hRESTS: an HTML Microformat for Describing RESTful Web Services

Jacek Kopecký
STI Innsbruck
Innsbruck, Austria
jacek.kopecky@sti2.at

Karthik Gomadam
kno.e.sis Center
Wright State University
Dayton, Ohio, USA
karthik@knoesis.org

Tomas Vitvar
STI Innsbruck
Innsbruck, Austria
tomas.vitvar@sti2.at

Abstract

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 tool support for developers who use these services. We propose a microformat called hRESTS (HTML for RESTful Services) for machine-readable descriptions of Web APIs, backed by a simple service model. The hRESTS microformat describes main aspects of services, such as operations, inputs and outputs. We also present two extensions of hRESTS: SA-REST, which captures the facets of public APIs important for mashup developers, and MicroWSMO, which provides support for semantic automation.

1. Introduction

The Web has gone through a great change 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.

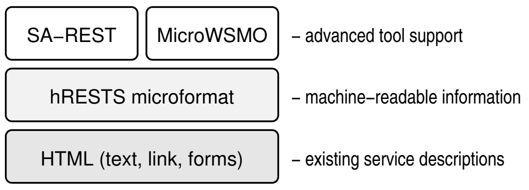

Figure 1. hRESTS layer cake

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, geographic coordinates, calendar events, ratings etc. In this paper, we propose a microformat called hRESTS (HTML for RESTful Services) for machine-readable descriptions of Web APIs, backed by a simple service model. As depicted in Figure 1, the hRESTS microformat captures machine-processable service descriptions, building on the HTML service documentation aimed at developers. We also show that hRESTS is a good basis (and a common model) for extensions, such as SA-REST, providing support for describing various facets of Web APIs, or MicroWSMO, adding means for semantic Web service automation.

The remainder of this paper is structured as follows: Section 2 introduces an example service/API that we use for demonstrating hRESTS. In Section 3, we discuss the common ways of describing RESTful Web services. Section 4 details our service model — how we view RESTful Web services. Section 5 uses this model to define hRESTS, our microformat for machine-readable service descriptions. Sections 6 and 7 sketch SA-REST and MicroWSMO, two possible applications of hRESTS that use it and extend it for various kinds of semantic annotations. In Section 8, we discuss some related work, and Section 9 concludes the paper.

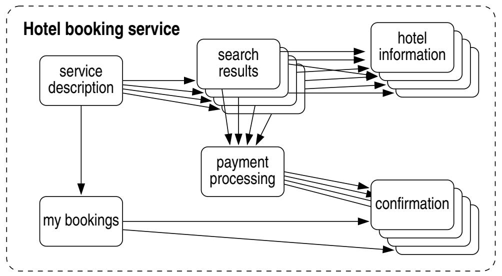

Figure 2. Structure of an example service

2. Example RESTful Web Service

Web APIs and RESTful Web services are hypermedia applications consisting of interlinked resources, oriented towards machine consumption. The orientation towards machine consumption manifests mainly in that the interactions between RESTful services and their clients are generally done with 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 their structure and behavior, RESTful Web services are very much like common Web sites. [10]

Figure 2 illustrates an example RESTful hotel booking service, with its resources and the links among them. 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.

The service description resource 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 link to search results resources, one 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 with the search results. 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 program 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 an input 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 enables 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 search results.

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

The service description also contains a link to a page with the bookings of the current user (which requires authentication functionality). With such a resource available to them, client applications no longer need to store the information about performed bookings locally.

Search results are a list of concrete rates available at the hotels in the given location, for the given dates and the number of guests. Each item of the list contains a link to further information about the hotel (e.g. the precise location, star rating and other descriptions), and a form for booking the rate, which takes as input the payment details (e.g. credit card information) and an identification of the guest who is going to 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.

Finally, the "my bookings" resource links to the confirmations of the bookings done by the authenticated user. The confirmations may further provide a way of canceling the reservation (not shown in the picture).

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 the service in a dashed box.

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.

In Figure 3, we extract the operations available in our 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.

While the resources of a service (the *nouns*) form the hypermedia graph (shown in Fig. 2), a programmer making a

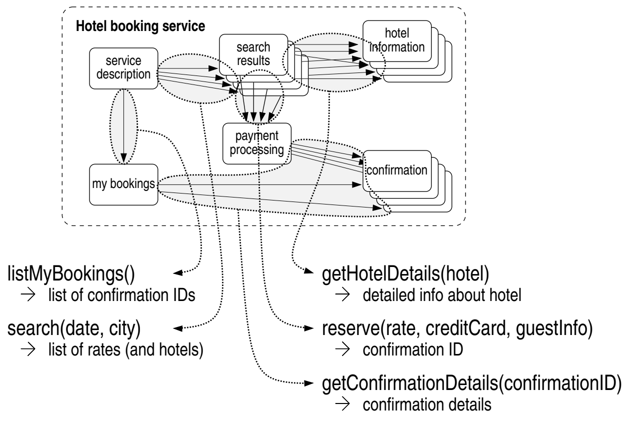

Figure 3. Operations of the example service

mashup or an automated client program rather thinks of the operations that can be invoked (the *verbs*, shown in Fig. 3); therefore public RESTful Web services are generally called APIs and are described in terms of the operations. The following might be a typical operation description:

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

3. Describing Web APIs and RESTful Web Services

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 use any particular service.

