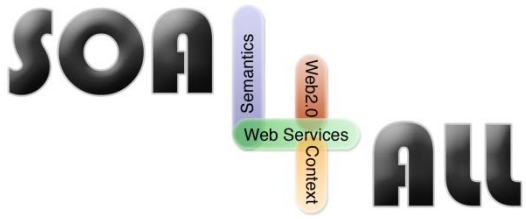

Project Number: **215219**

Project Acronym: **SOA4All**

Project Title: **Service Oriented Architectures for All**

Instrument: **Integrated Project**

Thematic Priority: **Information and Communication Technologies**

D5.4.2 Second Service Ranking Prototype

Activity N:	A2 - Core Research and Development	
Work Package:	WP5 – Service Location	
Due Date:		31/08/2010
Submission Date:		31/08/2010
Start Date of Project:		01/03/2008
Duration of Project:		36 Months
Organisation Responsible of Deliverable:		KIT
Revision:		1.0
Author(s):	Sudhir Agarwal (KIT), Martin Junghans (KIT), Barry Norton (KIT)	
Reviewers:	Usman Wajid (UNIMAN), Jacek Kopecky (OU)	

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	**x**
PP	Restricted to other programme participants (including the Commission)	
RE	Restricted to a group specified by the consortium (including the Commission)	
CO	Confidential, only for members of the consortium (including the Commission)	

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
1.0	04.08.2010	Initial version	Martin Junghans
1.1	06.08.2010	Internal review, updated example and screenshot	Martin Junghans
1.2	09.08.2010	Section preliminaries	Sudhir Agarwal
1.3	18.08.2010	Filled in content	Barry Norton
1.4	26.08.2010	Accounted for reviews	Barry Norton

Table of Contents

List of Figures

List of Tables

Glossary of Acronyms

Acronym	Definition
D	Deliverable
EC	European Commission
MSM	Minimal Service Model
NFP	Non-Functional Property/Parameter
POSM	Procedure-Oriented Service Model
WP	Work Package

Executive summary

In a world of 'billions of services', as envisioned by SOA4All, it is not sufficient merely to aid users in finding services as an ad hoc task. It has been long-acknowledged that ranking will be required to help users find the best offerings in a service economy, but in the work described in this deliverable we go further. Using the latest SOA4All developments, such as service templates and a scalable service repository, we provide an infrastructure for a long-term, evolving relationship between service consumers and providers according to particular consumer needs, documented in a service template. We call this approach the Discovery Cloud, or DisCloud. The template becomes a permanent resource, not just a transient one for an ad hoc request. Matching services are ranked at each request, reflecting the real-time information available from the crawler and the developing SOA4All analysis platform. This development has produced a reusable object model for service descriptions and templates, called WSMO-Lite for Java, or WSL4J We also describe development on the plans for a fuzzy logic-based approach to specifying preferences that, by integration with this platform, will aid users in guiding the rank that is most useful for them.

1. Introduction

In a world of 'billions of services', as envisioned by SOA4All, it is not sufficient merely to aid users in *finding* services as an ad hoc task. It has been long-acknowledged that *ranking* will be required to help users find the best offerings in a service economy. SOA4All has proposed three approaches to ranking. An 'objective' ranking approach based on metrics collected by the crawler, also developed in WP5. A 'subjective' ranking approach where service descriptions can express rule-based metrics to give values to non-functional properties (NFPs) and basic consumer preferences can be expressed over these. A sophisticated fuzzy logic and rule-based ranking approach where metrics and other NFPs are first fuzzified, to aid understanding by consumers, and then preferences can be expressed by flexible rules. Integration between these approaches allows the objective metrics to be used also in subjective user preferences, and has led to a new infrastructure, the Discovery Cloud (DisCloud), with which we hope to make scalable discovery and ranking scalable to allow for the foreseen rapid expansion in services.

1.1 Purpose and Scope

In this deliverable we describe the latest developments on the ranking approaches, especially the fuzzy logic-based integration on which development is now underway, and a novel integration approach that reuses the latest developments in general SOA4All technology, including the Service Template model and the iServe service repository.

