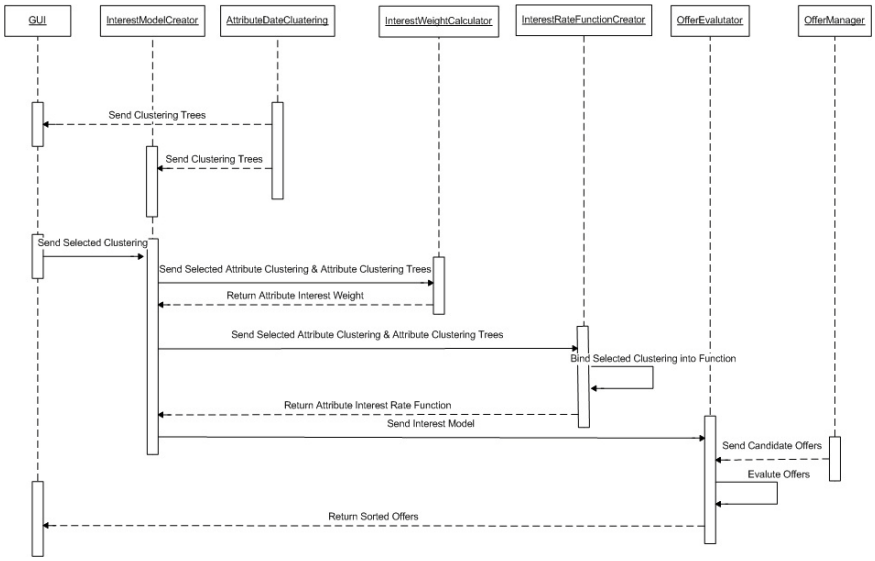

Fig. 13. Client Side: Offer Evaluation

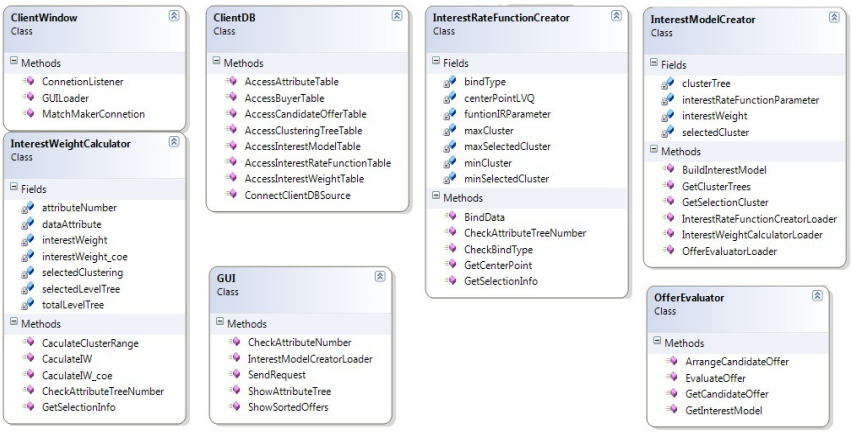

Fig. 14. Server Side Class Diagram

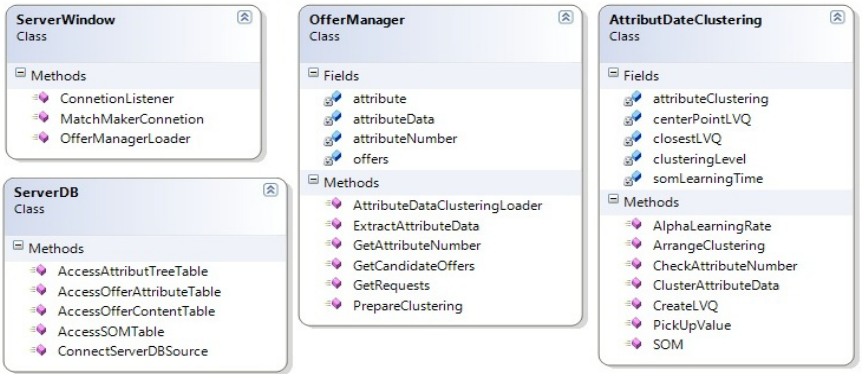

Fig. 15. Client Side Class Diagram

8 CONCLUSION AND FUTURE WORK

In this paper, we showed the benefits of sorting the request-matched offers according to the buyer's interests and needs. Our interest model provides a solution to existing matchmaking systems and avoids the linear matching problems. Adopting an economic method, we produced a simple and automated model to determine the best matched offer based on the buyer's selections.

One possible direction of this work is to include the interest learning [19] in our system. The main purpose of this learning is to update the interest model to fit the buyer's interests instantly. A learned interest model will be able to determine the best offer in these two situations: the buyer shifts his interests, or new offers are added in our database.

References

1. Hahn, C., Nesbigall, S., Warwas, S., Zinnikus, I., Klusch, M., and Fischer, K. Model-Driven Approach to the Integration of Multi-Agent Systems and Semantic Web Services. Enterprise Distributed Object Computing Conference Workshops. IEEE. 314-324 (Sept. 2008)
2. Wang, H., and Li. Z. Z.: A Semantic Matchmaking Method of Web Services Based on SHOIN^+ (D)*. Services Computing. IEEE. 26-33 (Dec. 2006)
3. Kawamura, T., Hasegawa, T., Ohsuga, A., Paolucci, M., and Sycara, K.: Web services lookup: a matchmaker experiment. IT Professional. IEEE. Vol. 7, No. 2, 36-41 (Mar-Apr 2005)
4. Qiu, T., and Li, P. F.: Web Service Discovery Based on Semantic Matchmaking with UDDI. In Proceedings of the The 9th International Conference. Young Computer Scientists. 1229-1234 (Nov. 2008)
5. Bai, D. W., Liu, C. C., Peng, Y. and Chen, J. L.: Web Services Matchmaking with Incremental Semantic Precision. Wireless Communications, Networking and Mobile Computing. IEEE. 1-4 (Sept. 2006)
6. Bai, D., Fei, A. G., and Cai, S. F.: Semantic Matchmaking of Web Services Constraint Conditions. Wireless Communications, Networking and Mobile Computing. IEEE. 1-5 (Sept.2009)
7. Huang, R., Zhuang, Y. W., Zhou, J. L., and Cao, Q. Y.: Semantic Web-based Context-aware Service Selection in Task-computing. In Proceedings of the WMSO '08 International Workshop. 97-101 (Dec. 2008)
8. Bellur, U., and Vadodaria, H.: Web Service Ranking Using Semantic Profile Information. In Proceedings of the ICWS 2009. IEEE. 872-879 (July 2009)
9. Essex, D.: Matchmaker, matchmaker. ACM Communications, ACM. Vol. 52, 16-17 (May 2009),
10. Kohonen, T.: The Self-Organizing Map, 3rd Edition. Springer (2001)
11. Chen, Y. Y., and Young. K. Y.: Applying SOM as a Search Mechanism for Dynamic System. Decision and Control, IEEE. 4111- 4116 (Dec. 2005)
12. McFadden, D.: Economic Choices. American Economic Association. American Economic Review. Vol. 91, No. 3, 351-378 (Jun. 2001)
13. McFadden, D., and Zarembka, P.: Conditional logit analysis of qualitative choice behavior. Academic Press. Frontiers in Econometrics. 105-142 (1974)
14. Ha, S. H., and Park, S. C.: Matching buyers and suppliers: an intelligent dynamic exchange model. IEEE. Intelligent Systems. Vol. 16, No. 4, 28- 40 (Jul-Aug 2001)
15. Wang, X., Vitvar, T., Kerrigan, M., and Toma, I.: A QoS-aware selection model for semantic web services. In Proceedings of the 4th Int. Conference on Service-Oriented Computing. 390–401 (2006)
16. Yu, Q., and Reiff-Marganiec, S.: Non-functional property based service selection: a survey and classification of approaches. In Proceedings of the Non Functional Properties and Service Level Agreements in SOC Workshop (Nov. 2008)
17. QoS for Web Services: Requirement and Possible Approaches. http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/
18. Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl
19. Wei, Y.Z., Moreau, L., Jennings, N.R.: Learning users' interests by quality classification in market-based recommender systems. Knowledge and Data Engineering. IEEE Transactions. Vol.17, No. 12, 1678- 1688 (Dec. 2005)

Anatomy of a Semantic Web-enabled Knowledge-based Recommender System

Daniele Dell'Aglio, Irene Celino, and Dario Cerizza

CEFRIEL – Politecnico of Milano, Via Fucini 2, 20133 Milano, Italy
{name.surname}@cefriel.it

Abstract. Knowledge-based Recommender Systems suggest to users items of their interest, on the basis of some understanding of both items' characteristics and users' profiles. In order to properly work, this kind of recommender systems need a thorough modeling of items and users; the usual barrier to their development is, therefore, the availability of the necessary knowledge and its maintenance over time.
With this respect, Semantic Web technologies can be of great help: not only knowledge technologies and languages can be employed to build the knowledge base, but the large availability of open and linked data about a growing variety of fields and topics, published on the Web of Data, further simplifies the modeling step for recommender systems.
In this paper, we present our concept of Semantic Web-enabled Recommender System, based on the retrieval from the linked data Web of the necessary pieces of knowledge about items and users. We illustrate the general structure of this new family of Knowledge-based Recommender Systems and we explain how we concretely followed this approach to develop a tool to recommend Web services in the context of the SOA4All project. We also offer our considerations about the strengths, the current limitations and the possible extensions of our proposal.

