UNIVERSITY OF JYVÄSKYLÄ

Lecture 9: Semantic Web Services

TIES4520 Semantic Technologies for Developers
Autumn 2023

University of Jyväskylä

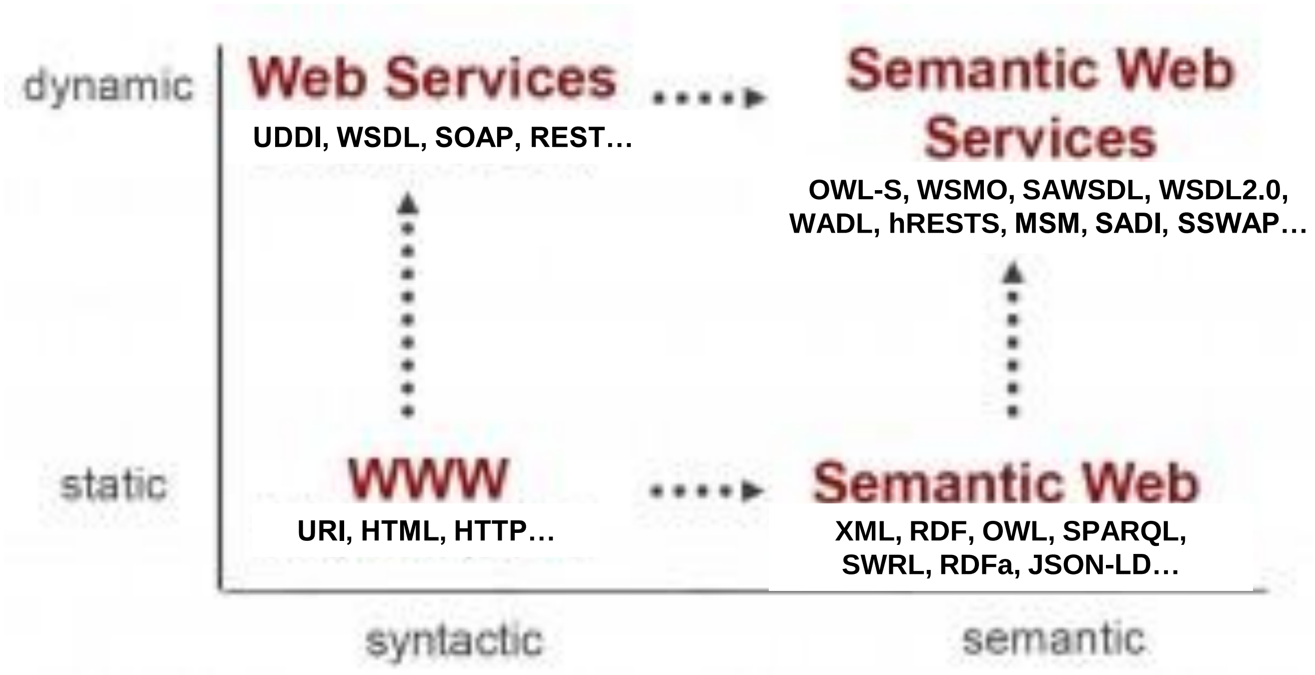
Khriyenko Oleksiy

What is Semantic Web Services?...

- *Web services annotated with semantics. Semantic Web services extend the existing Web services with semantic capability.*
- *Semantic Web services can be defined as web services whose descriptions are annotated by machine-interpretable ontologies so that other software agents can use them without having any prior 'built-in' knowledge about how to invoke them.*
- *Self-contained, self-describing, semantically marked-up software resources that can be published, discovered, composed and executed across the Web in a task-driven semiautomatic way.*
- *Ontology-based descriptions of Web services that could be processed by ontology reasoning tools.*
- *Integrate Web services technology with machine supported data interpretation, using ontologies as a data model, to enable automatic discovery, selection, composition, and Web-based execution of services.*
- *Semantic Web Services are Web Services that can specify not only their interfaces, but also describe in full, under the form of OWL-based ontologies, their capabilities, and the prerequisites and consequences of their use.*
- *Semantic enrichment of current Web Services allowing automated discovery, composition and execution of services.*
- *When Web services and their related messages are semantically described (capabilities, interfaces) with appropriate ontologies, they are called semantic Web services.*
- *Research in the Semantic Web services area is addressing the problems of automated service discovery, composition, execution and similar problems by augmenting the existing Web services standards by additional semantic layer(s). The additional semantic focuses on enabling "... semantically transparent services which will make it possible for clients to successfully use services that are dynamically discovered without prior negotiations between client and service developers". Most of the proposed mechanisms rely on extending the current WS descriptions with descriptions in some logical formalism more suitable for capturing the explicit meaning of provided Web services. Such formalisms typically rely on formal ontologies and research conducted in the Semantic Web area.*
- *Semantic Web Services, like conventional web services, are the server end of a client–server system for machine-to-machine interaction via the World Wide Web. Semantic services are a component of the semantic web because they use markup which makes data machine-readable.*

Link: *https://www.igi-global.com/dictionary/semantic-web-services/26380*

Semantic Web Services



Semantic Web Services

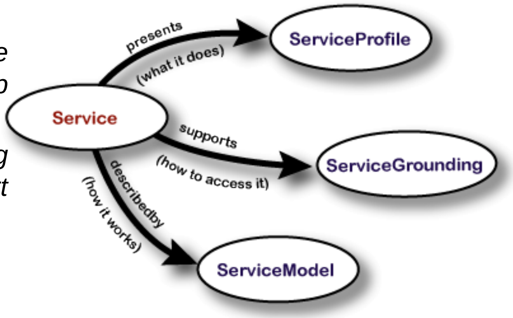
In Semantic Web, users and software agents should be able to discover, invoke, compose, and monitor Web resources offering particular services and having particular properties, and should be able to do so with a high degree of automation if desired. Powerful tools should be enabled by service descriptions, across the Web service lifecycle. ***Semantic Web Services***, like conventional web services, are system for machine-to-machine interaction via the Web, but additionally support semantic meaning of the messages making data machine-readable in a detailed and sophisticated way.

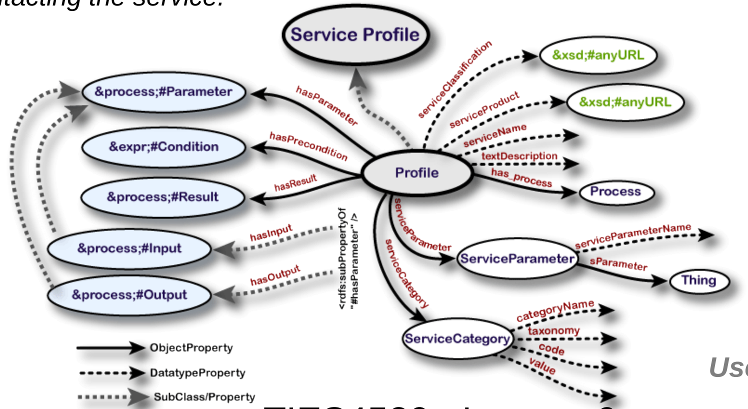
- *OWL-S (formerly DAML-s) defines an ontology for the domain of Web services that provides tags which can be used for describing actual Web Services (http://www.w3.org/Submission/OWL-S). OWL-S services are mapped to WSDL operations, inputs and outputs of OWL-S are mapped to WSDL messages. Top level ontology consists of:*

Service Profile – tells "what the service does“, includes a description of what is accomplished by the service, limitations on service applicability and quality of service, and requirements that the service requester must satisfy to use the service successfully;

Service Model – tells a client how to use the service, by detailing the semantic content of requests, the conditions under which particular outcomes will occur, and, where necessary, the step by step processes leading to those outcomes;

Service Grounding - specifies the details of how an agent can access a service. Typically a grounding will specify a communication protocol, message formats, and other service-specific details such as port numbers used in contacting the service.





Useful link: *Practical tutorial on Semantic Web Services [1]*

Semantic Web Services

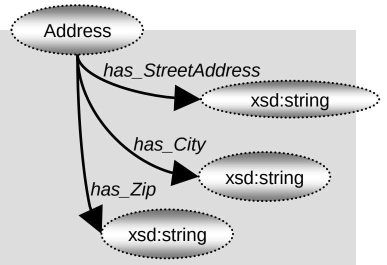
Semantic Web Services, like conventional web services, are system for machine-to-machine interaction via the Web, but additionally support semantic meaning of the messages making data machine-readable in a detailed and sophisticated way.

- *WSDL-S extends WSDL with extending attributes (e.g. wssem:modelReference, wssem:schemaMapping, wssem:precondition, wssem:effect, wssem:category) and provides a lightweight (comparing to OWL-S) approach to create Semantic Web Service descriptions (http://www.w3.org/Submission/WSDL-S/);*
- *SAWSDL defines a set of extension attributes for the WSDL and XML Schema (http://www.w3.org/2002/ws/sawsdl/)(relevant tools: http://easywsdl.ow2.org/extensions-sawsdl.html). It does not specify a language for representing the semantic models, but provides mechanisms by which concepts from the semantic models can be referred using annotations, specify how to translate an XML schema element to/from an ontology instance in another language using following extensibility attributes:*
  - *modelReference - to specify the association between a WSDL or XML Schema component and a concept in some semantic model;*

```
<wsdl:operation name="order"
    sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/ontology/rosetta#RequestPurchaseOrder">
    <wsdl:input element="OrderRequest"/>
    <wsdl:output element="OrderResponse"/>
    <wsdl:fault name="ItemUnavailableFault" element="AvailabilityInformation"
        sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/ontology/rosetta#ItemUnavailable"/>
</wsdl:operation>
```

  - *liftingSchemaMapping and loweringSchemaMapping - to specify mappings between WSDL Type Definitions in XML and semantic data and back respectively. Lowering and Lifting scripts can be written in XSLT together with SPARQL. Later, XSPARQL (which combines XQuery and SPARQL) have been recognized as a more effective language for this purpose (http://www.w3.org/Submission/xsparql-language-specification).*

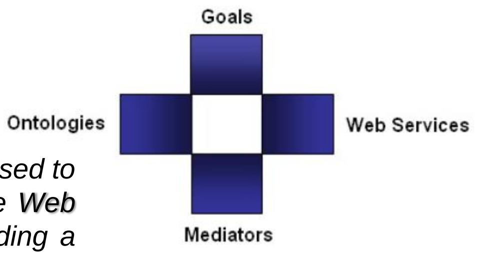
```
<complexType name="POAddress"
    sawsdl:modelReference = "http://www.w3.org/2002/ws/sawsdl/spec/ontology/purchaseorder#Address"
    sawsdl:liftingSchemaMapping = "http://www.w3.org/2002/ws/sawsdl/spec/mapping/POAdress2Ont.xslt"
    sawsdl:loweringSchemaMapping = "http://www.w3.org/2002/ws/sawsdl/spec/mapping/Ont2POAddress.xsparql" >
    <all>
     <element name="streetAddr1" type="string" />
     <element name="streetAddr2" type="string" />
     <element name="zipCode" type="string" />
     <element name="city" type="string" />
    </all>
</complexType>
```



Semantic Web Services

Semantic Web Services, like conventional web services, are system for machine-to-machine interaction via the Web, but additionally support semantic meaning of the messages making data machine-readable in a detailed and sophisticated way.

