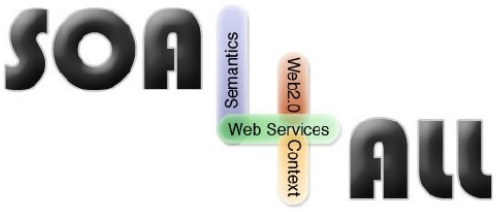

Project Number: **215219**
Project Acronym: **SOA4All**
Project Title: **Service Oriented Architectures for All**
Instrument: **Integrated Project**
Thematic Priority: **Information and Communication Technologies**

D3.4.3
MicroWSMO and hRESTS

Activity:	Activity 2 — Core R&D Activities	
Work Package:	WP3 — Service Annotation and Reasoning	
Due Date:		M12
Submission Date:		10/3/2009
Start Date of Project:		01/03/2008
Duration of Project:		36 Months
Organisation Responsible for Deliverable:		UIBK
Revision:		1.0
Author(s):	Jacek Kopecký	UIBK
	Tomas Vitvar	UIBK
	Dieter Fensel	UIBK
Reviewer(s):	Maria Maleshkova	OU
	Barry Bishop	UIBK

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

Version History

Version	Date	Comments, Changes, Status	Authors, Contributors, Reviewers
0.9	3/2/2009	CMS WG draft	All authors
1.0	9/3/2009	Final version after internal reviews	Jacek Kopecký (UIBK), Maria Maleshkova (OU), Barry Bishop (UIBK)

TABLE OF CONTENTS

LIST OF FIGURES

GLOSSARY OF ACRONYMS

Acronym	Definition
API	Application Programming Interface
D	Deliverable
EC	European Commission
GRDDL	Gleaning Resource Descriptions from Dialects of Languages
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
ID	Identifier
JSON	JavaScript Object Notation
OWL	Web Ontology Language
RDFS	RDF Schema
RDF	Resource Description Framework
REST	Representational State Transfer
SAWSDL	Semantic Annotations for WSDL and XML Schema
SWS	Semantic Web Services
URI	Uniform Resource Identifier
WADL	Web Application Description Language
WSDL	Web Services Description Language
WSMO	Web Service Modeling Ontology
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language
XSLT	XSL Transformations

1 EXECUTIVE SUMMARY

The Web 2.0 wave brings, among other aspects, the Programmable Web: increasing numbers of Web sites provide machine-oriented APIs and Web services. However, most APIs are only described with text in HTML documents. The lack of machine-readable API descriptions affects the feasibility of semantic annotation and the application of Semantic Web Service automation technologies. This deliverable describes MicroWSMO, a semantic annotation mechanism for RESTful Web services, based on a microformat called hRESTS (HTML for RESTful Services) for machine-readable descriptions of Web APIs, and backed by a simple service model.

Since there is no generally accepted machine-processable description language for RESTful Web services, the deliverable contains a definition of the hRESTS microformat which serves as a rough equivalent to WSDL. hRESTS naturally ties into the WSMO-Lite minimal RDF model from D3.4.2.

On top of hRESTS, this deliverable also defines the MicroWSMO microformat that adds SAWSDL-like annotations to hRESTS service descriptions. In effect, this deliverable provides a parallel to the stack of WSDL and SAWSDL, only aimed at RESTful services.

On hRESTS and MicroWSMO, we can apply WSMO-Lite service semantics and thus integrate RESTful services with WSDL-based ones. Most SOA4All components need not even distinguish between service descriptions that are in WSDL/SAWSDL or in hRESTS/MicroWSMO because both kinds are effectively WSMO-Lite.

2 INTRODUCTION

The Web has gone through great changes since it became popular, evolving from an infrastructure for static content of pages consumed by individuals to a communication platform where individuals, companies, and devices alike provide, consume and synthesize content and services on a massive scale. The value of Web applications is no longer only in providing content to consumers but also in exposing functionality through increasing numbers of public APIs designed for machine consumption. Both Web applications and APIs follow the Web architecture style called REST (Representational State Transfer [1]), and public APIs on the Web are often called "RESTful Web services".

Web application APIs are generally described using plain, unstructured HTML documentation useful only to a human developer. Finding suitable services, composing them ("mashing them up"), mediating between different data formats etc. are currently completely manual tasks. In order to provide tool support or even a degree of automation, we need the API descriptions to be machine-readable.

An "adaptation of semantic XHTML that makes it easier to publish, index, and extract semi-structured information", called *microformats* [6], is an approach for annotating mainly human-oriented Web pages so that key information is machine-readable. On top of microformats, GRDDL [2] is a mechanism for extracting RDF information from Web pages, particularly suitable for processing microformats. There are already microformats for contact information, calendar events, ratings etc.

In this deliverable, we define a microformat called HTML for RESTful Services, in short hRESTS, for machine-readable descriptions of Web APIs, backed by a simple service model in RDF. As depicted in Figure 2.1, the hRESTS microformat captures machine-processable service descriptions, building on the HTML service documentation aimed at developers. We further define MicroWSMO, an extension of hRESTS that adds means for semantic Web service automation. hRESTS can support other extensions as well, such as SA-REST [10], intended for enabling tool support, especially faceted browsing and discovery of services by client developers.

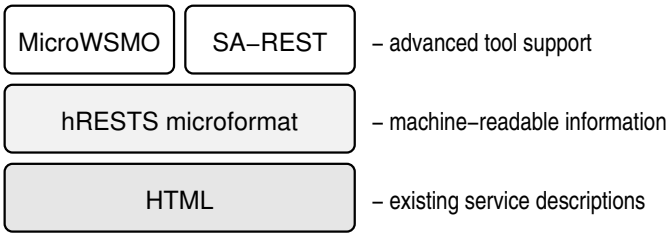

Figure 2.1: MicroWSMO in hRESTS layer cake

A recent W3C Recommendation called RDFa [7] specifies a mechanism for embedding RDF data in HTML. RDFa shares some of its use cases with microformats, but with different design principles: where microformats aim to be especially easy to use for Web content authors, RDFa is better prepared for proliferation of data vocabularies. Because our microformats (hRESTS and its extension MicroWSMO) are based on RDF models, we also discuss how RDFa can be used in lieu of microformats to express the machine-processable descriptions of RESTful services.