1 Introduction and Motivation

Recommender systems are becoming more and more commonly used to help users serendipitously find items they were (implicitly or explicitly) looking for. From a user's point of view, recommendations are seen as suggestions that are proactively provided by the system, in a timely fashion. In order to be effectively useful, the recommendations should be accurate, as to "foresee" a user's needs.

Recommender systems are usually classified by the recommendation technique they use [9]:

- *Collaborative Filtering Recommender Systems* [30]: given a user, they find users with similar behavior to predict items of interest;
- *Content-based Recommender Systems* [25]: they usually employ a classifier to predict items' similarity;
- *Demographic Recommender Systems*: they compute users' similarity using demographic information (age, location, etc.);
- *Knowledge-based Recommender Systems* [8]: they build a knowledge base with a model of the users and/or items in order to apply inference techniques and find matches between users' need and items' features.

Additionally, another category of systems, the *Hybrid Recommender Systems* [9], tries to join the advantages of two or more techniques described above. In this paper we focus on the Knowledge-based Recommender Systems.

Knowledge-based Recommender Systems offers some advantages with respect to the other techniques. First, they need a *minimal amount of users*, i.e. they do not require a huge amount of data to compute recommendations, differently from other classes of Recommender Systems; moreover, they do not suffer the so-called *cold start problem*: when a new user/item is added with its description, the system is immediately able to compute recommendations for the new user/item; finally, by their very nature, they are able to generate *proofs for the recommendations*, i.e. they are able to "explain" the motivation behind an item proposal on the basis of the user/item modeling at disposal.

The main drawback of Knowledge-based Recommender Systems, however, consists in the modeling, building and maintenance of the knowledge base: a correct and up-to-date description of items to be recommended, as well as of users to which provide suggestions must be ensured, in order to guarantee a high level of recall and precision of the generated recommendations. Several elements can change over time: new users and new items can be added, old users and old items can require an updated description, the domain knowledge can evolve or be modified to take into consideration new features, the set of policies defined to compute recommendations can be revised to better meet users' needs and requirements, and so on.

This knowledge base creation and maintenance require a lot of effort, with a heavy human intervention. A special attention must also be given to assure the consistency, quality and reliability of the modeled knowledge. Therefore, in this context, it is quite natural to think about applying technologies and tools coming from the Semantic Web community to help and support this phase. Some efforts in this directions were already explored (as we report in Section 5); still, a comprehensive work to design a Semantic Web-enabled Knowledge-based Recommender System is missing.

In this paper, we present a holistic approach to apply Semantic Web technologies and the knowledge coming from the Web of Data to support and enhance the knowledge base modeling as well as all other phases of the recommender life cycle. Indeed, we believe that the recent availability of large amounts of linked data can definitely offer new opportunities to build innovative and enriched recommender systems. Moreover, the widespread uptake of Semantic Web standards – RDF [15], SPARQL [29] and also the recently published RIF [6] W3C Recommendation – provides new favorable circumstances to improve recommendation algorithms and tools, by leveraging on users' and items' semantics.

The rest of the paper is structured as follows: in Section 2 we describe our concept for a *Semantic Web-enabled Recommender System* and we introduce the scenario in which we applied our proposed approach to recommend Web services, in the context of the SOA4All project. The details of our approach are offered in Section 3, in which we explain how the Web of Data can be leveraged to build a knowledge base, and in Section 4, where we illustrate how Semantic

Web technologies can help in computing recommendations; those sections include also the description of the actual realization of our conceptual approach in the Web service recommendation scenario. Finally, in Section 5, we introduce some previous works to combine Semantic Web technologies with Recommender Systems and in Section 6 we conclude with our considerations about open issues and current limitations and we provide hints for future extensions.

2 Our Concept of Semantic Web-enabled Recommender System

For the last years, the Semantic Web community has been working to realize the *Web of Data*, i.e. to publish structured information and datasets on the Web and interlinking them. Thanks to the popularity of Tim Berners-Lee linked data meme [3] and to the growing interest and coordinated effort of the Web community, a first nucleus of this Web of Data has been built and constantly updated and enriched since early 2007 to constitute the so-called "Linking Open Data dataset cloud" or simply *LOD Cloud*[1], which comprises around 4.7 billion RDF statements, connected by 142 million RDF links (as of May 2009 [4]).

We believe that the Web of Data should be considered as an interesting source of information to be used by Knowledge-based Recommender Systems. The LOD Cloud is a huge public source where information can be found to describe several kinds of items, users and domains. Accessing and exploiting the Web of Data can allow the partial automation of the knowledge base creation and maintenance, simplifying the modeling and profiling of items and users. Furthermore, the computation tasks to generate recommendations, operating on the knowledge base, can be performed and enhanced by the use of Semantic Web tools like SPARQL processors, reasoners or rule-based systems.

In this paper we propose a holistic approach to design and to realize a Knowledge-based Recommender System using Semantic Web technologies. In Section 2.1 we explain the high level architecture of such a system, which will be detailed in Sections 3 and 4. The scenario in which we demonstrate the applicability of our approach is introduced in Section 2.2; we will use this application scenario also in the following sections to explain how we realized in practice a prototype of our Semantic Web-enabled Recommender System.

2.1 High-level Architecture of a Semantic Web-enabled Recommender System

A recommender system is usually part of a more complex application (like a Web site), so it exchanges data with other components. It provides its functionalities to a set of known users about a predefined set of items; both those sets can be modified over time (users and items can be removed or added).

Our concept of recommender system processes information from both private and public sources. We call *private data* the information about users and items generated by other parts of the application; with *public data* we identify the data published and freely accessible on the Web of Data.

[1] Cf. `http://lod-cloud.net/`.

Figure 1 shows the schematic high-level architecture of our Semantic Web-enabled Recommender System, that consists of two main components, the Model Builder and the Recommendation Engine.

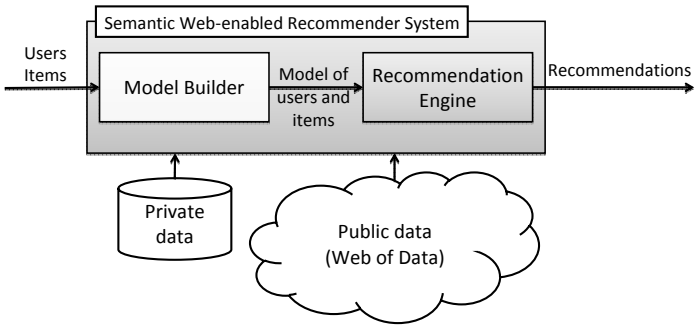

Fig. 1. Schema of our Semantic Web-enabled Recommender System

The *Model Builder* is related to the knowledge base building and maintenance; its goal is to create a model containing the descriptions of items' and users' profiles. Additionally, this model should contain information about how users (and their interests) are related to items (and their features). In order to build such a model, this component queries and interacts with the available data sources, both private and public ones, and identifies the knowledge "bits" useful to describe the items to be recommended and the users. For what regards the items, the Model Builder should look for available descriptions, categorizations and classifications, reviews and ratings, related entities, etc.; it should also reconstruct the users' profiles by retrieving useful information about their tastes and their preferences.

The second component, the *Recommendation Engine*, is devoted to the computation of item suggestions; it receives as input the model generated by the Model Builder and analyzes this knowledge to find semantic connections between users and items. The assumption is that, if a user profile can be connected to an item description by a set of semantic links, this means that that item is a good candidate for recommendation. Still, the component should also check for specific characteristics of those connection "paths" in order to compute a score – the so-called *utility value* – that represents the system's degree of confidence in the usefulness of such recommendation for the user.

