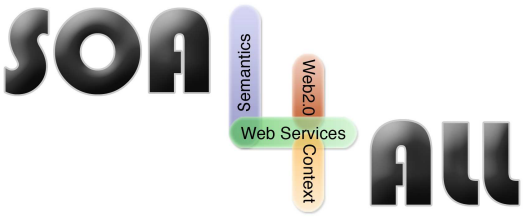

Project Number:	**215219**
Project Acronym:	**SOA4ALL**
Project Title:	**Service Oriented Architectures for All**
Instrument:	**Integrated Project**
Thematic Priority:	**Information and Communication Technologies**

D5.4.1 First Service Ranking Prototype

Activity:	Activity 2 - Core Research and Development	
Work Package:	WP 5 - Service Location	
Due Date:		M18
Submission Date:		03/09/2009
Start Date of Project:		01/03/2008
Duration of Project:		36 Months
Organisation Responsible of Deliverable:		UIBK
Revision:		1.0
Authors:	Ioan Toma UIBK Nathalie Steinmetz SEEKDA Holger Lausen SEEKDA Sudhir Agarwal UKARL Martin Junghans UKARL	
Reviewers:	Guillermo Alvaro Rey ISOCO Gianluca Ripa CEFRIEL	

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	**X**

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
1.0	03.06.2009	Initial Version	Ioan Toma
2.0	15.08.2009	Integration of initial UIBK contribution	Ioan Toma
3.0	18.08.2009	Adding seekda Section 2.1	Nathalie Steinmetz
4.0	19.08.2009	Adding initial Section 3	Nathalie Steinmetz
5.0	19.08.2009	Adding initial seekda Section 4.1	Nathalie Steinmetz
6.0	20.08.2009	Minor changes and conclusion	Martin Junghans
7.0	21.08.2009	Finishing seekda Section 4.1	Nathalie Steinmetz
8.0	22.08.2009	Integration of UKARL contribution Adding Executive summary, rework Introduction, Conclusions	Ioan Toma
9.0	22.08.2009	Minor updates	Nathalie Steinmetz
9.0	23.08.2009	Minor changes and final version release	Ioan Toma
10.0	28.08.2009	Internal Review	Guillermo Alvaro Rey (iSOCO), Gianluca Ripa (CEFRIEL)
11.0	31.08.2009	Restructured Section 4.3. Examples, future work, novelty added, review comments addressed.	Martin Junghans (UKARL)
12.0	02.09.2009	Address reviewers comments	Ioan Toma
13.0	03.09.2009	Review comments addressed	Nathalie Steinmetz
14.0	03.09.2009	Final version	Ioan Toma

Table of Contents

List of Listings

List of Figures

Glossary of Acronyms

Acronym	Definition
API	Application Programming Interface
D	Deliverable
EC	European Commission
NFP	Non-Functional Property
REST	Representational State Transfer
RDF	Resource Description Framework
SOAP	Simple Object Access Protocol
WP	Work Package
WSDL	Web Service Description Language
WSML	Web Service Modeling Language
WSMO	Web Service Modeling Ontology
XML	Extensible Markup Language
QoS	Quality of Service

Executive summary

Within this deliverable we present a set of service ranking approaches that complement each other and integrate nicely into an overall solution for ranking services on large scale. We introduce first, the techniques used to gather information useful for the ranking process. Such information is obtained by crawling relevant data from the Web and also by monitoring how services perform in terms of different criteria. We propose a set of models for various aspects considered at ranking time, models that provide the terminology to describe non-functional properties or Quality of Service aspects of services and requests. Additionally we propose extensions based on Fuzzy-Logic rules, to model services and user requests that enable a more expressive mechanism from the user perspective allowing vagueness.

The core contribution of the deliverable is an integrated ranking prototype based on several ranking approaches These approaches are using the data gathered through crawling and monitoring and formalized using the proposed models. The first method computes global ranks for services, score that is computed based on other ranks determined by related documents, monitoring data and WSDL metrics or confidence of the Web API classification. Another ranking approach proposed in this deliverable is using non-functional properties ontologies and logic rules for modeling of non-functional properties aspects. It computes ranking scores for services based on user specified non-functional properties and aggregates non-functional properties values into aggregated scores. Finally, the third approach proposes a fuzzy logic based ranking mechanism that considers an extended model of preferences including vagueness information. Last but not least, we have identified the need to develop a ranking mechanism that integrates the three individual ranking approaches mentioned above.

1. Introduction

The service-oriented approach on how to create system solutions for various customers' problems has received in the last years increasing attention from both industry and academia. The notion of service is at the very center of this approach, abstracting from the underlying software implementation and hardware resources. The software and resources are thus seen as services in a service-oriented architecture (SOA). Embracing the service-oriented paradigm rises however a set of new and tough challenges especially in the context of large systems such as the Web. The Web itself is evolving in what we call a Service Web, in which billions of parties are exposing and consuming services via advanced Web technology.

Building an effectively usable and domain independent service delivery platform requires scalable solutions for each of the challenges introduced by the Service Web vision including service discovery, ranking and selection, composition, etc. Among these tasks, discovery and ranking are core building blocks. As with most of the search products available on the market, it is not only important to determine the relevant results, but it is as well extremely important to provide the results in a relevant order. This is exactly the purpose of service ranking task, which complements the discovery task. While problems such as discovery ([2], [3], etc.) and composition ([4], etc.) for Semantic Web Services have been intensively studied, the service ranking problem, has rather gathered not so much attention. However, we argue that service ranking in an important task in the overall service usage process and thus it needs to be treated accordantly.

A ranking system computes a service rank to each set of services. The service rank is a quantitative metric that in some way shows the "importance" of a service within services network. To determine the rank of a service many types of information can be considered, ranging from unstructured information about services available online, monitoring data, to more formal descriptions based on various logics. Moreover service ranks can be computed independently or depending on the user request. A categorization of ranking approaches was proposed in [14] based on two aspects: (1) *local* or *global*, depending whether local or global knowledge is needed and (2) *absolute* or *relative*, depending whether the ranking score is absolute (i.e., independent of the user request) or refers to a particular client request. Considering these dimensions four types of ranking can be defined: (1) *local & absolute*, (2) *local & relative*, (3) *global & absolute*, and (4) *global & relative*.

In this deliverable, we propose three ranking approaches that cover the aspects mentioned above. First we introduce a global and absolute multi-valued ranking approach to compute service ranks. The approach is global, using information gathered from the Web and monitoring data, as well as absolute, as the rank of a service is not dependent on user's queries. Multi-value approaches are important for service ranking. Very often many different measurements need to be integrated into a service rank thus ranking systems are required to be multi-dimensional in nature. All three approaches that we introduce in this deliverable are actually multi-dimensional in nature. The second approach is based on the evaluation of multiple non-functional properties of services expressed as logical rules. It falls mainly in the relative category since the ranking is done according to user's query being often local since local knowledge is sufficient to perform the ranking. This approach puts emphasis on importance of *non-functional properties* aspects at ranking. The *non-functional descriptions* capture conditions that need to be fulfilled by the client in order to consume for example the functionality of a service. The third approach brings a set of modifications and extensions to the second approach. A combination of Description Logics and Fuzzy Logic rules is proposed as logical formalism to express user requests and services. This approach belongs to the same categories as the second approach.

The rest of the deliverable is organized as follows. Section 2 describes the methods developed to gather Web data, as well as methods for modeling and describing service properties used during the ranking process. Section 3 discusses the need of an integrated approach for ranking services, approach that combines and decides when each method should be used. The ranking approaches are described in details in Section 4. Section 5 contain details about how where to download, how to install and run the current prototype Finally, in Section 6 we summarize our contribution for the first service ranking prototype and describe possible extensions to be considered for the second version of the ranking prototype. The technical work presented in this deliverable has been published as scientific publications at international conferences. Appendix 1 contains a list of such scientific publications co-authored by the authors of the current deliverable.

2.Gathering data and modeling descriptions relevant for ranking

In this section we present our approaches for gathering data used by the ranking task. Furthermore we present different methods for modeling various types of information that are used as well at ranking.

2.1 Feature Aggregation Based on Ontologies

This section describes the data aggregation for the ranking approach that is based on feature aggregation using ontologies. To start with, we want to outline the novel aspects regarding this approach. We use semantic technologies to aggregate various aspects related to Web Services in a unified model, aspects that encompass information that is available about services by analysing their description and their hyperlink relations, by talking to their hosting server, etc. We do not rely on handcrafted, manually added, information, but only take into account real-world information that is anyway available. Also we do not assume – un-realistically- that we know how a service behaves on execution, what functionality he delivers, etc. (as would be the case in a 'man-in-the-middle' approach, where we would assume to have such knowledge before doing the ranking).

Within SOA4All we gather the services by crawling the Web for Web Services complying either to the WSDL standard or following a RESTful approach (a.k.a Web APIs). Together with the services we foster the Web for related documents, e.g. service descriptions, help pages, etc. The data that is resulting from the crawler comes together with RDF metadata that describes amongst others the relation from services and their related documents (in the case of WSDL services) or that tells us to what extent we believe that a certain Web resource is a Web API (in the case of RESTful services). More details on the crawl data and the corresponding RDF metadata can be found in [6]. The service meta-data can be used for a multi-value ranking approach, taking into account aspects like the number and the quality of related documents, classification scores of Web APIs, "live" monitoring data and metrics from the WSDL descriptions, as e.g. how much documentation is provided for a service.

