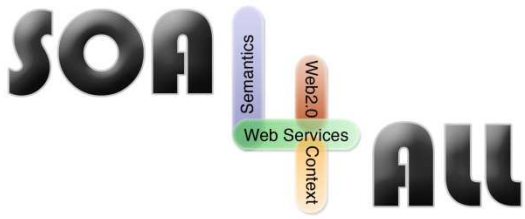

Project Number: **215219**

Project Acronym: **SOA4All**

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

Instrument: **Integrated Project**

Thematic Priority: **Information and Communication Technologies**

D5.4.3 Second Service Ranking Prototype

Activity N:	A2 - Core Research and Development	
Work Package:	WP5 – Service Location	
Due Date:		30/04/2011
Submission Date:		28/04/2011
Start Date of Project:		01/03/2008
Duration of Project:		38 months
Organisation Responsible of Deliverable:		KIT
Revision:		1.4 / Final for submission
Author(s):	Sudhir Agarwal (KIT), Martin Junghans (KIT), Barry Norton (KIT), José María García (USE)	
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	21.04.2011	Initial version	Steffen Stadtmüller (KIT)
1.1	26.04.2011	Preference-based Ranking added	José María García (US)
1.2	27.04.2011	Consolidation	Barry Norton (KIT)
1.3	27.04.2011	Final review for submission	Julia Wells (ATOS)
1.4	28.04.2011	Update of tables pages refs	Julia Wells (ATOS)

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 the development of both a fuzzy logic-based approach to specifying preferences and a model that provides a super-structure to preferences to accommodate all the fore-going approaches into the Discovery Cloud.

This deliverable is a revision to D5.4.2; Section 3.1, on the integrated preference model, is new and Section 4.2 is completely revised and includes details on the distributed implementation of discovery and ranking and performance evaluation.

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, a novel integration approach that reuses the latest developments in general SOA4All technology, including the Service Template model and the iServe service repository, and a new approach to modelling user preferences that covers the existing work including the fuzzy logic-based approach.

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. A large-scale evaluation has been carried out of DisCloud-based discovery and ranking.

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 the work described in Section 3.1 has been introduced. Section 4 describes the work to date on integrated ranking; Section 4.2 described the work in the last period on large-scale evaluation of the integrated approach. 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 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. 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.

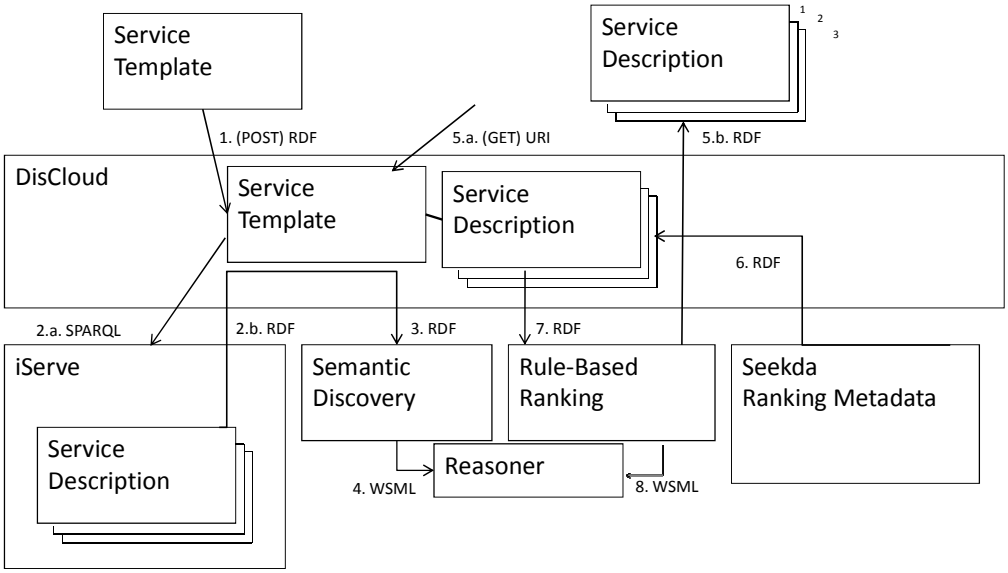

Figure 1: Discovery Cloud Architecture and Interactions.

