A Distributed Hybrid Matchmaker for IoT Services

G. Cassar, P. Barnaghi and K. Moessner

Centre for Communication Systems Research
University of Surrey
Guildford, GU2 7XH, UK

Abstract: The use of service-oriented computing paradigm in Internet of Things research has recently received significant attention to create a semantic service layer that supports virtualisation of and interaction among “Things”. Using service-based solutions will produce a deluge of services that provide access to different data and capabilities exposed by different resources. The heterogeneity of the resources and their service attributes require efficient solutions that can discover services and match them to the data and capability requirements of different users. We propose a distributed hybrid service matchmaking method that combines our previous work on probabilistic service matchmaking using latent semantic analysis with a weighted-link analysis based on logical signature matching. The hybrid method can overcome semantic synonymy in semantic service description which usually presents the biggest challenge for semantic service matchmakers. The results show that the proposed method performs better than existing solutions in terms of precision ($P@n$) and normalised discounted cumulative gain ($NDCG_n$) measurement values.

Keywords: availability, business, monitoring, reliability, services.

1. Introduction

Transparent and seamless access to large volumes of smart devices and resources is one of the main challenges in the Internet of Things (IoT) [1]. A service-oriented approach provides a promising solution for enabling access to smart devices through loosely coupled Web services [6]. Resources such as sensors, actuators, and other mobile devices can be represented as Web services, providing common interfaces that allow users or machines to access their capabilities and/or data through the Internet. We term the services exposed by the connected Things in the physical world as IoT Services.

Service Discovery is a research challenge that has sparked various works in service oriented computing [26] as other high-level service oriented concepts such as service composition, provisioning [12], and adaptation highly rely on accuracy of the service discovery results. With the emerging practice of exposing IoT sensors and actuators as Web services [6,30,12], service discovery has become a topic of great importance to IoT research. Service discovery in the IoT is more challenging than discovery on enterprise Web service platforms where reliable service resources can be abundant. The IoT services run on sensor nodes that are limited in processing capabilities and energy (e.g., limited battery life); communication between services running on mobile devices and gateways is also error prone and in many cases unreliable; the changes of the surrounding environments also have significant impact on performance of such services [30]. In the light of the challenges discussed, we recognise that to be useful in dynamic environments, a service discovery solution needs to extend from keyword-based matchmaking and take an automated approach [23] where machines inter-
pret the meaning behind the service description data and matchmaking is performed based on both functional and non-functional attributes of a service [5].

Service discovery solutions generally consist of three components [16,26]: service representation, service matchmaking method, and discovery architecture. In this paper we will discuss IoT service representation and distributed service discovery architecture, however, the main focus of the paper is on the core challenge of providing service matchmaking [16].

A common practice in semantic service matchmaking is to take advantage of machine-interpretable annotations in the service descriptions to match the semantic input/output (IO) signature of a service to a service request [16]. Methods based on logical reasoning tend to be very accurate given its solid mathematical basis. However, strictly matching the semantic signature alone may lead to false negatives [15]. Another known limitation of logic-based approaches is that when two concepts are semantically synonymous but defined differently in their terminological definitions, the similarity between the two is not captured by the subsumption hierarchy and a reasoner would fail to find the match between the two [17]. The limitations of logic-based semantic service matchmakers influenced the creation of a separate category of non-logic-based semantic matchmakers. Non-logic-based semantic service discovery approaches [27,10,22,25] aim to reduce the complexity of semantic matchmaking by analysing the frequency of occurrence of certain terms within service descriptions and determine semantics which are implicit in service descriptions. These approaches generally use techniques such as graph matching, linguistic analysis, data mining, and information retrieval (IR) [21] to process the meta-data provided in service descriptions in terms of vectors. However, this transformation results in the loss of the machine-interpretable semantics found in semantic service descriptions. Furthermore non-logic-based semantic matchmakers do not possess the logic-based functions to determine whether the IO parameters of a service are compatible with the requirements of the user. Hybrid semantic matchmakers [15,17,18,24,9,14,13] combine the advantages of Non-Logic-based techniques with the fine grained reasoning capabilities of Logic-based techniques. Klusch et al. [17] state that the objective of this hybrid semantic matchmaking is to appropriately exploit both crisp logic-based and non-logic-based semantic matchmaking where using each of the solutions alone could fail. Although literature suggests that hybrid matchmakers always outperform logic-based and non-logic-based matchmakers, in terms of precision and ranking the most relevant services at the top of search results, existing hybrid matchmakers are still using non-logic-based components that do not take advantage of the semantic data in semantic service descriptions.