In a next step, for the second ranking prototype, we will enlarge the number of criteria that we take into account. We expect that especially for the Web APIs we will take into account more metrics; we will base our improvements on evaluations of the current approach.

In the following we will first outline what RDF metadata we use for ranking; next we will describe the WSDL metrics and the monitoring data that we build upon and will in a last step provide an overview on existing and new ontologies that we use for modeling the ranking. The data described in this section is used for the 'Multi-valued ranking approach' as described in Section 4.1. Each of the following subsections will introduce the crawl metadata that it relies upon, if any. Section 2.1.4 will then describe in detail the new ontology elements that we use to describe the ranking.

We use the following namespaces and prefixes in the above mentioned subsections:

- *Service-Finder Service Ontology:* `sf` – "http://www.service-finder.eu/ontologies/ServiceOntology#"
- *seekda Crawl Ontology:* `sco` – "http://seekda.com/ontologies/CrawlOntology#"
- *seekda Ranking Ontology:* `sro` – "http://seekda.com/ontologies/RankingOntology#"
- *XML Schema:* `xsd` – "http://www.w3.org/2001/XMLSchema#"

2.1.1 Crawl Meta-data

In the case of WSDL services (and related resources) the meta-data delivered by the crawler consists mainly of annotations to the single Web documents, tying them on the one side to a

service and describing on the other side of what kind the relation to the service is. We will use the following information for ranking:

- Number of related documents per service
- Kind of relation from document to service

The meta-data is stored using elements of the Service-Finder Service Ontology[1], as shown in Listing 1 as RDF triples. `DirectInLink`, `DirectOutLink` and `TermVectorSimilarityAssociation` are sub-classes of a `DocumentAnnotation`.

```
<sf:DirectInLink > <sf:isAboutEntity> <sf:Service>
<sf:DirectInLink > <sf:belongsToDocument> <sf:Document>
<sf:DirectOutLink> <sf:isAboutEntity> <sf:Service>
<sf:DirectInLink > <sf:belongsToDocument> <sf:Document>
<sf:TermVectorSimilarityAssociation> <sf:isAboutEntity> <sf:Service>
<sf:TermVectorSimilarityAssociation> <sf:belongsToDocument> <sf:Document>
```

Listing 1: WSDL service and related documents meta-data used for ranking

In the case of Web APIs the crawl metadata describes some specific features of the Web document (e.g. number of external links, number of camel-case tokens) and provides (a) single scores that specify to what extent the two crawl classifiers (see [6]) believe that a given resource is a Web API and (b) a confidence score that is built from the single scores for convenience reasons. For ranking we will use the following information:

- What is the Web API Confidence score of a document?
- Which crawler classifier has classified the document as Web API?

The meta-data is stored using elements of the Service-Finder Service Ontology and of the seekda Crawl Ontology[2], as shown in Listing 2 as RDF triples. `Annotation` is the super-class of a `DocumentAnnotation`, the two classifiers that are used within the crawler, `SVMClassifier` and `WebAPIEvaluator`, are instances of the `Agent` and `AnnotatableEntity` is a super-class of `Service`.

```
<sf:DocumentAnnotation> <sf:hasScore> <xsd:number>
<sf:Document> <sco:hasWebAPIConfidenceScore> <xsd:number>
<sf:Annotation> <sf:source> <sf:Agent>
<sf:Annotation> <sf:isAboutEntity> <sf:AnnotatableEntity>
<sf:DocumentAnnotation> <sf:belongsToDocument> <sf:Document>
```

Listing 2: Web API meta-data used for ranking

2.1.2 WSDL Metrics

A WSDL describes a Web Service from an “operational” point of view: services, their operations, messages, message formats, endpoints, network bindings, etc. While this information as such is not useful for ranking the services, the documentation of the single elements is worth being taken into account (`<documentation>` tag within both WSDL 1.1 and WSDL 2.0): a well documented WSDL improves the ranking of the corresponding

[1] http://www.service-finder.eu/ontologies/ServiceOntology

[2] http://seekda.com/ontologies/CrawlOntology.rdfs

service.

We take into account the documentation of the service and of the operations. The data is stored using elements of the Service-Finder Service Ontology, as shown in Listing 3 as RDF triples.

```
<sf:Service> <sf:hasDescription> <xsd:string>
<sf:Operation> <sf:hasDescription> <xsd:string>
<sf:Service> <sf:implementsInterface> <sf:Interface>
<sf:Interface> <sf:hasOperation> <sf:Operation>
```

Listing 3: WSDL meta-data used for ranking

2.1.3 Monitoring Information

Interesting criteria for service ranking are related to Quality of Service information. One such information is the availability of services, i.e. their liveliness. This data is monitored and stored by seekda (Web Service search engine at http://seekda.com) on a daily basis. The availability is based upon the endpoint of a service and is only available for WSDL services, not for Web APIs. Monitoring the liveliness of a service does not mean that the functionality of the service is tested in any kind; it expresses whether the server where the service is hosted is reachable or not, checks at the same time whether the server is correctly implementing the SOAP protocol, whether the page needs an authentication, and more, based on the HTTP response codes.

The availability data will be delivered as RDF dump on a weekly basis. It will contain the average (percentaged) availability of a service over the last 6 months, the last month and the last week (if possible). Listing 4 shows the elements that we use from the Service-Finder Service Ontology to store this data.

```
<sf:Endpoint> <sf:availabilityLast6Months> <xsd:number>
<sf:Endpoint> <sf:availabilityLastMonth> <xsd:number>
<sf:Endpoint> <sf:availabilityLastWeek> <xsd:number>
<sf:Service> <sf:hasEndpoint> <sf:Endpoint>
```

Listing 4: Monitoring meta-data used for ranking

2.1.4 Ranking Ontologies

To structure and store the data as described above we rely, as already mentioned, on ontologies. Where possible, we reuse the Service-Finder Service Ontology and the seekda Crawl Ontology. Furthermore we have developed a new seekda Ranking Ontology that allows us to express the new ranking specific information that is not yet expressable within the other two ontologies (shown in Listing 5).

```
<sf:Service> <sro:hasNumberOfRelatedDocuments> <xsd:number>
<sf:Service> <sro:hasRelatedDocsRank> <xsd:number>
<sf:Service> <sro:hasWebAPIScoreRank> <xsd:number>
<sf:Service> <sro:hasWSDLMetricRank> <xsd:number>
<sf:Service> <sro:hasMonitoringRank> <xsd:number>
<sf:Service> <sro:hasGlobalRank> <xsd:number>
```

Listing 5: seekda Ranking Ontology

All ranking values are expressed by numbers between 0 and 1, 1 being the best-possible

rank. We calculate a rank for each of the criteria that we take into account from the crawl data and the monitoring. Then we calculate a final rank for each service. Details on how the ranks are calculated will be provided in Section 4.1.

2.2 Non-functional properties models

In this section we briefly introduce our approach for semantically describing NFPs of services. Non-functional properties capture important aspects of services. They are often used as input data for many service related tasks, including service ranking.

The service descriptions that we consider are annotated using WSMO-Lite [5], a lightweight semantic approach for describing services. WSMO-Lite distinguishes between various aspects of a service description. More precisely WSMO-Lite identifies five aspects as being central to a service description, namely *functional, behavioral, information, technical,* and *non-functional*. The *functional descriptions* contain the formal specification of what exactly a service can do. The *behavioral descriptions* are about how the functionality of the service can be achieved in terms of the interaction with the service and in terms of the functionality required from other Web services. The *information model* defines the data model for input, output and fault messages. The *technical descriptions* define messaging details, such as message serializations, communication protocols, and physical service access points. Finally, *non-functional descriptions* define any incidental details specific to a service provider, or the service implementation or its running environment.In this section we discuss the latter, as this kind of properties are most relevant at ranking. One important challenge is how to model non-functional properties of services.

WSML-Lite introduces an element called *wl:NonFunctionalParameter* that is used as a placeholder for a concrete domain specific non-functional property. However it does not provide a specific way how to model non-functional properties of services, this being outside the scope of WSMO-Lite. Our approach for modeling non-functional properties semantically extends WSMO-Lite. We first defined a set of ontologies that provide the terminology needed for specifying non-functional properties of services. Ontologies describing the non-functional properties domain can be imported and concepts, relations, instances can be used in service descriptions. The set of non-functional properties ontologies that we have created is available at [1]. The ontologies have been modeled in WSML and include models for: locative, temporal, availability, obligation, price, payment, discounts, rights, trust, quality of service, security, intellectual property, rewards, provider, measures and currency aspects. RDF versions of the ontologies are as well available.

Having provided ontologies for non-functional domains, we developed a model how to describe non-functional aspects of services. They are modeled similar to capabilities, more precisely by means of logical expressions, i.e., axiom. The schematic model is provided in Listing 6

```
Class nonFunctionalProperty
    hasAnnotations type annotations
    hasDefinition type axiom
```

Listing 6: Non-functional property model

To exemplify how non-functional properties of a service can be described let's consider a shipping service. The shipping service allows requestors to order a shipment by specifying, senders address, receivers address, package information and a collection interval during which the shipper will come to collect the package.