3.1 Integrated Preference Based Ranking Approach

In order to take full advantage of the three developed ranking techniques detailed in the D5.4.1 deliverable, a user should be able to express preferences using every facility those ranking techniques provide, at the same time. In order to achieve this goal, an integrated ranking approach have been developed, so that a user can define and compose preferences using a generic and expressive model that integrate preference definitions used in the other ranking techniques. This integrated ranking approach can be viewed as a façade to access available ranking techniques using a common, unique access point to them.

3.1.1 Preference Model

The preference model used in this approach is an adaptation of a comprehensive, user-friendly model described in [1]. Basically, the user can express atomic preferences using different preference terms that are handled internally by the corresponding ranking approach, and then composite preferences can be used to compose those terms, defining the relationship between previously expressed atomic preferences. Figure 2 shows a UML representation of this preference model.

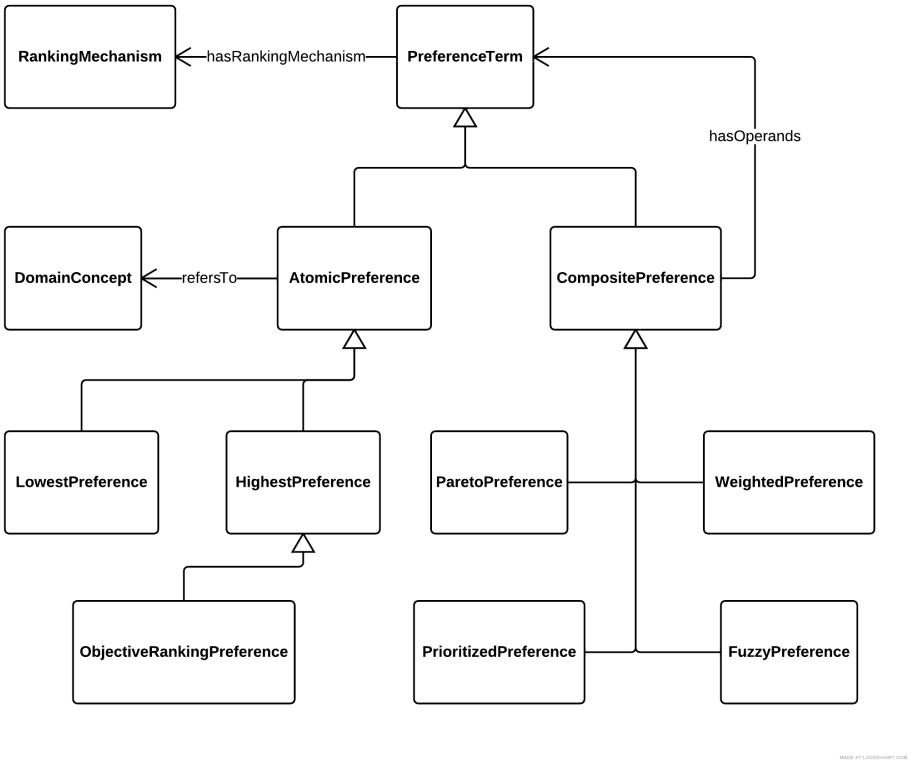

Figure 2: Simplified UML representation of the preference model

Essentially, each preference term is handled by a corresponding ranking mechanism, namely 'objective' ranking metrics, multi-criteria NFP-based ranking, and fuzzy logic based ranking, while more generic composite preferences are directly handled by the integrated ranking framework used in the implementation (see Ranking Implementation section). Note that fuzzy preferences representation is simplified in the diagram (see Fuzzy Logic Based Ranking Approach section for a more detailed description). The correspondences between preference terms and ranking mechanisms are summarized in Table 1.

Table 1 Correspondences between ranking mechanisms and preference terms

Preference Term	Ranking Mechanism
LowestPreference	MultiCriteriaRanking
HighestPreference	MultiCriteriaRanking
ObjectiveRankingPreference	ObjectiveMetricsRanking
ParetoPreference	DefaultParetoRanking
PrioritizedPreference	DefaultPrioritizedRanking
WeightedPreference	MultiCriteriaRanking
FuzzyPreference	FuzzyLogicBasedRanking

Atomic preferences are related to a domain-specific concept that represents a NFP that should be optimized to fulfil the user preference over it. For instance, a Lowest (a Highest) preference means that the user prefers an NFP value the lower (the higher) the better. These preferences mimic the ascending or descending order defined in the multi-criteria, NFP-based ranking approach, while using Weighted preferences the user can define each atomic preference interest value.