WSDL [13] is a standard for Web service description, yet it is not perceived as suitable for describing RESTful services, and few (if any) such services have a WSDL description. Instead, Web APIs are usually described in textual documentation (on dedicated Web pages). Nevertheless, in order to provide tool support or automation for consuming RESTful Web services, certain aspects of the service descriptions must be made machine-readable.

In this section, we briefly discuss how Web services are described using HTML text, links and forms.

3.1. Textual Descriptions

Web service documentation is most naturally available as Web pages. Textual documentation, such as the example operation description above, or real service descriptions at `docs.amazonwebservices.com/AmazonSimpleDB/2007-11-07/DeveloperGuide` and `flickr.com/services/api`, has all the details necessary for a human to be able to create a program that can use the service.

The operation description from above could be captured in HTML textual documentation like this:

> <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>

In order to tease out the technical details (operations, addresses, HTTP methods, input and output data formats), the textual documentation needs to be amended in some way, such as with our hRESTS microformat (Section 5).

3.2. Links and Forms

Beside the typical public Web API descriptions, the Web actually contains many simple machine-readable operation descriptions: hyperlinks and data input forms.

Theoretically, any hyperlink denotes an operation that uses the method GET on the given target address. Such operations have no input data, and they generally serve for data retrieval. Our example hotel reservation service has one data retrieval operation with no inputs: listMyBookings(). The description of this service can simply link to the "my bookings" resource and specify the format in which the list of bookings will be returned. The user can then very easily test this operation — simply click on the link.

However, obviously not all hyperlinks in the service documentation describe service operations. Therefore, a machine-readable service description should distinguish links representing data retrieval operations from other, non-operation links.

Operations on the Web can also be described with forms. A form specifies the address, the HTTP method (HTML forms support only GET and POST) and the input data. If a service has an operation that has a few simple inputs and uses the GET or POST method, such as the getHotelDetails operation above, the service description can include a form for invoking this operation, again so that a developer reading the description can easily test the operation.

Our service description microformat hRESTS, while primarily aimed at annotating textual descriptions, also supports annotation of hyperlinks and forms. If present, operation-describing links and forms provide most of the necessary machine-readable information about the operation (with the notable exception of the output data format), and hRESTS takes advantage of that.

4. Model for RESTful Web Services

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 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, a RESTful Web service can be decomposed into its operations, which can be considered independently from the structure of the hypertext (cf. Fig. 3).

This leads us to a service model shown in Figure 4. A Web service has a number of operations, each with potential inputs and outputs, and a hypertext graph structure

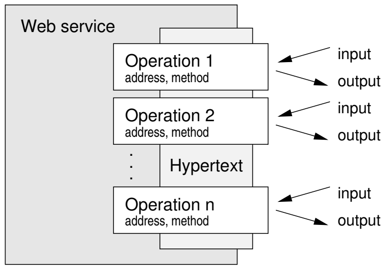

Figure 4. Functional model of a Web service

where the outputs of one operation may link to other operations. This model presents the requirements for what our machine-readable description of RESTful Web services must be able to represent. Unsurprisingly, the model is very similar to the structure of WSDL, only instead of hypertext, WSDL services use the terms "process" or "choreography" for the sequencing of operations.

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

Listing 1 shows an RDFS realization of our 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.

A machine-readable description of a Web service 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. We sketch different kinds of annotations in Sections 6 and 7.

5. hRESTS: Machine-readable Web API and Service Descriptions

The purpose of the hRESTS microformat is to provide a machine-readable representation of common Web service and API descriptions. Section 4 shows our model for this machine-readable information. But first, let us quickly describe how microformats work.

Microformats take advantage of existing XHTML facilities such as the `class` and `rel` attributes to mark the fragments of interest in a Web page. A calendar microformat marks 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.

```
@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:Message  a  rdfs:Class .
hr:hasAddress  a  rdf:Property ;
   rdfs:domain  hr:Operation ;
   rdfs:range  hr:URITemplate . # a datatype for URI templates
hr:hasMethod  a  rdf:Property ;
   rdfs:domain  hr:Operation ;
   rdfs:range  hr:HTTPMethod .
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 .
```

Listing 1. Service model in RDFS/N3

A microformat translates the hierarchy of HTML elements into a hierarchy of objects and their properties. For instance in hRESTS, an element with the class `service` is expected to contain (child or descendant) elements with the class `operation`, representing the service's operations, and one element with the class `label`, specifying the name of the service. Further details on how microformats work can be found at `microformats.org`.