Listing 7 contains a concrete example on how to describe one non-functional property of a service from the Shipment Discovery scenario of the SWS Challenge[3] (i.e., Muller), namely obligations. The listing contains only the specification of obligations aspects without any functional, behavioral or any other non-functional descriptions of the service. In an informal manner, the service obligations can be summarized as follows: in case the package is lost or damaged Muller's liability is the declared value of the package but no more than 200$. Following our model for NFPs, Muller's obligations are expressed as logical rules in WSML. In a similar way other non-functional properties can be described.

```
// namespaces and prefixes
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix wl: <http://www.wsmo.org/ns/wsmo-lite#> .
@prefix so: <http://sws-ranking/Shipment.wsml#> .
@prefix loc: <http://www.wsmo.org/ontologies/nfp/locationNFPOntology#> .
@prefix pr: <http://www.wsmo.org/ontologies/nfp/priceNFPOntology#> .
@prefix xs: <http://www.w3.org/2001/XMLSchema#> .
@prefix wsml: <http://www.wsmo.org/wsml/wsml-syntax#> .
@prefix ex: <http://example.org/onto#> .
// ontology example
<> rdf:type wl:Ontology.
so:ShipmentConfirmation rdf:type rdfs:Class .
so:hasPackage rdf:type rdf:Property ;
rdfs:domain so:ShipmentConfirmation ;
rdfs:range so:Package .
so:hasPrice rdf:type rdf:Property ;
rdfs:domain so:ShipmentConfirmation ;
rdfs:range pr:AbsoultePrice .

so:Package rdf:type rdfs:Class .
so:hasLength rdf:type rdf:Property ;
rdfs:domain so:Package ;
rdfs:range xs:float .
so:hasWidth  rdf:type rdf:Property ;
rdfs:domain so:Package ;
rdfs:range xs:float .
so:hasHeight  rdf:type rdf:Property ;
rdfs:domain so:Package ;
rdfs:range xs:float .
so:hasWeight rdf:type rdf:Property ;
rdfs:domain so:Package ;
rdfs:range xs:float .
so:hasStatus rdf:type rdf:Property ;
rdfs:domain ex:Package ;
rdfs:range so:PackageStatus .
so:hasDeclaredValue rdf:type rdf:Property ;
rdfs:domain ex:Package ;
rdfs:range pr:MonetrayAmount .

so:PackageStatus rdf:type rdfs:Class .
so:ShipmentOrderRequest rdf:type rdfs:Class .
so:hasRequestedPackage rdf:type rdf:Property ;
rdfs:domain so:ShipmentOrderRequest ;
```

[3] http://sws-challenge.org/

```
rdfs:range so:Package .
so:hasDestination rdf:type rdf:Property ;
rdfs:domain so:ShipmentOrderRequest ;
rdfs:range loc:Address .

//Muller is shipping only packages with a weight lower than 100
ex:MullerPrecondition rdf:type wl:Condition ;
rdf:value
"?shipmentOrderRequest[hasRequestedPackage hasValue ?package, hasDestination
hasValue ?address]
memberOf so#ShipmentOrderRequest and
?package[hasWeight hasValue ?w] memberOf so:Package and
?w < 100"
wsml:AxiomLiteral .
ex:MullerEffect rdf:type wl:Effect ;
rdf:value
"?shipmentConfirmation[hasRequestedPackage hasValue ?package] memberOf
so#ShipmentConfirmation"
wsml:AxiomLiteral .

// non-functional property example
pr:AbsolutePrice rdfs:subClassOf wl:NonFunctionalParameter .
ex:ShippingPrice rdf:type pr:AbsolutePrice ;
pr:hasAmount "30"xs:integer.
pr:hasCurrency "euro"cur:Euro.

// classification example
ex:ShippingService rdf:type wl:ClassificationRoot .
ex:AsiaShippingService rdfs:subClassOf ex:ShippingService .
ex:EuropeShippingService rdfs:subClassOf ex:ShippingService .
```

Listing 7: Example of service non-functional property in WSML-Lite

In Section 4.2 we explain how such descriptions are processed and used in a ranking prototype.

2.3 Extension of User Preference Modeling

In this section a Fuzzy logic approached for modeling user preferences is introduced. More precisely we use fuzzy IF-THEN rules to express user preferences and relationships between values of non-functional properties. Fuzzy logic enables the modeler to express vagueness in user preferences which is sometime desired. We give a short introduction to fuzzy IF-THEN rules in Section 2.3.1 and discuss the modeling of categories and user request using this logic respectively in Sections 2.3.2 and 2.3.3.

2.3.1 Fuzzy IF-THEN Rules

Fuzzy IF-THEN rules were introduced and applied for practical optimization problems by L. A. Zadeh [7]. Formally, an optimization problem can be described as the a function f that computes an optimal solution in the space spanned by its parameters $x_1,...,x_n$. Classical optimization methods for determining f were too complex to be applicable for practical problems, since f is in many cases extremely non-linear or even non-differentiable or not even continuous. However, it was known that for many practical problems a good approximation $\bar{f}$ of f is sufficient to solve the purpose. Fuzzy IF-THEN rules are a new way for determining $\bar{f}$ much more efficiently than the classical approximation methods.

A Fuzzy IF-THEN rule base can be formally described as follows. Given linguistic variables $X_1,...,X_m$ with $m \geq 1$ and $Y_1,...,Y_n$ with $n \geq 1$. We assign the linguistic terms $T_{\mu 1},...,T_{\mu m}$

with each linguistic variable X_μ with $1 \leq \mu \leq m$ and the linguistic term T'_ν with each linguistic variable Y_ν with $1 \leq \nu \leq n$.

A common form of a Fuzzy IF-THEN rule base is as follows.

$$IF\ X_1 = T_{11}\ and \ldots and\ X_m = T_{m1}\ THEN\ Y_1 = T'_1$$
$$\vdots$$
$$IF\ X_1 = T_{1n}\ and \ldots and\ X_m = T_{mn}\ THEN\ Y_n = T'_n$$

Note, that the IF part of a rule can also contain “or” and “not” in addition to “and” for connecting atomic terms of the form $X_i = T_{ij}$.

In order to apply a Fuzzy IF-THEN rule base to a concrete practical problem, a semantic interpretation of a Fuzzy IF-THEN rule base must be fixed. This means a complex premise (IF part) composed of atomic terms as defined above need to be mapped to a fuzzy set. Such a fuzzy set is computed with the help of fuzzy sets $F_{1\nu}, \ldots, F_{m\nu}$ that are interpretations of the linguistic terms $T_{1\nu}, \ldots, T_{m\nu}$.

There are basically two principles for interpreting a fuzzy rule base as a fuzzy set, namely “First Inference Then Aggregation (FITA)” and “First Aggregation Then Inference (FATI)”. It has been shown in [10] that the two principles are equivalent.

In FITA, each rule is processed as follows:

- Compute the degree of fulfillment of each term in the premise.
- From each computed degree in the previous step, compute the overall degree of fulfillment of the whole premise. While computing the overall degree, the individual degrees are combined according to the Boolean operators “and” “or” and “not” used for combining the terms.
- Compute a new fuzzy set from the fuzzy set representing the linguistic term in the conclusion of the rule and the overall degree of fulfillment of the premise of the rule.

At the end of above steps that represent inference, we have for each rule a new fuzzy set. Now, in the aggregation step, these fuzzy sets are aggregated to one fuzzy set. Thus, at the end of FITA, we obtain one fuzzy set that represent the solution of the problem. The final solution is obtained by performing defuzzificaiton on the final fuzzy set.

In case of FATI, the steps of inference and aggregation are exchanged. That is, first all the fuzzy sets in the conclusions of the rules are combined to one fuzzy set and then a new fuzzy set is computed from the aggregated fuzzy set and the degree of fulfillments of the terms in the premise of the rules.

2.3.2 Modeling Categories as Fuzzy Membership Functions

We define a membership function μ_C for a category C as a finite and non-empty set of points $(x,y) \in \mathbf{R}^2$, with $y \in [0,1]$. x denotes individuals of a concept. We assume concepts whose instances can be mapped to an interval scale and instances to be real numbers. In case, C is infinite, we can use special values $MINVAL$ and $MAXVAL$ for denoting minimum and maximum possible values for x. The following axioms define such a membership function:

$$Point \sqsubseteq \top \sqcap \exists x \sqcap \exists y\, \mathbf{n}\geq_0 (y)\ \mathbf{n}\leq_1 (y)$$

$$MF \sqsubseteq \top \sqcap \exists p.Point$$

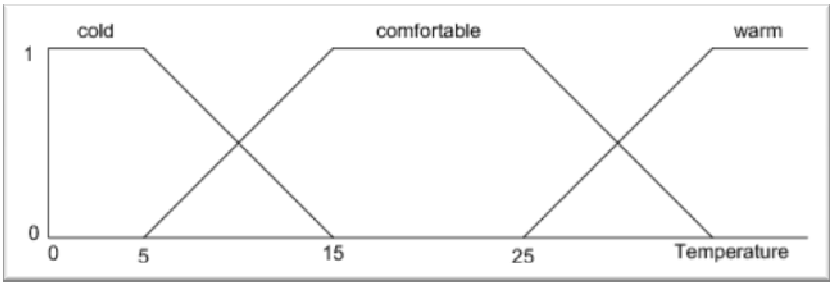

Figure 1. Example membership functions.