The objective ranking metrics approach is actually an optimization of ranking metrics, so it is handled similarly to a highest preference, but the referred domain concept to optimize is one of the available metrics. Finally, users can compose preferences by balancing their fulfilment degree (a Pareto preference) or prioritizing some preferences over others (a Prioritized preference). See [1] for further details.

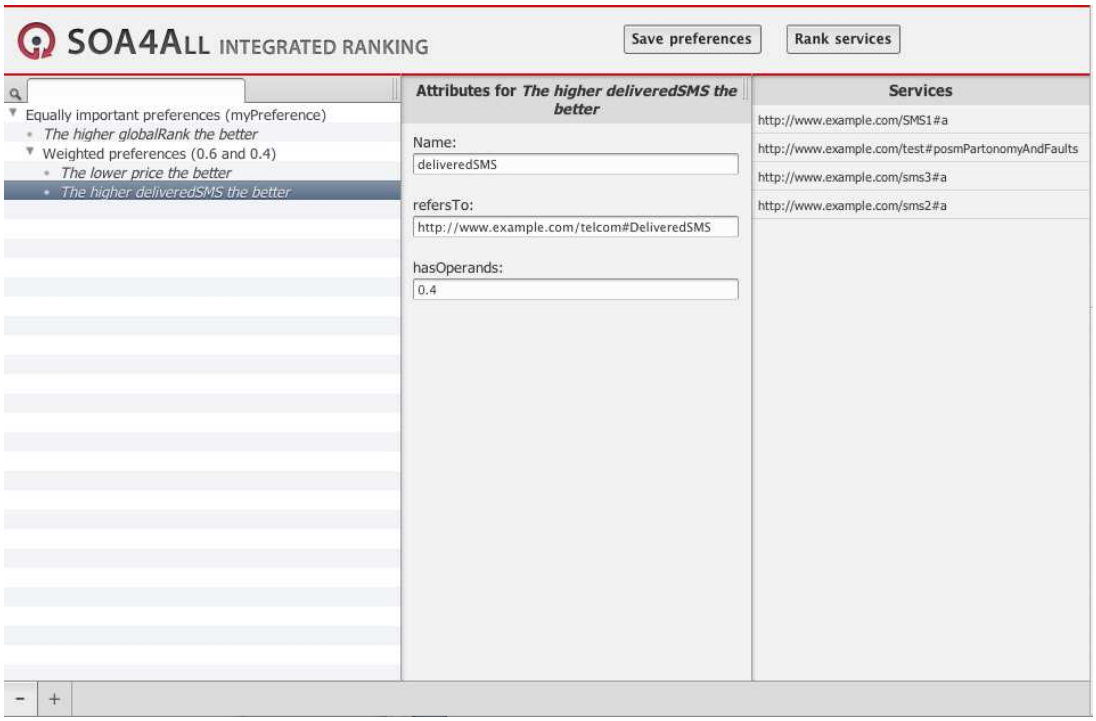

Figure 3. Screenshot of the preference definition UI

3.1.2 User Interface

A user interface to define preferences and rank services accordingly have been developed, using the Google Web Toolkit and based on Universidad de Sevilla's *AcME* modelling toolkit[1]. This interface allows the user to easily define preferences based on the discussed model. For instance, in Figure 3, a user has defined a preference that balance the importance of a higher global rank with a multi-criteria preference over a lower price (with an interest value of 0.6) and a higher number of delivered SMS (with an interest value of 0.4).

Additionally, the user interface can also be used to test the integrated preference based ranking implementation, so a set of pre-loaded services can be ranked in terms of the created preferences. Using the "Rank services" button, the resulting ranking of services is shown. In the next section this integrated ranking implementation is introduced.