We propose a hybrid semantic service matchmaking method for the IoT services. The proposed method combines our previous work on probabilistic service matchmaking using latent semantic analysis [4] with a logical signature matchmaking. The logical signature matchmaking method is based on the concept of individual *Links* between a source parameter and a destination parameter (defined in Section 5.2). This method provides an added flexibility needed when searching for candidate IoT services to be used in complex operations such as IoT service composition or IoT service provisioning. The hybrid method can overcome semantic synonomy in IoT service description which usually present the biggest challenge for semantic service matchmakers. The evaluation results show that the proposed method performs better than existing solutions in terms of precision ($P_{@n}$) and normalised discounted cumulative gain ($NDCG_{n}$) measurement values.

The rest of this paper is structured as follows. Section 2 describes the IoT service modelling framework. In Section 3, we briefly explain our previous work on probabilistic service matchmaking. Section 4 discusses a probabilistic method for organising repositories into clusters. Section 5 presents the use of *links* and a weighted-link measure for matching the IO signature of an IoT service to a request. Section 6 discusses the distributed service discovery architecture and how the probabilistic service matchmaking and the weighted-link measure can be combined to create a hybrid semantic matchmaker. Sections 7 and 8 perform a comparative study between our method and existing semantic service matchmakers and describe the evaluation results respectively. We discuss the merits and limitations of our method and describe the future work in Section 9.
2. Service Representation

Semantic service modelling provides a machine interpretable framework for representing many aspects (e.g., functional, non-functional and transactional attributes) of services. The semantic Web service community has developed several models for semantically describing general Web services such as the Ontology Web Language for Services (OWL-S)\(^1\) and Web Service Modelling Ontology (WSMO)\(^2\). The work in [8] proposed the 'Entity-Device-Resource' model for representing IoT resources and services based on the Semantic Sensor Network ontology [7]. However, these heavyweight and complex models are not suitable for describing IoT services. IoT services exposed by IoT resources mostly have limited computation capabilities and often operate in dynamic and constrained physical environments; therefore, they are far less reliable and stable compared to the carefully designed and maintained Web services. Their logic is much simpler and their output usually represents observation and measurement of features of interest of physical entities (therefore, service models have to be associated with IoT resources). Despite these characteristics, in a service oriented IoT, they also need to participate in service composition and the issues on effective service adaptation and compensation mechanisms become prominent.

For these reasons, a semantic IoT service representation model preferably needs to be lightweight to facilitate computation (experiences in ontology design shows that well-designed lightweight ontologies have the potential to be widely adopted), in particular efficient service discovery, composition and adaptation given the stunning number of IoT resources and services. The service model should be associated with the model of its exposing resource and provide constructs for linking to concepts in domain knowledge base (e.g., Geonames ontology\(^3\)) or the linked data\(^4\). We have developed a lightweight description ontology for IoT service based on the existing research and the aforementioned requirements (See Figure 1).

![Figure 1. Overview of the lightweight IoT service description model.](image-url)

The service model is also designed to be independent of any particular service technologies (i.e., SOAP/WSDL and RESTful services) based on the analysis of their commonalities and distinctiveness. The OWL-S model for SOAP/WSDL services is designed using the 'Profile-Process-

---

\(^1\) [http://www.w3.org/Submission/OWL-S/](http://www.w3.org/Submission/OWL-S/)

\(^2\) [http://www.wsmo.org/](http://www.wsmo.org/)

\(^3\) [http://www.geonames.org/ontology/](http://www.geonames.org/ontology/)

\(^4\) [http://linkeddata.org/](http://linkeddata.org/)
Grounding’ pattern and much of the complexity stems from the process modelling. On the contrary, the hREST model [19] for RESTful service is too simple: it does not include a profile and grounding which are important for service discovery and access. Our service description model represents a trade-off between these two: being lightweight and service technology independent while at the same time providing sufficient modelling constructs for represent service on the IoT (Details of the ontology can be found at: http://purl.oclc.org/net/unis/IoT.est-Service.owl). We refer to the design pattern as ‘Profile-Model-Grounding’: Profile and Grounding are adapted from the OWL-S and refined (so it can also be used for RESTful services); the Model excludes the process modelling and is based on the atomic service modelling in OWL-S and RESTful service modelling in hREST. Another advantage of our service model is that although it is not fully compatible with existing modelling methods, it can be easily transformed to each other using a simple program. It should be noted that many of the existing works on semantic service matchmaking [15], [9] and [4] are based on the OWL-S model. For evaluation and comparison purposes, we use a dataset of OWL-S service descriptions in this work. The main focus of our work is to show how service descriptions can be used to efficiently discover IoT services in a distributed environment (irrespective of what service description model is used).