2.2 Application Scenario: Recommending Web Services

SOA4All[2] is a European research project aimed to provide a comprehensive framework that integrates SOA, context management, Web principles, Web 2.0 and semantic technologies into a service delivery platform. SOA4All tools can be used by developers to discover, compose and reuse services in order to build a new application. In this scenario, the users of the system are developers looking

[2] Cf. `http://www.soa4all.eu/`.

for services, which are the items to be recommended. For example, a developer who wants to build an e-commerce website is interested in finding pre-existing services to be reused, like a shopping cart manager or a payment service.

Within the SOA4All project, we applied the approach sketched above to realize a Semantic Web-enabled Recommender System[3] to suggest developers (users) services of their interest (recommended items). Developers are identified by their OpenID, which is an unambiguous Web identifier; by using this identifier, the Model Builder retrieves a user description from the Web of Data which complements the "private" data about the user. Moreover, the services to be recommended have their own URIs to identify them and are semantically annotated with a categorization ontology; this lets the Model Builder find related information. Finally, the Recommendation Engine processes the SOA4All users' profiles and the services' description to find if and how they are semantically connected; if such connections exist, the engine computes their utility values in order to detect which services can be interesting for the system users.

It is worth noting that, since our Recommender System is a Semantic Web application that accesses distributed linked data sources, we developed it on top of LarKC[4] [11], an integrated and pluggable platform for Web-scale semantic computing. We configured a set of LarKC "workflows" to search for relevant sources on the (Semantic) Web and to interact with them to retrieve the desired data. A LarKC workflow is activated when it receives a request under the form of SPARQL query.

3 Building a Knowledge Base with the Web of Data

The Model Builder creates a model containing users' profiles and items' descriptions by processing the available data sources, both private and public data. This component can be further divided in three parts, as depicted in Figure 2: the User Profiler, the Item Profiler and the Linker.

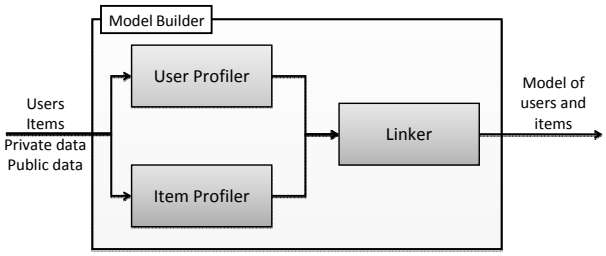

Fig. 2. The Model Builder and its components

The *User Profiler* and the *Item Profiler* respectively build a description of the users and a description of the items; the main task of the Linker is to connect

[3] Cf. `http://etechdemo.cefriel.it/rs-answers-service/`.
[4] Cf. `http://www.larkc.eu/`.

the users to the items, adding the missing data to complete the "paths" between users, their tastes and interests on the one hand, and items and their features on the other hand.

3.1 Generic Entity Profiler

Since both User and Item Profilers aim at building a semantic description of a user/item respectively, to better clarify their role, we firstly describe a generic *Entity Profiler*. Then, in Sections 3.2 and 3.3, we detail the specificities of those two components within the Model Builder.

An Entity Profiler can be characterized by the entities to be described and by some specification of the desired description of those entities, e.g. in terms of the requested entity attributes. To reconstruct the entity description, the Entity Profiler accesses both the private sources and the public data published on the Web of Data; the Entity Profiler queries and traverses those knowledge sources and retrieves the needed data; in case, it interlinks the gathered data to reconstruct a complete model of the entities to be described.

We assume that the entities to profile can be identified by URIs; this kind of identifier can be used to retrieve more information from the Web. We also consider that the entity profile will be expressed in terms of RDF triples and RDF links to external sources. In order to build such a profile, the Entity Profiler can be configured to specify what kind of profile data should be fetched; the basic assumption is a SPARQL query in the form `DESCRIBE <entity-URI>`, but configuring this component could mean defining a detailed `CONSTRUCT` query. In the latter case, the `CONTRUCT` clause could be used to transform the original retrieved data into a common format that is more suitable for the subsequent elaborations. When the entity description is fetched from several heterogeneous sources, this transformation is needed to homogenize the different pieces of data.

For what regards the data sources, the Entity Profiler should first search in the application private data; then it should look for additional data from external sources. Those public sources can either be manually identified or dynamically discovered; in the former case, the Entity Profiler can be configured with a list of known data sources (to be used, for example, in the `FROM` clause of SPARQL queries), while, in the latter case, the component should query a Semantic Web search engine service (like Sindice[5] [22] or Watson[6] [10]) and then interact with the returned list of sources to retrieve the additional information.

A final note about the "storage" of entity profiles: the most natural approach would be to store all the reconstructed profiles locally; this is also the most common approach in existing recommender systems. Growing the amount of data gathered from public sources, however, this could be unfeasible. Being the Web of Data distributed by nature, moving from a dataset to a connected one is always possible by following RDF links. A Semantic Web-enabled Recommender System could therefore have a local knowledge base for the RDF links to external

[5] Sindice `http://sindice.com/`.
[6] Watson `http://watson.kmi.open.ac.uk/`.

public datasets, leaving in their original locations a large part of the public data. Incorporating only the links to the remote locations lets the system keep its knowledge base small and manageable, without hindering the possibility to access the public data on the Web. Of course, a pragmatic solution could include partial local caching of remote data.

3.2 User Profiler

The *User Profiler* is the instantiation of the generic Entity Profiler with the goal of reconstructing the user profile. Usually, recommender systems adopt two different strategies to derive a user profile: by analyzing implicit user feedbacks and by processing explicit user inputs.

The *implicit feedbacks analysis* [14] allows to define users' needs by collecting data about their behavior, for example capturing navigation links, bookmarks and so on. The resulting user profile in this case can be inaccurate, but the advantage is that it is reconstructed without bothering users, e.g. when they cannot or do not want to provide personal details. On the other hand, when users insert specific information about themselves and their preferences, they build an *explicit profile* describing their interests. As a consequence, the system gets a very accurate user profile based on what users voluntarily provide.

Our concept of Semantic Web-enabled Recommender System does not require the adoption of either specific technique. The User Profiler addresses the challenge of *complementing* a user description, derived via implicit/explicit techniques, with additional public information from the Web of Data. Therefore, the following considerations are valid in both scenarios.

The Web of Data allows for a relevant *enrichment* of the profiles, through the discovery of different kinds of information. Not only the User Profiler can look for specific information about users and their tastes; it can also enrich the description of users' needs, by retrieving more detailed descriptions of interest topics. The availability of such information is due to the growing success of social networks; Web users are more and more accustomed to describe their profile on a multitude of different platforms and the Semantic Web community has investigated how to automatically derive a structured user profile from the social Web (e.g. extracting users' interests from their Facebook pages [27]).

In particular, we refer to *FOAF*[7] [7] profile information which, in the LOD Cloud, represents one of the largest datasets. To recommender systems, FOAF can offer useful kinds of information, like relations between users (`foaf:knows` links between profiles) and users' topics of interest (`foaf:interest` property values). Retrieving the FOAF profiles of the recommender system's users can be a successful way to obtain additional data to enrich users' profiles; for example, turning to FOAF data can be very useful when the system has no available information about new users (the so-called user cold start problem). Unfortunately, existing FOAF profiles do not always contain useful information, but we believe that, with the growing adoption of this vocabulary, more and more relevant information about people will be made available on the Web of Data.

[7] Cf. `http://www.foaf-project.org`.

Reconstructing SOA4All developers' profiles. In the scenario of service recommendation introduced in Section 2.2, system users are developers identified by their OpenID. In this context, therefore, we used this identifier to find an available FOAF profile, since FOAF specification defines the inverse functional property `foaf:openid`. If such a profile is found, we retrieve `foaf:interest`s, thus collecting the useful hints to understand users' preferences.

Since we want to build a profile that lets the recommender system provide useful service suggestions, we do not limit the developers' profiling to their FOAF description. Starting from the values of the `foaf:interest` property, we try to identify related DBpedia resources [5]. In this way, as explained later, we pave the way for an easier discovery of semantic connections between developers and the services to be recommended. An example of FOAF profile expressed in N3 retrieved from the Web of Data starting from an OpenID could be the following:

```
<user-uri> foaf:openid   <user-openid> ;
           foaf:interest dbp:ClassicalMusic .
```

Finally, the retrieved profile data are converted into a common format expressed by the Weighted Interest Ontology[8]; this data model lets us keep track of the different interest degree of a user with regards to a topic. Moreover, this ontology could be used to tell apart the different "contexts" of a user's profile: a person could be interested in books in his private life and in music for professional reasons. The example above should therefore be converted as follows:

```
<user-uri> foaf:openid   <user-openid> ;
           wi:preference [
                 rdf:type  wi:WeightedInterest ;
                 wi:topic  dbp:ClassicalMusic ;
                 wi:weight "1" ;
                 wi:context "professional life"
           ] .
```

3.3 Item Profiler

As the User Profiler builds users' description, the *Item Profiler* performs similar actions on the recommendable items. The goal of this component is to reconstruct a description of the items, collecting their relevant features and properties.