3.1.3 Implementation

The developed integrated preference based ranking approach uses preferences defined after the presented model in order to rank a set of discovered services. As described before, each preference term is handled by a particular ranking mechanism. In order to correctly call each mechanism, compose the results, and manage in general the integrated ranking process, the implementation is based on the *PURI* framework, developed by Universidad de Sevilla as a working implementation of the model discussed in [1].

PURI[2] stands for Preference-based Universal Ranking Integration framework, and provides facilities to integrate several ranking mechanisms by using an extended preference model. The integrated ranking approach adapts the PURI framework, integrating the three ranking approaches described in this deliverable.

This implementation is published as a web service that provides a method called rank that takes a set of services to rank and the user preference defined after the discussed preference model. Concretely, this method firstly analyses the user preference. Then, service ranking for each preference term is delegated to the corresponding ranking mechanism presented in Table 1. The adaptation of PURI framework that has been developed is responsible to both the delegation mechanism and the composition of ranked results for each preference term. Finally, the method returns the requested ranked list of services.

[1] AcME development has been supported by the European Commission (FEDER) and Spanish Government under CICYT project SETI (TIN2009-07366) and PPP project SMARTGRID. More information can be found at http://www.isa.us.es/acme

[2] PURI is currently under development, supported by the European Commission (FEDER) and Spanish Government under CICYT project SETI (TIN2009-07366). An early prototype described in [2] can be found at http://www.isa.us.es/upsranker

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[3], 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.

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

4.2 DisCloud

The Discovery Cloud, or DisCloud2, stores service descriptions as well as service templates, both consisting of a pair of SPARQL graph patterns. In case of a service description, these patterns are representing the input and output of the described service. In case of a service template the patterns are representing the input a client can provide to invoke a service and the output the client requests.

The DisCloud offers a RESTful repository API: Service descriptions and templates can be POSTed, returning the usual HTTP status codes:

- **201 Created** – the service description/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).

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

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

Every submitted template is stored and matched with all currently stored service descriptions. Jena[4], a Java framework for building Semantic Web applications, and ARQ[5], a query engine for Jena, is used to implement the matching mechanism. To determine if a service description matches a template it is calculated, if the graph representing the input in the service description is 'contained' in the graph representing the input in the template; equivalent to checking whether the promised input is enough to invoke the described service. Analogously it is determined if the graph pattern representing the output of the template is contained in the graph representing the output in the service description. This is equivalent to checking, if the output of the described service is enough to satisfy the demands declared in the template.

To allow for a sophisticated ranking, instead of just a binary discovery decision, additionally two other metrics calculated: The *predicate subset ratio*, which measures the fraction of predicates used in the input graph pattern of the service description, that are also used in the input graph pattern in the template (and vice versa for output graph patterns). The *resource subset ratio* analogously measures the fraction of resources in subject or object position of the triple patterns in one graph that are used in the other.

These two metrics provide a continuum of matches be expressing to what degree, the template and the service description use the same vocabulary. To calculate the two ratios for input and output respectively ARQ SPARQL SELECT queries are executed over the skolemized graphs to extract the set of predicates (resources) used in the graph patterns. Each set of metrics, generated in this way, for every combination of template and service descriptions is tagged with an identifier consisting of a combination of respective service description URI and template URI.

To allow service descriptions to be updated, or to populate the system with new service

[4] http://openjena.org

[5] http://jena.sourceforge.net/ARQ/

descriptions, an analogous process is employed. A submitted (via HTTP POST) service description is stored in the system and matched with all service templates. The resulting metrics are also tagged with an identifier and complement the already existing results. Thus, every combination of template and service description has a set of results that is persistently saved and can be retrieved from the system via HTTP GET.

Apache Hadoop[6], the open-source implementation of Google MapReduce, is used to provide for scalability. The Hadoop software allows for parallel computation of the metrics on different nodes in a cluster of machines.

OpenCirrus[7], a research testbed for cloud-based systems, is used to deploy the DisCloud. This environment provides an "infrastructure as a service"-cloud (IaaS) to easily create and configure virtual machines that act as independent computers. These machines are used to set up a Hadoop cluster. This implies, that the cluster runs on top of a cloud, further abstracting from actual physical hosts.