3. Probabilistic Service Matchmaking

The work in this paper builds upon our previous work on service search and matchmaking and the ranking of search results [4]. In [4] we showed how semantic concepts can be extracted from OWL-S service descriptions and mapped into a latent factor space using a technique called Latent Dirichlet Allocation (LDA) [3]. LDA is an unsupervised machine-learning technique which uses a generative probabilistic model to map high-dimensional count vectors (such as the distribution of semantic concepts describing the services in a repository) to a lower dimensional representation in latent variable space. An abstract overview of our approach is shown in Figure 2.

For every service description $s_i$, the model associates unobserved latent factors $z_1, z_2, ..., z_k$ with the probability of concept $c_j$ appearing in $s_i$. The generative model of LDA can be represented as:
\[ P(c_j) = \sum_{k=1}^{K} P(c_j|z_k) P(z_k) \]  

(1)

where \( P(z_k) \) is the probability that latent factor \( k \) is sampled for concept \( j \) and \( P(c_j|z_k) \) is the probability of sampling concept \( j \) given latent factor \( k \).

The LDA model assumes that the probability distributions \( P(c|z) \) and \( P(z) \) follow a Dirichlet distribution: \( \Phi^{(k)} = P(c|z) \) and \( \Theta^{(i)} = P(z) \) respectively.

Concepts describing functional parameters and profile data are extracted from OWL-S service descriptions using a reasoner and listed in a Service Transaction Matrix which represents the probability distribution \( P(s,c) \) of concepts \( c \) over service descriptions \( s \). Using the observed data from the service transaction matrix, the parameters \( \Phi \) and \( \Theta \) can be estimated using a method based on Gibbs Sampling described in [28]. This algorithm was implemented using the LingPipe\(^5\) toolkit.

The learned model describes each service as a vector of latent factors (as shown in Figure 2). Using this method, a request \( R \) (also following the OWL-S model) containing semantic definitions can be converted into latent factor space and matched accurately to the services in a service registry by computing the vector similarity of each service vector to the request vector. We compute this similarity using the proximity measure called Multidimensional Angle (also known as Cosine Similarity); a measure which uses the cosine of the angle between two vectors [25]. The multidimensional angle between a vector containing the distribution of latent factors \( p \) of a service and a vector containing the distribution of latent factors \( q \) of a query can be calculated using Equation 2.

\[
\cos(p, q) = \frac{p \cdot q}{\|p\| \cdot \|q\|} = \frac{\sum_{i=1}^{f} p_i q_i}{\sqrt{\sum_{i=1}^{f} p_i^2} \cdot \sqrt{\sum_{i=1}^{f} q_i^2}}
\]  

(2)

where \( f \) is the number of latent-factors.

In this work, we use the probabilistic matchmaking method to search for the list of services that most accurately match to a request. The list of search results returned by the probabilistic matchmaking is then passed on to the logical signature matchmaking method (described in the next section) which ranks the services more accurately by checking the compliance of their IO signature with the IO signature of the request.

4. Probabilistic Clustering

As the number of service descriptions stored in a registry increases, efficiently finding service descriptions becomes a challenging task. By organising the service data into clusters, services become easier and thus faster to discover [25]. Clustering is an approach that organises a complex dataset into a series of simpler sets (clusters), where the members of each set have common features. 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.

We propose using the latent factors learned from the probabilistic model to group the services into clusters. We create \( K \) clusters; where \( K \) is the number of generated latent factors (i.e. a cluster for each latent factor). The vector of latent factors describing each service is used to determine which

\(^5\)http://alias-i.com/lingpipe/
latent factor best describes the service. The service is then assigned to the cluster corresponding to that latent factor. If a service has more than one latent factor that is related to it, the service will be assigned to each of the clusters that correspond to these latent factors.

This approach gives us a number of advantages over classical clustering algorithms. The dimensionality of the model is reduced as all services can be described in terms of a small number of latent factors rather than a large number of concepts. The algorithm is also more scalable and can be applied to large datasets 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 clusters easily without high computational requirements. 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 service request.

5. Logical Signature Matchmaking

Logical signature matching has been used in different works to verify whether the IO parameters of a service are compatible with the IO parameters of a request [16]. A common approach to logical signature matchmaking is to define a set of rules (filters) which dictate what kind of logical relationships are acceptable between the IO parameters of a service and the IO parameters of a request [15].