The DisCloud service template repository is introduced to manage the long-term, evolving relationship between service consumers and providers according to particular consumer needs, documented in a service template. The template becomes a permanent resource, not just a transient one for an ad hoc request. Functionally-matching services are pre-computed, and will be updated on an on-going basis as new descriptions are added to iServe, for efficiency. The matching services are then ranked at each request, reflecting the real-time information available from the crawler and the developing SOA4All analysis platform

1.2 Structure of the document

This document is structured as follows. Section 2 recalls the three approaches to ranking proposed by WP5. Section 3 describes the integration achieved, and explains its novelty, during the last period. Section 4 describes the work to date on implementing fuzzy logic and rule-based ranking. Section 5 gives an overview of the current uses made of ranking in the project and Section 6 describes on-going work and the general outlook.

2. Preliminaries

In this section, we give short overviews of the three 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. The approaches are described in more detail in the D5.4.1 deliverable.

2.1 'Objective' Ontology-based Feature Aggregation for Multi-valued Ranking

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

2.1.1 Related Documents Rank

This rank is based on the crawl meta-data that is delivered by the crawler and is calculated based on the following information: (1) How many related documents does a service have? (2) How is the document related to a specific service?

In a first step we calculate the number of related documents per service. This value is stored using the `hasNumberOfRelatedDocuments` relation of the seekda Ranking Ontology. Now the related documents rank is calculated. The final rank is stored for each service using the `hasRelatedDocsRank` relation of the seekda Ranking Ontology.

2.1.2 WSDL Mertrics 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. 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.

2.1.3 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. The availability score is a number between 0 and 1 that is set depending on the endpoint check result. In between 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. The rank is stored for each service using the `hasMonitoringRank` relation of the seekda Ranking Ontology:

2.1.4 Web API 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. The rank is based on the following information: (1) What is the Web API Confidence score of a document? (2) Which crawler classifier has classified the document as Web API?

To calculate the rank we need to extract both the score and the component that has assigned the score. Based on first evaluations of the classifiers, we deem the score of the SVM classifier more important than the one of the Web API Evaluator.

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

2.1.6 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 use the seekda Ranking Ontology to store and distribute the service ranks. In the meanwhile we have a Java component that calculates the ranks.

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.

2.2 'Subjective' Multi-criteria Ranking based on Non-Functional Properties

Non-functional properties specified in the user request and service descriptions are formalized by means of logical rules using terms from NFP ontologies. 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.

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 and evaluates them using the WP3 reasoning engine which supports WSML rules. A quadruple structure is built that 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 of non-functional properties weighted by importance values. The results are collected in a set of tuples, each tuple containing the service id and the computed score. Finally, the scores are ordered as specified by the user and the final list of services returned.

2.2.1 Implementation

The multi-criteria ranking approach takes as input a set of services annotated using the WSMO-Lite ontology and a user request using the new Service Template model. The result is presented in a form of ordered list of services. Furthermore, for each service in the list additional information 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 = "templateURI") String template) 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.

2.3 'Subjective' Fuzzy Logic Based Ranking Approach

This process of computing a fuzzy logic-based rank of a service consists of four main steps: (1) Fuzzification (2) Inferencing (3) Aggregation and (4) Defuzzification. The user preferences are specified as fuzzy IF-THEN rules.

During fuzzification, the crisp values of the non-functional properties of the service are fuzzified, i.e. their fuzzy memberships in the fuzzy sets associated with the properties are computed. During inferencing, each fuzzy IF-THEN rule is processed and a degree of fulfillment of the rule is computed. The fuzzy set in the conclusion of a rule are chopped at the level that equals to degree of fulfillment of the premise of the rule. During aggregation, the chopped fuzzy sets in the conclusion of the rules are aggregated. The aggregated fuzzy set denotes the rank of the service as a fuzzy set, which is then defuzzified to a crisp value between 0 and 1 to obtain the actual rank of the service.

The novelties of the fuzzy logic based service ranking approach can be summarized as follows.

1. Expressivity: This approach is capable to model complex preferences and thus to consider relationships 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. Practicability: 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 requirements rather vaguely.