- *WSMO (Web Service Modeling Ontology) (http://www.w3.org/Submission/WSMO) is a conceptual model for relevant aspects related to Semantic Web Services (capabilities, orchestration and choreography, goals, ontologies, etc.) that provides an ontology based framework, which supports the deployment and interoperability of Semantic Web Services. WSMO identifies four main top-level elements:*
  1. *Ontologies that provide the terminology used by other elements;*
  2. *Goals that state the intentions that should be solved by Web Services;*
  3. *Web Services descriptions which describe various aspects of a service;*
  4. *Mediators: to resolve interoperability problems.*



In contrast to OWL-S, it has built-in syntax for encoding the boolean formulas that are used to describe the pre-conditions and post-conditions of the services. It is presented in the Web Service Modeling Language (WSML) - a formalization of the WSMO ontology, providing a language within which the properties of Semantic Web Services can be described.

WSMX execution environment provides an architecture including discovery, mediation, selection, and invocation and has been designed including all required supporting components enabling an exchange of messages between requesters and the providers of services.

- *It has a lightweight ontology WSMO-Lite - the next evolutionary step after SAWSDL, filling the SAWSDL annotations with concrete semantic service descriptions. WSMO-Lite (http://www.w3.org/Submission/WSMO-Lite/)* is inspired by the WSMO ontology, focusing on a small subset of it to define a limited extension of SAWSDL. *WSMO-Lite addresses the following requirements:*
  - *Identify the types and a simple vocabulary for semantic descriptions of services (a service ontology).*
  - *Define an annotation mechanism for WSDL using this service ontology.*
  - *Provide the bridge between WSDL, SAWSDL and (existing) domain-specific ontologies such as classification schemas, domain ontology models, etc.*

Semantic Web Services

Stirred by ***Web 2.0***, classical Web Services has significantly evolved with the proliferation of ***Web APIs*** – ***RESTful services*** (when confirm to the ***REST*** *(Representational State Transfer)* architectural style).

- *Web APIs are generally described using plain, unstructured HTML*
- *There are some formats to represent Web API in machine-readable form:*
  - ***WSDL 2.0*** *– an extension of WSDL to support the rest HTTP operations could be considered as acceptable method of documenting RESTful services (http://www.w3.org/TR/wsdl20/). But it is still operation-based description for resource-oriented services, contains some difficulties with specifying the message exchange formats;*
  - ***WADL*** *- a machine-readable XML description of HTTP-based web applications (http://www.w3.org/Submission/wadl/). As opposed to WSDL, WADL models the resources provided by a service, and the relationships between them in the form of links. A service is described using a set of resource elements. Each resource contains descriptions of the inputs and method elements, including the request and response for the resource. The main difficulty of using WADL descriptions is that they are complex, in comparison to text-based documentation, and require that developers have a certain level of training and tool support that enables the application development on top of WADL;*
  - ***hRESTS*** *(HTML for RESTful Services) microformat for machine-readable descriptions of Web APIs [2][3].* ***hRESTS*** *enables the marking of service properties including operations, inputs and outputs, HTTP methods and labels, by inserting HTML tags within the HTML. As an alternative to using hRESTS,* ***RDFa*** *- enables the embedding of RDF data in HTML.* ***RDFa*** *is similar to using* ***microformats****, but is somewhat more complex and offers more HTML marking options, as opposed to hRESTS.*
- *Approaches, based on making existing RESTful service descriptions machine-processable by using HTML tags are simpler and more lightweight as opposed to WSDL and WADL. In addition, they can be applied directly on already available descriptions, rather then creating new service descriptions from scratch. The adoption by developers is also easier, since the creation of a machine-processable restful service description is equivalent to Web content creation or modification.*
- *Web APIs require supportive infrastructure/tools to provide more sufficient descriptions in machine and human readable form. (e.g.,* ***HAL*** *- a format that simplify developer's work with API. HAL provides a set of conventions for expressing hyperlinks in either JSON or XML : http://stateless.co/hal_specification.html);*
- *One of the registries of Web APIs: http://www.programmableweb.com*

Semantic Web Services

HTML:

```
<p>
   Description of the ACME Hotels service:
</p>
<p>
   The operation <code> getHotelDetails </code> is
   invoked using the method GET at <code>
   http://example.com/h/{id} </code>, with the ID of
   the particular hotel replacing the parameter
   <code> id </code>. It returns the hotel details in
   an <code> ex:hotelInformation </code> document.
</p>
```

hRESTS:

```
<div class="service" id="svc">
  <p> Description of the <span class="label" > ACME
  Hotels </span> service:
  </p>
  <div class="operation" id="op1">
    <p> The operation <code class="label" >
    getHotelDetails </code> is invoked using the
    method <span class="method"> GET </span> at <code
    class="address"> http://example.com/h/{id}
    </code>, with <span class="input">the ID of the
    particular hotel replacing the parameter
    <code> id </code>.</span> It returns <span
    class="output">the hotel details in an <code>
    ex:hotelInformation </code> document.</span>
    </p>
  </div>
</div>
```

hRESTS service model:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix hr: <http://www.wsmo.org/ns/hrests#> .

hr:Service a rdfs:Class .
hr:hasOperation a rdf:Property ;
  rdfs:domain hr:Service ;
  rdfs:range hr:Operation .
hr:Operation a rdfs:Class .
hr:hasInputMessage a rdf:Property ;
  rdfs:domain hr:Operation ;
  rdfs:range hr:Message .
hr:hasOutputMessage a rdf:Property ;
  rdfs:domain hr:Operation ;
  rdfs:range hr:Message .
hr:hasAddress a rdf:Property ;
  rdfs:domain hr:Operation ;
  rdfs:range hr:URITemplate .
hr:hasMethod a rdf:Property ;
  rdfs:domain hr:Operation ;
  rdfs:range hr:HTTPMethod .
hr:Message a rdfs:Class .
hr:URITemplate a rdfs:Datatype .
hr:HTTPMethod a rdfs:Class .
hr:GET a hr:HTTPMethod .
hr:POST a hr:HTTPMethod .
hr:PUT a hr:HTTPMethod .
hr:DELETE a hr:HTTPMethod .
```

Semantic Web Services

HTML:

```
<p>
   Description of the ACME Hotels service:
</p>
<p>
   The operation <code> getHotelDetails </code> is
   invoked using the method GET at <code>
   http://example.com/h/{id} </code>, with the ID of
   the particular hotel replacing the parameter
   <code> id </code>. It returns the hotel details in
   an <code> ex:hotelInformation </code> document.
</p>
```

hRESTS:

```
<div class="service" id="svc">
  <p> Description of the <span class="label" > ACME
  Hotels </span> service:
  </p>
  <div class="operation" id="op1">
    <p> The operation <code class="label" >
    getHotelDetails </code> is invoked using the
    method <span class="method"> GET </span> at <code
    class="address"> http://example.com/h/{id}
    </code>, with <span class="input">the ID of the
    particular hotel replacing the parameter
    <code> id </code>.</span> It returns <span
    class="output">the hotel details in an <code>
    ex:hotelInformation </code> document.</span>
    </p>
  </div>
</div>
```

RDF description:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix hr: <http://www.wsmo.org/ns/hrests#> .
@prefix jk: <http://members.sti2.at/jacekk/hrests/src.xhtml#> .

jk:svc a hr:Service;
  rdfs:isDefinedBy
  <http://members.sti2.at/jacekk/hrests/src.xhtml>;
  rdfs:label ”ACME Hotels”;
  hr:hasOperation jk:op1 .
jk:op1 a hr:Operation;
  rdfs:label ”getHotelDetails”;
  hr:hasMethod hr:GET;
  hr:hasAddress
  ”http://example.com/h/{id}”^^hr:URITemplate;
  hr:hasInputMessage [ a hr:Message ];
  hr:hasOutputMessage [ a hr:Message ] .
```

hRESTS ***does not*** *actually provide machine-readable information about the inputs and outputs…*

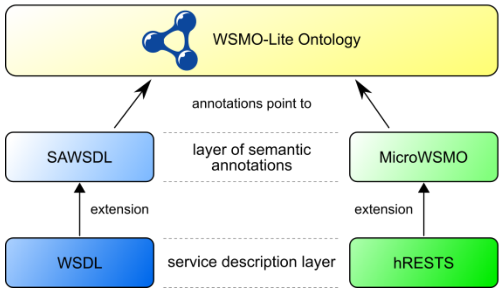
Semantic Web Services

SA-REST originally proposed as an RDFa-based annotation mechanism, is an extension of hRESTS that supports the description of different service properties (various facets of Web APIs) to support search and filtering for programmers:

- *data formats (e.g., XML, JSON, GData and ATOM/RSS, etc.);*
- *programming language bindings.*

MicroWSMO is a SAWSDL-based extension of hRESTS with the elements (model, lowering, lifting) which can be associated with the elements of a hRESTS definition.