Since an item is any entity that could be recommended to the user, the specific kind of information to be retrieved strongly depends on the concrete scenario and cannot be generalized. Nonetheless, in the following we offer some hints on how Semantic Web technologies can help this component. In particular, we focus on enriching the "private data" about items with additional details from the Web of Data; as with users' profiles, we assume each item can be identified by a URI to let the Item Profiler search for related information.

For what regards the data sources to be queried, it is of course hard to indicate a source that can prove useful in any possible case. As explained in

[8] Weighted Interest Ontology `http://xmlns.notu.be/wi/`.

Section 3.1, the location of data sources of interest can be reached through the use of a Semantic Web search engine, by looking for information directly related to the item or relevant with respect to some domain-specific concept or entity.

It can also be advisable, however, to retrieve information from the most common generic sources, like the already cited DBpedia – "the crystallization point for the Web of Data" [5] – or Freebase[9]. The connection of an item description to those broad and common sources, in turn, can result in an easier discovery of semantic connections with users' profiles, as illustrated in Section 3.4.

Reconstructing SOA4All services' descriptions. In the SOA4All scenario, the recommendable items are services. Each service – either SOAP or REST – is identified by a URI and semantically described; the basic service data can be retrieved through the iServe[10] linked data endpoint developed in SOA4All. In particular, services are annotated with regards to the Service-Finder Category ontology[11], which describe general purpose classifications of services. An example of service profile retrieved in such a way is the following:

```
<service-uri> rdf:type sf:Service ;
              sawsdl:modelReference sfcat:Music .
```

As illustrated for user profiling, service description could be turned in a common format suitable for recommendation computing. A possible expressive model is offered by the Service-Finder ontology[12] as follows:

```
<service-uri> rdf:type sf:Service ;
              sf:hasCategoryAnnotation  [
                       sf:hasCategory sfcat:Music ;
                       sf:strength    "1"
              ] .
```

3.4 Linker

The output of the two profilers consists of two RDF graphs (Figure 3), one containing users' profiles and one describing recommendable items and their features. As described in Section 4, computing recommendations is the process of analyzing the relations between users and items, thus, the goal of the *Linker* is to supply the additional RDF triples describing the relations between users (or their interests) and items (or their features). Some of these connections could already be part of items/users profiles; otherwise, the Linker should look for the missing links. In [9], those three models to be connected are named *User Knowledge*, *Catalog Knowledge* (data about items' features) and *Functional Knowledge* (data about the mapping between user interests and items).

To discover useful connections, the Linker could perform two operations on the Web of Data: inference and RDF paths search. Applying *inference* on User

[9] Freebase `http://www.freebase.com/`.
[10] iServe `http://iserve.kmi.open.ac.uk/`.
[11] Cf. `http://www.service-finder.eu/ontologies/ServiceCategories`.
[12] Cf. `http://www.service-finder.eu/ontologies/ServiceOntology`.

and Catalog Knowledge, the Linker could find hidden relations among the described entities. In addition, inference could improve the RDF path search by adding new "starting points" to the data discovery.

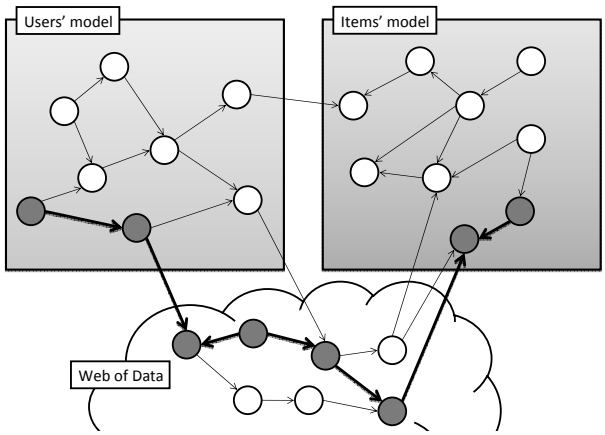

Fig. 3. Linker connection discovery

The *RDF path search* consists in the discovery of the missing links between users' and items' profiles on the Web of data. This means that, if additional connections exist, they could as well be published on the Web of Data. In literature, some works to perform RDF path search on the LOD Cloud already exist.

RelFinder[13] [16, 12] is an application that receives as input two entities and looks for paths among them. It constructs a set of SPARQL queries that ask for a path between the two entities with a predefined orientation and length; then it submits the queries to a SPARQL endpoint to retrieve the paths. RelFinder works if both the entities are accessible through the same endpoint.

SCARLET [28] copes with the problem of looking for connections between two concepts. First it looks for an ontology that contains both concepts: if it exists, the ontology is processed to derive the relation between the concepts (e.g., disjointness or equivalence). If it doesn't, SCARLET recursively looks for a set of ontologies that allows to link the two inputs.

The idea behind SCARLET could be extended from concepts to generic entities. The Linker could act in a similar way: given the two resources for which a connection should be found, it should firstly identify the datasets potentially containing the RDF paths; then, traversing the datasets by following a specific policy (for example using only a set of predefined properties), it should try to find the connections. It is worth noting that an exhaustive search for paths is not an issue, since the Linker's aim is to find useful paths for the computation of the recommendations.

Linking services to SOA4All users. As explained above, services are annotated by the use of the Service-Finder Category ontology. Those categories are

[13] RelFinder `http://relfinder.dbpedia.org/`.

mapped to DBpedia categories[14], by the use of SKOS [20] mapping properties (specifically, `skos:closeMatch`). This existing mapping enables the connection between any service annotated with the Service-Finder Category ontology with the Web of Data. Our Linker, therefore, starting from the service categorization recalls the link between such categorization and DBpedia categories from the Service-Finder mapping, fetching additional knowledge, as follows:

```
sfcat:Music skos:closeMatch dbpcat:Music .
```

Moreover, DBpedia categories constitute a taxonomy, in which categories are related via the `skos:broader` mapping property. DBpedia topics are then related to DBpedia categories by a `skos:subject` predicate. Thus, the Linker can find a path between user interests (expressed as DBpedia topics) and service categorizations (mapped to DBpedia categories). The following triples complete the path between the user and the item of the previous listings:

```
dbp:ClassicalMusic    skos:subject  dbpcat:ClassicalMusic .
dbpcat:ClassicalMusic skos:broader  dbpcat:MusicGenre .
dbpcat:MusicGenre     skos:broader  dbpcat:Music .
```

4 Computing Recommendations with the Semantic Web

The last element in our vision for a Semantic Web Recommender System is of course the component to generate suggestions of interesting items to the users: the *Recommendation Engine*. On the basis of the RDF graph that connects users to items as computed by the Model Builder, this component must derive a list of recommendations, i.e. a list of user-item pairs qualified by a *utility value* which represents the confidence in the predicted user/item correlation.

The approach of our Recommendation Engine is oriented to find *meaningful relations* between users and items in the descriptive graph. The first step is thus identifying the relevant path(s) – in terms of RDF triples – that connects a user with an item within the graph reconstructed by the Linker. It is possible that no path exists to connect a user to an item: in this case, the Engine does not produce any recommendation for that pair; on the other hand, when multiple paths exist, all paths or only a subset of them can be considered. In any case, each path must be evaluated and given a score.

In order to judge if an item can be of interest for a user, the bare existence of a path between them is not enough; indeed, the path should contain evidence of the recommendation "utility". To this end, Semantic Web technologies can play again an important role: the RDF path in fact is not only a route connecting two points in a graph, but it consists in a "semantic" description of the reasons why the user and the item are linked.

Under this perspective, the Recommendation Engine must check the "content" of the user-item connection path, to identify signs of potential utility. For

[14] Cf. `http://www.service-finder.eu/ontologies/SFC2DBpedia`.

example, it could verify if the path contains triples that express interest, liking or importance (e.g. the user *likes* a topic which is related to the item); on the contrary, it should make sure that the path does not contain expressions of disapproval or distaste (e.g. the user *dislikes* a subject to which the item refers). Finally, it could take into account the user "context" or "role" to give higher scores to paths relevant for the current user needs. Therefore, the Recommendation Engine should *verify a set of constraints* within the RDF path(s); the satisfaction of each constraint leads to the attribution of a score and the combination of all scores constitutes the utility value for the user-item pair.