3. Integrated Ranking Approach

In order to bring together together the three approaches to ranking, and to take best advantage of SOA4All developments such as Service Templates and the RESTful repository for service descriptions, we have introduced an integrated ranking approach based on the following principles:

- The various 'objective' ranking metrics, made available via the crawler as described in Section 2.1.1, should be dynamically encoded in NFPs attached to semantic service descriptions in such a way that they are available (for users to include in their preferences) in the subjective ranking described in Section 2.1.2.
- The rules applied to derive metrics as NFPs for service descriptions in the approach described in Section 2.1.2 should be made available for 'fuzzification' in the fuzzy ranking approach described in Section 2.1.3.
- A repository-oriented approach to Service Templates has been investigated, where:
    - templates are stored as permanent resources, which may be private (brokered only for the uploading client) or shared;
    - the functional match with services, i.e. the discovery approach described in D5.3.2 should be carried out on upload and kept up-to-date with new services, with discovery against each brokered service triggered by notifications of new uploads from iServe;
    - ranking should be carried out at request, taking advantage of the all three ranking approaches – i.e. based on the subjective preferences of the requester, but allowing preferences to be specified over the up-to-date metrics used for objective ranking.

The intended architecture and interaction is as shown in Figure 1, as described in the following section.

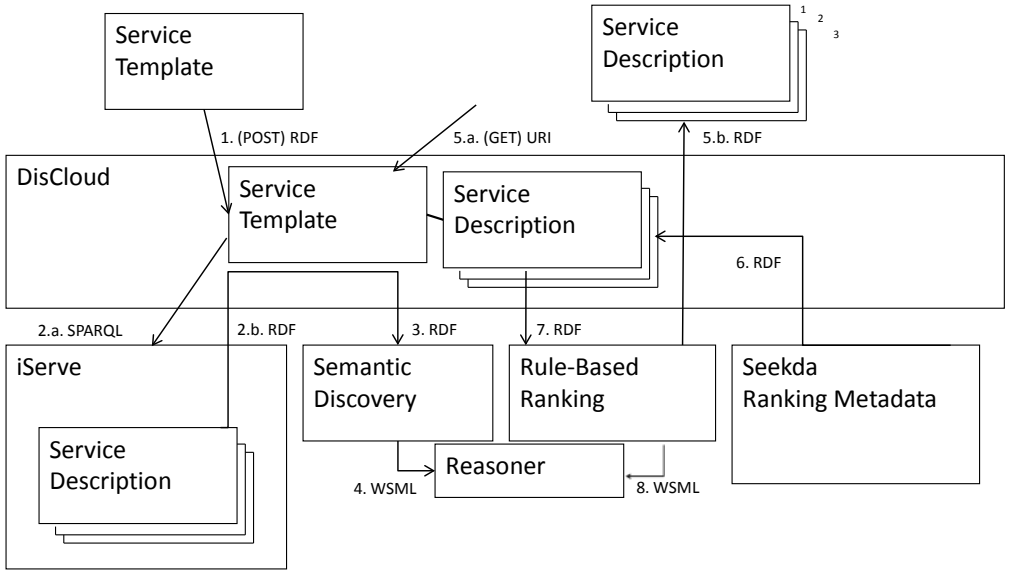

Figure 1: Discovery Cloud Architecture and Interactions.

4. Implementation

The implementation is intended to support the interaction shown in Figure 1, and so create an effective service template repository called DiscoveryCloud, or DisCloud, as follows:

1. The client formalises their functional requirements using the SOA4All Service Template model, and their preferences using the chosen preference model, encode these together in RDF and HTTP POST them to the new repository.

2.a. The functional classification(s) requested in the service template are used to pre-select potential service matches from iServe, using a SPARQL query;

2.b. The appropriate service definitions are retrieved, in RDF;

3. Semantic discovery is extended, as described in D5.3.2, to have an interface based on the new Service Template model;

4. Semantic discovery encodes the appropriate WSML queries to refine the set of matching services that are then stored.

5.a. The repository waits (or rather, when available, uses iServe upload notifications to keep an up-to-date list of functional matches) until the user requests information on the status of the template (with an HTTP GET);

6. At this point the latest objective ranking metrics are retrieved from the seekda crawler via the metadata repository API and are used to update the RDF description of the services, adding further NFPs;

7. The updated (MSM) service descriptions are passed to the WSML or Fuzzy rule-based ranking engine

5.b. The ranked services are returned to the user.

4.1 WSL4J

In order to support validation of service templates, and the addition of preference models to MSM (which has been renamed Procedure-Oriented Service Model, or POSM[1], as a result of the proposed RPC Service Model of D3.4.6), a parsing (from RDF serialisations) Java object model was developed.