- *modelReference is used to link any part of a service description with its semantic properties (points to a concept in some semantic model).*
- *liftingSchemaMapping – transformation of the XML data produced by the web service into RDF data .*
- *loweringSchemaMapping - vice versa to liftingSchemaMapping.*



SA-RESR example:

```
<div class=”service” id=”svc”>
<p>The output format of the operations of the
  <code class=”label”>ACME Hotels</code>
  service is <span class=”data-format”>
  JSON </span>. Client libraries are available
  in <span class=”p-lang-binding”>Java</span>
  and <span class=”p-lang-binding”>PHP</span>.
</p>
</div>
```

MicroWSMO example:

```
<div class=”service” id=”svc”>
  <p><span class=”label”>ACME Hotels</span> is
  a <abbr class=”mref”
  title=”.../ecommerce/hotelReservation”> hotel
  reservation</abbr> service.</p>
  ...
  <div class=”operation” id=”op1”>
    <p> ...
    <span class=”input”>A particular hotel ID
      replaces the param <code class=”mref”
      title=”.../onto.owl#Hotel”>id</code>
      (<a rel=”lowering”
      href=”.../hotelID.xslt”>lowering</a>).
    </span>. ... </p>
  </div>
</div>
```

Semantic Web Services

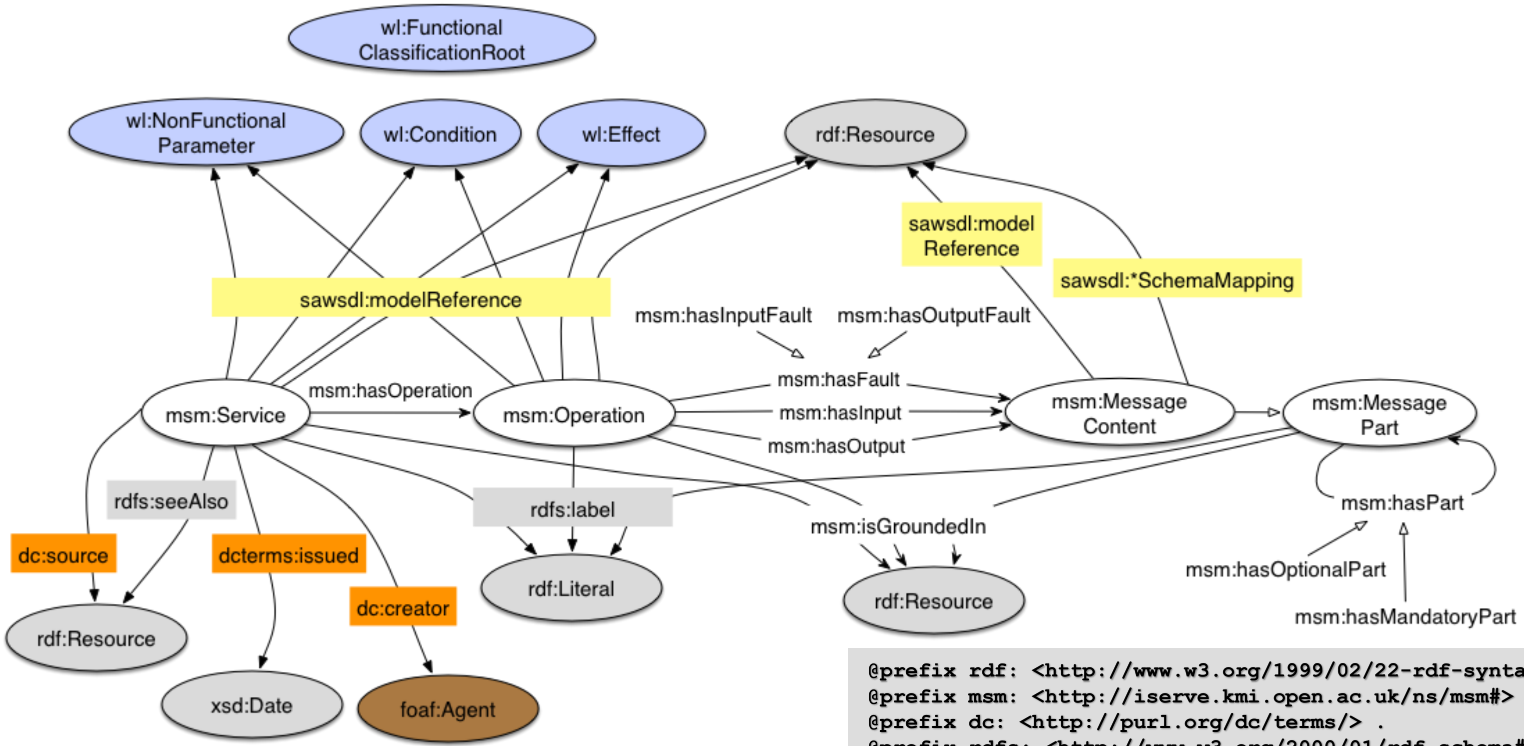
WSMO-Lite Ontology is a lightweight approach to the semantic annotation of Web service descriptions.

```
@prefix  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix  rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix  owl:  <http://www.w3.org/2002/07/owl#> .
@prefix  sawsdl:  <http://www.w3.org/ns/sawsdl#> .
@prefix  wl:  <http://www.wsmo.org/ns/wsmo-lite#> .
@prefix  xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix  dc: <http://purl.org/dc/terms/>.

wl: a owl:Ontology ;
    ...
# classes for use in semantic annotation of service descriptions
wl:FunctionalClassificationRoot  rdfs:subClassOf
rdfs:Class ;
    rdfs:label "Functional Classification Root"@en ;
    rdfs:description "instances of this class are roots of
functionality classification hierarchies"@en .
wl:NonfunctionalParameter  rdf:type  rdfs:Class ;
    rdfs:label "Nonfunctional Parameter"@en ;
    rdfs:description "instances of this class are concrete
nonfunctional properties of services"@en .
wl:Condition  rdf:type  rdfs:Class ;
    rdfs:label "Condition"@en ;
    rdfs:description "preconditions of services and
operations"@en .
wl:Effect  rdf:type  rdfs:Class ;
    rdfs:label "Effect"@en ;
    rdfs:description "effects (postconditions) of services
and operations"@en.
```

```
# the following two are deprecated definitions (as of May 2013),
they haven't been found useful
wl:Ontology  rdf:type  rdfs:Class ;
    rdfs:subClassOf  owl:Ontology ;
    owl:deprecated "true"^^xsd:boolean ;
    rdfs:label "Ontology"@en ;
    rdfs:description "information model ontology
(deprecated)"@en.

# property for identifying potentially relevant ontologies
wl:usesOntology  a  rdfs:Property ;
    rdfs:domain  wl:Service ;
    rdfs:subPropertyOf  rdfs:seeAlso ;
    owl:deprecated "true"^^xsd:boolean ;
    rdfs:label "uses Ontology"@en ;
    rdfs:description "pointer from a semantic
description to an ontology that it uses
(deprecated)"@en ;
    rdfs:isDefinedBy wl: .

# Additional useful definitions, not defined as part of WSMO-Lite

# SAWSDL properties (repeated here for completeness)
sawsdl:modelReference  rdf:type  rdf:Property .
sawsdl:liftingSchemaMapping  rdf:type  rdf:Property.
sawsdl:loweringSchemaMapping  rdf:type  rdf:Property.

# Minimal Service Model used to be copied here but is no longer (as
of May 2013)
```

Semantic Web Services

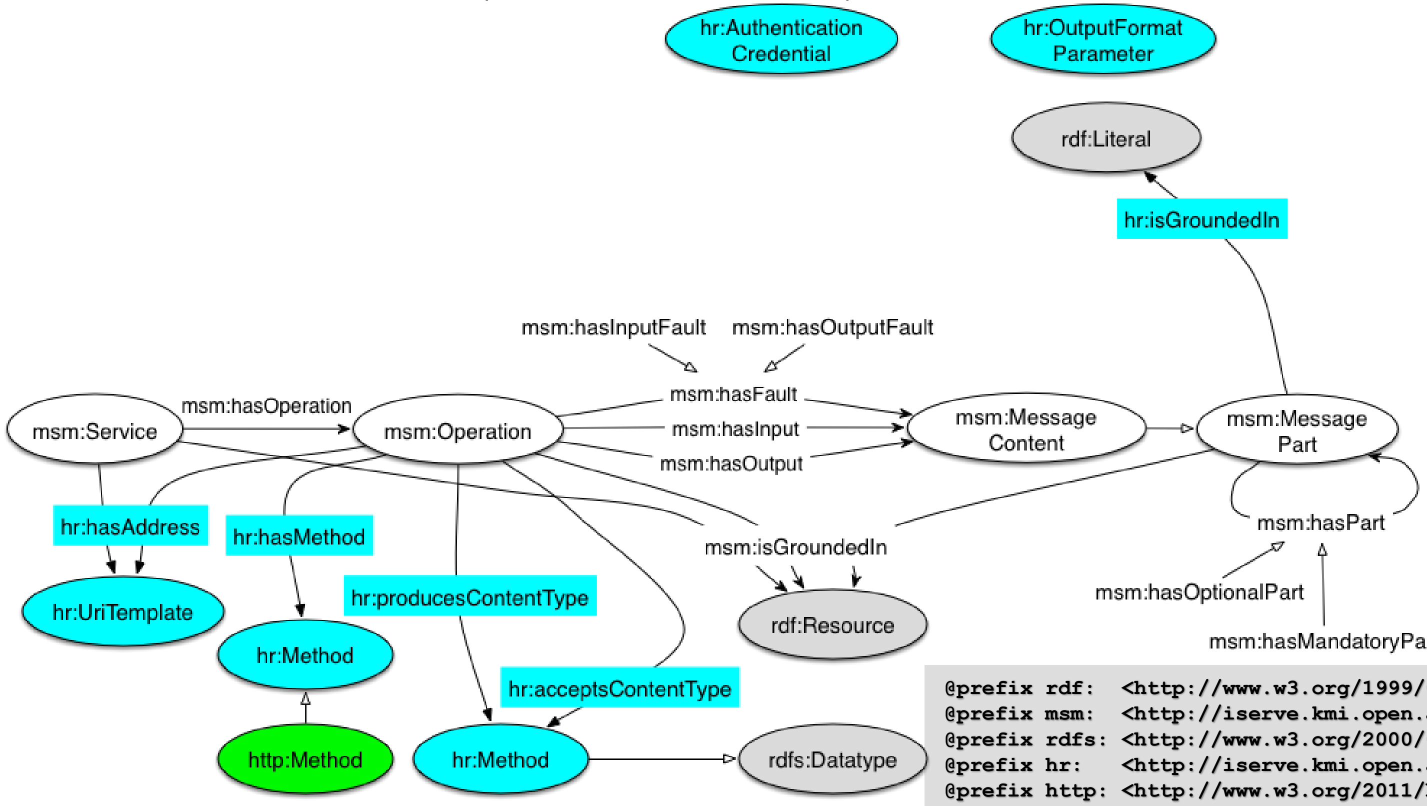
Minimal Service Model (MSM) *represents an operation-based approach towards describing Web APIs. It is a simple RDF(S) ontology that supports the annotation of common Web API descriptions. It also aims to enable the reusability of existing Semantic Web service approaches by capturing the maximum common denominator between existing conceptual models for services.*



```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix msm: <http://iserve.kmi.open.ac.uk/ns/msm#> .
@prefix dc: <http://purl.org/dc/terms/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix wl:  <http://www.wsmo.org/ns/wsmo-lite#> .
@prefix sawsdl:  <http://www.w3.org/ns/sawsdl#> .
```