The logical signature matchmaking filters most commonly used in logic-based service matchmaking research are Exact, Plug-in, Subsumes, Subsumed-by, and LFail. Klusch and Kapahnke [15] describe these filters, measuring the degree of match between a request R and a service S, as follows:

- **Exact**: S matches exactly to R when the inputs and outputs of S are equivalent to the inputs and outputs R. An exact match is described by:
  \[
  \forall S_{\text{In}} \in \text{in}(S) \exists R_{\text{In}} \in \text{in}(R) : (S_{\text{In}}, R_{\text{In}}) \in \text{in}(S) \equiv \text{in}(R) \\
  \land \forall R_{\text{Out}} \in \text{out}(R) \exists S_{\text{Out}} \in \text{out}(S) : (R_{\text{Out}}, S_{\text{Out}}) \in \text{out}(R) \equiv \text{out}(S)
  \]

- **Plug-in**: S plugs into R when the inputs of R are a direct sub-class of S and the outputs of S are a sub-class of the outputs of R. A plug-in match is described by:
  \[
  \forall S_{\text{In}} \in \text{in}(S) \exists R_{\text{In}} \in \text{in}(R) : (S_{\text{In}}, R_{\text{In}}) \in \text{in}(S) \subseteq \text{in}(R) \\
  \land \forall R_{\text{Out}} \in \text{out}(R) \exists S_{\text{Out}} \in \text{out}(S) : (R_{\text{Out}}, S_{\text{Out}}) \in \text{out}(R) \supseteq \text{out}(S)
  \]

- **Subsumes**: R subsumes S when the inputs of R are a sub-class of the inputs of S and the outputs of S are a sub-class of R. A subsumes match is described by:
  \[
  \forall S_{\text{In}} \in \text{in}(S) \exists R_{\text{In}} \in \text{in}(R) : (S_{\text{In}}, R_{\text{In}}) \in \text{in}(S) \subseteq \text{in}(R) \\
  \land \forall R_{\text{Out}} \in \text{out}(R) \exists S_{\text{Out}} \in \text{out}(S) : (R_{\text{Out}}, S_{\text{Out}}) \in \text{out}(R) \supseteq \text{out}(S)
  \]

- **Subsumed-by**: R is subsumed by S when the inputs of R are a sub-class of the inputs of S and the outputs of S are a direct super-class of R. A subsumed-by match is described by:
  \[
  \forall S_{\text{In}} \in \text{in}(S) \exists R_{\text{In}} \in \text{in}(R) : (S_{\text{In}}, R_{\text{In}}) \in \text{in}(S) \supseteq \text{in}(R) \\
  \land \forall R_{\text{Out}} \in \text{out}(R) \exists S_{\text{Out}} \in \text{out}(S) : (R_{\text{Out}}, S_{\text{Out}}) \in \text{out}(R) \subseteq \text{out}(S)
  \]

- **LFail**: S and R fail to match if none of the above filters apply.

However, this kind of matchmaking takes into consideration the whole IO signature and can only calculate the degree of match between one service and one request.

While we agree that logical signature matchmaking is important to check that the IO signature of a service is compatible before using it for a task that requires specific IO parameters, we argue that complex mechanisms such as service composition or service provisioning require a more flexible approach than the rigid matchmaking filters discussed in [15]. We build our logical signature matchmaking method based on the concept of individual Links between a source parameter and a destination parameter.
5.1. Links

We define a link as a logical relationship between two IO parameters. A link has a source parameter \( Source \) and a destination parameter \( Destination \) and is denoted as \( Link(Source, Destination) \). The links in automated service matchmaking can represent a possible connection between two services, the relevancy of an input of a service to one of the input parameters specified in a service request, or the ability of a service to generate one of the outputs specified in a service request. The definition of links defined in this section derives from the definition of Causal Links [20].

Given a domain ontology model \( \tau \), a causal link between the output parameter \( A_out_i \) of service \( A \) and the input parameter \( B_in_j \) of service \( B \) can belong to five different categories:

1. **Exact**: if \( A_out_i \) and \( B_in_j \) are equivalent concepts;
   
   formally, \( \tau \models A_out_i \equiv B_in_j \).

2. **Plug-In**: if \( A_out_i \) is a sub-class of \( B_in_j \);
   
   formally, \( \tau \models A_out_i \subseteq B_in_j \).

3. **Subsumes**: if \( A_out_i \) is a super-class of \( B_in_j \);
   
   formally, \( \tau \models A_out_i \supseteq B_in_j \).

4. **Intersection**: if the intersection of \( A_out_i \) and \( B_in_j \) is satisfiable; formally, \( \tau \not\models A_out_i \cap B_in_j \perp \).

5. **Disjoint**: if \( A_out_i \) and \( B_in_j \) are incompatible;
   
   formally, \( \tau \models A_out_i \cap B_in_j \not\perp \).

Our definition of a link differs from causal links in that we specify that every link has a source parameter and a destination parameter and does not always necessarily exist only from an output of a service to the input of another service. While causal links are only applied to service composition where the output of one service is linked to the input of another service, our definition of a link can also be used to perform logical signature matchmaking between a service and a request. Checking individual links makes it possible to assess the degree of match between a service and a request more flexibly compared to rigid logic filters such as those described in [15].