Alongside the developments described for DisCloud this was then extended to:

- directly load service descriptions from iServe;
- directly load service templates from DisCloud;
- automatically load and encode ranking metrics from the seekda metadata repository;
- serialise models back to RDF (n3/Turtle and RDF/XML syntax).

As well as being used in the development of DisCloud, this object model was used to update the interfaces of all semantic ranking and discovery components.

Thereafter, having demonstrated its general utility, it was renamed WSL4J (WSMO-Lite for Java) and contributed to WP3 for maintenance.

The Javadocs for WSL4J have been included in Annex A.

[1] http://www.wsmo.org/ns/posm/

4.2 DisCloud

The Discovery Cloud, or DisCloud[2], offers a RESTful repository API for the maintenance of service templates, compatible with the API of iServe.

Service templates can be POSTed, in n3/Turtle or RDF/XML syntax, and are validated, returning the usual HTTP Status codes:

- **201 Created** – the template has been stored and its new representation can be retrieved from the URI returned in the **Location** HTTP header field;
- **400 Bad Request** – the model passed did not validate;
- **401 Unauthorised** – the client did not pass relevant authorisation header (DisCloud is being integrated with the authorisation system of the SOA4All Studio).

Service templates are stored internally in a triple store. Technically, the REST interface is realised using the Jersey[3] implementation of JAX-RS[4], while the repository is currently realised using the standard Sesame[5] implementation of the SAIL API[6], accessed via the RDF2Go API[7]. This means that DisCloud can easily be upgraded, once stable, to use the SOA4All semantic space via the recent implementation of this API.

Stored service templates can later be DELETEd, with the normal HTTP responses (depending on authorisation):

- **200 OK** – template will no longer be available;
- **401 Unauthorised** – the client is not the owner of the resource.

Until deletion, each template is directly GETtable, resulting in a Linked Data-style redirect:

- **303 See Other** – location depends on Accept HTTP header field:
  - http://km.aifb.kit.edu/services/DisCloud/templates/{id}/data field matches media type 'application/rdf+xml', 'text/n3' or 'text'ttl', etc.,
  - http://km.aifb.kit.edu/services/DisCloud/templates/{id}/page field matches media type 'text/html' or similar;
- **401 Unauthorised** – the client is not the owner of the resource.

Upon storage, discovery is triggered and the URIs for matched services are linked to the stored service template via further triples.

On retrieval of matching services (via GET on http://km.aifb.kit.edu/services/DisCloud/templates/{id}/matches), DisCloud parses back the relevant service template from the underlying repository using WSL4J object model (eu.soa4all.wsl4j.ServiceTemplate.ServiceTemplate.createFromDisCloud), loads the attached services from iServe (...wsl4j.rpc.MSMService.createFromIServe), retrieves the latest metrics (...wsl4j.WSML.SeekdaRankingNFP.updateNFPs) and dispatches to the relevant ranking engine based on the classification of the associated preference model (e.g. ...wsl4j.WSML.RuleBasedRanking).

[2] Available at http://km.aifb.kit.edu/services/DisCloud

[3] https://jersey.dev.java.net/

[4] https://jsr311.dev.java.net/

[5] http://www.openrdf.org/

[6] http://www.openrdf.org/doc/sesame2/system/ch05.html

[7] http://semanticweb.org/wiki/RDF2Go

4.3 Fuzzy Based Service Ranking

This service ranking approach proposes a fuzzy logic based ranking mechanism that considers an extended model of preferences including vagueness. In D5.4.1 we introduced a fuzzy logic approach for modeling user preferences. We use fuzzy IF-THEN rules to express user preferences and relationships between values of non-functional properties. Then, fuzzy logic based ranking approach features the abilities:

- to express vagueness while expressing preferences using linguistic terms instead of crisp values;
- to assign crisp property values to different categories by specifying overlapping fuzzy sets membership functional that model these categories;
- to create complex preferences constructed by the combination of simple terms.

We implemented the ranking component as a Web service[8]. It provides two public methods. One method 'addPropertyClasification' that lets users add categorizations of properties. This method requires a property name and a set of membership functions over those categories. As depicted in Figure 3, each category is represented by a unique category name and four characteristic points of the membership function.