Semantic Web Services

Minimal Service Model (MSM) *additionally targets to support both traditional Web services, as well as Web APIs with procedural view on resources, so that they can be handled in a unified way. The* ***hRESTS*** *vocabulary, extends the MSM with specific attributes for operations so as to allow modeling additional details necessary for Web APIs such as grounding, URI templates to support URI construction for invocation, HTTP methods used, etc.*



```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix msm:  <http://iserve.kmi.open.ac.uk/ns/msm#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix hr:   <http://iserve.kmi.open.ac.uk/ns/hrests#> .
@prefix http: <http://www.w3.org/2011/http#> .
```

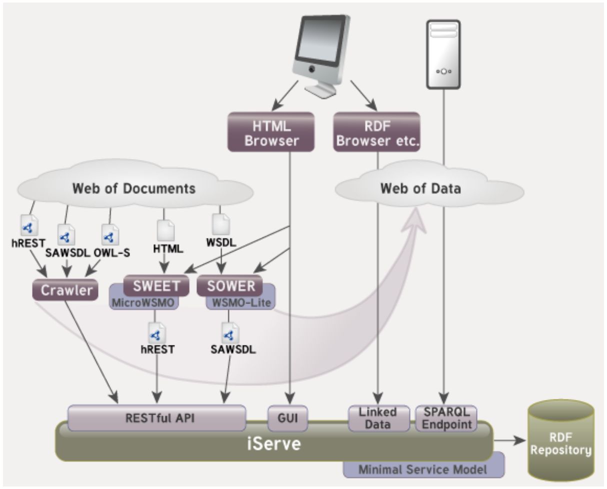
Semantic Web Services

- **Service-to-Service (S2S) interaction:**

Services Discovery and Publication [9][10]

iServe is a service warehouse which unifies ***service publication***, ***analysis***, and ***discovery*** through the use of lightweight semantics as well as advanced discovery and analytic capabilities:

- *publishes **Semantic Web Services** as **Linked Data** (no matter their original format);*
- *provide suitable support for publishing and querying services in expressive and extensible manner;*
- *the registry transparently supports the discovery of heterogeneous services, mainly **WSDL services** (described in WSDL and SAWSDL) and **Web APIs** (documented with hRESTS). Additional support is also provided for OWL-S services.*
- *uses as its core the **Minimal Service Model**, a minimal vocabulary for describing services in RDF which abstracts us away from the original approach used for annotating the services, such as SAWSDL, WSMO-Lite, MicroWSMO or OWL-S.;*
- *provides a range of advanced service analysis and discovery techniques. For instance, **iServe** currently supports input/output discovery using RDFS and SKOS reasoning, functional classification-based discovery with RDFS reasoning, and similarity-based discovery given services' textual descriptions.*
- *URL: http://iserve.kmi.open.ac.uk/ , http://kmi.github.io/iserve/latest/index.html and https://github.com/kmi/iserve/releases*



iServe *provides:*

- *iServe Browser - web-based application allowing users to browse, query and upload services to iServe;*
- *SPARQL endpoint where all the data hosted in iServe can be accessed and queried;*
- *RESTful API that enables creating, retrieving and querying for services directly from applications;*
- *SWEET and SOWER - web-based applications supporting the annotation of Web APIs and WSDLs based on MicroWSMO and WSMO-Lite respectively, which directly provide the means for publishing the annotations in iServe.* (*http://sweet.kmi.open.ac.uk*)

Semantic Web Services

- **Service-to-Service (S2S) interaction:**

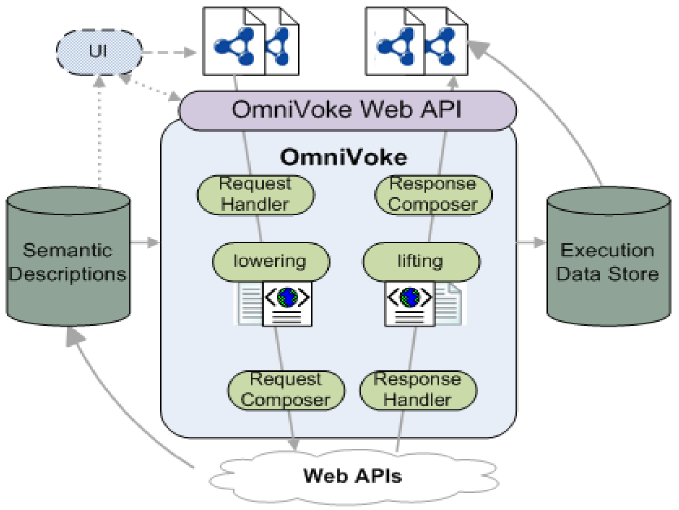
Services Invocation

Google APIs Discovery Service (*https://developers.google.com/discovery/*) gives possibility to build client libraries, IDE plugins, and other tools that interact with Google APIs. It provides a lightweight, JSON-based API that exposes machine-readable metadata about Google APIs. Google has built several tools using the service: ***Google API client libraries*** (client libraries in various languages for accessing Google APIs) and ***Google APIs Explorer*** (an interactive web-based tool for exploring Google APIs).

However, such platforms that provide machine-readable metadata about APIs enabling the auto-generation of client software support only a predefined set of APIs and provide very limited coverage…

OmniVoke [11] is an invocation engine which provides a single interface for invoking linked services (*http://omnivoke.kmi.open.ac.uk*). The engine takes RDF data as input and returns RDF data as a response, enabling a seamless integration of linked services within applications as linked data producers and/or consumers.

- *For services that do not handle RDF natively, the engine uses **lowering** and **lifting** schema mappings as declared on the service description in order to transform, respectively, the RDF input into the suitable data format the underlying endpoint accepts and viceversa. Currently, **OmniVoke** embeds an **XSPARQL** engine to this end.*
- ***OmniVoke** works with the **Extended Minimal Service Model** (http://iserve.kmi.open.ac.uk/ontology/msm1.0.rdfxml), which extends the initial model with invocation-specific service descriptions. In particular, the model define input data grounding (**:isGroundedIn**), which specifies whether the input values are transmitted as part of the URI, HTTP headers or the HTTP request message body (applicable to HTTP POST and PUT requests), as well as the message format when input is transmitted as message body (**:producesContentType** and **:acceptsContentType**).*
- *Since a majority of the Web APIs require some form of authentication, **OmniVoke** uses an **authentication ontology** that enables the annotation of authentication information as part of a semantic Web API description so that authentication can be seamlessly handled by OmniVoke (http://omnivoke.kmi.open.ac.uk/authentication).*



Semantic Web Services

*Previously mentioned description languages have tackled the semantic enrichment of function-based services, such as WSDL and REST, by using domain knowledge describing mainly technical interfaces. In fact, **legal aspects**, **pricing models**, and **service levels** are all elements which need to be explicitly described when dealing with cloud services. Therefore, efforts were redirected to the development of new languages to capture business and operational perspectives beside the technical one. **USDL** and **Linked USDL** are a family of languages that provide a comprehensive view on services to be used by providers, brokers, and consumers when searching, evaluating, and selecting services.* *[4]*

Unified Service Description Language (USDL) is a platform-neutral language for describing services consolidated from SAP Research projects. The kinds of services targeted for coverage by ***USDL*** include human services (e.g., consultancy), business services (e.g. purchase order requisition), software services (e.g., WSDL and RESTful services), infrastructure services (e.g., CPU and storage services), etc. [5]

Linked USDL is a remodeled version of ***USDL*** that builds upon the Linked Data principles and the Web of Data. This effort is therefore most concerned with remodeling the existing ***USDL*** specification as an RDF(S) vocabulary that could better support machines in trading services on the Web. To maximize the potential interoperability Linked USDL adopts, where possible, existing RDF(S) vocabularies such as ***GoodRelations***, the ***Minimal Service Model*** and ***FOAF*** to name a few. ***Linked USDL*** is inline with other Linked Data centric initiatives around services such as the work on ***Linked Services***. [6] (*http://linked-usdl.github.io/ , https://github.com/linked-usdl*)

Semantic Web Services

Linked Services are services described as Linked Data [8].

- *service descriptions, their inputs and outputs, their functionality, and their non-functional properties are described in terms of (reused) light weight RDFS vocabularies and exposed following* Linked Data *principles*
  - *Use URIs as names for thing;*
  - *Use HTTP URIs so that people can look up those names;*
  - *When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL);*
  - *Include links to other URIs, so that they can discover more things.*
- *services, with appropriate infrastructure support, can consume RDF from the Web of Data, and, if necessary, can also generate additional RDF to be fed back to the Web of Data.*

Semantic Web Services

- **Service-to-Service (S2S) interaction:**

Semantic Automated Discovery and Integration (SADI) - is a lightweight set of fully standards-compliant Semantic Web service design patterns [12] (*http://sadiframework.org/content/about-sadi/*).

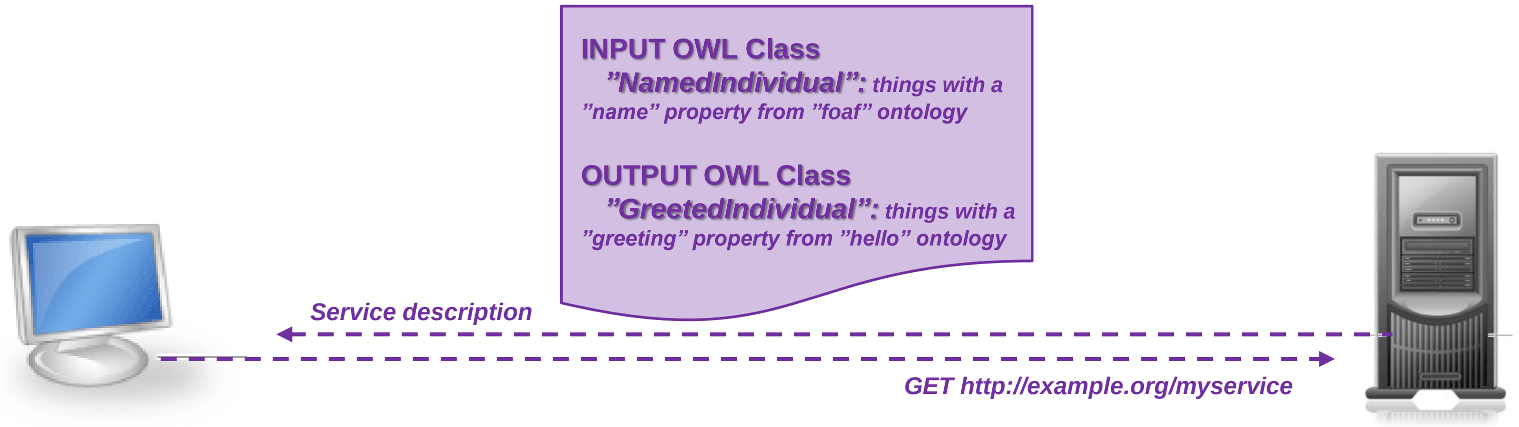
SADI simply consists of a number of recommendations for how services themselves should be implemented and described in order to achieve a set of useful, interoperable behaviors that can be leveraged by existing Web service standards:

- *SADI Web services are **stateless** and **atomic**;*
- *SADI service endpoints respond to **HTTP GET** by returning the interface definition of the service;*
- *Service interfaces (i.e., inputs and outputs) **are defined in terms of OWL-DL classes**; the property restrictions on these OWL classes define what specific data elements are required by the service and what data will be provided by the service, respectively;*
- *SADI services consume and produce data in **RDF format**;*
- *SADI services are invoked through plain **HTTP POST of RDF data** to the service endpoint. All information required for service invocation must be present in the data itself, because SADI uses a non-parameterized POST - i.e. does not use the HTTP FORM encoding;*
- *Input RDF data - data that is compliant with (i.e. classifies into) the input OWL Class definition – is “decorated” or “annotated” by the service provider to include new properties until it fulfills the Class definition of the service’s output OWL Class. Importantly, in so doing, the **URI of the input OWL Class Instance is preserved and becomes the URI of the output OWL Class Instance**.*

Links:
- *https://metacpan.org/dist/SADI-Simple*
- *https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3212890/*
- *https://jbiomedsem.biomedcentral.com/articles/10.1186/2041-1480-5-46*

Semantic Web Services

■ Service-to-Service (S2S) interaction:


INPUT OWL Class
"NamedIndividual": things with a "name" property from "foaf" ontology
OUTPUT OWL Class
"GreetedIndividual": things with a "greeting" property from "hello" ontology
Service description
GET http://example.org/myservice

SADI service description:

```
@prefix : <http://www.mygrid.org.uk/mygrid-moby-service#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix schema: <http://www.w3.org/2001/XMLSchema#> .
@prefix ser: <http://sadiframework.org/examples/hello#> .
@prefix h-ser: <http://sadiframework.org/examples/hello> .
@prefix h-ont: <http://sadiframework.org/examples/hello.owl#> .

h-ser: a :serviceDescription ;
    :hasOperation ser:operation ;
    :hasServiceDescriptionText "A simple Hello" ;
    :hasServiceNameText "Hello"^^schema:string .
ser:operation a :operation ;
    :inputParameter ser:input ;
    :outputParameter ser:output .
ser:input a :parameter;
    :objectType h-ont:NamedIndividual .
ser:output a :parameter;
    :objectType h-ont:GreetedIndividual .
```

Ontology to describe a service (hello.owl):

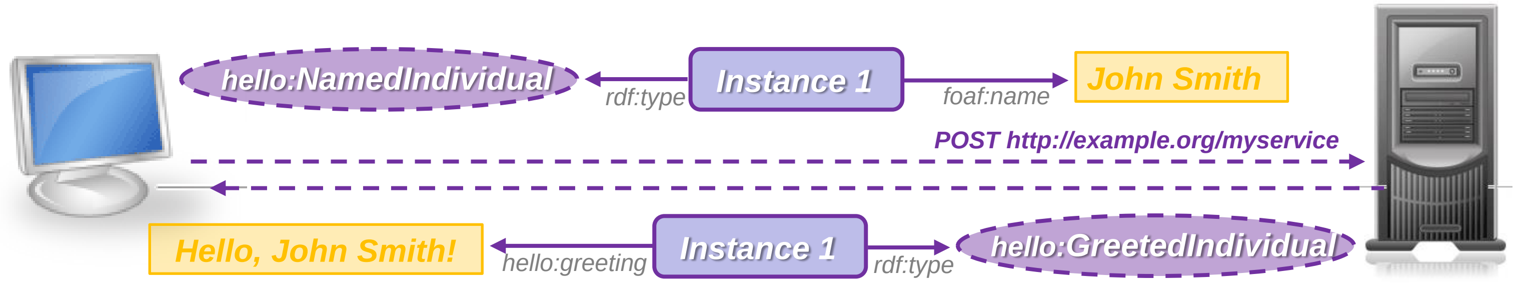
```
@prefix : <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix schema: <http://www.w3.org/2001/XMLSchema#> .

<> a :Ontology ;
   :imports <http://xmlns.com/foaf/0.1/name> .
<#NamedIndividual> a :Class ;
   :equivalentClass [
      a :Restriction ;
      :minCardinality "1"^^schema:int ;
      :onProperty <http://xmlns.com/foaf/0.1/name> ] .
<#GreetedIndividual> a :Class ;
   :equivalentClass [
      a :Restriction ;
      :minCardinality "1"^^schema:int ;
      :onProperty <#greeting> ] .
<#greeting> a :DatatypeProperty .
```

Semantic Web Services

- **Service-to-Service (S2S) interaction:**

Invocation HTTP message to invoke the "hello world" service:

```
POST /examples/hello HTTP/1.1
Host: sadiframework.org

<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:foaf="http://xmlns.com/foaf/0.1/"
   xmlns:hello="http://sadiframework.org/examples/hello.owl#">
   <hello:NamedIndividual rdf:about="http://sadiframework.org/examples/hello-input.rdf#1">
      <foaf:name>John Smith</foaf:name>
   </hello:NamedIndividual>
</rdf:RDF>
```



Response HTTP message sent in response to the invocation message:

```
HTTP/1.1 200 OK
Content-type: application/rdf+xml

<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:hello="http://sadiframework.org/examples/hello.owl#">
   <hello:GreetedIndividual rdf:about="http://sadiframework.org/examples/hello-input.rdf#1">
      <hello:greeting>Hello, John Smith!</hello:greeting>
   </hello:GreetedIndividual>
</rdf:RDF>
```

Semantic Web Services

- **Service-to-Service (S2S) interaction:**

Simple Semantic Web Architecture and Protocol SSWAP (pronounced "swap") is a lightweight, document-centric protocol and architecture for the Semantic Web [13][14].

It aims to combine web services and semantic web technologies to enable high-throughput discovery, assessment, and integration of data and services between distributed parties.

- *Using SSWAP, users can create systems to enable creation, discovery and execution of RESTful and other web services;*
- *SSWAP utilizes OWL ontologies to describe the features and capabilities of web services and standard HTTP methods to execute these web services;*
- *In SSWAP a service represents nothing more than a mapping from its inputs to its outputs;*
- *SSWAP ontology is not domain-specific and can be combined with arbitrary domain ontologies to represent the relevant types of services, inputs, or outputs.*

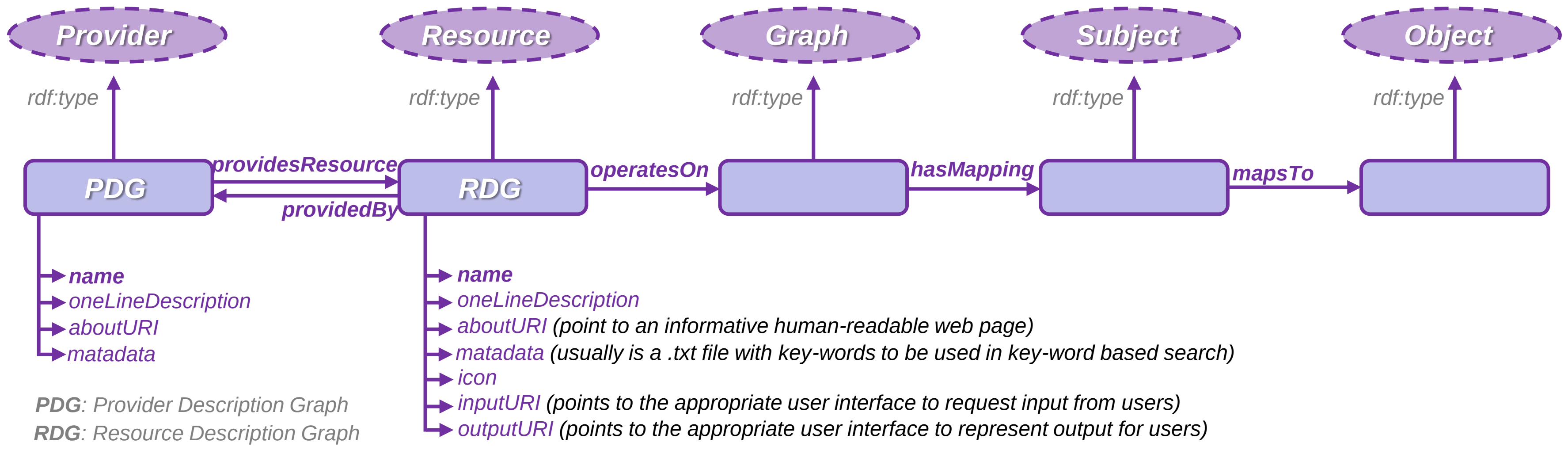
Semantic Web Services

- **Service-to-Service (S2S) interaction:**

SSWAP architecture is based on five basic concepts: Provider, Resource, Graph, Subject, and Object:

- *Provider. Providers typically correspond to organizations that own and publish resources;*
- *Resource. Resources can be arbitrary resources like web pages or ontologies or datasets, but they are primarily used to describe web services. The descriptions of resources are stored in documents that are referred as a Resource Description Graph (RDG);*
- *Graph. Graph concept describes transformation performed by the service.*
  - *It defines a mapping from a Subject (input) to an Object (output).*
  - *Multiple inputs and multiple outputs can be represented using multiple graphs or multiple mappings in a single graph, respectively.*
- *Subject and Object. Subject and Object are input and output respectively.*

SSWAP protocol: *https://sourceforge.net/p/sswap/wiki/protocol/*


Provider
Resource
Graph
Subject
Object
rdf:type
rdf:type
rdf:type
rdf:type
rdf:type
PDG
providesResource
providedBy
RDG
operatesOn
hasMapping
mapsTo
name
oneLineDescription
aboutURI
matadata
name
oneLineDescription
aboutURI (point to an informative human-readable web page)
matadata (usually is a .txt file with key-words to be used in key-word based search)
icon
inputURI (points to the appropriate user interface to request input from users)
outputURI (points to the appropriate user interface to represent output for users)
PDG: Provider Description Graph
RDG: Resource Description Graph

Semantic Web Services

Service-to-Service (S2S) interaction:

Resource Description Graph (RDG) is a document that stores the description of resources.