As already stated in the introduction of fuzzy logics, one of the biggest advantages of fuzzy logics is that fuzzy rules are easier to define and understand for humans, since they contain linguistic variables and linguistic terms, e.g., “temperature = cold” rather than precise values like “temperature = 5℃”. A linguistic term is a membership function that maps each possible value of the linguistic variable to a real number between 0 and 1. Each linguistic term of a linguistic variable covers a range of possible values that the linguistic variable can take and the actual advantage of fuzzy inferencing lies in the smooth transition between linguistic terms covering adjacent value ranges. Therefore, even if fuzzy sets can be seen as generalizations of classical (crisp) sets, the advantage of fuzzy logic can be taken only for those concepts whose instances can be ordered. Figure 1 shows the linguistic terms “cold”, “comfortable” and “warm” modeled as membership functions for a linguistic variable “Temperature”.

2.3.3 Modeling User’s Goals as Fuzzy Rules

A goal can be regarded as those NFPs that a Web service should fulfill in order to be accepted for further consideration. We specify different levels (categories) of acceptance with fuzzy membership functions. Thus, a user’s goal is just a set of fuzzy IF-THEN rules. The IF part is a combination of linguistic terms of the properties that are important for the user, e.g., “IF (temperature = warm)”. The linguistic terms can be combined by using conjunction ($\sqcap$), disjunction ($\sqcup$) and negation ($\neg$). The THEN part is just one of the categories of acceptance. Intuitively, a fuzzy rule describes which combination of property values a user is willing to accept to which degree, where property values and degree of acceptance are fuzzy sets, i.e., vague. An example from the industrial and process and control domain is the following IF-THEN rule, which might control the opening of windows.

IF (temperature = hot $\sqcap$ humidity = high) THEN (window = ajar)

3. Overall SOA4All Ranking mechanism

As we did already outline in the introduction (Section 1), ranking is a very important aspect – one of the building blocks - in any service delivery platform. A good service rank eases the task of the service consumer to choose the fitting service for his needs, especially if we work with services on a very large scale (i.e. Web scale). We have developed a ranking mechanism within SOA4All that builds upon three different approaches:

A. ontology-based feature aggregation for multi-valued ranking

B. multi-criteria ranking based on non-functional properties, and

C. fuzzy logic based ranking

Approach A takes into account on the one hand service meta-data that is based on the actual service descriptions available on the Web and on the other hand meta-data that is resulting from a constant monitoring of the liveliness of the services. The resulting rank will be calculated “offline”, i.e. it will be provided as batch to the SOA4All ranking component (see Section 4.1 for more details).

Approach B works over semantically annotated non-functional properties of the services, mainly properties that are to be provided by the service providers. The semantic non-functional properties descriptions are evaluated online and results values are aggregated in order to compute a score for each services. The rank that is calculated based on the non-functional properties can be directly influenced by the potential service customers (i.e. the users of the SOA4All Studio).

Approach C works as well over semantically annotated non-functional properties descriptions of services. User requests are represented using fuzzy logic that enables a higher degree to express vagueness and relations between non-functional properties.

Approaches A and B are implemented as an own “component”, as will also become visible in Section 4, within the implementation descriptions. Approach C is not yet implemented, but will be implemented as independ component. The single approaches work independently of each other and do rather complement each other. Approach A is executed first on a large set of services, as the rank is calculated beforehand and does not require any computation effort at run-time. This approach can though be used as “filter” before the more expensive ranking approaches B and C are executed.

The way the approaches complement each other is shown in Figure 2. They do not base their rank on the same underlying data and take into account different aspects of services.

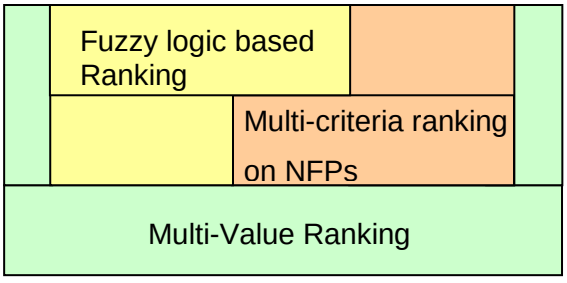

Figure 2. Integration of the three ranking approaches.

We plan to combine the three approaches for the second ranking prototype. We will therefore develop a way that provides the user with best-possible choice for his ranking approach on the one hand and transparency regarding the ranking results on the other hand.

4. Ranking approaches

The three different approaches for service ranking, namely the multi-valued ranking approach, the multi-criteria ranking based on non-functional properties, and the fuzzy logic based ranking are described in this section.

4.1 Ontology-based Feature Aggregation For Multi-valued ranking

In Section 2.1 we have outlined what aspects of services we aggregate to build our ontology-based feature aggregation for multi-valued ranking approach. The data is aggregated into a unified model using mainly three ontologies, the Service-Finder Service Ontology, the seekda Crawl Ontology and the seekda Ranking Ontology. We do not only use the ontologies for the meta-data that is needed to calculate the rankings but as well to structure and store the rank values. The fact that we have all the service meta-data that we need for our ranking available as semantic data allows each 'semantic-aware' client to build its own ranking – based on the same service meta-data – creating individual rules (e.g. using SPARQL).

In the following we will present the way we combine the data gathered as described in Section 2.1 to get a global ranking value (Section 4.1.1), will provide an overview over the implementation of the approach and the resulting data format (Section 4.1.2) and will outline the novelties of the approach (Section 4.1.3) and will present some future work (Section 4.1.4).

4.1.1 Rules for a global multi-valued rank

The ontology-based feature aggregation for multi-valued ranking approach differs for the two types of services we support in SOA4All: WSDL services and Web APIs. For WSDL services we first calculate three independent ranking values (based on crawl meta-data like info on related documents, WSDL metrics and monitoring data) that are then combined to one global rank. For Web APIs we so far only take into account one value: the Web API confidence score.

The following subsections present the rules that we apply to calculate the single rank values (described in a procedural pseudo-code). We estimate that an implementation using a procedural language is more performant than using a declarative one. Nevertheless it is as well possible to describe and implement the rules in a declarative language, or, e.g., using SPARQL.

Related Documents Rank

This rank is based on the crawl meta-data that is delivered by the crawler, as shown in Listing 1 and will be calculated based on the following information:

- How many related documents does a service have? We need to check the document annotations that belong to a service and then count the unique documents that are tied to the annotations (as more annotations can refer to the same document).
- How is the document related to a specific service? Documents can be direct inlinks or direct outlinks of the WSDLs that belong to a service, or the connection to the service can be coming from a term vector analysis of the documents and the service.

In a first step we thus need to calculate the number of related documents per service. To do so it is not enough to just take the number of document annotations as one document might have several annotations (e.g. a document that has a `DirectOutLink` annotation and a `TermVectorSimilarityAssociation`). We need to first extract all `DirectInLink`, `DirectOutLink` and `TermVectorSimilarityAssociation` annotations. This way we get the identifiers of all documents that correspond to the annotations. Now we count the documents, counting multiple occurences of the same document only once. This value is then stored using the `hasNumberOfRelatedDocuments` relation of the seekda Ranking

Ontology (see Section 2.1.4).

Now the related documents rank is calculated as follows (described in pseudo-code) in Listing 8:

```
// get the average number of related docs per service
average = totalNumberOfRelatedDocs / numberOfServices;
// get the root mean square deviation of the distribution of related docs
per service
sumDeviationFromAverage = 0;
for (Service s : allServices) {
    sumDeviationFromAverage += s.numberOfRelatedDocs – average;
}
variance = power(sumDeviationFromAverage, 2) / numberOfServices -1;
rootMeanSquareDeviation = positiveSquareRoot(variance);
// get max outliers values
maxOutlier = average + (2.5 * rootMeanSquareDeviation);
// take into account the kind of relation from document to service. If a
// service has a number of related documents that is outside of the max
// outlier value we set the number to the average of related documents per
// service in order to not allow spam to influence the ranking value
for (Service s : allServices) {
    temporaryRank = 0;
    if (s.numberOfRelatedDocs > maxOutlier) {
        s.numberOfRelatedDocs = maxOutlier;}
    if (s.hasInlink) {
        temporaryRank = s.numberOfRelatedDocs * 5;}
    if (s.hasTermVectorAssociatedDoc) {
        temporaryRank += s.numberOfRelatedDocs * 4;}
    if (s.hasOutlink) {
        temporaryRank += s.numberOfRelatedDocs * 2;}
    s.finalRank = temporaryRank / maxOutlier / 11;
}
```

Listing 8: Calculation of the Related Documents Rank

The single values that are used for the single kinds of related documents to calculate the temporary rank are currently experimental. These might be changed on a frequent basis until we discover the values that seem optimal for our needs. The final rank is stored for each service using the `hasRelatedDocsRank` relation of the seekda Ranking Ontology.

WSDL Metrics Rank

This rank is based on metrics that we extract from the WSDL descriptions. We currently take into account the documentation of (a) the service element, and (b) the operations. The rank is calculated as follows in Listing 9:

```
for (Service s : allServices) {
    finalRank = 0;
    if (s.hasServiceDocumentation) {finalRank = 1;}
    if (s.hasOperationDocumentation) {finalRank += 3;}
    s.finalRank = finalRank/4;
}
```

Listing 9: Calculation of the WSDL Metrics Rank

We put more importance on the documentation of the single operations than of the service

documentation, as we think that the operation might contain useful information regarding the functionality provided by the operation and regarding its invocation. We currently do not differentiate between whether all operations of a service are documented or only one or some. The final rank is stored for each service using the `hasWSDLMetricRank` relation of the seekda Ranking Ontology.