2.1 Alignment with SOA4All Architecture and Use Cases

MicroWSMO is the description language that will be used in SOA4All for semantic descriptions of RESTful Web services. These descriptions will be stored in a service registry, represented in RDF according to the model defined WSMO-Lite and extended in Section 4.1, and they will be used by the semantic automation components.

The use cases deal with Web services, both WSDL-based and RESTful. MicroWSMO will be used by the use cases to describe the RESTful services, combined with the WSMO-Lite service ontology.

2.2 Structure of the deliverable

The remainder of this deliverable is structured as follows: Section 3 introduces an example service/API that we use for demonstrating hRESTS and MicroWSMO. Section 4 defines hRESTS, our microformat for machine-readable service descriptions, and it also discusses the use of RDFa in lieu of the microformat. Section 5 adds MicroWSMO, an extension of hRESTS towards support for semantic automation. In Section 6, we describe an openly available XSLT implementation of a parser for our microformats. Finally, in Section 7, we conclude with a discussion of some related work and future plans.

3 Example RESTful Web Service

Web APIs and RESTful Web services are hypermedia applications consisting of interlinked resources (Web pages) that are oriented towards machine consumption. In their structure and behavior, RESTful Web services can be very much like common Web sites [8]. Both common Web sites and RESTful Web services use HTTP [5] as the communication protocol. The orientation of RESTful Web services towards machine consumption manifests mainly in the data formats: clients generally interact with RESTful services by sending structured data (e.g. XML, JSON[1]), as opposed to the standard Web document markup language, HTML, which is a human-oriented presentation language.

In this section, we introduce an example RESTful Web service. First, in Section 3.1 we describe the service in terms of its hypertext structure. In Section 3.2 we show that we can view the service as a set of operations, independent from its hypertext structure. Finally, in Section 3.3, we show how our example service would typically be described.

3.1 Example Service as Hypertext

Figure 3.1 illustrates an example RESTful hotel booking service, with its resources and the links among them. Together, all these resources form the hotel booking service; however, the involved Web technologies actually work on the level of resources, so *service* is a virtual term here and the figure shows it as a dashed box.

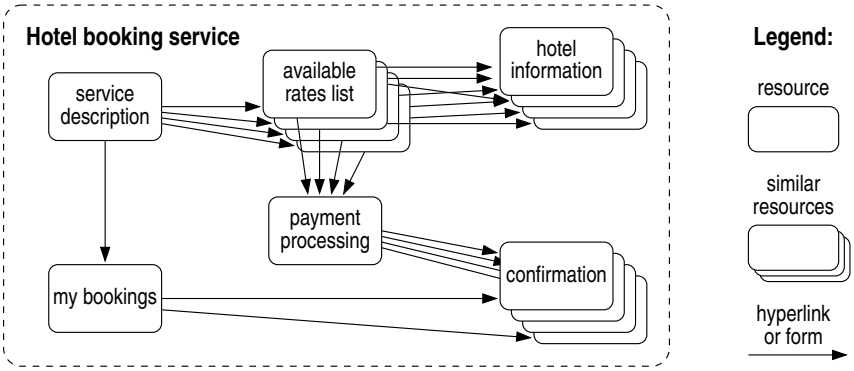

Figure 3.1: Structure of an example service

The “service description” is a resource with a stable address and information about the other resources that make up the service. It serves as the initial entry point for client interaction. In a human-oriented Web application, this would be the homepage, such as `http://hotels.example.com/`.

The existence of such a stable entry point lowers the coupling between the service and its clients, and it enables the evolution of the service, such as adding or removing functionality. A client need only rely on the existence of the fixed entry point, and it can discover all other functionality as it navigates the hypermedia. In contrast, in service-description-driven distributed computing technologies, such as WSDL-based Web services, the client is often programmed against a given service description before it uses the service, making it harder to react dynamically to changes of the service.

[1] JavaScript Object Notation, see `http://www.json.org/`

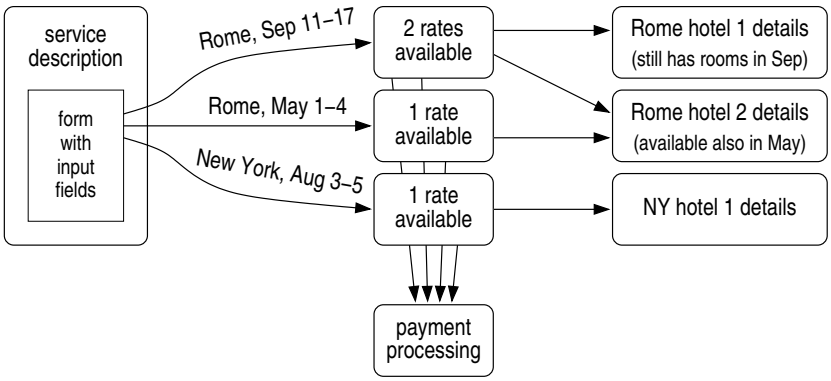

Figure 3.2: Detail of example service resources

The service description resource of our example service contains a form for searching for available hotels, given the number of guests, the start and end dates and the location. The search form serves as a parametrized hyperlink to search results resources that list the available rates, as detailed in Figure 3.2; one resource per every unique combination of the input data. The form prescribes how to create a URI that contains the input data; the URI then identifies a resource that returns the list of available hotels and rates. As there is a large number of possible search queries, there is also a large number of results resources, and the client does not need to know that all these resources are likely handled by a single software component on the server.

The search results are modeled as separate resources (as opposed to, for instance, a single data-handling resource that takes the inputs in a request message), because it simplifies the reuse of the hotel search functionality in other services or in mashups (lightweight compositions of Web applications), and it also supports caching of the results. With individual search results resources, creating the appropriate URI and retrieving the results (with HTTP GET) is easier in most programming frameworks than POSTing the input data in a structured data format to one Web resource, which would then reply with the list of available hotels and rates.