The second method '`rankServices`' computes the ranking according to preferences specified by the user. The signature of the method is defined as follows. The method receives a set of service IDs used to identify the semantic service description, and the user preference expressed by a set of fuzzy IF-THEN rules. As a result, the method returns an ordered list of service IDs with ascending degree of fulfillment of the preferences. In the following, we will provide insights on the modeling preferences and property categories with fuzzy sets within the user interface.

4.3.1 Modeling Preferences

Preferences of the user are represented by a set of fuzzy IF-THEN rules; one rule for each category of the acceptance property at most. The conclusion of the rule, i.e., the THEN part, refers to exactly one category of the acceptance property. The grammar to express IF-THEN rules is given in Table 1 in Backus-Naur Form. `PropertyName` and `PropertyCategory` refer to the name of a property like 'ex:responseTime' in the namespace abbreviated by 'ex' and a category like 'low' that should be defined for the rsp. property. In this example, a term is 'responseTime=low'.

Table 1 : Grammar of Fuzzy IF-THEN Rules.

```
<Rule>          ::=  'IF' <Body> 'THEN' <Head>
<Body>     ::=  <Expression>
<Expression>    ::=  <Term> | '(' <Expression> ')' |
<Conjunction>
                | <Disjunction> | <Negation>
<Conjunction>   ::=  <Expression> 'AND' <Expression>
```

[8] The service ranking Web service is available at http://km.aifb.kit.edu/services/soa4all-discovery/axis2/services/FuzzyServiceRanking?wsdl

```
<Disjunction>    ::=   <Expression> 'OR' <Expression>
<Negation> ::=    'NOT' <Expression>
<Term>     ::=   PropertyName '=' PropertyCategory
<Head>     ::=   'Acceptance=' PropertyCategory
```

The simple syntax of the rules allows to express complex preferences using conjunctions, disjunctions, negations, and nesting. In order to process given user preferences, a parser translates the preferences specified in a user interface into the Java object model as shown in Figure 3 representing preferences internally.

4.3.2 Modeling Property Value Categorization

The value range of each property that occurs in the preferences of the user must be categorized in order to allow users to refer to fuzzy sets (identified by the name of the category) instead of crisp values. For example, the property responseTime can be categorized into the three categories low, medium, and high. Each of these categories is modeled as a fuzzy set by a membership function. The specification of four points allows for the creation of trapezoids in the two-dimensional space that represent the membership of crisp values in the property range to the respective category. Figure 2 depicts the membership functions for the above-mentioned example with property responseTime. Left and right shoulder functions used for the categories low and high, respectively, denote a membership of 1 for infinitely rsp. small and large property values on the horizontal axis. Figure 3 depicts the relation of properties, categories, and the characteristic points of membership functions.

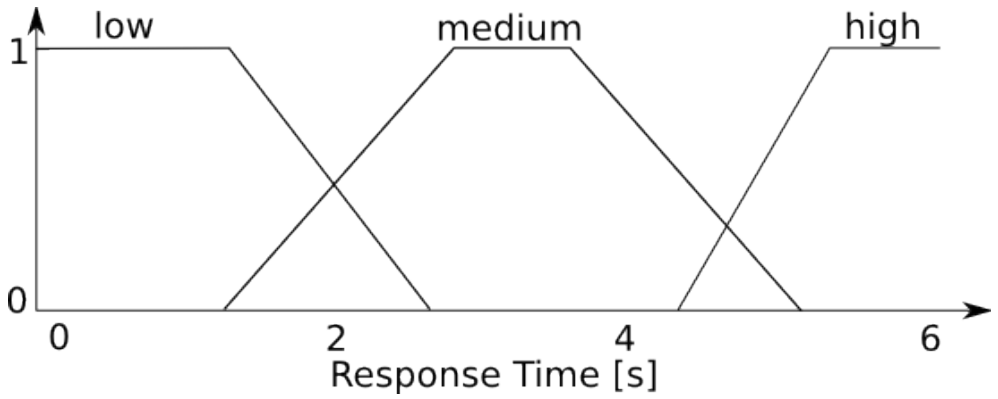

Figure 2: Categories of property response time modeled by membership functions.