Pragmatically, those constraints to be verified can be expressed in a number of different ways using Semantic Web technologies, including SWRL [13] or RIF [6] rules. In the simplest case, those constraints can be expressed as *triple patterns* to be verified via a *SPARQL query* [29]: the query verifies specific path characteristics, like triples with specific predicates. To configure a Recommender Engine, a set of constraints should be provided, each one qualified by a score that, in case of verification, contributes to the utility value computation. Once all those computations take place, making item recommendations for a specific user means selecting the user-item pairs with the highest utility values.

Moreover, when suggesting those recommendations to the user, the Recommendation Engine could also provide a *proof* for its choice, i.e. the path connecting the user to the recommended item. This proof lets the user understand the "semantics" of the recommendation and check its validity; in turn, this could enable the possibility to gather feedbacks from the user about the soundness of a recommendation, thus paving the way for an improvement of the user/item model (e.g. the user could add/modify his interests in order to get more tailored suggestions).

Computing service recommendations scores. In our service recommendation scenario, we compute the utility value by analyzing the paths. On the one hand, we rely on the knowledge about users and items: user interests are "weighted" as exemplified before and service annotations have a "strength" representing how much the employed automatic annotator software is confident about the categorization. On the other hand, the RDF path connecting a user to a service contains other information, like the DBpedia topics and categories.

For each user-service couple, the system computes a correlation value out of all the paths connecting them. Given a path, a first utility value is computed considering specific characteristics of the RDF path, like the number of `skos:broader` relations between DBpedia categories (the fewer relations, the higher the score) and the presence of DBpedia "top" categories (if present, the connection path is probably not very meaningful). Then, this value is combined with the interest's weight and the categorization's strength, in order to compute a global utility value for the path.

5 Related works

Semantic Web and Recommender Systems are two research fields with several points of contact: in literature, it is possible to find several works related to the study of their interaction [26].

A first way to use Semantic Web technologies in recommender systems is to *describe users' profiles and items' features*. Middleton, Alani and De Roure in [19] use ontologies to model the topics taxonomy of research papers (the items to be recommended) and users' interests, in order to compute recommendations and to identify meaningful groups of users (communities of practice). In [2], Bannwart et al. present OMORE, a Content-based Recommender System for movies; it runs as a Firefox plug-in and it predicts how much a movie could be of interest for a user. When running, the application analyzes the Web pages the user views; if a page is related to a movie, OMORE processes it, retrieving the features of the movie through the LOD Cloud and, thus, profiling the user.

Regarding the use of Semantic Web tools for the *generation of recommendations*, Abel et al. in [1] present a recommender system for an on-line community with the goal of suggesting relevant discussions to the users. To process the recommendations, the system can select among several collaborative filtering algorithms; those algorithms are exposed as Semantic Web Services described using OWL [18]. To choose the best algorithm, the system employs a Semantic Web rule-based engine (with rules defined in SWRL [13]).

In [17], Manikrao and Prabhakar present a recommender system for Web services. One of the core components of their system is a semantic matcher, that operates on a knowledge-base in order to match users' needs with services. Knowledge-based Recommender System can take advantage of Semantic Web tools like reasoners and inference engines; to our best knowledge, however, the use of those technologies in recommender systems is still quite limited.

Another way Semantic Web could help to build a Knowledge-based Recommender System is *to build recommendations' proof*. Passant and Decker in [23] present dbrec, a recommender system to compute music recommendations. The system gets description of musicians and music bands from DBpedia and processes the retrieved RDF graph in order to compute a semantic similarity value (called Linked Data Semantic Distance) among artists. When users navigate the dbrec Web site, the system shows the reasons behind the recommendations of similar musicians.

Finally, Semantic Web technologies can be successfully employed to *integrate data from heterogeneous sources*. Passant and Raimond in [24] propose the use of the data in the LOD Cloud to build a music recommender system. They analyze three different kinds of data available on the LOD Cloud that could be used to compute recommendations: social-network data (the FOAF-o-sphere), descriptions of artists and bands and user-defined tags. Those data are distributed on several sources, such as DBpedia or MusicBrainz, and they are integrated and interlinked by following the Linked Data principles [3].

Another example of Semantic Web-based integration can be found in [31], where Szomszor et al. cope with the problem of building a unified knowledge-

base to compute recommendations from two datasets (IMDb and Netflix). To achieve their goal, they employ RDF and make the integrated data accessible via a SPARQL endpoint.

6 Conclusions

In this paper we presented our vision of how a Knowledge based Recommender System could be based on and exploit all the capabilities of Semantic Web technologies. We explained how the model of users and items can be enriched with the knowledge from the Web of Data and how Semantic Web tools can be employed to elaborate data and support generating recommendations. We designed the structure and the main modules that compose such a system at a conceptual level, giving hints on how this can be realized in practice; additionally, we illustrated how we concretely built a demonstrator of such a Semantic Web-enabled Recommender System in the context of service recommendations in the SOA4All project.

Our main goal with this paper was to present several opportunities to leverage the results of the Semantic Web community in the Recommender Systems field; we hope that other researchers can take our concept and analysis as reference for further investigations on the possible interplay of the two "worlds".

The pure adoption of Semantic Web technologies, however, does not solve some existing open issues in building a Knowledge-based Recommender System. In particular, since the quality of recommendations is directly correlated with the *quality of data*, relying on the knowledge in the LOD Cloud means trusting the quality of its contents. To dispel the concerns about the quality of data and the level of "trust" of the linked data sources on the Web, the Semantic Web community is putting a lot of effort in the definition and standardization of "*provenance*" descriptions[15] [21]. With the arrival of massive amounts of Semantic Web data, information about the origin of that data becomes an important factor in telling apart reliable data from low-quality information.

In an open and inclusive environment such as the Web, information is often contradictory or questionable; the distributed and redundant nature of the Web puts at risk not only the *consistency of knowledge*, but also its *completeness and reliability*. To overcome those problems, usually people make trust judgments based on what they know about the data provenance. However, when provenance information is not available or is not enough, the problem of managing inconsistency remains open. This, in turn, has a direct consequence on the possibility to evaluate the generated recommendations, e.g. in terms of *precision* and *recall*. On the other hand, the employment of semantic descriptions paves the way to a recommendation evaluation based on the *meaning* of data.

All in all, our proposed approach is far from being complete. Our future work will be devoted to extend and improve the current proof of concept, to evaluate it and to detail the lessons learned from the adoption of our approach. In our roadmap, we would like to investigate in the direction of deriving user profiles from implicit knowledge (as in our previous work [32]) and of leveraging the

[15] Cf. `http://www.w3.org/2005/Incubator/prov/`.

semantic relations between users and between items. Finally we would like to explore the feasibility of realizing a hybrid system [9] by joining our Semantic Web-enabled Recommender System with a Collaborative Filtering approach.

Acknowledgments

This research has been partially supported by the *SOA4All* (FP7-IST-215219), *LarKC* (FP7-IST-215535) and *Service-Finder* (FP7-IST-215876) EU co-funded projects.