Monitoring Rank

This rank is based on the liveliness information of a service, e.g., is the server reachable, does it correctly implement the SOAP protocol, etc. This liveliness information is delivered by seekda on a weekly basis as shown in Listing 4. The availability score is a number between 0 and 1 that is set depending on the endpoint check result. The score is, e.g., 0 for read time-outs or errors and 1 if, based on the resulting payload (e.g., XML fault), we are rather sure to be talking to a WSDL over SOAP. Inbetween different scores are set to express pages that are not found, pages that require a login or an authentication, etc., mostly based on the HTTP response code.

We get the average service availability score for different time periods: last week, last month and last 6 months. We assume that the long-time availability of a service is more relevant than only the short-time availability over one week. It is important to note that this rank does not state anything about whether the functionality that the service announces is correctly implemented or not. The rank is calculated as follows in Listing 10 and is stored for each service using the `hasMonitoringRank` relation of the seekda Ranking Ontology:

```
for (Service s : allServices) {
    finalRank = ((s.availabilityLastWeek * 1.5) + (s.availabilityLastMonth
* 2.5) + (s.availabilityLast6Months * 6)) / 10;
}
```

Listing 10: Calculation of the Monitoring Rank

WebAPI Rank

For ranking Web APIs we currently only take into account the Web API confidence score. This score is calculated based on two classifiers within the crawler that check whether a Web resource might be a Web API or not. In the future we might extend the metrics for the Web API ranking. The rank is based on the crawl meta-data that is delivered by the crawler, as shown in Listing 1 and will be calculated based on the following information:

- What is the Web API Confidence score of a document? This score is a final confidence score that is calculated (based on the crawler administrators' estimation and experience) from the single scores provided by the Web API classifiers.
- Which crawler classifier has classified the document as Web API? As described in [6], one classifier (SVM Classifier) has been trained on a set of data and automatically classifies documents based on this training. The other classifier (Web API Evaluator) performs structural and term vector analyses of the resources and assigns scores for specific indicators.

To calculate the rank we thus need to extract both the score and the component that has assigned the score. Based on first evaluations of the classifiers (see [6] for details), we deem the score of the SVM classifier more important than the one of the Web API Evaluator. Listing 11 shows how the rank is calculated:

```
for (Service s: allRESTServices) {
    finalRank = 0;
    if (s.hasSVMClassifierAnnotation) {
        finalRank = s.hasWebAPIConfidenceScore * 3;}
    if (s.hasWebAPIEvaluatorAnnotation) {
        finalRank += s.hasWebAPIConfidenceScore * 1;}
    s.finalRank = finalRank/4;
}
```

Listing 11: Calculation of the WebAPI Rank

Global Rank

As already mentioned above, the calculation of the global rank differs depending on whether the ranked service is a WSDL-based service or a Web API. For WSDL services we calculate the global rank based on the Related Documents Rank, the WSDL Metrics Rank and the Monitoring Rank. The single ranks are numbers between 0 and 1, and from these we calculate the global rank as follows in Listing 12, putting equal relevance on the availability of documentation (related documents being estimated more important than the documentation within the WSDL) and on the liveliness of a service. The global rank is stored for each service using the `hasGlobalRank` relation of the seekda Ranking Ontology.

```
for (Service s : allServices) {
   s.globalRank = (s.hasRelatedDocsRank * 0.35) + (s.hasWSDLMetricRank *
0.15) + (s.hasMonitoringRank * 0.5);
}
```

Listing 12: Global Rank Calculation for WSDL-based services

For Web APIs, the calculation is simple: the WebAPI Rank is at the same time the global rank of the service.

4.1.2 Implementation

The service ranks produced by seekda take as input meta-data in RDF triples format and returns the ranks in the same way. We will use the seekda Ranking Ontology (as introduced in Section 2.1.4) to store and distribute the service ranks. Inbetween we have a Java component that calculates the ranks as described in Section 0.

Together with the single ranks we will distribute the meta-data triples that the ranks are based upon. As both the single ranks and the global rank are values between 0 and 1, all reasoners that can do ordering on numbers are able to work with the ranks. The RDF data will be delivered as dump on a weekly basis by seekda. The triples will then be added to the SOA4All semantic spaces and will be available to the Studio.

4.1.3 Novelty

The approaches we follow with our multi-valued ranking in general and the export of the ranking related data in a semantic format have three major advantages (and novelties):

1. Each RDF aware client can understand the rational of a ranking, i.e. it can work with the final ranks, but can as well analyse the data on which the ranking is based. This is a transparent approach that is very unlike most of the currently available approaches. It allows in the end each client to perform, if desired, its own ranking calculation with the service meta-data that is provided.
2. Our ranking approach covers and combines many aspects of a service, like its documentation, the existence of related information that is available on the Web, the liveliness data (which is a strong QoS criteria), and, in the case of Web APIs ranking,

the classification score of services. Most existing ranking approaches concentrate rather on individual ranking criteria like availability, reputation, etc.

3. We get one part of our service meta-data from the Service Crawler, which extracts this information in its postprocessing process. The other part relies on a daily monitoring of the connectivity of the services; valuable data which is provided by seekda and which is – up to our knowledge – only monitored in that detailed way by the seekda Web Service search engine[4].

4.1.4 Future Work

We plan to extend the multi-valued ranking approach in the future with community data. That is we will use statistical data coming from the SOA4All Studio to improve the ranking. Such data can include views of services, edits of services, usage of single services in composition, invocations, etc.

Furthermore we will evaluate the current ranking approach and implement improvements on the existing ranks based on this.

4.2 Multi-criteria ranking based on non-functional properties

In this section we present a multi-criteria ranking approach that considers multiple non-functional properties. We describe first the algorithm implemented as part of the approach. An optimization of the presented approach is presented as well in this section. The section contains as well details about the implementation, novelty of the approach and future work.

4.2.1 Algorithm

Non-functional properties specified in the user request and service descriptions are formalized by means of logical rules using terms from NFP ontologies as described in Section 2.2. The logical rules used to model NFPs of services are evaluated, during the ranking process, by a reasoning engine. Additional data is required during this process: (1) which NFPs the user is interested in, (2) the importance of each of these NFPs, (3) how the list of services should be ordered (i.e., ascending or descending) and (4) concrete instance data. The non-functional properties values obtained by evaluating the logical rules are sorted and the ordered list of services is built.

The algorithm for multi-criteria ranking based on non-functional properties is presented in *Listing* 13.

[4] http://seekda.com

```
Data: Set of services S_Ser, Goal G.
Result: Order list of services L_Ser.
begin
1   Ω ⟵ ∅, where Ω is a set of tuples [service,score];
2   λ = extractNFPs(G), where λ is a set of tuples [nfp, importance];
3   G_Know = extractInstancesKnowledge(G);
4   d = extractOrderingSense(G);
5   β ⟵ ∅, is a set of quadruples [service,nfp,nfpvalue,importance];
6   for s ∈ S_Ser do
7       for nfp ∈ λ do
8           imp = λ.getImportance(nfp);
9           if nfp ∈ s.nfps then
10              rule = extract(nfp, s);
11              nfpvalue = evaluateRule(rule, G_Know);
12              β = β ∪ [s, nfp, nfpvalue, imp];
            else
13              β = β ∪ [s, nfp, 0, 0];
14  for s ∈ β do
15      score_s = 0;
16      for nfp ∈ β do
17          nfpvalue = β.getNFPValue(s, nfp);
18          nfpvalue_max = max(β.npf);
19          score_s = score_s + imp * nfpvalue / nfpvalue_max;
20      Ω = Ω ∪ [s, score_s];
21  L_Ser ⟵ sort(Ω, d);
end
```

Listing 13: Multi-criteria ranking for NFPs

First, a set of tuples containing non-functional properties and their associated importance is extracted from the user request (line 2). Considering the user request example provided in Listing 14, the list of non-functional properties and their importance are extracted from the ontology. In our example, the list of non-functional properties includes *obligations* with a 0.1 importance value, *price/discounts* with a 0.6 importance value, *delivery time* with a 0.2 importance value, and *rewards* with a 0.1 importance value. If no importance is specified the default value is consider to be 0.5, which stands for a moderate interest in the respective non-functional property. The importance is a numeric value ranging from 0 to 1, where 1 encodes the fact that the user is extremely interested in the non-functional property and 0 encodes the opposite. Instance data from the goal is extracted (line 3) and a knowledge base is created. Given the user request in Listing 14, the extracted instance data contains information about the receiver, the package and the destination address. The last step in extracting relevant information for the ranking process is to identify how the results should be ordered i.e., ascending or descending (line 4).

```
// namespaces and prefixes
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix wl: <http://www.wsmo.org/ns/wsmo-lite#> .
@prefix so: <http://sws-ranking/Shipment.wsml#> .
@prefix loc: <http://www.wsmo.org/ontologies/nfp/locationNFPOntology#> .
@prefix pref: <http://www.wsmo.org/ontologies/nfp/preferenceOntology#> .
```

```
@prefix xs: <http://www.w3.org/2001/XMLSchema#> .
@prefix wsml: <http://www.wsmo.org/wsml/wsml-syntax#> .
@prefix ex: <http://example.org/onto#> .
// ontology example
<> rdf:type wl:Ontology.

ex:prefObligation rdf:type pref:Preference;
 pref:hasNonFunctionalProperty pref:Obligation.
 pref:hasInterestValue “0.1”xs:float.
 pref:hasOrder pref:Descending.

ex:prefPriceAndDiscounts rdf:type pref:Preference;
 pref:hasNonFunctionalProperty pref:PriceAndDiscounts.
 pref:hasInterestValue “0.6”xs:float.
 pref:hasOrder pref:Ascending.

ex:prefDeliveryPrice rdf:type pref:Preference;
 pref:hasNonFunctionalProperty pref:DeliveryPrice.
 pref:hasInterestValue “0.2”xs:float.
 pref:hasOrder pref:Ascending.

ex:prefRewards rdf:type pref:Preference;
 pref:hasNonFunctionalProperty pref:Rewards.
 pref:hasInterestValue “0.1”xs:float.
 pref:hasOrder pref:Descending.

ex:ShipmentRequest1 rdf:type so:ShipmentRequest.
so:hasPackage ex:GumblePackage.

ex:GumblePackage rdf:type so:Package;
so:hasLength “10”xs:float.
so:hasWidth  “2”xs:float.
so:hasHeight  “3”xs:float.
so:hasWeight  “150”xs:float.
```