4.3.3 Utilization of Semantic Service Descriptions

Non-functional properties including their actual values are derived from the service descriptions. The service IDs are used to retrieve the corresponding semantic service description from the SOA4All service repository using its RESTful interface[9]. The WSMO-Lite [3] based service description may contain non-functional properties (using the concept `wl:NonFunctionalParameter` defined in WSMO-Lite). These non-functional properties are expressed by a concept of a domain ontology and is associated

[9] http://iserve.kmi.open.ac.uk/resource/services/<service ID>

with a concrete value. Within the computation of the ranking, each service is reduced to a set of key value pairs (see Figure 3).

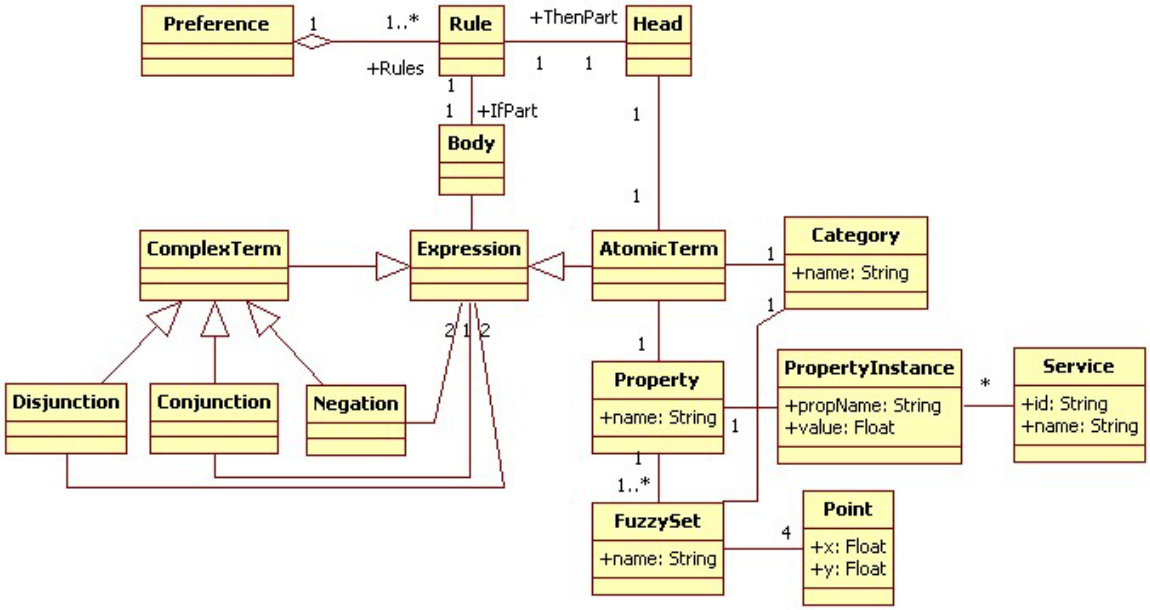

Figure 3: UML diagram of the data objects model.

4.3.4 User Interface

The Web based user interface for modeling preference and property categorization is developed with the Google Web toolkit. To model preferences, fuzzy IF-THEN rules are described by the user. Therefore, the Web based user interface provides a form that allows to enter A rule bodies within A text fields. The number A of text fields is derived from the number A of categories of the property Acceptance. For instance, let the property Acceptance be categorized by the four different levels of acceptance: poor, good, super, excellent. The preferences can be expressed in four fuzzy rules. Each rule holds another category of Acceptance in its rule head. As the number of rules, and the rule heads are already known, the user only enters up to four expressions (see `<Expression>` in Table 1) in the text fields which are marked with the corresponding level of acceptance. Editing the rule body expressions is assisted by auto completion for the keywords defined in the grammar and the property names defined in domain ontologies.

Figure 4 show a screenshot of a Web based user interface that allows to model the categories of a property. In the depicted example, three categories represented by three labeled membership functions model were added to the diagram. A trapezoid, which is determined by four points, can be arbitrarily adjusted by the drag and drop functionality of the four characteristic points. The user interface enforces, that acceptance of at each of the points is in the interval [0,1] depicted on the vertical axis. The property value range on the horizontal axis can be adjusted. Further, for acceptance=0, the horizontal extent of a trapezoid must be equal or larger than for higher acceptance values.