We argue that a **Subsumes** link between an output \( A_out_i \) of service \( A \) and the input \( B_in_j \) of service \( B \) cannot be used in practical cases because the super-class of a parameter is more general and may consist of other sub-classes of parameters that \( B_in_j \) is not compatible with and would result in service \( B \) not being able to work properly. The same argument applies to **Intersection** links. The only instance in which a **Subsumes** link is applicable occurs when the output of a service is linked to an output of a request. In such a case, if an **Exact** or **Plug-In** link does exist, providing a super-class of the desired output parameter as the final output is better than not providing any output at all. Thus in our work, a link can belong to only one of the four categories defined below. Let \( \tau \) be a domain ontology model. Let \( Source \) be a source IO parameter concept and let \( Destination \) be an IO parameter concept that \( Source \) can be linked to. Then, the type of link between \( Source \) and \( Destination \): \( Link(Source, Destination) \) can be classified as one of the four categories explained below:

![Figure 3. Link-Weight matching example.](image-url)
1. **Exact**: Source is an exact match to Destination if 
   \( \tau \models Source \equiv Destination \).
2. **Plug-In**: Source plugs into Destination if 
   \( \tau \models Source \sqsubseteq Destination \).
3. **Subsumes**: Source subsumes Destination if 
   \( \tau \models Source \sqsupseteq Destination \).
4. **Disjoint**: Source is not related to Destination in any of the above ways.

Note that although we did not drop the Subsumes link, we only allow such links to link the output of a service to the output of a request.

5.2. Weighted-Link Matchmaking

We propose Weighted-Link Matchmaking as a means to measure the logical signature match between a service \( S \) and a request \( R \). A weighted-link match operates separately on each one of the IO parameters making the logical signature of a service. For matching the inputs of a request to the inputs of a service (an input-input link), the total link score that can be assigned to a link \( T_{\text{w}_{\text{in}}} \) depends on the number of inputs of the service i.e.

\[
T_{\text{w}_{\text{in}}} = \frac{1}{|\text{in}(S)|} \tag{3}
\]

For matching the outputs of a service to the outputs of a request (an output-output), the total link score that can be assigned to a link \( T_{\text{w}_{\text{out}}} \) depends on the number of outputs specified in the request i.e.

\[
T_{\text{w}_{\text{out}}} = \frac{1}{|\text{out}(R)|} \tag{4}
\]

The maximum weight given to an input-input link depends on the number of inputs of the service rather than the inputs of the request because the highest priority here is to make sure that all the inputs necessary for the service to operate can be satisfied. If one of the inputs is missing, the service cannot be used properly while it is ok to leave one of the inputs specified by the request unused. Conversely the maximum weight given to an output-output link depends on the number of outputs specified in the request. The reason behind this is that the important aspect is whether a service can generate all the outputs required by the request. In automated systems, it could be acceptable that a service generates an extra output if that output is not used. What matters is that all the outputs specified in the request are ultimately generated and supplied to the service consumer.

We define a weight function \( w_f \) that assigns a weight to the strength of a link between a source parameter \( S_{\text{rc}} \) and a destination parameter \( D_{\text{st}} \) depending on the type of the link.

\[
w_f(\text{Link}(S_{\text{rc}}, D_{\text{st}})) = \begin{cases} 
1.0, & \text{if } \text{Link}(S_{\text{rc}}, D_{\text{st}}) = \text{Exact} \\
\alpha, & \text{if } \text{Link}(S_{\text{rc}}, D_{\text{st}}) = \text{Plug-In} \\
\beta, & \text{if } \text{Link}(S_{\text{rc}}, D_{\text{st}}) = \text{Subsumes} \\
0.0, & \text{if } \text{Link}(S_{\text{rc}}, D_{\text{st}}) = \text{Disjoint}
\end{cases} \tag{5}
\]

where \( \alpha \) and \( \beta \) are penalising weights that allow the user to bias the algorithm towards preferred link types.