References

1. F. Abel, I. I. Bittencourt, N. Henze, D. Krause, and J. Vassileva. A Rule-Based Recommender System for Online Discussion Forums. In *Proceedings of the 5th international conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH '08)*, pages 12–21, Berlin, Heidelberg, 2008. Springer-Verlag.
2. T. Bannwart, A. Bouza, G. Reif, and A. Bernstein. Private Cross-page Movie Recommendations with the Firefox add-on OMORE. In *8th International Semantic Web Conference (ISWC2009)*, Washington DC, USA, October 2009.
3. T. Berners-Lee. *Linked Data – Design Issues.* Online at http://www.w3.org/DesignIssues/LinkedData.html, 2006.
4. C. Bizer, T. Heath, and T. Berners-Lee. Linked Data – The Story So Far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.
5. C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia – A Crystallization Point for the Web of Data. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 7:154–165, 2009.
6. H. Boley, G. Hallmark, M. Kifer, A. Paschke, A. Polleres, and D. Reynold. *RIF Core Dialect – W3C Recommendation.* Available at http://www.w3.org/TR/rif-core/, June 22th, 2010.
7. D. Brickley and L. Miller. *FOAF Vocabulary Specification 0.97.* Available at http://xmlns.com/foaf/spec/, January 1st, 2010.
8. R. Burke. Knowledge-Based Recommender Systems. *Encyclopedia of Library and Information Science*, 69(32), 2000.
9. R. Burke. Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.
10. M. d'Aquin, C. Baldassarre, L. Gridinoc, S. Angeletou, M. Sabou, and E. Motta. Characterizing Knowledge on the Semantic Web with Watson. In *Proceedings of the 5th International Workshop on Evaluation of Ontologies and Ontology-based Tools (EON2007), co-located with the ISWC2007*, pages 1–10, Busan, Korea, 2007.
11. D. Fensel, F. van Harmelen, B. Andersson, P. Brennan, H. Cunningham, E. Della Valle, F. Fischer, Z. Huang, A. Kiryakov, T. Kyung-il Lee, L. School, V. Tresp, S. Wesner, M. Witbrock, and N. Zhong. Towards LarKC: a Platform for Web-scale Reasoning, 8 2008.
12. P. Heim, S. Hellmann, J. Lehmann, S. Lohmann, and T. Stegemann. RelFinder: Revealing Relationships in RDF Knowledge Bases. In *Proceedings of the 4th International Conference on Semantic and Digital Media Technologies (SAMT 2009)*, volume 5887 of *Lecture Notes in Computer Science*, pages 182–187. Springer, 2009.
13. I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML.* Available at http://www.w3.org/Submission/SWRL/, 2004.
14. D. Kelly and J. Teevan. Implicit feedback for inferring user preference: a bibliography. *SIGIR Forum*, 37(2):18–28, 2003.

15. G. Klyne and J. J. Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax – W3C Recommendation.* Available at http://www.w3.org/TR/rdf-concepts/, 2004.
16. J. Lehmann, J. Schüppel, and S. Auer. Discovering Unknown Connections - the DBpedia Relationship Finder. In *Proceedings of the 1st SABRE Conference on Social Semantic Web*, 2007.
17. U. S. Manikrao and T. V. Prabhakar. Dynamic Selection of Web Services with Recommendation System. In *Proceedings of the International Conference on Next Generation Web Services Practices (NWESP '05)*, page 117, Washington, DC, USA, 2005. IEEE Computer Society.
18. D. L. McGuinness and F. van Harmelen. *OWL Web Ontology Language Overview – W3C Recommendation.* Available at http://www.w3.org/TR/owl-features/, 2004.
19. S. E. Middleton, H. Alani, and D. C. De Roure. Exploiting Synergy Between Ontologies and Recommender Systems. In *Proceedings of the WWW2002 International Workshop on the Semantic Web*, 2002.
20. A. Miles and S. Bechhofer. *SKOS Simple Knowledge Organization System Reference – W3C Recommendation.* Available at http://www.w3.org/TR/skos-reference/, August 18th, 2009.
21. L. Moreau. The foundations for provenance on the web. *Foundations and Trends in Web Science*, November 2009.
22. E. Oren, R. Delbru, M. Catasta, R. Cyganiak, H. Stenzhorn, and G. Tummarello. Sindice.com: a document-oriented lookup index for open linked data. *International Journal of Metadata, Semantics and Ontologies*, 3(1):37–52, 2008.
23. A. Passant and S. Decker. Hey! Ho! Let's Go! Explanatory Music Recommendations with dbrec. In *ESWC Part II*, pages 411–415, 2010.
24. A. Passant and Y. Raimond. Combining Social Music and Semantic Web for Music-Related Recommender Systems. In *Proceedings of the 1st Workshop on Social Data on the Web (SDoW2008), co-located with the ISWC2008*, Karlsruhe, Deutschland, October 2008.
25. M. J. Pazzani and D. Billsus. Content-Based Recommendation Systems. In *The Adaptive Web*, pages 325–341, 2007.
26. E. Peis, J. M. M. del Castillo, and J. A. Delgado-López. Semantic Recommender Systems. Analysis of the state of the topic. *Hipertext.net*, 6:online, 2008.
27. M. Rowe and F. Ciravegna. Getting to Me – Exporting Semantic Social Network from Facebook. In *Proceedings of the 1st Workshop on Social Data on the Web (SDoW2008), co-located with the ISWC2008*, 2008.
28. M. Sabou, M. d'Aquin, and E. Motta. SCARLET: SemantiC relAtion discoveRy by harvesting onLinE onTologies. In *Proceedings of the 5th European Semantic Web Conference (ESWC2008)*, Tenerife, Spain, 2008.
29. A. Seaborne and E. Prud'hommeaux. *SPARQL Query Language for RDF – W3C Recommendation.* Available at http://www.w3.org/TR/rdf-sparql-query/, January 15th, 2008.
30. X. Su and T. M. Khoshgoftaar. A Survey of Collaborative Filtering Techniques. *Advances in Artificial Intelligence*, vol. 2009(Article ID 421425), 2009.
31. M. Szomszor, C. Cattuto, H. Alani, K. O'Hara, A. Baldassarri, V. Loreto, and V. D. Servedio. Folksonomies, the Semantic Web, and Movie Recommendation . In *Bridging the Gap between Semantic Web and Web 2.0 workshop, colocated with ESWC2007*, 2007.
32. A. Turati, D. Cerizza, I. Celino, and E. Della Valle. Analyzing User Actions within a Web 2.0 Portal to Improve a Collaborative Filtering Recommendation System. In *Web Intelligence/IAT Workshops*, pages 65–68. IEEE, 2009.

Behavioral Matchmaking of Semantic Web Services*

Zijie Cong and Alberto Fernández

CETINIA, Universidad Rey Juan Carlos, Madrid, Spain
zijie@ia.urjc.es, alberto.fernandez@urjc.es

Abstract. Service matchmaking is an integral link of service discovery, composition, invocation and other similar tasks under Service-Oriented Architecture (SOA). Most current approaches measure the degree of match of two services based merely on their I/O pairs which could leads to false results. This paper presents an approach for matchmaking in Semantic Web Services (SWS) that considers each service as a sub-graph of the semantic network of the ontology formed by inputs, outputs, pre- and post-conditions with contribution of syntactical information such as keywords and textual descriptions. The similarity between services is defined as the similarity between these graphs. The aim of this approach is to reveal the internal work flow and intention of service, i.e. behavior, thus it agrees with human intuition to a larger extent than existing approaches.

1 Introduction

The original intention of adding semantic annotations to web services is to improve the automation of service discovery, selection, invocation and interoperation by letting service descriptions to be machine-processable [12]. One integral part of such automation is matchmaking among services.

Various approaches have been proposed in previous studies. Without concerns about semantics of its components, one primitive method to calculate the similarity of services is based on the syntactical information - e.g. keywords, tag-clouds and textual descriptions.

For services with semantic information, inputs/outputs (I/O) matching is a common method for measuring the similarity. Inputs and outputs of a semantic service are instances of ontological concepts. The similarity of two services is determined by the subsumption relation the taxonomy tree between corresponding concepts of I/O pair. The result is a degree of semantic similarity, such as EXACT, PLUG-IN, SUBSUMES and FAIL [11]. Some studies, such as [9], aimed to achieve higher robustness and precision by combining both semantic and syntactical approaches.

* Work partially supported by the Spanish Ministry of Science and Innovation through grants TIN2009-13839-C03-02 and CSD2007-0022(CONSOLIDER-INGENIO 2010)

More recently, various graph based approaches have been proposed. In [7], a service was considered as a composition of processes and thus could be represented as a finite-state machine (FSM), the similarity between services was defined as the similarity between two FSMs. Like other similar graph-based approaches [6,5], it concentrated on structural similarity of services instead of the semantic similarity of atomic units of functionality.

This paper presents a novel but preliminary approach for service matchmaking. The main rational behind this approach is that a service could be considered as a sub-graph (Service Behavioral Graph) of a semantic network which maps input concepts to output concepts via elements specified in conditions, retrieved from textual description, it reveals the behavior of services which could be a more intuitive option for calculating the degree of match of services.

The rest of the paper is organized as follows. Section 2 shows the motivation of this work with an example of 2 service descriptions using a shared ontology with 20 concepts and 10 relations. The concept and main components of Service Behavioral Graph are defined in section 3, algorithms for obtaining those components are also shown. Section 4 describes the calculation of the degree of match between two services and how it compromises with other studies. Finally, in section 5 we conclude our current work with a discussion and future plans.

2 Motivation

Although an appropriate measurement of degree of match is difficult to define, it is consensus that the result of matching should agree with human intuition. Inputs and outputs sometimes may not provide sufficient information about service's behavior, and relying solely on them may lead to false results. An example is presented in the rest of this section.