Search results are presented as a list of concrete rates available at the hotels in the given location, for the given dates and the number of guests, as also shown in Figure 3.2. Each item of the list contains a link to further information about the hotel (e.g. the precise location, star rating, guest reviews and other descriptions), and a form for booking the rate, which takes as input the payment details (such as credit card information) and an identification of the guest(s) who will stay in the room. The booking data is submitted (POSTed) to a payment resource, which processes the booking and redirects the client to a confirmation resource. The content of the confirmation can serve as a receipt.

The service description resource also contains a link to “my bookings”, a resource listing the bookings of the current user (which requires authentication functionality). This resource links to the confirmations of the bookings done by the authenticated user. With such a resource available to them, client applications no longer need to store the information about performed bookings locally.

The confirmation resources may further provide a way of canceling the reservation (not shown in the picture, could be implemented with the HTTP DELETE method).

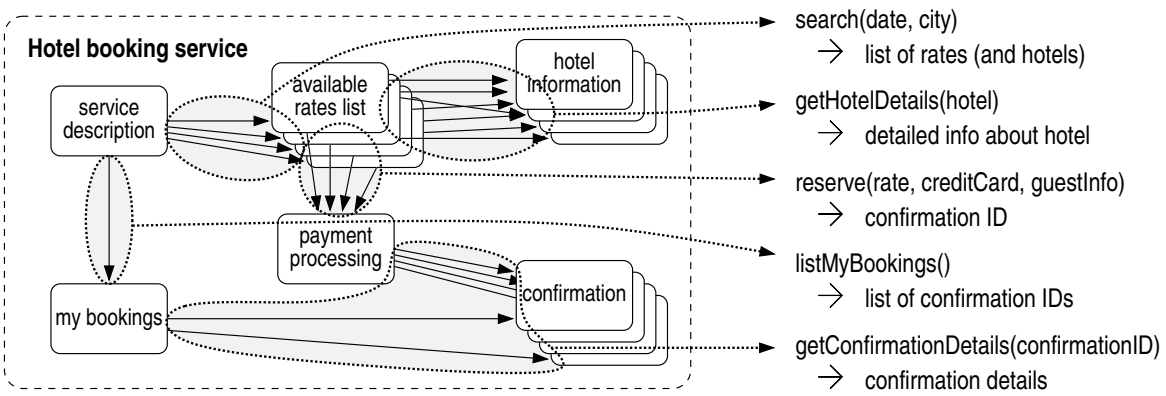

Figure 3.3: Operations of the example service

3.2 Turning Hypertext into Operations

So far, our description of the example hotel reservation service has focused on the hypermedia aspect: we described the resources and how they link to each other. Alternatively, and in fact more commonly, we can also view the service as a set of operations available to the clients — as an API.

The resources of the service (the *nouns*) form a hypermedia graph (shown in Fig. 3.1). The interaction of a client with a RESTful service is a series of operations (the *verbs* or *actions*) where the client sends a request to a resource and receives a response that may link to further useful resources. Importantly, the links need not be only simple URIs, but they can also be *input forms* that indicate the URI, the HTTP method, and the input data.

The graph nature of a hypermedia service guides the sequence of operation invocations, but the meaning of a resource is independent of where it is linked from; the same link or form, wherever it is placed, will always lead to the same action. Therefore, the operations of a RESTful Web service can be considered independently from the graph structure of the hypertext.

In Figure 3.3, we extract the operations present in our example service. The search form in the homepage represents a search operation, the hotel information pages linked from the search results can be viewed as an operation for retrieving hotel details, the reservation form for any particular available rate becomes a reservation operation, and so on.

3.3 HTML Description of the Example Service

Web APIs, or indeed services of any kind, need to be described in some way, so that potential clients can know how to interact with them. While Web applications are self-describing to their human users, Web services are designed for machine consumption, and someone has to tell the machine how to consume any particular service.

Public RESTful Web services are universally described in human-oriented documentation[2] using the general-purpose Web hypertext language HTML, which is the medium of choice for dissemination of information about Web APIs, along with a vast majority of other textual content.

[2]For instance, see `flickr.com/services/api` and `docs.amazonwebservices.com/AmazonSimpleDB/2007-11-07/DeveloperGuide`

```
1  <h1>ACME Hotels service API</h1>
2  <h2>Operation <code>getHotelDetails</code></h2>
3
4  <p> Invoked using the method GET at http://example.com/h/{id} <br/>
5    <strong>Parameters:</strong> <code>id</code> – the identifier of the particular hotel
6    <br/>
7    <strong>Output value:</strong> hotel details in an
8    <code>ex:hotelInformation</code> document
9  </p>
```

Listing 3.1: Example HTML service description

Typically, such documentation will list the available operations (calling them API calls, methods, commands etc.), their URIs and parameters, the expected output and error conditions and so on; it is, after all, intended as the documentation of a programmatic interface.

The following might be an excerpt of a typical operation description:

> ACME Hotels service API
> **Operation** `getHotelDetails`
>
> Invoked using the method GET at `http://example.com/h/{id}`
> **Parameters:** `id` - the identifier of the particular hotel
> **Output value:** hotel details in an `ex:hotelInformation` document

In HTML, the description can be captured as shown in Listing 3.1. Such documentation has all the details necessary for a human to be able to create a client program that can use the service. In order to tease out these technical details, the textual documentation needs to be amended in some way, such as with our hRESTS microformat, shown in the following section.

In the hypertext of the example service, the service has five operations but only two are directly accessible from the service description resource. All five operations can be described in a single HTML document, however, the client may not know any concrete hotel identifiers to invoke getHotelDetails(), or any confirmation ID to invoke getCofirmationDetails(). The client may save hotel or confirmation identifiers and use them later to invoke these operations without going through availability searches or the list of “my bookings”; this behavior is equivalent to how bookmarks work in a Web browser.

4 hRESTS: HTML FOR RESTful WEB SERVICES

We have seen that a RESTful Web service can be viewed as a hypertext graph of interlinked resources, or as a set of operations to be invoked by the client. While navigating the hypertext graph is natural for the human users, programmatic clients deal rather with the operations, even though they can use the links in the response messages.