For example, from Figure 3b., if we select \( \alpha = 0.8 \) the degree of match between input parameter \( \text{In}_R_1 \) of the request and input parameter \( \text{In}_S_1 \) of the service is \( \text{Link}(\text{In}_R_2, \text{In}_S_1) = \text{Exact} \), thus \( w_f(\text{Link}(\text{In}_R_1, \text{In}_S_1)) = 1. \)
The weighted-link score is calculated using the equation:

$$Link_{Score}(S_{rc}, D_{st}) = T_{wx} w_f(Link(S_{rc}, D_{st}))$$  \hspace{1cm} (6)$$

where $x \in \{in, out\}$ depending on whether the link is an input-input link or an output-output link. The total matching score $Match_{Logic}(S, R)$ between service $S$ and request $R$ is given by adding the weighted-link score of all the links between $S$ and $R$:

$$Match_{Logic}(S, R) = \sum_{IO} Link_{Score}(S_{rc}, D_{st})$$  \hspace{1cm} (7)$$

### 6. Distributed Hybrid Matchmaker

In this section, we explain the discovery architecture and the hybrid matchmaking approach proposed in this paper. We propose an architecture (shown in Figure 4) with a number of distributed registries (the number of registries depending on the demography of the network or the load requirements) and one central manager entity. Before the system is deployed, a subset of available service descriptions is used to train the machine learning techniques and learn the distribution over concepts for each latent factor $P(c|z)$ as discussed in Section 3. These probability distributions are then propagated to the different registries so that each registry can convert new service descriptions and service requests to latent factor space. The manager then assigns a number of clusters to each repository so that each repository can be responsible of only a small set of clusters. The registries use the folding-in algorithm [31] to compute the probability distribution over latent factors for their stored service descriptions and then assign each service description to the relevant cluster(s) as discussed in Section 4. If a service description does not belong to a cluster maintained by the registry processing it, the service description is forwarded to the registry that handles the relevant cluster(s). Communication between the distributed registries can be carried out using HTTP GET/POST messages.

When a new service description/request is submitted to one of the registries, the registry will use folding-in to determine the distribution of latent factors for the service description/request and decide whether to handle the request within that registry (if the distribution of latent factors indicates it is related to one of the clusters handled by this registry) or whether to forward it to another registry that handles clusters that are more relevant to the service description/request. When a service request is finally submitted to the registry that is most relevant to it, the hybrid matchmaking component is used to search within that cluster.

The hybrid matchmaking relies on the probabilistic matchmaking component described in Section 3 to find a short list of candidate IoT services which is then passed to the weighted-link matchmaking component described in Section 5.2 to accurately arrange the results. The probabilistic component is first used to match the IoT services to the request based on latent factors extracted from the underlying concepts in the IoT service descriptions. This approach helps to identify statistical similarity between a service and a request and can find relevant candidate services that would otherwise have been omitted by strict logic matchmaking [4]. The probabilistic component then passes a short list of results to the logic-based component, thus restricting the scope of search for this component and reducing the complexity of the matchmaking. The size of the short list is specified by the user depending on the number of service required. The logic-based component verifies the IO signature of each candidate service and calculates the weighted-link score. Finally, the results from the logic based are ranked based on their weighted-link score. When two services score the same, the score from the probabilistic component is used as a tie-breaker. The final ranked list of results is presented to the client.
Each registry also notifies the manager of any new concepts appearing in service descriptions/requests so that when the number of new concepts observed hits a predefined threshold, the manager retriggers the machine-learning algorithm to relearn the latent factors and the whole process is repeated again. This is a limitation in the current approach and our future work will focus on solving this issue. Another limitation is that as the number of observed concepts increases, the number of latent factors required to accurately represent the service information will need to be gradually increased as well. Our current method uses empirical parameters to set the variables to determine the number of latent factors [11]. However, future work will investigate the usage of non-parametric probabilistic topic models that allow the number of latent factors to increase gradually as new concepts are observed without the need of re-learning the latent factors [2,29].

7. Evaluation

Many of the existing works on semantic service matchmaking are based on the OWL-S model [15, 17]. In order to compare our approach with state-of-the-art service matchmakers, we perform the comparative analysis in this paper using the OWL-S service retrieval test collection OWLS-TC v3.06. The close relationship between the OWL-S service model and the IoT service modelling framework was explained in Section 2. The dataset consists of 1007 service descriptions defined in OWL-S form. The services are divided into seven categories and a total of 29 OWL-S queries are provided together with a relevant answer set for each query. The answer set for each query consists of a list of relevant service and each service \( i \) has a graded relevance value \( label(i) \in \{1, 2, 3\} \) where 3 denotes a high-relevance to the query and 1 denotes a low-relevance. Table 1 shows the number of services and queries belonging to each of the seven categories.