Listing 14: Example of user request

Once the preprocessing is completed each service is assessed in order to determine whether the non-functional properties specified in the user request are available in service description. If this is the case, the algorithm extracts the corresponding logic rules (line 10) and evaluates them (line 11) using a reasoning engine which supports WSML rules (e.g. IRIS[5]). A quadruple structure is built (line 12 and 13). This contains the computed value and its importance for each service and non-functional property.
An aggregated score is computed for each service by summing the normalized values (line 17) of non-functional properties weighted by importance values (line 18). The results are collected in a set of tuples, where each tuple contain the service id and the computed score (line 20). Finally, the scores are ordered as specified by the user and the final list of services is returned (line 21).

4.2.2 Implementation

The multi-criteria ranking approach takes as input a set of services annotated using the WSMO-Lite ontology and a user request/goal, all in RDF format. The result is presented in a form of ordered list of services. Furthermore, for each service in the list additional information

[5] http://www.iris-reasoner.org/

can be provided such as the score for each non-functional property requested by the user as well as the aggregated score. The implementation uses the IRIS reasoner to evaluate the values of non-functional properties. The multi-criteria ranking approach is implemented as a Java component and is exposed as a web service.

The high level interface for the ranking component is provided below.

```
@WebMethod(operationName = "rank")
@WebResult(name = "rankedServices")
String[] rank(@WebParam(name = "services")  String[] services,
                  @WebParam(name = "goalURI") String goal) throws
RankingException;
```

In the above method signature the input array of Strings represents the IDs of the services being ranked and the output array of Strings represents the same IDs of services but in this case the services (IDs) are ranked according to user preferences available in the goal description.

4.2.3 Novelty

The novelties of our proposed multi-criteria ranking approach are summarized below:

1. We introduce a ranking algorithm for services that uses, on one hand ontological representations of non-functional properties, and on the other hand multiple non-functional properties dimensions.
2. The user is presented not only with the list of service ranks but also with an explanation of why the service rank was computed in the way it was computed. The results of invoking the multi-criteria ranking component contain the score for each requested non-functional property as well as the overall aggregated score.

4.2.4 Future Work

We are currently in the process of developing an optimized version of our multi-criteria ranking approach for WSMO-Lite annotated services using rank aggregation techniques. A consensus mechanism that integrates rank lists produces by individual rank engines into a communally agreed rank list will form the basis of the optimized approach. Another research direction that is currently investigated is the usage of data coming from social web sites in the ranking process.

4.3 Fuzzy Logic Based Ranking Approach

In this section we provide the fuzzy logics based Web service ranking approach. We therefore introduce the conceptual steps in order to compute a ranking. As this approach is not implemented yet, we then introduce a concrete example that illustrates this ranking approach. In consistency with the two prior presented approaches, we show the novelty and future work of this approach at the end of this section,

4.3.1 Algorithm

In order to calculate minimum, maximum, addition, subtraction, multiplication and division of two real numbers, we need the predicates MIN , MAX , ADD , SUB , MUL and DIV of arity three (two inputs and one output), respectively. We need one more predicates LEQ of arity two to evaluate whether the first number is less than or equal to the second number.

Let O represent the concept that represents the acceptance and let O be categorized in k categories represented by $O_1 \dots O_k$. Further, there exists k rules $R_1 \dots R_i \dots R_k$, where R_i has O_i as conclusion. Further, let μ_X represent the membership function for the category X . In the following, we show how we calculate $RANK(O,a,r)$, the ranking r of the individual a

with respect to objective O . We will use FITA principle (First Inference Then Aggregation) instead of FATI (First Aggregation Then Inference) for the interpretation of fuzzy rules. [10] has shown that the two principles are equivalent.

Fuzzy Inferencing. For a given individual a and a set of rules $R_1, \ldots, R_k$, we calculate for each rule R_i, with what degree the individual a fulfils the rule R_i. Note, that we assume that the objective has been divided into k categories. The result of this step is the set of k new membership functions.

The premise of a rule is any arbitrary combination of conjunction, disjunction, or negation of property categories, that is membership functions. The conclusion of a rule is exactly one category of the objective, again a membership function. The reasoner fetches offers from the metadata repository and calculates for each offer and each rule the degree to which the offer fulfils the rule. The degree of fulfillment of a rule is calculated by the following semantics as suggested by Zadeh in [11]. Let μ_A and μ_B denote two membership functions, then

$$(\mu_A \sqcap \mu_B)(a) \equiv \min\{\mu_A(a), \mu_B(a)\},$$

$$(\mu_A \sqcup \mu_B)(a) \equiv \max\{\mu_A(a), \mu_B(a)\},$$

$$\neg\mu_A(a) \equiv 1 - \mu_A(a).$$

Calculating $\mu(a)$, the membership of a to μ . Considering that our membership functions are just a set of points in $\mathbf{R^2}$, we can calculate the membership of x , with $x_1 \leq x \leq x_2$ by calculating y value for x on the line passing through (x_1, y_1) and (x_2, y_2) by the following formula

$$y = \frac{y_1 - y_2}{x_1 - x_2}(x - x_1) + y_1 .$$

This formula can be represented as DL rules as

$$CALCY(x_1, y_1, x_2, y_2, x, y) \equiv SUB(y_1, y_2, dy) \sqcap SUB(x_1, x_2, dx) \sqcap DIV(dx, dy, s) \sqcap SUB(x, x_1, d) \sqcap MULT(s, d, p) \sqcap$$

.

Now, the only thing that remains is how to find the correct pair of points and how to use the above formula in the calculation of the membership function. Let f be a membership function. Assuming that a membership function f consists of n points $(x_1, y_1), \ldots, (x_n, y_n)$ in $\mathbf{R^2}$, we insert $n - 1$ rules in the rule base where the i -th rule looks like as follows

$$MU(f, a, m) \equiv LEQ(f.x_i, a) \sqcap EQ(a, f.x_{i+1}) \sqcap CALCY(f.x_i, f.y_i, f.x_{i+1}, f.y_{i+1}, a, m).$$

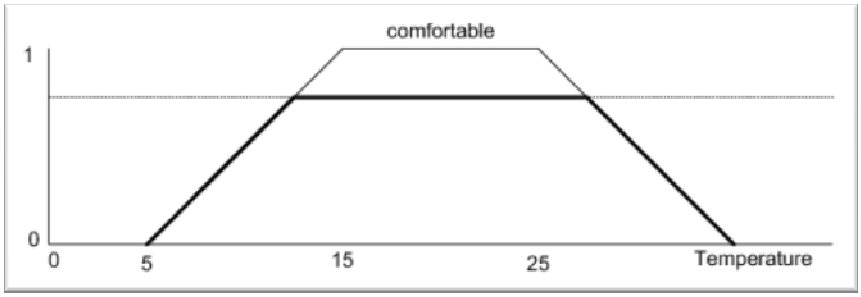

Figure 3. Fuzzy inferencing.

Calculating the Degree of Fulfillment of a Rule. The previous step yields a number $m^a_{R_i}$ between 0 and 1 that represents the degree of fulfillment of the rule R_i by the individual a. In this step, we construct a new membership function $\mu^a_{O_i}$ by cutting the part of the membership function μ_{O_i} which is higher than $m^a_{R_i}$ (cf. Figure 3).

Algorithm 1 Calculating $\mu^a_{O_i}$

for (x, y) a point in μ_{O_i} **do**
  add $(x, \min(y, m^a_{R_i}))$ to $\mu^a_{O_i}$
end for
add all points (x, y) to $\mu^a_{O_i}$ if (x, y) is the intersection of a line passing through two adjacent points in μ_{O_i} and the line passing through the points $(MINX, m^a_{R_i}))$ and $(MAXX, m^a_{R_i}))$

Algorithmically, $\mu^a_{O_i}$ can be constructed by performing the following steps. Let $MINX$ and $MAXX$ denote the smallest and largest value of x in μ_{O_i}. Refer to Algorithm 1.

Similarly, these steps are performed for other rules, such that we have in end as many new membership functions as there are categories in the objective. We denote these membership functions by $\mu^a_{O_1} \dots \mu^a_{O_k}$.

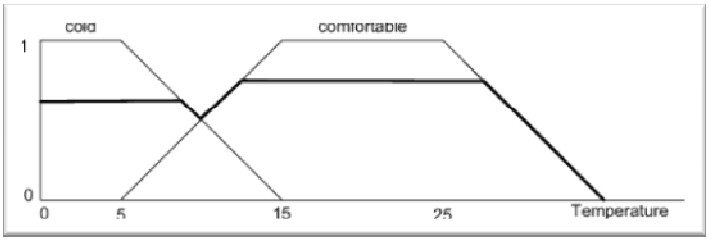

Figure 4. Fuzzy Aggregation.