When a template, or service description, is submitted, Hadoop calculates the matching metrics by transferring and executing the code, that implements the matching mechanism together with the submitted template, to the nodes where the service descriptions (templates) are stored, rather than moving the data to the code.

After calculation of the metrics, a timestamp and the identifier are assigned to every set of metrics, which are persistently saved on cloud. Since the system also allows for updating templates and service descriptions by re-submitting a new version of them with the same identifier the newly generated results are compared with the results that are already stored. If a submitted service description is tagged with the same URI than a preexisting service description (i.e., an update is intended), some of the generated results will also have the same identifiers. In this case the older results are deleted, which can be checked by using the mentioned timestamps.

To evaluate the performance of the DisCloud a generator was developed to create random pairs related SPARQL graph patterns within boundaries, set by certain parameters, described below. These graph patterns can be interpreted as input and output of service descriptions or templates.

For evaluation 10 000 service descriptions have been generated and deployed on the DisCloud. The graph patterns in these service descriptions are composed with a random number between 5 and 50 of triple patterns. The triple patterns for every respective pair are generated with resources in subject or object position, randomly drawn out of a local resource pool consisting of 10 to 50 different (URI-identified) resources. These local resources are randomly drawn out of a global pool of 500 resources. The predicates in the triple patterns of the tuples are also randomly drawn out of 3 to 25 different predicates in a local predicate pool. And again this local pool is randomly chosen out of a global pool of 250 predicates. So the difference between the local and global pools is, that the global pools of resources and predicates are used for all tuples, whereas the local predicates and resources are only consistent for both of the graph patterns within a tuple. This approach is chosen to establish a credible relationship between input and output.

Additionally the generator uses variables, rather than resources, in subject or object position with a probability of 0.3 in each case. A variable is used in predicate position with a

[6] http://hadoop.apache.org

[7] http://opencirrus.org

probability of 0.2. In every tuple between 2 and 10 different variables are used. Since variables are already locally valid within one tuple, no global variable pool to draw from is needed. Additionally a generated pair of graph patterns is used as template with the same parameters.

Then the matching process for the generated template over all service descriptions was triggered on the DisCloud using different setups with one, two, five, eight and ten Hadoop worknodes on the OpenCirrus testbed. The execution time needed for the matching itself as well as the overall execution time was measured. The latter includes the time needed to compare the newly calculated with the preexisting metric sets (i.e. the update mechanism). To account for fluctuations in network traffic we measured the matching on each setup twice. The results are shown in Table 1 and Table 2.

Table 2: measurements of exclusive matching time

Worknodes	Execution	Time (sec)	Mean (sec)	Standard deviation (sec)	Standard error (sec)
1	1.	394.3	395	1	0.7
	2.	395.8			
2	1.	223.6	221.5	3	2.1
	2.	219.3			
5	1.	120.6	122.4	2.4	1.7
	2.	124			
8	1.	121.7	119.5	3.2	2.3
	2.	117.2			
10	1.	81.4	81.8	0.5	0.4
	2.	82.1			

Table 3: measurements of overall execution time

Worknodes	Execution	Time (sec)	Mean (sec)	Standard deviation (sec)	Standard error (sec)
1	1.	477.3	470.3	9.9	7.0
	2.	463.3			
2	1.	283.7	280.4	4.7	3.3
	2.	277			
5	1.	169.7	162.9	9.6	6.8
	2.	156			
8	1.	155.3	161.2	8.2	5.9
	2.	167.1			
10	1.	134	127.8	8.7	6.2
	2.	121.7			

The calculation of the metrics alone took between 81.4 sec, on ten worknodes, and 395.8 sec, on one worknode. The overall execution time was measured between 121.7 sec using ten worknodes, and 477.3 sec on one worknode. This observation shows, that a high scalability is achieved, which cannot only be attributed to the increase of computation resources (i.e., adding additional CPUs and memory with every worknode), but also to the strategic distribution and execution of matching tasks.

For the overall execution time the standard deviation ranges between 4.7 sec and 9.9 sec, which results in a standard error between 3.3 sec and 7 sec. For the exclusive matching process the standard deviation is measured between 0.5 sec and 3.2 sec, which results in a

standard error between 0.4 sec and 2.3 sec. Those values clearly indicate the stability of the system.