In this section, we specify hRESTS, a microformat that can be used to structure existing RESTful Web service documentation so that key pieces of information are machine-processable. This microformat serves as the basis for extensions that introduce additional information (in the form of annotations), such as MicroWSMO (see Section 5) and SA-REST [10].

We start in Section 4.1 by identifying the key pieces of technical information that are present in the textual documentation, from which we form an RDF model for descriptions of RESTful services. In Section 4.2, we define the syntax of the hRESTS microformat, and Section 4.3 shows how hRESTS information can be marked up in HTML documentation using RDFa in lieu of our microformat.

4.1 Minimal Service Model

The interaction of a client with a RESTful *service* such as the one in our example is a series of *operations* where the client sends a *request* to a *resource* (using one of the HTTP *methods* GET, POST, PUT or DELETE), and receives a *response* that may *link* to further useful resources. The emphasized words indicate the key concepts that the clients encounters:

- *service* is the service (a set of related resources) that the client deals with,
- *operation* is a single action that the client can perform on that service,
- *resource* determines the address (URI) where the operation is invoked,
- *method* captures the HTTP method for the operation,
- *request* and *response* are the messages sent as input and output of the operation,
- *links*, especially in the output messages, make up a run-time hypertext graph of related resources.

This leads us to a service model shown in Figure 4.1. A Web service has a number of operations, each with potential inputs and outputs, and a hypertext graph structure where

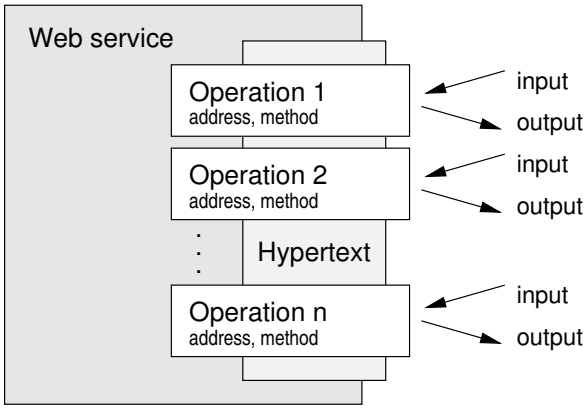

Figure 4.1: Functional model of a RESTful Web service

the outputs of one operation may link to other operations. This model captures the requirements for what we need to represent in a machine-readable description. Unsurprisingly, the model is very similar in its structure to WSDL [13], only instead of hypertext, WSDL services use the terms “process” or “choreography” for the sequencing of operations. More importantly, this model naturally builds on the minimal service model of WSMO-Lite [12].

An operation description specifies an address (a URI or a parametrized URI template[1]), the HTTP method (GET, POST, PUT or DELETE), and the input and output[2] data formats. In principle, the output data format can be self-describing (self-description is a major part of Web architecture), but the API documentation should specify what the client can expect.

While at runtime the client interacts with concrete resources, the service description may present a single operation that acts on many resources (e.g. getHotelDetails() which can be invoked on any hotel details resource), therefore an operation specifies an *address* as a URI template whose parameters are part of the input data.

Listing 4.1 shows an RDFS realization of this service model, together with the operation properties described above. Services, their operations, and messages can also have human-readable names, which can be attached using the `rdfs:label` property. Note that we reuse the WSMO-Lite minimal RDF service model vocabulary which captures the WSDL components; from the point of view of this minimal model, hRESTS is roughly equivalent to WSDL.

Once a machine-readable description of a Web service is available, it can be further annotated with additional information, such as semantic descriptions (the functionality of operations, the meaning of the input and output data), or nonfunctional properties (e.g., the price of using the service, QoS guarantees, security and privacy policies). Such annotations extend the utility of service descriptions.

4.2 hRESTS Microformat Syntax

The purpose of hRESTS is to provide a machine-readable representation of common Web service and API descriptions. The preceding section shows our model for this machine-readable information. Here, we proceed to define the syntax of a microformat that realizes the model in the HTML service documentation.

Microformats take advantage of existing XHTML facilities such as the `class` and `rel` attributes to mark up fragments of interest in a Web page, making the fragments easily available for machine processing. For example, a calendar microformat marks up events with their start and end time and with the event title, and a calendaring application can then directly import data from what otherwise looks like a normal Web page. Further details on how microformats work can be found at `microformats.org`.

The hRESTS microformat is made up of a number of HTML classes that correspond directly to the various parts of our service model. To help illustrate the following detailed definitions of the hRESTS classes and the structural constraints on hRESTS descriptions defined at the end of this section, in Listing 4.2 we show hRESTS annotations of the sample HTML service description shown in Listing 3.1.

In the following detailed definitions, we refer to RDF classes and properties from the service model (Listing 4.1) using the prefixes `wsl` and `hr`.

[1] URI templates are defined for instance in WSDL 2.0 HTTP Binding in [14] Section 6.8.1.1.

[2] While HTTP defines request and response messages, we call them *input* and *output* messages in hRESTS for compatibility with WSMO-Lite and WSDL.

```
1  @prefix hr:   <http://www.wsmo.org/ns/hrests#> .
2  @prefix rdf:  <http://www.w3.org/1999/02/22−rdf−syntax−ns#> .
3  @prefix rdfs: <http://www.w3.org/2000/01/rdf−schema#> .
4  @prefix wsl:  <http://www.wsmo.org/ns/wsmo−lite#> .
5  @prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
6
7  # classes and properties of the WSMO−Lite minimal service model
8  wsl:Service a rdfs:Class .
9  wsl:hasOperation a rdf:Property ;
10   rdfs:domain wsl:Service ;
11   rdfs:range wsl:Operation .
12 wsl:Operation a rdfs:Class .
13 wsl:hasInputMessage a rdf:Property ;
14   rdfs:domain wsl:Operation ;
15   rdfs:range wsl:Message .
16 wsl:hasOutputMessage a rdf:Property ;
17   rdfs:domain wsl:Operation ;
18   rdfs:range wsl:Message .
19 wsl:Message a rdfs:Class .
20
21 # hRESTS properties added to the above model
22 hr:hasAddress a rdf:Property ;
23   rdfs:domain wsl:Operation ;
24   rdfs:range hr:URITemplate .
25 hr:hasMethod a rdf:Property ;
26   rdfs:domain wsl:Operation ;
27   rdfs:range xsd:string .
28
29 # a datatype for URI templates
30 hr:URITemplate a rdfs:Datatype .
```

Listing 4.1: hRESTS service model in RDFS/N3

The `service` class on block markup (e.g. `<body>`, `<div>`), as shown in the example listing on line 1, indicates that the element describes a service API. An element with the class `service` corresponds to an instance of `wsl:Service`. A service contains one or more operations and may have a label (see below).