Aggregation. The aggregation step consists of taking the maximum of all the $\mu^a_{O_i}$ that we obtained in the previous step. Let μ^a_O denote such a membership function. Refer to Figure 4 and Algorithm 2.

Algorithm 2 Calculating μ_O^a

for $i = 1$ TO k **do**
  for (x, y) in $\mu_{O_i}^a$ **do**
    calculate $y_1 = \mu_{O_j}^a(x), \ldots y_k = \mu_{O_k}^a(x)$
    add $(x, \max\{y_1, \ldots, y_k\})$ to μ_O^a
  end for
  add all points (x, y) to μ_O^a if (x, y) is the intersection of a line passing through two adjacent points in μ_{O_i} and a line passing through two adjacent points in μ_{O_j} with $i \neq j$.
end for

Defuzzification. There are a few defuzzification strategies like the following.

- The *max criterion method* finds the point at which the membership function is a maximum.
- The *mean of maximum* takes the mean of those points where the membership function is at a maximum.
- The *centre of area* method which finds the centre of gravity of the solution fuzzy sets.

[13] states, that there is no systematic procedure for choosing a defuzzification strategy. We use the centre of area since it is the most common method. The x coordinate of the center of area of a membership function μ with n points can be calculated by the following formula (refer to Figure 5). The overall acceptance of an offer is then equal to the value of the x -coordinate of the center of gravity of the with the formula

$$\frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} y_i}.$$

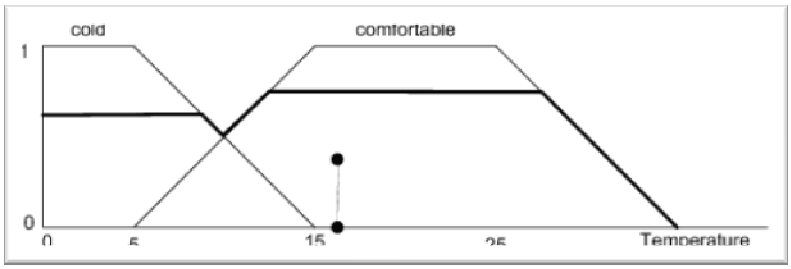

Figure 5. Defuzzification.

Let $SUMXY(f, s)$ denote that the sum of products of the x and y of all the points of f is s and $SUMY(f, t)$ denote the sum of y of all the points in f. We can calculate the value α as the value of the x coordinate of the center of area of μ by

$$CALCX(\mu, \alpha) \equiv SUMXY(\mu, s) \sqcap SUMY(\mu, t) \sqcap DIV(s, t, \alpha).$$

Now, we calculate $RANK(O, a, r)$ by

$$RANK(O, a, r) \equiv CALCX(\mu_O^a, r).$$

After defuzzification we get a $\prec P, I \succ$ structure, which is one of the most traditional preference models [12]. The model consists of two relations. On the one hand, the

asymmetric relation denoted by P representing the preference relation that orders any two individuals a_1 and a_2 such that the statement "a_1 is preferred over a_2" is true. And on the other hand, the reflexive and symmetric relation denoted by I representing the indifference relation that orders any two individuals a_1 and a_2 such that the statement "a_1 and a_2 are indifferent" is true.

The derived preference structure is a weak order structure, because it meets the following conditions $\forall a_1, a_2 \in \Delta$

$$P(a_1, a_2) \text{ iff } f(a_1) > f(a_2),$$

$$I(a_1, a_2) \text{ iff } f(a_1) = f(a_2),$$

where $f(a_1)$ and $f(a_2)$ represent r_1 and r_2 such that $RANK(O, a_1, r_1)$ and $RANK(O, a_2, r_2)$ hold, respectively.

4.3.2 Example

The basic idea of this approach is now explained by an example. Assuming that the user "Albert" searches for Web services that book flights. A knowledge base provided by the ranking component features a metadata repository that already contains ontologies, which include the concept of $FlightBookingServices$. This concept has the non-functional properties $price$ and $space$ per passenger. Using syntax of description logics [15], that is,

$$FlightBookingServices \sqsubseteq \top \sqcap \exists price \sqcap \exists space .$$

Step 1 – Albert specifies a fuzzy goal. He is interested in a flight that is either

(i) cheap and comfortable, or
(ii) affordable and comfortable, or
(iii) cheap and normally spaced.

Albert defines the membership functions as shown in Figure 6. Let us assume Albert uses four categories bad, $fair$, $good$, and $super$ for the objective function. Albert's goal can be formulated as:

$$bad \equiv (affordable \sqcap tight) \sqcup (expensive \sqcap tight) \sqcup (expensive \sqcap normal)$$

$$fair \equiv (cheap \sqcap tight) \sqcup (affordable \sqcap normal) \sqcup (expensive \sqcap comfortable)$$

$$good \equiv (cheap \sqcap normal) \sqcup (affordable \sqcap comfortable)$$

$$super \equiv (cheap \sqcap comfortable)$$

Step 2 – The goal is relaxed to a crisp request description by replacing the fuzzy membership function by an interval. The according crisp request looks like follows:

$$\geq_{0.5} (space) \sqcap \leq_{MAXVAL} (space) \sqcap \geq_{0} (price) \sqcap \leq_{300} (price)$$

$$\geq_{0.6} (space) \sqcap \leq_{MAXVAL} (space) \sqcap \geq_{180} (price) \sqcap \leq_{550} (price)$$

$$\geq_{0.25} (space) \sqcap \leq_{0.6} (space) \sqcap \geq_{0} (price) \sqcap \leq_{300} (price)$$

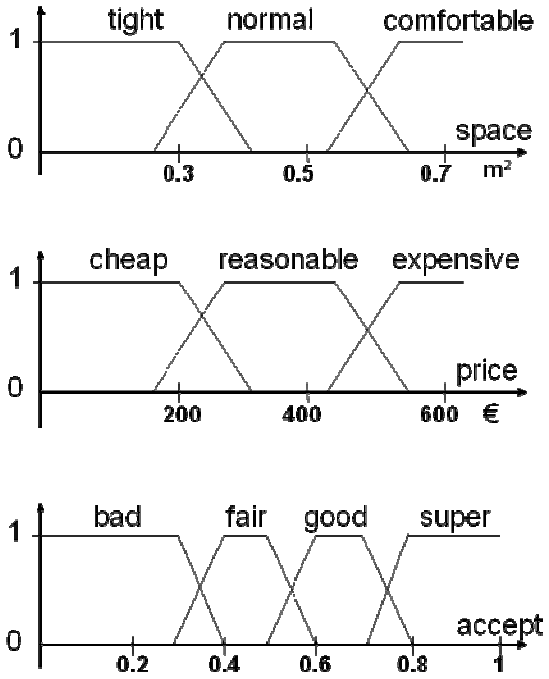

Figure 6. Membership function for seat space and price and objectives functions.

Step 3 – The crisp requests is transformed into a query that can be processed by the reasoner. Therefore it maps the concepts of space and price to the concepts used by the Web service vendor or annotator.

Step 4 – Let assume that there are three flight booking Web services matching the fuzzy query.

Flight1: $space = 0.55m^2$, $price = 400€$

Flight2: $space = 0.3m^2$, $price = 300€$

Flight3: $space = 0.7m^2$, $price = 550€$

Step 5 – In this step, the computation as described conceptually above is executed in order to compute the ranking of available flight booking Web services.

The original fuzzy request is sent to the DL reasoner, which already knows the rules that map fuzzy-DL into crisp DL. The reasoner fetches Web service offers from the metadata repository and calculates for each offer and each rule the degree to which the offer fulfils the rule. The obtained degrees for an offer are then used along with the categories of the objective function to calculate the aggregated area that the offer covers. The overall acceptance of an offer is then equal to the value of the x-coordinate of the center of gravity of the area (cf. Figure 7). For our three offers the degree of acceptance is calculated as follows:

$$\frac{0.4 + 0.5 + 0.059 + 0.079}{1 + 1 + 0.1 + 0.1} = 0.4718$$

$$\frac{0.3 + 0.111 + 0.171 + 0.059 + 0.079}{1 + 1 + 0.3 + 0.3 + 0.1 + 0.1} = 0.257$$