---

6http://www.semwebcentral.org/projects/owls-tc/
Table 1. Number of Services and Queries for each domain.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Services</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Education</td>
<td>284</td>
<td>6</td>
</tr>
<tr>
<td>Food</td>
<td>34</td>
<td>1</td>
</tr>
<tr>
<td>Medical</td>
<td>73</td>
<td>1</td>
</tr>
<tr>
<td>Travel</td>
<td>165</td>
<td>6</td>
</tr>
<tr>
<td>Communication</td>
<td>58</td>
<td>2</td>
</tr>
<tr>
<td>Economy</td>
<td>359</td>
<td>12</td>
</tr>
<tr>
<td>Weapon</td>
<td>40</td>
<td>1</td>
</tr>
</tbody>
</table>

The probabilistic method (based on LDA) described in Section 3 and the hybrid method described in Section 6 are compared with a text-matching approach powered by Apache Lucene\(^7\) and also methods from the OLWS-MX 2.0\(^8\) hybrid semantic Web service matchmaker (M0, M3, and M4) [17]. M0 is a logic-based approach and M3 and M4 are hybrid approaches which use both logic and non-logic based methods. In the next section, the probabilistic method based on LDA is labeled LDA and the hybrid method is labeled LDA + Logic.

We also investigated how assigning the services to different number of clusters at the same time effects the performance of the search and ranking mechanism. By assigning all service descriptions to more than one cluster, purity becomes a confusing measure because each cluster will now contain service descriptions from a wider variety of categories rather than a very specialised set. The hybrid method with different number of cluster assignments is evaluated by comparing the averaged Precision at n \((P@n)\) and the Normalised Discounted Cumulative Gain \((NDCG_n)\) values over all 29 service requests for different numbers of cluster assignments.

For the hybrid method, in the experiments we have given a higher weight to Plug-In links and have penalized Subsumes links. The parameters \(\alpha\) and \(\beta\) are set to 0.8 and 0.4 respectively based on heuristic measures. The size of the short list which should be specified by the user was set to 40 services since our evaluations were carried out to up to 40 services retrieved. To compare the distributed service matchmaking method (discussed in Section 6) with the other service matchmaking methods we kept the number of cluster assignments fixed to 3.

All experiments were carried out on a computer with Intel(R) Core(TM)2 Duo T7500 2.2GHz CPU, 2GB RAM, and running Microsoft Windows 7 x86. The sample queries are all in the form of OWL-S templates and contain the semantic requirements together with a text description of the queried functionality. For the text-based approach, the text descriptions of the service attributes are retrieved from the query templates and used as the query string.

We evaluated our approach by calculating the Precision at n \((P@n)\) and the Normalised Discounted Cumulative Gain \((NDCG_n)\) for the results obtained for each of the sample queries. These are standard evaluation techniques used in Information Retrieval to measure the accuracy of a search mechanism with respect to completeness of the results returned.

### 7.1. Precision @ n

Precision is a measure used to evaluate the results of the search and matchmaking process. Precision @ n is a measure of the precision of the system taking into account the first \(n\) retrieved services. Precision reflects the number of retrieved services which are relevant to the search. The precision for a set of retrieved services is given by:

\[
\text{precision} = \frac{|\{\text{RelevantServices} \cap \{\text{RetrievedServices}\}\}|}{|\{\text{RetrievedServices}\}|} \quad (8)
\]

\(^7\)http://lucene.apache.org/
\(^8\)http://semwebcentral.org/projects/owls-mx/
where the set of relevant services to a given query is defined in the OWLS-TC v3.0 test collection. Only services with a graded relevance value of 3 were considered for this evaluation.

### 7.2. Normalised Discounted Cumulative Gain

$NDCG_n$ is a measure that takes into account the graded relevance of each service retrieved. This measure is particularly useful for evaluating ranking results since not all services in a relevance set are of the same relevance to the query. The $NDCG_n$ for $n$ retrieved services is given by Equation 9.

$$NDCG_n = \frac{DCG_n}{IDCG_n}$$

where $DCG_n$ is the Discounted Cumulative Gain and $IDCG_n$ is the Ideal Discounted Cumulative Gain.

The $IDCG_n$ is found by calculating Discounted Cumulative Gain of the ideal first $n$ returned services for a given query. The $DCG_n$ is calculated by Equation 10.

$$DCG_n = \sum_{i=1}^{n} \frac{2^{label(i)} - 1}{\log_b(1 + i)}$$

where $n$ is the number of services retrieved, $label(i)$ is the graded relevance of the service in the $i$th position in the ranked list, $b$ is the Discounting Factor which models the user’s persistence (e.g. impatient: $b = 2$; persistent: $b = 14$).

$NDCG_n$ gives higher scores to systems which rank services with higher relevance first and penalizes systems which return services with low relevance. In our experiments we set $b = 2$ and used graded relevance scheme with values from 3 (high relevance) to 1 (low relevance).