The `operation` class, also used on block markup (e.g. `<div>`), indicates that the element contains a description of a single Web service operation, as shown in the listing on line 3. An element with this class corresponds to an instance of `wsl:Operation`, attached to its parent service with `wsl:hasOperation`. An operation description specifies the address and the method used by the operation, and it may also contain description of the input and output of the operation, and finally a label.

The `address` class is used on textual markup (e.g. `<code>`, shown on line 6) or on a hyperlink (`<a href>`) and specifies the URI of the operation, or the URI template in case any inputs are URI parameters. Its value is attached to the operation using `hr:hasAddress`. On a textual element, the address value is in the content; on an abbreviation, the expanded form (the *title* of the abbreviation) specifies the address; and on a hyperlink, the target of the link specifies the address of the operation.

The `method` class on textual markup (e.g. `<span>`, shown on line 5) specifies the HTTP method used by the operation. Its value is attached to the appropriate operation using the property `hr:hasMethod`.

Both the address and the method may also be specified on the level of the service, in which case these values serve as defaults for operations that do not specify them. In

```
1  <div class="service" id="svc">
2   <h1><span class="label">ACME Hotels</span> service API</h1>
3   <div class="operation" id="op1">
4    <h2>Operation <code class="label">getHotelDetails</code></h2>
5    <p> Invoked using the <span class="method">GET</span>
6     at <code class="address">http://example.com/h/{id}</code><br/>
7     <span class="input">
8      <strong>Parameters:</strong>
9      <code>id</code> – the identifier of the particular hotel
10    </span><br/>
11    <span class="output">
12     <strong>Output value:</strong> hotel details in an
13     <code>ex:hotelInformation</code> document
14    </span>
15   </p>
16  </div></div>
```

Listing 4.2: Example hRESTS service description

absence of any explicit value for method, the default is GET. The RDF form of the service model reflects the default values already applied, that is, an instance `wsl:Service` will never have either `hr:hasMethod` or `hr:hasAddress`.

The `input` and `output` classes are used on block markup (e.g. `<div>` but also `<span>`), as shown on lines 7 and 11, to indicate the description of the input or output of an operation. Elements with these classes correspond to instances of `wsl:Message`, attached to the parent operation with `wsl:hasInputMessage` and `wsl:hasOutputMessage` respectively. While hRESTS does not provide for further machine-readable information about the inputs and outputs, extensions such as MicroWSMO (cf. Section 5) and SA-REST [10] add more properties here.

In principle, the output data format can be self-describing through the metadata the client receives together with the operation response, but it is, in general, useful for API descriptions to specify what the client can expect; hence our `output` class.

The `label` class is used on textual markup to specify human-readable labels for services and operations, as shown on lines 2 and 4 in the example listing. The value is attached to the appropriate service or operation using `rdfs:label`.

Additionally, elements with the classes `service` or `operation` can carry an `id` attribute, which is combined with the URI of the HTML document to form the URI identifier of the particular service or operation. This will allow other statements to directly refer to these instances.

The definitions above imply a hierarchical use of the classes within the element structure of the HTML documentation. The following is a complete list of structural constraints on the hierarchy of elements marked up with hRESTS classes. It reflects the structure of our service model, amended with the defaulting of the `address` and `method` properties:

1. No XHTML element has two or more hRESTS classes at the same time.
2. No element with the class `service` is a descendant[3] of an element with any hRESTS class.
3. Either there is no element with the class `service` in the document, or every element with the class `operation` is a descendant of an element with the class `service`.

[3]The term *descendant* is defined for XML/HTML elements in XPath [15].

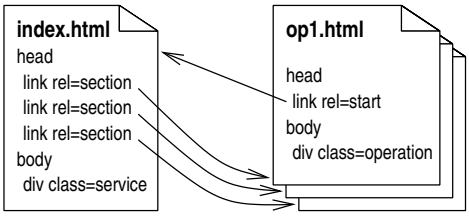

Figure 4.2: Service description in multiple documents

4. No element with the class `operation` is a descendant of an element with an hRESTS class other than `service`.
5. Every element with the class `address`, `method` or `label` is a descendant of an element with either the class `service` or the class `operation`.
6. Every element with the class `input` or `output` is a descendant of an element with the class `operation`.
7. No element with any of the classes `address`, `method`, `input`, `output` or `label` is a descendant of an element with an hRESTS class other than `service` and `operation`.

A single HTML document can define multiple services; such a document will contain multiple elements with the class `service`. Conversely, and this is a common occurrence, multiple documents can together make up the description of a single service. Indeed, textual service documentation is often split into a number of interlinked pages that describe the service as a whole, the individual operations, data types, error conditions, specific authentication mechanisms etc. In such cases, the Web page describing an operation (or a group of operations) will not contain any element with the class `service` because it is described elsewhere. The documents that together make up the service description should contain metadata links (either `<link>` elements in the `<head>` section, or `<a href>` elements in the body) pointing to the first document in the set (`rel="start"` as defined by HTML [4]) and to the documents that make up the set (`rel="section"`) — such links can help a crawler to find related pieces of the service description.

Such a situation is illustrated in Figure 4.2, which shows an overview page on the left that talks about the service as a whole, and three pages on the right that describe one operation each. The `start` and `section` relation links tie the pages together, which can be interpreted in our RDF model as a description of a single service with three operations.