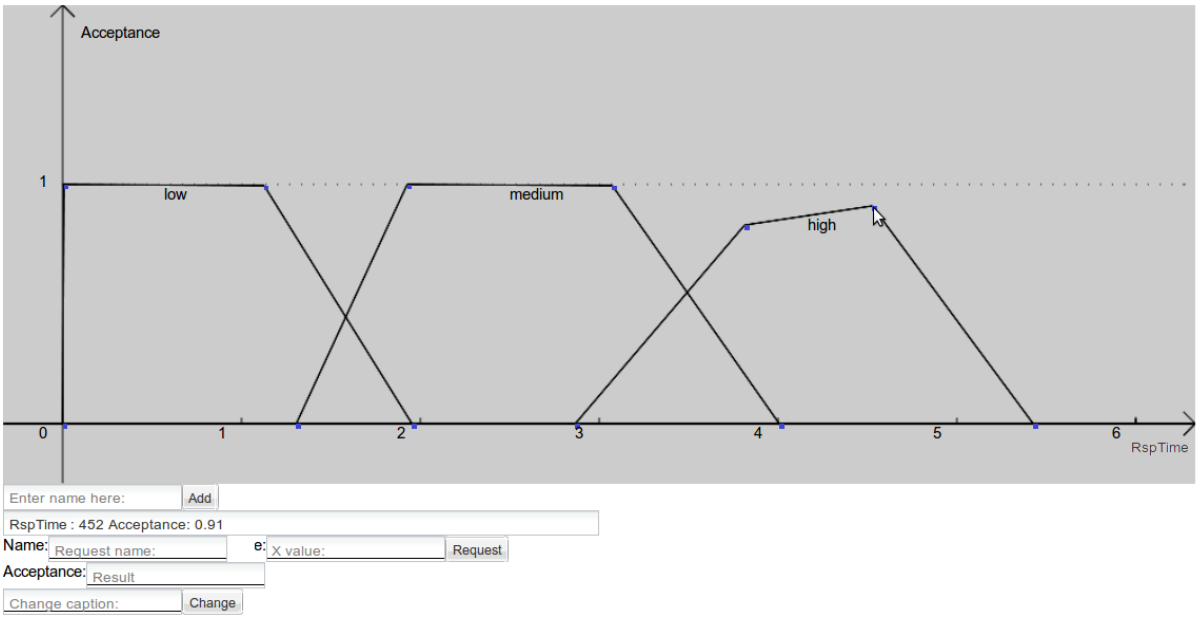

Figure 4: Screenshot of the user interface to define categories of a property by membership functions.

The result of the ranking method is a ranked list of services. These are displayed in an ordered list within the discovery user interface that is already described in D3.4.2.

5. Use of Ranking within SOA4All

5.1 Use in Other Components

Service Construction

The use of ranking in service construction goes hand-in-hand with the documented use of discovery. The repository-based approach has been evaluated with respect to the aims of WP6 and would match particularly well where the template owner can explicitly register which services they're prepared to use, at run-time, and retrieve a dynamic rank based just on these (as discussed in Section 6).

5.2 Deliverable relation with the use cases

All use cases potentially use ranking, together with discovery, especially: (WP7) in locating relevant services within an enterprise (when, again the management of a 'short-list' of services, and the dynamic provision of a rank would be advantageous) and within e-Government scenarios, which is the longest-standing use of discovery and ranking in use case demonstrators; (WP8) in considering the objective ranking metrics (uptime and reliability, etc.) of third party services (SMS, etc.) in geographical regions where Ribbit does not provide these, as in the demonstration prepared for the M24 review. WP9 has offered the most concrete new use for DisCloud-based templates, as the long-term brokerage model of shared templates, representing payment services, etc., fits particularly well and this will be further investigated.

6. Conclusions and Outlook

In this deliverable we have introduced the latest work on ranking, which has been oriented towards development on the planned fuzzy ranking approach, and on integration. The integration has itself produced two major new artifacts: the WSL4J object model, which has been handed over to WP3 for maintenance, and the DisCloud service template repository, which will be the basis of further work in the remainder of the project.

Plans for extension, based on DisCloud involve the following:

- full integration of the fuzzy ranking approach, as this is completed;
- provision for set-up of an 'approved shortlist' of matching services against a template, and ability to retrieve a rank just across those services;
- work on scalability of discovery and ranking based on this repository.