### 8. Results

Figure 5 and Figure 6 show the comparison of $P@n$ and $NDCG_n$ scores for the distributed service matchmaker with different number of cluster assignments. In both cases, LDA + Logic on Full Registry represents the best case scenario where the process checks every service in the registry. The distributed service matchmakers with different cluster assignments show the effect of restricting the scope of search on the average $P@n$ and $NDCG_n$ scores of the method. The curves are labeled LDA + Logic with $X$ Cluster Assignment, where $X$ indicates the number of clusters a service can be assigned to. The distributed service matchmaker with one cluster assignment exhibits the least $P@n$ and $NDCG_n$ performance at five services retrieved while as we allow services to be assigned to more clusters (thus increasing the scope of the search), the $P@n$ and $NDCG_n$ performance start approaching that of LDA + Logic on Full Registry.

The comparisons of average $P@n$ and $NDCG_n$ scores for all of our methods and the state-of-the-art service matchmakers are shown in Figures 7 and 8. The Precision@n results show that Pure Text-Matching and the logic-based OWLS-M0 were unable to find some of the relevant services that were not directly related to the queries through logic descriptions or keywords. LDA used the information captured in the latent factors to match services based on statistical similarity rather than just semantic or syntactic similarity and thus exhibited better precision than Text Matching and OWLS-M0. OWLS-M3 and OWLS-M4 also found more relevant services than the Pure Text-Matching and the logic-based OWLS-M0. The hybrid method successfully combined the merits of LDA-base matchmaking with weighted-link matching to accurately re-arrange the results thus outperforming all the other methods in terms of precision.

$NDCG_n$ evaluates the ranking mechanism and it is the most important measure for automated search and matchmaking engines. The top most relevant (i.e. the first five or ten) results retrieved by
a search and matchmaking engine are the main results that will be used by the client. The LDA and LDA + Logic matchmakers perform better than the other search and matchmaking mechanisms in this experiment. LDA + Logic holds a higher $NDCG_n$ than all other methods for any number of services retrieved, this reflects the accuracy of the hybrid ranking mechanism used by our method. Pure Text-Matching and OWLS-M0 have a low $NDCG_n$ because, as shown in the $P@n$ results, both mechanisms are unable to find some of the highly relevant services. OWLS-M3 and OWLS-M4 both exhibit a high $NDCG_n$ but they are outperformed by the LDA and LDA + Logic matchmakers. The higher $NDCG_n$ exhibited by our hybrid matchmaking methods reflects the accuracy of the hybrid ranking mechanism used in our methods. These results show that although the distributed service matchmaking does not perform as well as when the Hybrid matchmaker searches the full registry, it can still out perform all the other matchmaking methods making it a reliable matchmaker for a distributed environment such as the Internet of Things.

Figure 5. Comparison of $P@n$ scores for the Hybrid matchmaker with different numbers of Cluster Assignments for each service.

Figure 6. Comparison of $NDCG_n$ scores for the Hybrid matchmaker with different numbers of Cluster Assignments for each service.
9. Conclusions

Web services provide a suitable solution to enable machine-controlled and automatically structured service-oriented dynamic systems in Internet of Things. Semantic service matchmaking is the fundamental construct on which higher level service-oriented functionalities such as IoT service recommendation, composition, and provisioning are provided.

The hybrid semantic matchmaker for IoT Services proposed in this paper combines probabilistic matchmaking with a logical signature matchmaking method. The probabilistic component uses a latent semantic analysis model to extract latent factors from the IoT Service description data and uses these latent factors to overcome problems often encountered with logic-based techniques such as semantic synonomy. The logic-based component uses weighted-links to match the IO signature of a service to a request. This feature is important when specific input and output parameters are needed such as in service composition or service provisioning scenarios.
The proposed method exhibits higher performance than existing methods in terms of $P@n$ and $NDCG_n$. The weighted-link matchmaking provides a versatile approach for evaluating the degree of match of individual links and paves the way for the integration of the hybrid semantic service matchmaking method with higher-level service-oriented functionalities in the Internet-of-Things. The evaluation results show that although the distributed hybrid matchmaker does not provide the same level of performance as the hybrid matchmaker searching the whole registry, the distributed service matchmaker still performs better than the other methods and provides a solution that is accurate and can be distributed across different service repositories, making the architecture more suitable for a distributed environment such as the Internet of Things.

Future work will focus on creating an automated IoT Service composition solution that uses our hybrid semantic matchmaker to find candidate services for composition and/or compensation. We will also investigate the usage of non-parametric probabilistic topic models that allow the number of latent factors to increase gradually as new concepts are observed without the need of re-learning the latent factors (as discussed in Section 6).

Acknowledgments

The authors would like to thank Dr. Wei Wang from the University of Surrey for his comments and suggestions regarding IoT service modelling. This paper describes work undertaken in the context of the EU ICT IoT.est project (www.Ict-Iot.est.eu) contract number: 288385.

References