As a consequence, a Web page pointed to by a link with `rel="start"` should contain only a single element with the class `service` so that the assignment of the operations to the service is unambiguous.

4.3 hRESTS in RDFa

Alternatively to using our microformat to capture the service model structure in the HTML documentation of RESTful Web services, we can also employ RDFa [7] and directly use the RDF service model. RDFa specifies a collection of XML attributes for expressing RDF data in any markup language, and especially in HTML.

Since our service description data is ultimately processed as RDF, RDFa is directly applicable. In our case, the difference between the use of a microformat or RDFa boils down to several considerations:

```
1  <div typeof="wsl:Service" about="#svc"
2        xmlns:hr="http://www.wsmo.org/ns/hrests#"
3        xmlns:wsl="http://www.wsmo.org/ns/wsmo-lite#"
4        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
5   <h1><span property="rdfs:label">ACME Hotels</span> service API</h1>
6   <div rel="wsl:hasOperation"><span typeof="wsl:Operation" about="#op1">
7    <h2>Operation <code property="rdfs:label">getHotelDetails</code></h2>
8    <p> Invoked using the <span property="hr:hasMethod">GET</span>
9     at <code property="hr:hasAddress" datatype="hr:URITemplate"
10          >http://example.com/h/{id}</code><br/>
11    <span rel="wsl:hasInputMessage"><span typeof="wsl:Message">
12     <strong>Parameters:</strong>
13     <code>id</code> – the identifier of the particular hotel
14    </span></span><br/>
15    <span rel="wsl:hasOutputMessage"><span typeof="wsl:Message">
16     <strong>Output value:</strong> hotel details in an
17     <code>ex:hotelInformation</code> document
18    </span></span>
19   </p>
20  </span></div></div>
```

Listing 4.3: Example hRESTS description expressed using RDFa

- the microformat syntax is simpler and more compact than RDFa;
- HTML marked up with our microformat remains valid HTML, whereas RDFa currently only validates against the newest schemas;
- RDFa represents the full concept URIs and thus facilitates the coexistence of multiple data vocabularies in a single document, where microformats may run into naming conflicts;
- processing microformats requires vocabulary-specific parsers (such as our XSLT transformation described in Section 6), while parsing the RDF data from RDFa is independent from any actual data vocabularies.

Listing 4.3 shows the same description as Listing 4.2, using RDFa instead of our microformat. Apart from the differences in attribute names, the listing shows two bigger differences:

Lines 6, 11 and 15 show that in order to make the `rdf:type` of the instances explicit, RDFa requires explicit statements. Similarly, line 9 shows that the datatype of a literal needs to be specified explicitly. However, if we can rely on RDFS inference to fill in the property range classes, the type statements on lines 6, 11 and 15 can be omitted, resulting in shortening these lines to the following form:

```
<div rel="wsl:hasOperation" resource="#op1">
  <span rel="wsl:hasInputMessage">
  <span rel="wsl:hasOutputMessage">
```

Finally, lines 2–4 show that RDFa requires the appropriate namespace declarations in order to be able to form the full URIs of all the terms of our vocabulary.

Since RDFa has only been published as a W3C Recommendation very recently, it is difficult to judge whether it will be accepted as an alternative to microformats, where it would be appropriate. The use of GRDDL [2] can alleviate the problem of vocabulary-specific parsers for microformats, reducing the need for RDFa. On the other hand, tool support can lower the importance of microformats' simpler syntax, making the future-proof RDFa format more acceptable.

Processors consuming hRESTS descriptions should support both forms (the microformat and RDFa). After translation into RDF, the resulting data shows no significant differences.

5 MicroWSMO: Extending hRESTS with Semantic Annotations

The hRESTS microformat structures the HTML documentation of RESTful Web services so they are amenable to machine processing. The microformat identifies key pieces of information that are already present in the documentation, effectively creating an analogue of WSDL, which is used by messaging (non-RESTful) Web services. hRESTS forms the basis for further extensions, where service descriptions are annotated with added information to facilitate further processing. In this section, we present MicroWSMO, an extension of hRESTS that adds semantic annotations.

Because the hRESTS view of services (Section 4.1) is so similar to that of WSDL, we can adopt SAWSDL [9] properties to add semantic annotations. SAWSDL is an extension of WSDL that specifies how to annotate service descriptions with semantic information. It defines the following three XML attributes, along with RDF properties with the same names:

- `modelReference` is used on any component in the service model to point to appropriate semantic concepts[1] identified by URIs,
- `liftingSchemaMapping` and `loweringSchemaMapping` are used to associate messages with appropriate transformations, also identified by URIs, between the underlying technical format such as XML and a semantic knowledge representation format such as RDF.

Figure 5.1 illustrates the relation of MicroWSMO to SAWSDL, along with their positioning among the various service description specifications. MicroWSMO is a SAWSDL-like layer on top of hRESTS. WSMO-Lite [12] specifies an ontology for the content of SAWSDL annotations in WSDL; MicroWSMO annotations in hRESTS also point to instances of the WSMO-Lite ontology, since it captures service semantics independently of the underlying Web service technology (WSDL/SOAP or REST/HTTP). In effect, MicroWSMO is on a layer below WSMO-Lite, even though both use the acronym “WSMO” in their names.