$$\frac{0.304 + 0.416 + 0.216 + 0.304}{0.8 + 0.0 + 0.4 + 0.4} = 0.5167$$

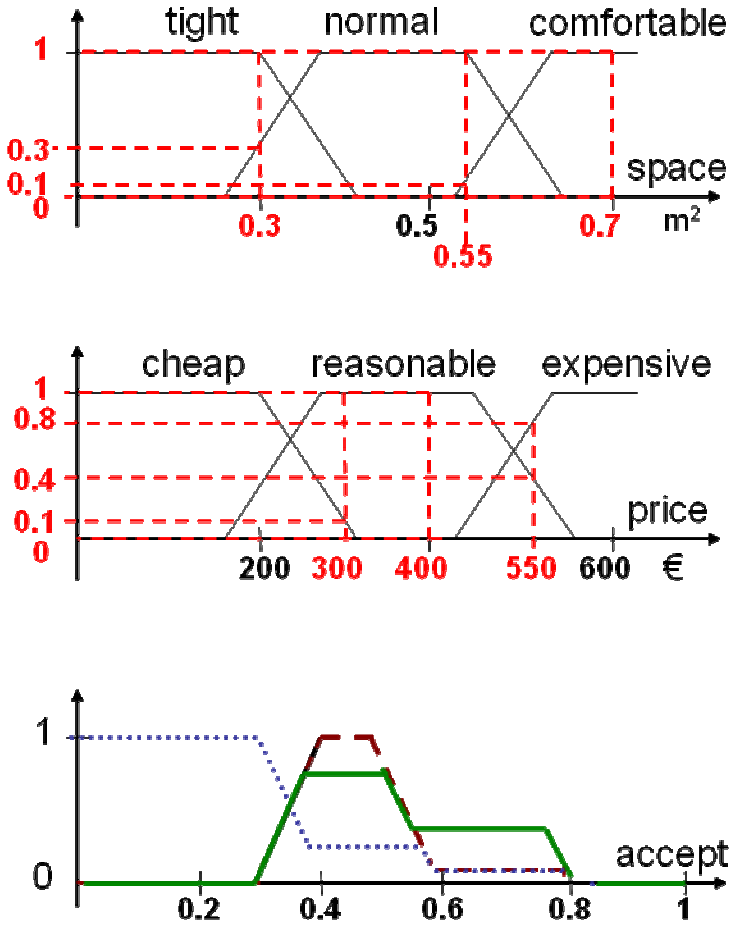

Figure 7. Memberships of values of the offers to fuzzy sets and aggregated areas.

Step 6 – The reasoner returns the list of Web services and their respective ranking. The sorted list can be presented to the user or used for further automated processing.

4.3.3 Novelty

The novelties of the fuzzy logic based service ranking approach was already mentioned in the introductory motivation of this approach as the novelties coincide with the approach's benefits. We briefly summarize the novel aspects introduced by the fuzzy logic based ranking of services.

1. Expressivity: This approach is capable to model complex preferences and thus to consider the relationship between different non-functional properties. For instance, the prior approaches did not allow users to formulate that a Web service with a high price and with a comparably large response time is not acceptable.
2. Efficiency: Using fuzzy logics introduces the well proven benefits low computational costs to compute a ranking. Considering the vast number of targeted Web service descriptions and the potential size of user preferences, the complexity of a Web service ranking algorithm is crucial for usability.

3. Indecisiveness: Users are not forced to formulate crisp preferences; they do not even need to be aware about specific values of a property. The fuzzy logic based approach allows users to formulate imprecise requirements.

4.3.4 Future Work

Obviously, the fuzzy-logics based service ranking approach needs to be implemented and validated in future steps. Notwithstanding, in this section, we want to discuss an important ingredient of this approach from the usability perspective that was not considered in the prior paragraphs, which presented the conceptual work.

Regarding the example from above, the question how a non-experienced user might model fuzzy sets, the membership functions as depicted in Figure 6, is not considered yet. Of course, expecting users to specify functions is unrealistic and we therefore have to provide a simple, perhaps visual, tool to increase usability and simplistically allow non-experts to model fuzzy sets.

We envision two approaches. First, the Studio provides a set of predefined patterns of fuzzy sets that can be selected and customized by users. Second, we provide a simple widget that allows the user to visually specify membership functions by moving characteristic points of the function plots with the mouse device (drag and drop). The trapezoid shape of a function plot, which is determined by only four characteristic points, is sufficient for modeling almost all kind of user preferences.

A user is able to model fuzzy sets by adding, moving, and stretching trapezoids, assigning labels to them (e.g., “cheap” when modeling the price). The predefined patterns of fuzzy sets can be modified in a similar fashion.

5. Installation

This section provides information on where to find details on how to download, install and run the current version of the ranking prototype.

As we have described in Section 4.1 the ontology-based feature aggregation for multi-valued ranking approach is based only on RDF meta-data triples (both the ranking input data and the output data). The RDF data will be delivered as dump on a weekly basis and can be accessed at: http://crawl.seekda.com/ranking/ .

The multi-criteria ranking based non-functional properties approach is available for download at: http://soa4all.sti2.at/index.php/WP5:_Service_Location#First_Ranking_Prototype

Details on how to install and run the multi-criteria ranking based non-functional properties approach are also available at:

http://soa4all.sti2.at/index.php/WP5:_Service_Location#First_Ranking_Prototype

As mentioned in Section 4.3.4, the fuzzy logic based ranking approach will be implemented as part of the second prototype. The corresponding deliverable will contain details on how to install and run each of the three updated approaches as well as the integrated implementation.

6.Conclusions

The first ranking prototype introduced in this deliverable represents the first step for the development of the SOA4All ranking component. Our approach comprises different aspects to fulfill the actual demand of the users of the SOA4All service delivery platform. Most notably, the first service ranking prototype takes service descriptions, service monitoring data, and user preferences into account. The overall approach is a multi-value approach which takes into account multiple criteria to compute services ranks. It integrates three distinct ranking approaches that complement each other by considering different information. The first ranking approach focuses on computing services ranks by evaluating service descriptions available on the Web and monitoring data, while the other two approaches focus more on semantic descriptions of services, more precisely on non-functional properties aspects. The second approach uses Logic Programming rules to model non-functional properties of services and requests, while the third expresses same aspects using Fuzzy Logics and Description Logics.

To sum up, we provided an integrated ranking solution for services based on three approaches. We have developed technical solutions for each of these approaches. Furthermore, two of the ranking approaches, namely ontology-based feature aggregation for multi-valued ranking and multi-criteria ranking based on non-functional properties are have been implemented and form the basis of the first service ranking prototype. For the second ranking prototype we plan to further develop the three approaches, to provide implementations for all of them and to develop and integrate the three approaches. The integrated implementation of the SOA4All ranking component will provide a novel approach that clearly distinguishes from state of the art ranking approaches since we holistically consider several aspects on services.

7.References

[1] http://www.wsmo.org/ontologies/nfp/
[2] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In I. Horrocks and J. Handler, editors, *1st Int. SemanticWeb Conference (ISWC)*, pages 333–347. Springer Verlag, 2002.
[3] K. Verma, K. Sivashanmugam, A. Sheth, and A. Patil. Meteor-s wsdi: A scalable p2p infrastructure of registries for semantic publication and discovery of web services. *Journal of Information Technology and Management*, 2004.
[4] J. Cardoso and A. P. Sheth. Introduction to semantic web services and web process composition. In *SWSWPC*, pages 1–13, 2004.
[5] T. Vitvar, J. Kopecky, and D. Fensel. WSMO-Lite: Lightweight Semantic Descriptions for Services on the Web, March 2008. http://www.wsmo.org/TR/d11/v0.2/.
[6] Nathalie Steinmetz, Holger Lausen, Manuel Brunner, Iván Martinez and Alex Simov. D5.1.3 Second Crawler Prototype. SOA4All deliverable, August 2009.
[7] Zadeh, Lotfi A.: *Outline of a new approach to the analysis of complex systems and decision processes*. IEEE Trans. on Systems, Man and Cybernetics, 3(1):28{44, 1973. Reprinted in [8].
[8] Yager, R. R., S. Ovchinnikov, R. M. Tong, and H. T. Nguyen (editors): *Fuzzy Sets and Applications | Selected Papers by L. A. Zadeh*. John Wiley & Sons, 1987.
[9] U. Straccia. Reasoning within fuzzy description logics. J. Artif. Intell. Res., 14:137–166, 2001.
[10] K.-H. Temme and H. Thiele. On the correctness of the principles of FATI and FITA and their equivalence. In 6th Int. Fuzzy Systems Association World Congress, pages 475–478, 1995.
[11] L. A. Zadeh. Fuzzy sets. Fuzzy sets, fuzzy logic, and fuzzy systems: selected papers by Lotfi A. Zadeh, pages 19–34, 1996.
[12] M. Oztürk, A. Tsoukias, and P. Vincke. Preference modelling. In J. Figueira, S. Greco, and M. Ehrgott, editors, Multiple Criteria Decision Analysis: State of the Art Surveys, pages 27–72. Springer Verlag, 2005.
[13] C. Lee. Fuzzy logic in control systems: Fuzzy logic controller, part ii. IEEE Transactions on Systems, Man and Cybernetics, pages 419–435, 1990.
[14] Gekas, J. Web Service Ranking in Service Networks. In *Sure, Y., and Domingue, J. (Eds.): The Semantic Web: Research and Applications, 3rd European Semantic Web Conference - ESWC'06*, Springer, LNCS, pages 501–510, 2006.
[15] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider, The Description Logic Handbook: Theory, Implementation, and Applications, Cambridge University Press, 2003.

Appendix 1

Based on the technical work presented in this deliverable a set of scientific publications have been published. They are listed below:

1. Toma, I., Roman, D., Fensel, D., Sapkota, B., and Gomez, J.M. A multicriteria service ranking approach based on Non-Functional Properties rules evaluation. In *ICSOC '07: Proceedings of the 5th international conference on Service-Oriented Computing*, pages 435–441. Springer-Verlag, 2007.

2. Toma, I., Roman, D., and Fensel, D. On describing and ranking services based on Non-Functional Properties. *Next Generation Web Services Practices, International Conference on*, 0:61–66, 2007.

3. Heymans, S., and Toma, T. Ranking services using Fuzzy hex programs. In *RR '08: Proceedings of the 2nd International Conference on Web Reasoning and Rule Systems*, pages 181–196, 2008. Springer-Verlag.