- *Providers publish their resource descriptions (RDGs) on the Web under resource URL;*
- *Clients can obtain these RDGs by dereferencing the URL for the resource;*
- *RDGs can be collected and stored by a discovery server that will provide a search service. A discovery server can locate RDGs either by crawling the Web or by allowing providers to submit the URL for their resources to the discovery server. The discovery server stores resource descriptions and any other domain ontology imported by the RDG in its knowledge base. In this way, SSWAP is similar to the Linked Open Data (LOD) approach, which also relies on URL-identified documents to contain machine processable information.*

*SSWAP's emphasis on a **canonical structured document** means that service description, querying for new services, service invocation, and service response all occur around the structure of the same, mutable RDF/XML document. This is distinct from traditional web service protocols (the way a service describes itself may say little about how to query and discover the service via a search engine, similarly, the way data is marshalled to invoke the service may have little connection to how the return data is packaged as a response).*

Resource Description Graph (RDG) example:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix sswap: <http://sswapmeet.sswap.info/sswap/> .
@prefix location: <http://www.example.org/locationOntology/> .
@prefix weather: <http://www.example.org/weatherOntology/> .
@prefix resource: <http://www.example.org/getTemperatureService/> .

resource:getTemperatureService
  rdf:type sswap:Resource ,
           weather:TemperatureInformationService;
  sswap:providedBy resource:resourceProvider ;
  sswap:name "Temperature lookup service" ;
  sswap:oneLineDescription "A service that accepts a
  location identified by a 5-digit zipcode or IP
  address and returns the current temperature in that
  location" ;
  sswap:operatesOn [
    rdf:type sswap:Graph ;
    sswap:hasMapping [
      rdf:type sswap:Subject, location:Location ;
      location:zipcode "" ;
      location:ipAddress "" ;
      sswap:mapsTo [
        rdf:type sswap:Object, weather:Temperature
      ]
    ]
  ].
```

Semantic Web Services

- **Service-to-Service (S2S) interaction:**

In *SSWAP* :

- *Performing an HTTP GET on a service's URL returns the service description - the RDG for that service.*
- *Removing the service's URL from the graph and sending the graph to a semantic search engine will query (i.e., discover) all similar services. This graph is called a Resource Query Graph (RQG).*
- *Adding input data to a SSWAP subject of the graph creates a graph ready for invocation which is called a Resource Invocation Graph (RIG). To invoke the service, the graph is POSTed to the service's URL or sent as a query string in an HTTP GET.*
- *The service's response is also embedded in the same graph. The response graphs is called a Resource Response Graph (RRG).*
- *A client may then simply remap the returned SSWAP object to another graph's SSWAP input and send it to another service, and so forth.*

The match between the service and the query is based on the semantic relations between the resource, subject, and object types. A discovery server will return all resources that are:

- *the same class or a subclass of the query graph's Resource. Ensures that results will contain all services that are tagged as at least as specific as requested (and maybe more specific);*
- *superclass of the query graph's Subject. Ensures that results will contain all services that operate on data that is guaranteed to subsume the input provided (and may operate on more general data);*
- *subclass of the query graph's Object. Ensures that results will contain all services that return data that is at least as specific as the requested output (and maybe more specific).*

Examples of RQG:

```
_:r rdf:type sswap:Resource, weather:TemperatureInformationService.
```

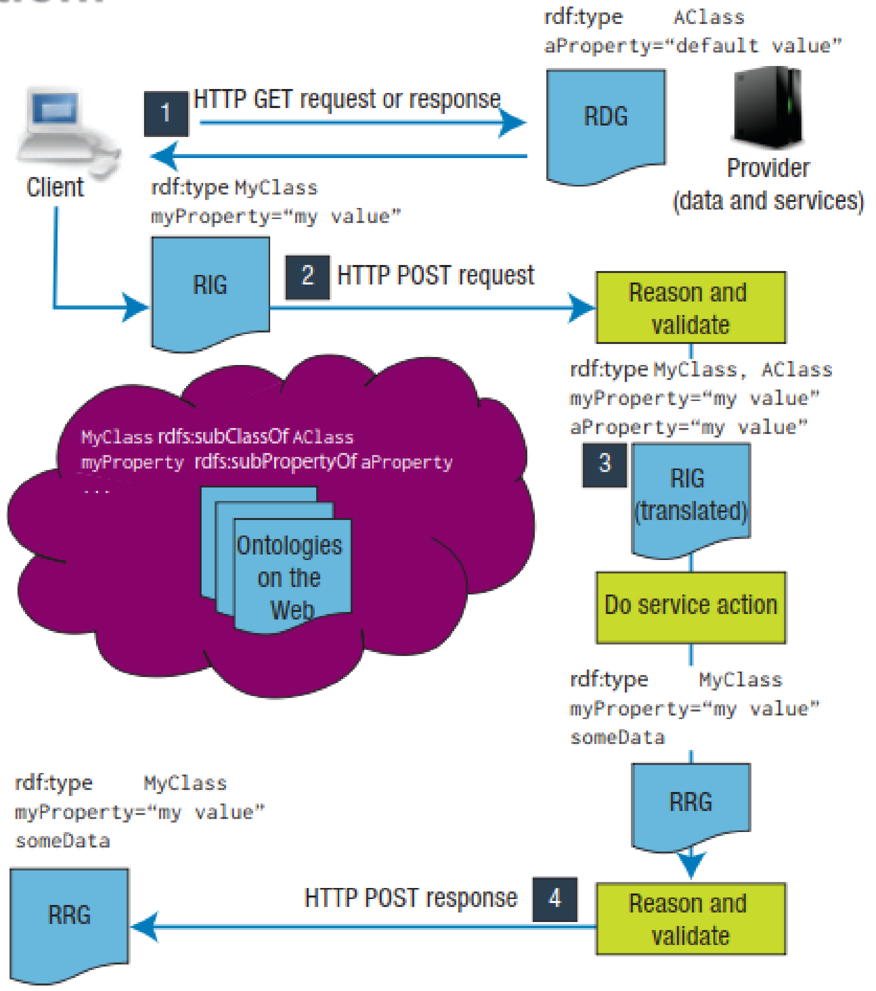
```
_:s rdf:type sswap:Subject, location:Location.
```

```
_:o rdf:type sswap:Object, weather:Temperature.
```

Semantic Web Services

- **Service-to-Service (S2S) interaction:**

USING LOOSELY COUPLED COMPUTATIONAL SEMANTICS FOR LOGIC-BASED SERVICE INTERFACES *[14]*



Semantic Web Services

■ Service-to-Service (S2S) interaction:

RDG:

```
…
resource:getTemperatureService
  rdf:type sswap:Resource ,
           weather:TemperatureInformationService;
  sswap:providedBy resource:resourceProvider ;
  sswap:name "Temperature lookup service" ;
  sswap:oneLineDescription "A service that accepts a location identified by a 5-
digit zipcode and returns the current temperature in that location" ;
  sswap:operatesOn [rdf:type sswap:Graph ;
                    sswap:hasMapping [rdf:type sswap:Subject,
                                               location:Location;
                                     location:zipcode "" ;
                                     location:ipAddress "" ;
                                     sswap:mapsTo [rdf:type sswap:Object,
                                                            weather:Temperature
                                                  ]
                                     ]
                    ].
```

RIG:

```
…
resource:getTemperatureService
  rdf:type sswap:Resource ,
           weather:TemperatureInformationService;
  sswap:providedBy resource:resourceProvider ;
  sswap:name "Temperature lookup service" ;
  sswap:oneLineDescription "A service that accepts a location identified by a 5-
digit zipcode and returns the current temperature in that location" ;
  sswap:operatesOn [rdf:type sswap:Graph ;
                    sswap:hasMapping [rdf:type sswap:Subject,
                                               location:Location;
                                     location:zipcode "40100" ;
                                     location:ipAddress "130.234.160.47" ;
                                     sswap:mapsTo [rdf:type sswap:Object,
                                                            weather:Temperature
                                                  ]
                                     ]
                    ].
```

RRG:

```
…
resource:getTemperatureService
  rdf:type sswap:Resource ,
           weather:TemperatureInformationService;
  sswap:providedBy resource:resourceProvider ;
  sswap:name "Temperature lookup service" ;
  sswap:oneLineDescription "A service that accepts a
location identified by a 5-digit zipcode and returns
the current temperature in that location" ;
  sswap:operatesOn [
    rdf:type sswap:Graph ;
    sswap:hasMapping [
      rdf:type sswap:Subject, location:Location ;
      location:zipcode "40100" ;
      location:ipAddress "130.234.160.47" ;
      sswap:mapsTo [
        rdf:type sswap:Object, weather:Temperature ;
        weather:value "30" ;
        weather:unit "celsius".
      ];
      sswap:mapsTo [
        rdf:type sswap:Object, weather:Temperature ;
        weather:value "86" ;
        weather:unit "fahrenheit"
      ]
    ]
  ].
```

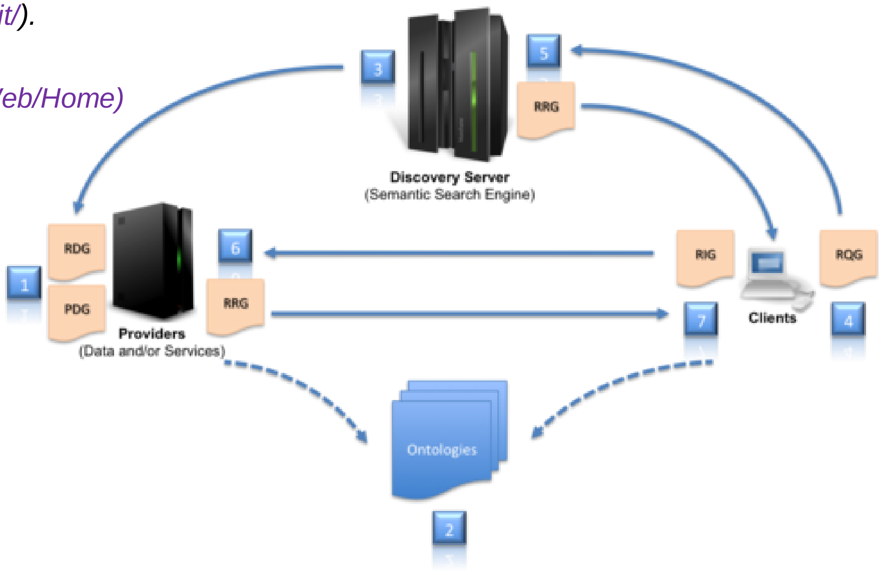
Semantic Web Services

Service-to-Service (S2S) interaction:

SSWAP web site (http://sswapmeet.sswap.info/) (https://sourceforge.net/p/sswap/wiki/protocol) presents iPlant's Semantic Web Platform that provides:

- *Discovering Semantic Web Services;*
- *Hosting Semantic Web Services;*
- *Several developer tools*
  (SSWAP Java API - http://sswap.iplantcollaborative.org/javadocs/ ;
  https://sourceforge.net/p/sswap/code/HEAD/tree/API-SDK/);
- *Platform to build pipelines of services;*
- *Semantic Pipeline RESTful API (https://sourceforge.net/p/sswap/wiki/api/;*
  https://pods.iplantcollaborative.org/wiki/display/SemanticWeb/Semantic+Pipeline+RESTful+API)
- *Semantic web service ontology portal (http://sswap.info/jit/).*

Wiki (https://sourceforge.net/p/sswap/wiki/Home/;
https://pods.iplantcollaborative.org/wiki/display/SemanticWeb/Home)



iPlant Semantic Web Service Architecture

Semantic Web Services

- **Service-to-Service (S2S) interaction:**

SSWAP Pipeline:

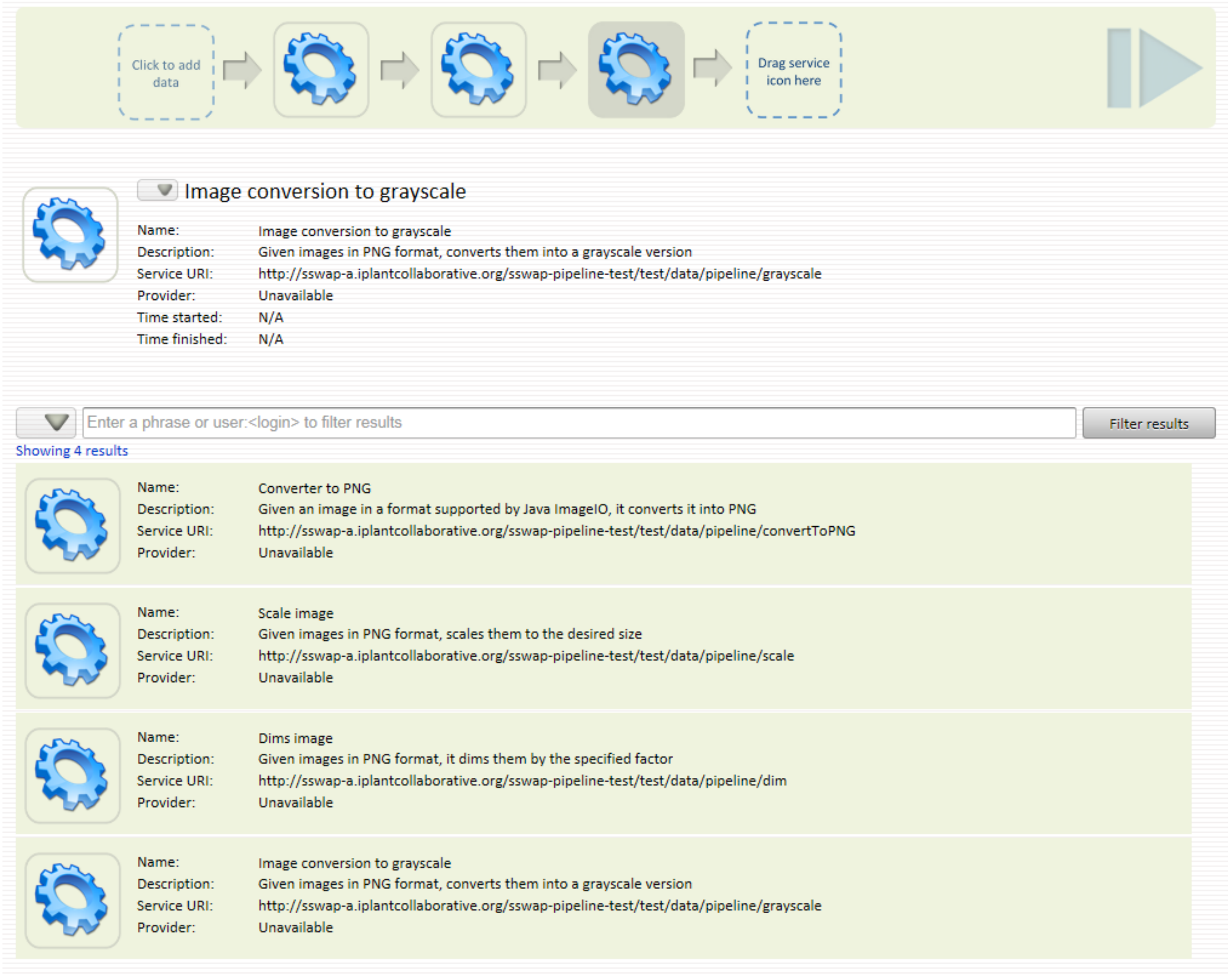


Image conversion to grayscale

Name:	Image conversion to grayscale
Description:	Given images in PNG format, converts them into a grayscale version
Service URI:	http://sswap-a.iplantcollaborative.org/sswap-pipeline-test/test/data/pipeline/grayscale
Provider:	Unavailable
Time started:	N/A
Time finished:	N/A

Enter a phrase or user:<login> to filter results

Filter results

Showing 4 results

Name:	Converter to PNG
Description:	Given an image in a format supported by Java ImageIO, it converts it into PNG
Service URI:	http://sswap-a.iplantcollaborative.org/sswap-pipeline-test/test/data/pipeline/convertToPNG
Provider:	Unavailable
Name:	Scale image
Description:	Given images in PNG format, scales them to the desired size
Service URI:	http://sswap-a.iplantcollaborative.org/sswap-pipeline-test/test/data/pipeline/scale
Provider:	Unavailable
Name:	Dims image
Description:	Given images in PNG format, it dims them by the specified factor
Service URI:	http://sswap-a.iplantcollaborative.org/sswap-pipeline-test/test/data/pipeline/dim
Provider:	Unavailable
Name:	Image conversion to grayscale
Description:	Given images in PNG format, converts them into a grayscale version
Service URI:	http://sswap-a.iplantcollaborative.org/sswap-pipeline-test/test/data/pipeline/grayscale
Provider:	Unavailable

Semantic Web Services

- **Service-to-Service (S2S) interaction:**

SSWAP Java API: Service Consumer side

- *RDG retrieving from service response to HTTP GET request.*

```
// execute HTTP GET request using service URL and use corresponding response to get RDG …
RDG rdg = null;
try {
   URI uri = new URI(serviceUrl);
   rdg = SSWAP.getResourceGraph(response.getEntity().getContent(), RDG.class, uri);
} catch (DataAccessException e1) {
   // TODO Auto-generated catch block
   e1.printStackTrace();
} catch (IllegalStateException e1) {
   // TODO Auto-generated catch block
   e1.printStackTrace();
} catch (URISyntaxException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
} catch (IOException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
}

SSWAPResource resource = rdg.getResource();
System.out.println("Resource name: " + resource.getName());
System.out.println("Resource oneline description: " + resource.getOneLineDescription());
SSWAPGraph graph = resource.getGraph();
SSWAPSubject subject = graph.getSubject();
```

Semantic Web Services

- **Service-to-Service (S2S) interaction:**

SSWAP Java API: Service Consumer side

- *RIG creation through assigning properties' values of graph Subject;*
- *RRG retrieving via RIG invocation.*

```
//SSWAPResource resource = rdg.getResource();
//SSWAPSubject subject = graph.getSubject();
…
Iterator<SSWAPProperty> iterator = subject.getProperties().iterator();
int i=0;
while (iterator.hasNext()) {
   SSWAPProperty property = iterator.next();
   SSWAPPredicate predicate = rdg.getPredicate(property.getURI());
   i++;
   subject.setProperty(predicate, "value"+i);
}

SSWAPGraph graph = resource.getGraph();
graph.setSubject(subject);
resource.setGraph(graph);

RIG rig = resource.getRDG().getRIG();
HTTPProvider.RRGResponse response = rig.invoke();
RRG rrg = response.getRRG();
```

Semantic Web Services

Service-to-Service (S2S) interaction:

SSWAP Java API: Service Consumer side

- *Retrieving Objects' properties from RRG.*

```
SSWAPResource resource = rrg.getResource();
SSWAPGraph graph = resource.getGraph();
SSWAPSubject subject = graph.getSubject();

Iterator<SSWAPObject> iteratorObjects =  subject.getObjects().iterator();
while (iteratorObjects.hasNext()) {
   SSWAPObject object = iteratorObjects.next();
   Iterator<SSWAPProperty> iteratorProperties = object.getProperties().iterator();
   while (iteratorProperties.hasNext()) {
      SSWAPProperty property = iteratorProperties.next();
      SSWAPPredicate predicate = rrg.getPredicate(property.getURI());
      String lookupName = getStrName(property.getURI());
      String lookupValue = getStrValue(object,predicate);
      System.out.println(""+lookupName+" : "+lookupValue);
   }
}
```

```
private String getStrValue(SSWAPIndividual sswapIndividual, SSWAPPredicate sswapPredicate) {
   String value = null;
   SSWAPProperty sswapProperty = sswapIndividual.getProperty(sswapPredicate);
   if ( sswapProperty != null ) {
             value = sswapProperty.getValue().asString();
             if ( value.isEmpty() ) { value = null; }
   } return value; }
private String getStrName(URI uri) {
   String[] parts = uri.toString().split("#");
   return parts[1]; }
```

Semantic Web Services

- **Service-to-Service (S2S) interaction:**

SSWAP Java API: Service side

- *Create your servlet class as an extension of SimpleSSWAPServlet.*

```
/**
 * Servlet implementation class MySSWAPServlet
 */
@WebServlet("/MySSWAPServlet")
public class MySSWAPServlet extends SimpleSSWAPServlet{
           private static final long serialVersionUID = 1L;