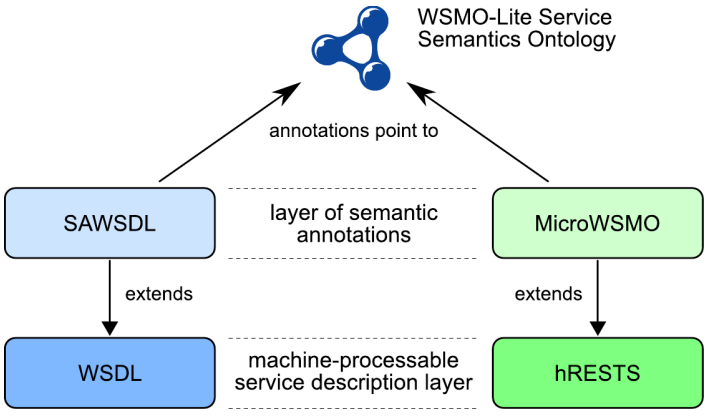

Figure 5.1: Relative positioning of WSMO-Lite and MicroWSMO

The WSMO-Lite ontology captures four aspects of service semantics: *information model* (a domain ontology) represents data, especially in input and output messages; *functional*

[1] SAWSDL [9] speaks about semantic concepts in general, which is not to be confused with the use of the term “concept” in WSMO to denote what is called “class” in OWL; a model reference can point to any element of a semantic description.

```
<div class="service" id="svc">
  <h1><span class="label">ACME Hotels</span> service API</h1>
    <p>This service is a
      <a rel="model" href="http://example.com/ecommerce/hotelReservation">
        hotel reservation</a> service.
    </p>
  <div class="operation" id="op1">
    <h2>Operation <code class="label">getHotelDetails</code></h2>
    <p> Invoked using the <span class="method">GET</span>
      at <code class="address">http://example.com/h/{id}</code><br/>
      <span class="input">
        <strong>Parameters:</strong>
        <a rel="model" href="http://example.com/data/onto.owl#Hotel">
          <code>id</code></a> – the identifier of the particular hotel
          (<a rel="lowering" href="http://example.com/data/hotel.xsparql">lowering</a>)
      </span><br/>
      <span class="output">
        <strong>Output value:</strong> hotel details in an
        <code>ex:hotelInformation</code> document
      </span>
    </p>
</div></div>
```

Listing 5.1: Example MicroWSMO semantic description

semantics specifies what the service does, by means of functionality classification or through preconditions and effects; *behavioral semantics* defines the sequencing of operation invocations when invoking the service; and *nonfunctional descriptions* represent service policies or other details specific to the implementation or running environment of a service.

To annotate a service description with the appropriate semantics, a model reference on a service can point to a description of the service's functional and nonfunctional semantics; a model reference on an operation points to the operation's part of the behavioral semantics description; and a model reference on a message points to the message's counterpart(s) in the service's information semantics ontology, complemented as appropriate by a pointer to a lifting or lowering schema mapping. See [12] for further details of the semantic service model.

SAWSDL annotations are URIs that identify semantic concepts and data transformations. Such URIs can be added to the HTML documentation of RESTful services in the form of hypertext links. HTML [4] defines a mechanism for specifying the relation represented by link, embodied in the `rel` attribute; along with `class`, this attribute is also used to express microformats. In accordance with SAWSDL, we introduce the following three new types of link relations:

- `model` indicates that the link is a model reference,
- `lifting` and `lowering` denote links to the respective data transformations.

Listing 5.1 illustrates the use of these link relations on semantic annotations added to the hRESTS description from Listing 4.2. In the following detailed definitions, we refer to the SAWSDL RDF properties using the prefix `sawsdl`[2].

The `model` link relation, on a hyperlink present within an hRESTS `service`, `operation`, `input` or `output` block, specifies a model reference (`sawsdl:modelReference`) from the respective component to its semantic description, as defined by WSMO-Lite.

[2]The prefix `sawsdl` refers to the namespace `http://www.w3.org/ns/sawsdl\#`

Listing 5.1 shows the use of the `model` link relation on lines 4 and 13. Line 4 specifies that the service does hotel reservations (the URI identifies a category in some classification of services), whereas line 13 defines the input of the operation to be an instance of the class Hotel, which is a part of the data ontology of this service.

The `lifting` and `lowering` link relations, on hyperlinks present within an hRESTS `input` or `output` block (corresponding to the properties `sawsdl:liftingSchemaMapping` and `sawsdl:loweringSchemaMapping`), specify the respective data transformations that map between the knowledge representation format of the client and the syntax of the wire messages of the service.

Listing 5.1 shows a link to a lowering transformation on line 15. The transformation would presumably map a given instance of the class Hotel into the ID that the service expects as a URI parameter. The description of concrete data lifting and lowering technologies is out of scope of this deliverable; it will be specified in more detail elsewhere as part of our future work.

Listing 5.1 shows the microformat syntax of MicroWSMO. In RDFa, the `rel` attribute would contain a namespace-qualified full name of the given SAWSDL property. For instance, line 4 would become:

<a **rel="sawsdl:modelReference"**
href="http://example.com/ecommerce/hotelReservation">

MicroWSMO and hRESTS, together with the WSMO-Lite ontology for service semantics, support automation of the use of RESTful Web services. Such automation has been researched under the name Semantic Web Services (SWS, [11]), where the aim is to use semantic technologies to help with the following tasks: *discovery* matches known Web services against a user goal and returns the services that can satisfy that goal; *ranking* orders the discovered services based on user requirements and preferences so the best service can be selected; *composition* puts together multiple services when no single service can fulfill the whole goal; *invocation* then communicates with a particular service to execute its functionality; and *mediation* resolves any arising heterogeneities.

6 Parser Implementation

In this section, we briefly describe an openly available XSLT stylesheet[1] that parses HTML documents with hRESTS and MicroWSMO microformat mark-up to produce the RDF form of the service description data.

In accordance with GRDDL, an XHTML document that contains hRESTS (and MicroWSMO) data can point to this stylesheet in its header metadata:

```
<head profile="http://www.w3.org/2003/g/data-view">
  <link rel="transformation"
        href="http://cms-wg.sti2.org/TR/d12/v0.1/20081202/xslt/hrests.xslt" />
  ... further metadata, especially page title ...
</head>
```

This header enables Web crawlers to extract the RDF form of the service description data, even if the crawlers are not specifically aware of the hRESTS and MicroWSMO microformats.