These results are not only valid for the matching a template over service descriptions, but also for populating the discovery system with a new service description, because they are syntactically equivalent and the process to submit a new service description is symmetrical to the process of submitting a new template.

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 5, 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 4 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'.

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

Table 4 : Grammar of Fuzzy IF-THEN Rules.

```
<Rule>          ::=  'IF' <Body> 'THEN' <Head>
<Body>     ::=  <Expression>
<Expression>    ::=  <Term> | '(' <Expression> ')' |
<Conjunction>
                | <Disjunction> | <Negation>
<Conjunction>   ::=  <Expression> 'AND' <Expression>
<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 5 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 4 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 5 depicts the relation of properties, categories, and the characteristic points of membership functions.

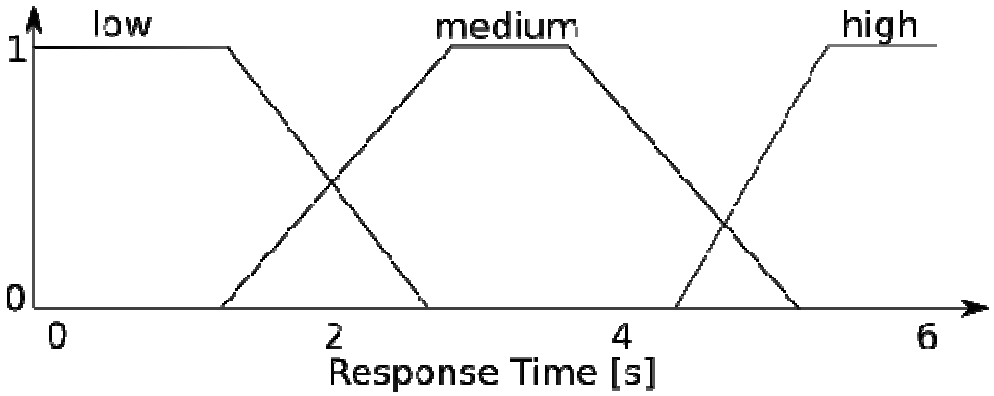

Figure 4: Categories of property response time modeled by membership functions.

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 [**Error! Reference source not found.**] based service description may contain non-unctional 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 with a concrete value. Within the computation of the ranking, each service is reduced to a set of key value pairs (see Figure 5).