Figure 1 illustrates an ontology of publication with 20 concepts and 10 relations connecting them, this ontology is adopted from [1].

Every service description used in this paper is a 4-tuple (T, I, O, Q), where:

$S_{(T)}$ is syntactical information of the service which may include keywords, tag-cloud or textual description.
$S_{(I)}$ is a set of input concepts.
$S_{(O)}$ is a set of output concepts.
$S_{(Q)}$ is a set of predicates that must be true *after* the execution of the service, i.e. post-conditions.

Due to the diversity of specifications and implementations of conditions in different service description approaches, in this paper, for the sake of simplicity, we consider these conditions as a conjunction of predicates that are defined in the ontology. A predicate is a binary relation between two concepts, such as *hasBirthday(Novelist, Date)*.

The preconditions are intentionally ignored as these conditions are usually checked before the execution of the service thus they do not concern with the actual behaviors.

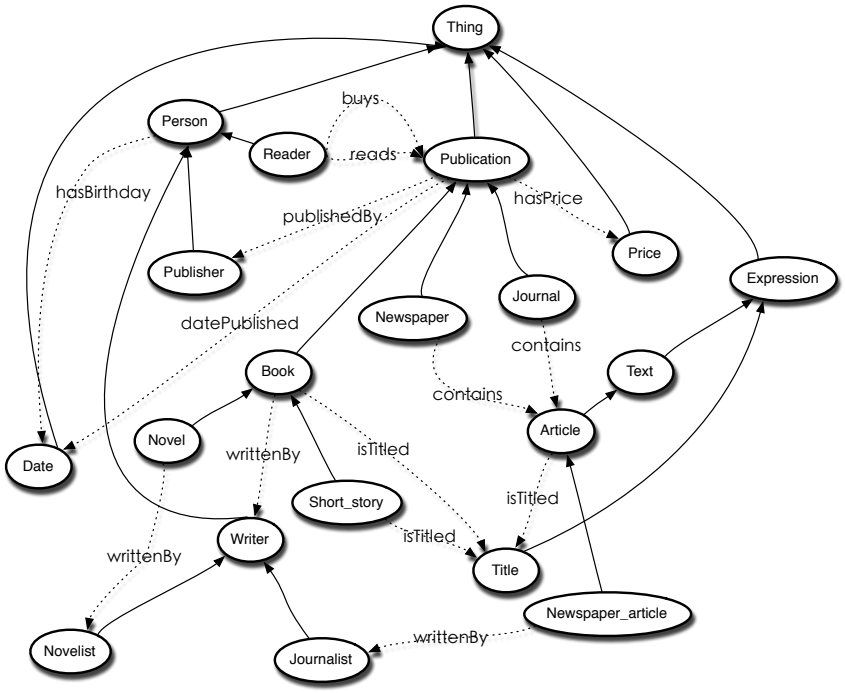

Fig. 1. An ontology of publications with 20 concepts and 10 relations, solid lines represent *subClassOf* relations.

To illustrate the problem with I/O matching approaches, we define two services in figure 2. By using I/O matching approaches such as [11], the matchmaker will not be able to distinguish between S_1 and S_2 as their inputs and outputs are identical, thus these two services matches exactly, even though the functionality of those two services is different.

Therefore the aim of our approach is to overcome the above limitations by exploiting the behavioral information of services.

3 Service Behavioral Graph (SBG)

To exploit the behavioral information of a service, we consider a service as a function that maps its inputs to its outputs. In Semantic Web Services, where inputs and outputs are ontological concepts, this mapping is usually defined by relations in the same domain ontology. As an ontology can be represented by a *multi-relational graph* where each vertex denotes a concept and each edge denotes a relation between concepts, a service thus can be further considered as a sub-graph of an ontology. More formally,

$$S_1 = \begin{cases} T = & returns\ the\ birthday\ of\ a\ given\ novelist \\ I = & \{Novelist\} \\ O = & \{Date\} \\ Q = & hasBirthday(Novelist,\ Date) \end{cases}$$

$$S_2 = \begin{cases} T = & published\ date\ of\ a\ novelist's\ earliest\ book \\ I = & \{Novelist\} \\ O = & \{Date\} \\ Q = & \emptyset \end{cases}$$

Fig. 2. Services using the ontology of publication

Definition 1. *(Service Behavioral Graph)* Let G be an ontology in its graph representation, $G = (V, \mathbb{E})$ where V is the set of concepts and $\mathbb{E}$ is the set of relations of heterogeneous types, where each relation is represented using a pair $< L, (V \times V) >$, where L is the label of the relation (e.g. *hasBirthday*). A service S is denoted as $G_S = \left(V^{'}, \mathbb{E}^{'}\right)$ where $V^{'} \subseteq V$ and $\mathbb{E}^{'} \subseteq \mathbb{E}$. Elements of $V^{'}$ and $\mathbb{E}^{'}$ are identified using the service description. This sub-graph of the ontology is referred as *Service Behavioral Graph* (SBG).

Figure 3 shows the SBGs of S_1 and S_2. These graphs can be discovered from the ontology graph using *critical elements* and *behaviorally correct paths*, which are defined in the following sections. Note that those graphs are different each other despite the fact that their I/O descriptions coincide.

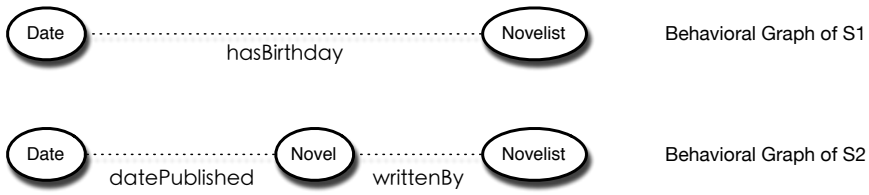

Fig. 3. SBGs of S_1 and S_2

3.1 Critical Elements

As we have mentioned in the beginning of this section the mapping from inputs to outputs is defined by the relations in the domain ontology. This mapping is, in fact, a set of paths from input concepts to output concepts, consisting of one or more relations. There may exist multiple paths between a pair of I/O concepts, therefore, finding proper paths is critical for describing the service's behavior correctly.

Such paths are determined by several components in the ontology which can be concepts or relations, and are referred as *critical elements* in this paper. *Q(Post-condition)* and *T(Syntactical information)* of service descriptions may offer some clues to determine these critical elements.

Syntactical Information Syntactical information is valuable for revealing service's behaviors. For example, even though $S_{1(I,O)} = S_{2(I,O)}$, the textual descriptions *(T)* differ these two services at human-readable level. To find the critical elements, syntactical information and ontological components' identifiers (ID or labels) need to be processed using information retrieval techniques [3] to transform them into a set of keywords with irrelevant words and morphological variants removed. Then components with keywords appeared in the syntactical information of the service are considered to be a critical element. For example, in S_1, relation *hasBirthday,* concepts *Novelist* are identified as critical elements because the words "birthday" and "novelist" have appeared in $S_{1(T)}$.

Post-conditions The post-conditions is a set of predicates that must be true after the execution of the service, for example, conditions that are specified in the *ConditionalOutput* or *ConditionalEffect* part of an OWL-S service description. These predicates often connect input elements with output elements, hence they reveal important information about service's behavior.

Figure 4 shows how *critical elements* can be determined and weighted. This function takes two arguments: O is the domain ontology used by service and S is the service description 4-tuple. Any postcondition which its domain and range are from inputs and outputs separately is considered to be critical elements with weight 1. Syntactical information such as textual descriptions are tokenized and stemmed. A normalized weight computed using TF-IDF [8] technique is assigned to each token. The TF-IDF weight measures the importance of certain words and their corresponding ontological elements in service, the calculation can be done with information of other services in the registry where service advertisements are registered, most commonly a UDDI registry [4]. Ontological elements corresponding to these tokens are considered as critical elements and assigned with weight of its token.

```
1: function CriticalElements(O, S)
2:     for all o ∈ O do
3:         o.weight = 0                                   ▷ Initialize weights to 0
4:     end for
5:     for all q ∈ S_(Q) do
6:         if range, domain of q are in S_(I) and S_(O) separately then
7:             q.weight = 1
8:         end if
9:     end for
10:    T ← TOKENIZE(S_(T))
11:    T ← STEM(T)
12:    T ← S_(T)\Stoplist                                 ▷ Remove common words
13:    for all t ∈ T do
14:        w ← TFIDF(t)                                   ▷ Calculate normalized tf-idf weight
15:        E ← ONTOLOGYELEMENTS(O, t) ∩ S_(I,O)
16:        for all e ∈ E do
17:            e.weight = w                               ▷ Assign weights to the elements
18:        end for
19:    end for
20:    return E
21: end function
```

Fig. 4. Algorithm for determining and weighting the critical elements

3.2 Behaviorally Correct Path (BCP)

To connect inputs with outputs, a path containing critical elements defined in the previous section needs to be found, we refer this path as a *behaviorally correct path (BCP)*.