The MicroWSMO description from Listing 5.1 is embedded in an XHTML document[2] that contains also the GRDDL transformation pointer. Listing 6.1 shows the GRDDL RDF view of the document.

```
1  @prefix hr:       <http://www.wsmo.org/ns/hrests#> .
2  @prefix rdfs:     <http://www.w3.org/2000/01/rdf-schema#> .
3  @prefix sawsdl:   <http://www.w3.org/ns/sawsdl#> .
4  @prefix wsl:      <http://www.wsmo.org/ns/wsmo-lite#> .
5  @prefix ex:       <http://cms-wg.sti2.org/TR/d12/v0.1/20081202/xslt/example.xhtml#> .
6
7  ex:svc a wsl:Service ;
8    rdfs:isDefinedBy <http://cms-wg.sti2.org/TR/d12/v0.1/20081202/xslt/example.xhtml> ;
9    rdfs:label "ACME Hotels" ;
10   sawsdl:modelReference <http://example.com/ecommerce/hotelReservation> ;
11   wsl:hasOperation ex:op1 .
12 ex:op1 a wsl:Operation;
13   rdfs:label "getHotelDetails" ;
14   hr:hasMethod "GET" ;
15   hr:hasAddress "http://example.com/h/{id}"^^hr:URITemplate ;
16   wsl:hasInputMessage [
17     a wsl:Message ;
18     sawsdl:modelReference <http://example.com/data/onto.owl#Hotel> ;
19     sawsdl:loweringSchemaMapping <http://example.com/data/hotel.xsparql>
20   ] ;
21   wsl:hasOutputMessage [ a wsl:Message ] .
```

Listing 6.1: RDF data extracted from Listing 4.2

Most of the listing is self-explanatory (for readers familiar with the N3 RDF syntax[3]). Note that our XSLT stylesheet adds an `rdfs:isDefinedBy` property (line 8) to the service with a pointer back to the HTML documentation that defines it; and also note the `[]` syntax for blank nodes on lines 16 and 21.

[1] `http://cms-wg.sti2.org/TR/d12/v0.1/20081202/xslt/hrests.xslt`
[2] `http://cms-wg.sti2.org/TR/d12/v0.1/20081202/xslt/example.xhtml`
[3] `http://www.w3.org/DesignIssues/Notation3.html`

7 Related Work and Conclusions

The Programmable Web needs machine-readable descriptions of the available Web services. With such descriptions, search engines can gather better information about existing services, and developers can easier use these services. Tools enabled by the existence of such descriptions can support the developer in using the Web APIs and mashing them up with others. With semantic annotations, the Web services and APIs can even be discovered and used automatically in Semantic Web Services systems.

There are several existing formats for machine-readable description of Web APIs, e.g. WADL [3] and even WSDL 2.0 [13], both amenable to SAWSDL annotations. Probably due to the perceived complexity of these XML formats, they do not seem to be gaining traction with API providers; service descriptions remain mostly in unstructured text. Therefore we propose hRESTS and MicroWSMO as a more accessible approach.

In this deliverable, we have defined a model of RESTful Web services and used that model to create the hRESTS microformat, which can make the critical parts of existing Web API documentation machine-readable. We have further defined MicroWSMO, an extension that builds on top of hRESTS to add semantic annotations, with direct support for lightweight semantics from WSMO-Lite.

As shown in this deliverable, MicroWSMO allows HTML service documentation to be annotated with service semantics in the same way that WSDL is annotated with SAWSDL. Thus we can treat RESTful services as first-class peers of WSDL-based services, and we can provide the same level of semantic automation.

To foster a wider adoption of hRESTS, we intend to follow the `microformats.org` process and to build community consensus on machine-readable descriptions of Web APIs. Similar steps may subsequently be planned for MicroWSMO.

ACKNOWLEDGEMENTS

The authors would like to thank to all the members of the Conceptual Models for Services working group[1] for their advice and input to this document.

[1] http://cms-wg.sti2.org/operation/members/

REFERENCES

[1] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000. Chair: Richard N. Taylor.

[2] Gleaning Resource Descriptions from Dialects of Languages (GRDDL). Recommendation, W3C, September 2007. Available at `http://www.w3.org/TR/grddl/`.

[3] Marc J. Hadley. Web Application Description Language (WADL). Technical report, Sun Microsystems, November 2006. Available at `https://wadl.dev.java.net/`.

[4] HTML 4.01 Specification. Recommendation, W3C, 1999. Available at `http://www.w3.org/TR/html401`.

[5] Hypertext Transfer Protocol – HTTP/1.1. Draft Internet Standard, IETF, June 1999. Available at `http://rfc.net/rfc2616.html`.

[6] R. Khare and T. Çelik. Microformats: a pragmatic path to the semantic web (Poster). *Proceedings of the 15th international conference on World Wide Web*, pages 865–866, 2006.

[7] RDFa in XHTML: Syntax and Processing. Recommendation, W3C, October 2008. Available at `http://www.w3.org/TR/rdfa-syntax/`.

[8] Leonard Richardson and Sam Ruby. *RESTful Web Services*. O'Reilly Media, May 2007.

[9] Semantic Annotations for WSDL and XML Schema. Recommendation, W3C, August 2007. Available at `http://www.w3.org/TR/sawsdl/`.

[10] Amit P. Sheth, Karthik Gomadam, and Jon Lathem. SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups. *IEEE Internet Computing*, 11(6):91–94, 2007.

[11] R. Studer, S. Grimm, and A. Abecker. *Semantic Web Services: Concepts, Technologies, and Applications*. Springer-Verlag New York, Inc. Secaucus, NJ, USA, 2007.

[12] Tomas Vitvar, Jacek Kopecký, and Dieter Fensel. WSMO-Lite: Lightweight Semantic Descriptions for Services on the Web. CMS WG Working Draft, February 2009. Available at `http://cms-wg.sti2.org/TR/d11/`.

[13] Web Services Description Language (WSDL) Version 2.0. Recommendation, W3C, June 2007. Available at `http://www.w3.org/TR/wsdl20/`.

[14] Web Services Description Language (WSDL) Version 2.0: Adjuncts. Recommendation, W3C, June 2007. Available at `http://www.w3.org/TR/wsdl20-adjuncts/`.

[15] XML Path Language (XPath) Version 1.0. Recommendation, W3C, November 1999. Available at `http://www.w3.org/TR/xpath`.