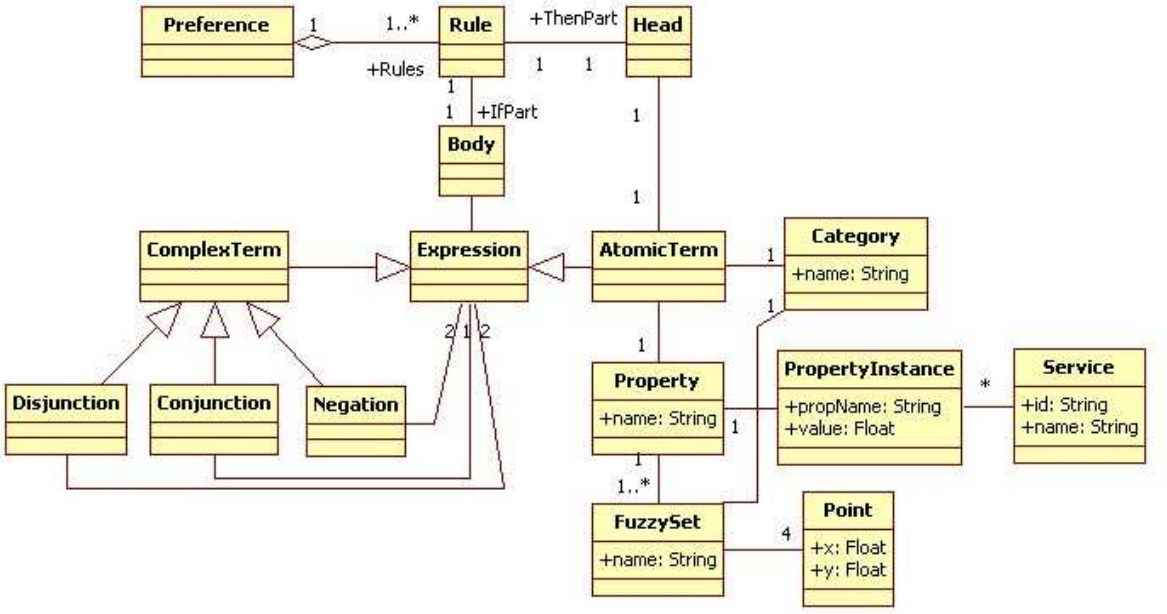

Figure 5: UML diagram of the data objects model.

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 4) 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 6 show a screenshot of a Web based user interface that allows to model the

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

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.

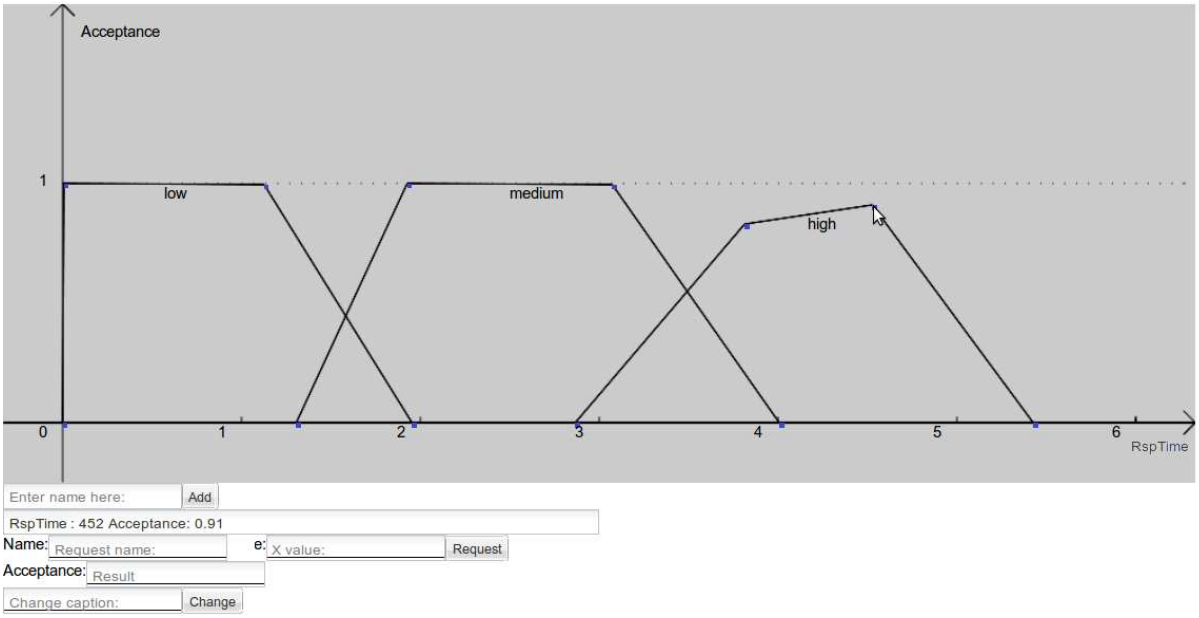

Figure 6: Screenshot of the user interface to define categories of a property by membership functions.

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, a unified preference model, 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.

Finally, to achieve scalability a distributed Hadoop-based implementation has been carried out and evaluated at large scale, as previously sketched in Deliverables 5.3.2 and 5.4.2.

7. References

1. J. M. Garcia, D. Ruiz, A. Ruiz-Cortés. A Model of User Preferences for Semantic Services Discovery and Ranking, in: L. Aroyo, G. Antoniou, E. Hyvönen, A. ten Teije, H. Stuckenschmidt, L. Cabral, T. Tudorache (Eds.), ESWC (2), Vol. 6089 of LNCS, Springer, 2010, pp. 1–14.
2. J. M. García, I. Toma, D. Ruiz, and A. Ruiz-Cortés. A service ranker based on logic rules evaluation and constraint programming, in: 2nd ECOWS Non-Functional Properties and Service Level Agreements in Service Oriented Computing Workshop, volume 411 of CEUR Workshop Proceedings, Dublin, Ireland, Nov 2008.

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	