Regarding scalability, the plan is to apply the MapReduce scheme (technically the Hadoop implementation) to the two points in the lifecycle of the repository that require significant computation, that is: the upload of a new template, and the notification of a new service. Further details on this issue are contained in the Conclusion and Outlook section of D5.3.2.

Annex A. Selected JavaDocs

Packages	
eu.soa4all.wsl4j	
eu.soa4all.wsl4j.rpc	
eu.soa4all.wsl4j.ServiceTemplate	
eu.soa4all.wsl4j.WSML	

All Classes

AnnotatedArtifact
Annotation
AnnotationException
Artifact
FunctionalClassificationAnnotation
Message
MessageContent
MessagePart
ModellingException
MSMService
OriginalMSMOperation
OriginalMSMService
ParseException
PartonomisedOperation
PartonomisedService
POSMService
Preference
ReferantAnnotation
RootArtifact
RPCOperation
RPCService
RuleBasedRankingNFP
RuleBasedRankingPreference
SeekdaRankingNFP
Service
ServiceTemplate
StructuralException
USEPreference
ValuedAnnotation
WSMLAnnotation

Class Hierarchy

- java.lang.Object
  - eu.soa4all.wsl4j.**Artifact**
    - eu.soa4all.wsl4j.**AnnotatedArtifact**
      - eu.soa4all.wsl4j.**RootArtifact**
        - eu.soa4all.wsl4j.**Service**
          - eu.soa4all.wsl4j.rpc.**RPCService**
            - eu.soa4all.wsl4j.rpc.**OriginalMSMService**
            - eu.soa4all.wsl4j.rpc.**PartonomisedService**
              - eu.soa4all.wsl4j.rpc.**MSMService**
              - eu.soa4all.wsl4j.rpc.**POSMService**
        - eu.soa4all.wsl4j.ServiceTemplate.**ServiceTemplate**
      - eu.soa4all.wsl4j.rpc.**RPCOperation**
        - eu.soa4all.wsl4j.rpc.**OriginalMSMOperation**
        - eu.soa4all.wsl4j.rpc.**PartonomisedOperation**
    - eu.soa4all.wsl4j.**Annotation**
      - eu.soa4all.wsl4j.**ReferantAnnotation**
        - eu.soa4all.wsl4j.**FunctionalClassificationAnnotation**
      - eu.soa4all.wsl4j.**ValuedAnnotation**
        - eu.soa4all.wsl4j.WSML.**WSMLAnnotation**
          - eu.soa4all.wsl4j.WSML.**RuleBasedRankingNFP**
            - eu.soa4all.wsl4j.WSML.**SeekdaRankingNFP**
    - eu.soa4all.wsl4j.rpc.**Message**
    - eu.soa4all.wsl4j.rpc.**MessagePart**
      - eu.soa4all.wsl4j.rpc.**MessageContent**
    - eu.soa4all.wsl4j.**Preference**
      - eu.soa4all.wsl4j.WSML.**RuleBasedRankingPreference**
      - eu.soa4all.wsl4j.ServiceTemplate.**USEPreference**

java.lang.Throwable (implements java.io.Serializable)

- java.lang.Exception
  - eu.soa4all.wsl4j.**ModellingException**
    - eu.soa4all.wsl4j.**AnnotationException**
    - eu.soa4all.wsl4j.**ParseException**
    - eu.soa4all.wsl4j.**StructuralException**

Package eu.soa4all.wsl4j

Class Summary	
AnnotatedArtifact	
Annotation	
Artifact	
FunctionalClassificationAnnotation	
Preference	
ReferantAnnotation	
RootArtifact	
Service	
ValuedAnnotation	

Exception Summary	
AnnotationException	
ModellingException	
ParseException	
StructuralException	

Package eu.soa4all.wsl4j.ServiceTemplate

Class Summary	
ServiceTemplate	
USEPreference	

Package eu.soa4all.wsl4j.rpc

Class Summary	
Message	Deprecated.
MessageContent	
MessagePart	
MSMService	
OriginalMSMOperation	Deprecated.
OriginalMSMService	Deprecated.
PartonomisedOperation	
PartonomisedService	
POSMService	
RPCOperation	
RPCService	

Package eu.soa4all.wsl4j.WSML

Class Summary	
RuleBasedRankingNFP	
RuleBasedRankingPreference	
SeekdaRankingNFP	
WSMLAnnotation	