In semantic networks, concepts are usually connected by heterogeneous links, including hierarchical relations as well as other relations. For similarity measuring purpose, it is necessary to have a unique path between two elements, and such path should not only contain the critical elements, but also be behaviorally correct.

In [2], Aleksovski et al. considered a path to be semantically correct if and only if no hierarchical links appear after a non-hierarchical one. For example, in figure 1, a path *{ShortStory, is_a, Book, writtenBy, Writer}* is semantically correct, while *{ShortStory, is_a, Book, writtenBy, Writer, is_a, Person}* is not.

In practice, however, there is a high possibility that no semantically correct path exists between two concept using Aleksovki's definition. Therefore, for the purpose of this paper, we define a behaviorally correct path as:

Definition 2. A *Behaviorally Correct Path (BCP)* is a path in a semantic network between two concepts containing *critical elements* with maximum one turn from non-hierarchical relation to hierarchical relation.

And two assumptions must be hold to ensure the existence of a BCP:

1. *Any relation in an ontology is invertible.*
   Relations have directions from range to domain. This assumption implies

that the graph representation of an ontology is undirected as for each relation there exists a inverse relation, e.g. if a relation *contains(Newspaper, Article)* exists, although not all articles are contained in newspapers, we assume a relation *ContainedIn(Article,Newspaper)* also exists.

2. *All relations are inheritable from a super-concept to a sub-concept.* This assumption implies that if there exists a relation p between concepts x and y, i.e. *p(x, y)*, and *is_ a(z, x)*, then *p(z, y)*. This eliminates the sequence of subsumption relations that might be appeared in the beginning of a BCP and also reduces the length of BCPs.

Together, definition 2, assumption 1 and 2, ensure that there always exist a behaviorally correct path between two concepts.

```
1: function SBG(O, S)
2:     SBG ← ∅
3:     CE ← CriticalElements(O, S)
4:     if |CE| > 0 then
5:         for all o ∈ S_(O) do
6:             Paths ← ∅
7:             for all i ∈ S_(I) do
8:                 Paths.append(BCP from o to i with maximum average weight)
9:             end for
10:            SBG.append(Path with maximum average weight in Paths)
11:        end for
12:    else
13:        SBG = S_(I) ∪ S_(O)
14:    end if
15:    return SBG
16: end function
```

Fig. 5. SBG Discovery

Figure 5 shows how a *SBG* is discovered. Firstly, if there are critical elements determined, for each pair of inputs and outputs, a BCP with maximum average weight is used to represent their behavioral connection. As not all input concepts contribute to the main behavior of the service, the path finding starts from output concepts and for each output concept, only one input concept is associated. The service behavioral graph is thus a set containing these paths.

If no critical elements can be identified, SBG will simply be a union of input and output concepts.

The SBGs of S_1 and S_2 were depicted in figure 3

4 Service Similarity

Paolucci et al. defined four degrees of matching: EXACT, PLUG-IN, SUBSUMES and FAIL, in their approach in [11] based on the hierarchical relation between

I/O pairs of service advertisement and request. They reflect the probability of conducting operation correctly of an advertised service and the satisfaction of its results with certain request. This approach guarantees the matched services can be invoked and operated correctly at lowest level, we will use these degrees as the baseline of our approach.

Algorithm in figure 6 computes the degree of match using approach from [11] at the beginning. This step eliminates the services that cannot be invoked and operated correctly even though their behaviors might be similar to certain extent. Also, this step guarantees that in the worst case, if no critical elements were found in the previous SBG discovery phase, i.e, SBGs are simply sets of input elements and output elements, the result is equivalent to Paolucci's approach.

```
1: function ServiceMatch(SBG^R, SBG^A)
2:     hierarchicalDegree ← HIERARCHICALMATCH(S^R, S^A)
3:     behavioralDegree ← 0
4:     if hierarchical = FAIL then
5:         return < FAIL, −1 >
6:     end if
7:     if SBG^R = S^R_(I,O) or SBG^A = S^A_(I,O) then
8:         return < hierarchicalDegree, −1 >
9:     end if
10:    for all Paths pr and pa in SBG^R and SBG^A do
11:        degree ← MAXPATHMATCH(pr, pa)
12:        if degree > behavioralDegree then
13:            behavioralDegree ← degree
14:        end if
15:    end for
16:    return <hierarchicalDegree, behavioralDegree>
17: end function
```

Fig. 6. Calculation of degree of match

The result of our approach is a pair, for example using the services presented in figure 2, the degree of match is <EXACT, 0.375>, the first element of this pair is the degree of match using Paolucci's approach, and the second element is the behavioral difference of two services, this structure provides requester more flexibility on interpreting the degree of match depends on their needs and environment.

This difference is computed based on the differences of paths where -1 indicates no behavioral matching has been done, 0 indicates an exact match. As a path is a sequence of concepts and relations, the differences of two paths can be defined as their edit distance. We use Levenshtein distance [10] in this paper as presented in figure 7, other distance metrics could also be used here such as Longest Common Sub-sequence (LCS).

1: **function** $PathMatch(P^R, P^A)$
2: $degree \leftarrow$ EDITDISTACE(P^R, P^A)
3: **if** $degree = 0$ **then**
4: **return** 0
5: **else**
6: **return** $\frac{degree}{P^A.length + P^R.length}$
7: **end if**
8: **end function**

Fig. 7. Distance between paths

5 Conclusion and Future work

This paper presents a novel but preliminary approach of calculating the degree of match between two services. This approach intends to reveal the behavioral information of services, and by comparing their similarity to achieve higher accuracy, robustness and in agreement with human intuition. The main notion behind this approach is that we consider a service as a sub-graph of semantic network that connects its inputs concepts and output concepts via critical elements, referred as *Service Behavioral Graph (SBG)*. We use syntactical information and conditions to determine the critical elements, and a SBG is discovered by exploiting these elements.

Experiments with actual realistic test cases are necessary to access the practicability of our approach. One expectable limitation of our approach is that it depends on the quality (in term of richness) of the ontology to a large extent which is highly unstable in practice. Also, in open environments, services may not use the same ontology to describe its functionality, so semantic alignments need to be performed. Our future work includes implementation, experiments and evaluation of this approach, also solving open issues such as efficient calculation of SBGs, reduction of the deviation caused by the instability of the quality of ontologies and refine the degree of match.

References

1. Owls-tc version 2.2 revision 2. http://projects.semwebcentral.org/projects/owls-tc/.
2. Z. Aleksovski, W. ten Kate, and F. van Harmelen. Exploiting the structure of background knowledge used in ontology matching. In *Ontology Matching Workshop at International Semantic Web Conference (ISWC)*. Citeseer, 2006.
3. G.G. Chowdhury. *Introduction to modern information retrieval*. Facet, 2004.
4. L. Clement, A. Hately, C. von Riegen, T. Rogers, et al. UDDI Version 3.0. 2. *UDDI Spec Technical Committee Draft*, 20041019, 2004.
5. J. Corrales, D. Grigori, and M. Bouzeghoub. Bpel processes matchmaking for service discovery. *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, pages 237–254, 2006.

6. D. Grigori, J.C. Corrales, and M. Bouzeghoub. Behavioral matchmaking for service retrieval: Application to conversation protocols. *Information Systems*, 33(7-8):681–698, 2008.
7. A. Günay and P. Yolum. Structural and semantic similarity metrics for web service matchmaking. In *Proceedings of the 8th international conference on E-commerce and web technologies*, pages 129–138. Springer-Verlag, 2007.
8. K.S. Jones et al. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 60:493–502, 2004.
9. M. Klusch, B. Fries, M. Khalid, and K. Sycara. Owls-mx: Hybrid owl-s service matchmaking. In *Proceedings of 1st Intl. AAAI Fall Symposium on Agents and the Semantic Web*, volume 142, 2005.
10. V. Levenshteiti. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics-Doklady*, volume 10, 1966.
11. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. *The Semantic Web (ISWC 2002)*, pages 333–347, 2002.
12. K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller. Adding semantics to web services standards. In *Proceedings of the International Conference on Web Services*, pages 395–401. Citeseer, 2003.