           @Override
           public void init(ServletConfig servletConfig) throws ServletException {
                      // always do this
                      super.init(servletConfig);
                      // do anything else here that needs to be done once, on servlet load
           }
           /*
            * We can override this method to connect this servlet with the
            * service class, or define the init-param 'ServiceClass' in web.xml
            *
            * @see info.sswap.api.servlet.SimpleSSWAPServlet#getServiceClass()
            */
           @SuppressWarnings("unchecked")
           @Override
           public <T> Class<T> getServiceClass() {
                      return (Class<T>) SSWAPService.class;
           }
}
```

Semantic Web Services

- **Service-to-Service (S2S) interaction:**

SSWAP Java API: Service side

- *Define location of RDG file in web.xml.*

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
id="WebApp_ID" version="3.0">
  <display-name>MySSWAPService</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
    <servlet-name>My SSWAP service</servlet-name>
    <servlet-class>mySSWAPService.MySSWAPServlet</servlet-class>
    <init-param>
      <param-name>RDGPath</param-name>
      <param-value>/res/mySSWAPServiceRDG</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>My SSWAP service</servlet-name>
    <url-pattern>/getService/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Semantic Web Services

■ Service-to-Service (S2S) interaction:

RDG:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:resource="http://localhost:8080/MySSWAPService/"
          xmlns:sswap="http://sswapmeet.sswap.info/sswap/"
          …
          xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<sswap:Resource rdf:about="http://localhost:8080/MySSWAPService/getService">
    <rdf:type rdf:resource="http://localhost:8080/MySSWAPService/res/mySSWAPServiceOntology.owl#TemperatureInformationService" />
        <sswap:providedBy rdf:resource="http://localhost:8080/MySSWAPService/res/resourceProvider" />
        <sswap:name>Temperature lookup service</sswap:name>
        <sswap:oneLineDescription>A service that accepts a location identified by a 5-digit zipcode and returns the current temperature in that location
                                                                                                    </sswap:oneLineDescription>
        <sswap:operatesOn>
            <sswap:Graph>
                <sswap:hasMapping>
                    <sswap:Subject>
                        <rdf:type rdf:resource="http://localhost:8080/MySSWAPService/res/mySSWAPServiceOntology.owl#Location" />
                        <ont:zipcode></ont:zipcode >
                        <ont:ipAddress></ont:ipAddress>
                        <sswap:mapsTo>
                            <sswap:Object>
                                <rdf:type rdf:resource="http://localhost:8080/MySSWAPService/res/mySSWAPServiceOntology.owl#Temperature" />
                                <ont:value></ont:value>
                                <ont:unit></ont:unit>
                            </sswap:Object>
                        </sswap:mapsTo>
                    </sswap:Subject>
                </sswap:hasMapping>
            </sswap:Graph>
        </sswap:operatesOn>
    </sswap:Resource>
</rdf:RDF>
```

Semantic Web Services

- **Service-to-Service (S2S) interaction:**

SSWAP Java API: Service side

- *SSWAPService class extends MapsTo.*

```
public class SSWAPService extends MapsTo {
        // initialize variables here

        @Override
        protected void initializeRequest(RIG rig) {
                rigGraph = rig;
                // if we need to check service parameters we could start here
        }
        @Override
        protected void mapsTo(SSWAPSubject translatedSubject) throws Exception {
                subject = translatedSubject;
                object = translatedSubject.getObject();
                // check an input (Subject) and fill an output (Object) through the
                //logic of the service
        }
}
```

Semantic Web Services

- **Service-to-Service (S2S) interaction:**

SSWAP Java API: Service side

- *Create extra Object in case of several outputs.*

```
/**
 * Creating new empty object result...
 */
SSWAPObject sswapObject = null;
sswapObject = assignObject(subject);
Iterator<SSWAPProperty> iterator = object.getProperties().iterator();
while (iterator.hasNext()) {
           SSWAPProperty property = iterator.next();
           SSWAPPredicate predicate = rigGraph.getPredicate(property.getURI());
           sswapObject.addProperty(predicate, "");
}
subject.addObject(sswapObject);

// fill the Object
```

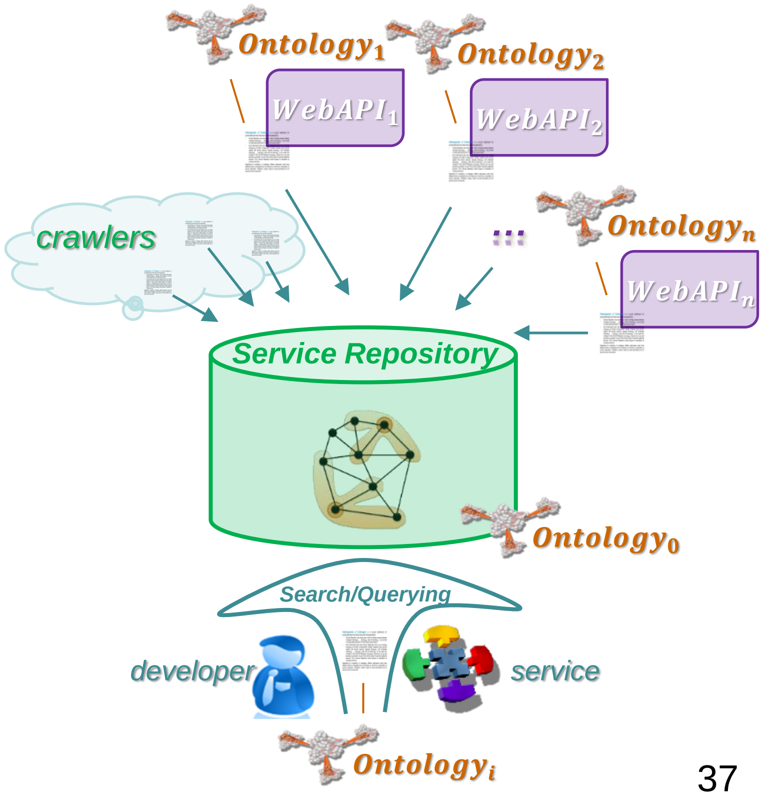
Semantic Web Services

- **Service-to-Service (S2S) interaction:**
  - *Registry: Service registry might be performed by service provider (through registry of service description to a Service Repository) or by crawlers that search for service descriptions in the Web.*
  - *Search/Querying: Search can be done by human (developer) as well as by other web service via querying correspondent service description. In case of human-driven search, Service Repository should provide correspondent template of service description and supportive tools for ontology browsing.*
  - *Instance matching: Service Repository does match of the queried service description with the descriptions of registered services by matching correspondent properties of the services. Some reasoning might be applied.*

Heterogeneity of Ontologies is a crucial bottleneck for automated service discovery and interoperability.

- *Service Repository may simply play a role of ontology mapper between ontologies ($\mathit{Ontology}_1$ … $\mathit{Ontology}_n$) and the $\mathit{Ontology}_i$; and provide correspondent alignment for further data transformation.*
- *More reasonable case when Service Repository has its own ontology $\mathit{Ontology}_0$ and does correspondent ontology mapping during service registry and service querying: mapping $\mathit{Ontology}_0$ with ontologies ($\mathit{Ontology}_1$ … $\mathit{Ontology}_n$) and with the $\mathit{Ontology}_i$. In this case own ontology of the Service Repository $\mathit{Ontology}_0$ should be as big and general as possible to cover all the domain areas of possibly registered services. Thus, Service Repository should support a mechanism of ontology extension.*

Depending on complexity of ontologies, different alignments might have different levels of probability that will influence on the level of automation of service interaction. Therefore, process might be semi-automated and will require human involvement.



References

All the links in the lecture materials:

- [1] Hakimpour F, Cong S, Damm D. E., ”A Practical Tutorial on Semantic Web Services”, URL: http://www.academia.edu/331267/A_Practical_Tutorial_on_Semantic_Web_Services
- [2] Kopecky J, Gomadam K. and Vitvar T. (2008). hRESTS: An HTML microformat for describing RESTful web services. In: 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, 9-12 Dec 2008, Sydney, Australia.
- [3] Kopeck J, Vitvar T, Fensel D, ”MicroWSMO and hRESTS”. Deliverable D3.4.3 of the FP7 Project ” Service Oriented Architectures for All (SOA4ALL)”. URL: http://sweet.kmi.open.ac.uk/pub/microWSMO.pdf
- [4] Jorge Cardoso and Carlos Pedrinaci, Evolution and Overview of Linked USDL. 6th International Conference Exploring Services Science, IESS 2015, Porto, Portugal, February 4-6, 2015, LNBIP, Vol. 201, Novoa, Henriqueta, Dragoicea, Monica (Eds.), 2015.
- [5] J. Cardoso, A. Barros, N. May, and U. Kylau. Towards a unified service description language for the internet of services: Requirements and first developments. In IEEE International Conference on Services Computing (SCC), pages 602 {609, July 2010.
- [6] Carlos Pedrinaci, Jorge Cardoso, and Torsten Leidig. Linked USDL: A vocabulary for web-scale service trading. In 11th Ext. Semantic Web Conference, Greece, 2014.
- [7] Andrade L. J. S., Prazeres C. V. S. (2015). LDaaSWS: Toward Linked Data as a Semantic Web Service. In Proceedings of the Tenth International Conference on Internet and Web Applications and Services (ICIW 2015), Brussels, Belgium, 2015, p. 56-62.
- [8] C. Bizer, T. Heath, and T. Berners-Lee, “Linked data - the story so far,” Int. Journal on Semantic Web and Information Systems (IJSWIS), 2009.
- [9] C. Pedrinaci, D. Liu, M. Maleshkova, D. Lambert, J. Kopecky, and J. Domingue, “iServe: a linked services publishing platform,” in CEUR Workshop Proceedings, vol. 596, 2010.
- [10] Rodriguez-Mier, P., Pedrinaci, C., Lama, M. and Mucientes, M. (2015) An Integrated Semantic Web Service Discovery and Composition Framework, IEEE Transactions on Services Computing
- [11] Ning Li, Carlos Pedrinaci, Maria Maleshkova, Jacek Kopecky, and John Domingue. OmniVoke: a framework for automating the invocation of Web APIs. In ICSC 2011 Fifth IEEE International Conference on Semantic Computing, 2011.

References

All the links in the lecture materials:

- [12] Wilkinson M. D., Vandervalk B. and McCarthy L. (2011). The Semantic Automated Discovery and Integration (SADI) Web service Design-Pattern, API and Reference Implementation. Journal of Biomedical Semantics, 2011, v2, n8.
- [13] Gessler DDG, Schiltz GS, May GD, Avraham S, Town CD, Grant D, Nelson RT 2009. SSWAP: A Simple Semantic Web Architecture and Protocol for semantic web services. BMC Bioinformatics, 10:309 doi:10.1186/1471-2105-10-309 (URL: http://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-10-309).
- [14] SSWAP: Enabling Transaction-Time Reasoning for Semantic Workflows. URL: http://ieeexplore.ieee.org/document/7274418/
- [15] Ly, Papa Alioune; Pedrinaci, Carlos and Domingue, John (2012). Automated information extraction from web APIs documentation. In: The 13th International Conference on Web Information System Engineering (WISE 2012), 28-30 November 2012, Paphos, Cyprus, pp. 497–511.

Relevant practical links:

- Java Servlets: http://www.tutorialspoint.com/servlets/servlets_tutorial.pdf, http://tutorials.jenkov.com/java-servlets/index.html
- AJAX: https://developer.mozilla.org/en-US/docs/AJAX/Getting_Started, http://www.w3schools.com/ajax/
- XSPARQL Tutorial: http://www.slideshare.net/net2-project/xsparql-tutorial, http://xsparql.deri.org
- REST Tutorial: http://rest.elkstein.org