Listing 2 contains the sample HTML service description shown earlier, extended with a sentence about the service, and annotated with the hRESTS microformat. The following subsection defines the classes of the hRESTS microformat, together with a few defaulting rules. Section 5.2 then discusses our XSLT implementation of a parser for the microformat, together with tools planned as future work.

```
<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>
```

Listing 2. Example hRESTS description

5.1. hRESTS Microformat Definition

The **`service` class** on block markup (e.g. `<body>`, `<div>`), as shown in the example listing on line 1, indicates that the element is a part of the hRESTS microformat, containing a Web service or API description. A service contains one or more operations and may have a label (see below).

The **`operation` class** can also be used on block markup (e.g. `<div>`) to indicate that the element contains a description of a Web service operation, as shown in the listing on line 4. 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 `operation` class can also be used on hyperlinks (`<a href>`) and on forms (`<form>`). A hyperlink operation specifies the address in the `href` attribute and the method is GET. A form operation specifies the address in the `action` attribute, the method in the `method` attribute, and the various input fields of the form specify the input of the operation.

The **`address` class** is used either on textual markup (e.g. `<span>`, shown on line 7) or on a link (`<a href>`) and specifies the URI (or the URI template in case any inputs are URI parameters) of the operation. On a textual element, the address is in the content; on a link, the target is the address.

The **`method` class** on textual markup (e.g. `<span>`, shown on line 6) specifies the HTTP method used by the operation.

The **`input` and `output` classes** are used on block markup (e.g. `<div>` but also `<span>`), as shown on lines 8 and 10, to indicate the description of the input or output of an operation. Apart from the potential label, hRESTS does not actually provide for further machine-readable information about the inputs and outputs; however, extensions such as SA-REST and MicroWSMO add more properties here.

Finally, the **`label` class** is used on textual markup to specify a human-readable label for a service, an operation or for a message, as shown on lines 3 and 5.

```
@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 ] .
```

Listing 3. RDF data extracted from Listing 2

Defaulting: when a Web page contains some operation descriptions but no element with class `service`, the parser should assume that the page describes a single service with no label. Further, a service may specify the default address and method for its operations, using the classes `address` and `method` outside an operation block.

Due to space constraints, further details on the hRESTS microformat will be avialable online at `http://esw.w3.org/topic/Semantic_Annotations_for_RESTful_Services`

5.2. Implementation

In accordance with GRDDL, we have implemented an XSLT stylesheet that extracts the RDF form of the hRESTS data from XHTML Web pages.[2] The hRESTS description from Listing 2 is embedded in an XHTML document[3] that references this stylesheet. The GRDDL RDF view of the document is shown in Listing 3.

Beside the parser that extracts the data from hRESTS-annotated Web pages, we could also implement a validator to check that a Web page adheres to the hierarchy of our service model, and that it includes all the necessary information: a service must have at least one operation to be useful, and an operation must specify an address and the HTTP method in order to be invokable.

We could also implement a Web crawler to look for hRESTS service descriptions and store them in a service registry. We plan to work on these tools as part of the follow-up work on SA-REST and MicroWSMO.

6. SA-REST: Support for Service Facets

In addition to operations with their inputs and outputs, API documents describe other facets including data formats and programming language bindings. Unlike the WSDL model, where XML is the only data format, RESTful services also use a variety of other data formats including JSON, GData and ATOM/RSS, as described in any particular API documentation. Additionally, APIs may also provide client libraries in various programming languages. SA-REST, originally proposed in [11] as an RDFa-based annotation mechanism, is an extension of hRESTS that supports the description of these different service properties.

The motivation for creating this extension stemmed out of our experience in creating the IBM sharable mashup framework [8] and a search engine for Web APIs [3]. We identified two key impediments that users often face in the process of creating mashups.

1. The task of integrating services that support different data formats, since mediating between data formats is not straightforward.

[2]`http://members.sti2.at/~jacekk/hrests/hrests.xslt`
[3]`http://members.sti2.at/~jacekk/hrests/src.xhtml`

2. Creating mashups of services with client libraries in different languages would mean that the developer either has to use the supported languages or to write the client in a neutral language. In both cases, the developer loses the advantage of having client libraries. Further, it is very hard to access popular services like Google maps without using their client libraries.

Developers prefer to use services that are homogeneous in their data formats and client libraries, since it helps them to avoid these impediments. The SA-REST extension defines classes for describing the data format and programming language binding properties of Web APIs, thereby allowing developers to search in homogeneous groups.

The **`data-format`** **class** is used on textual markup and specifies the data format used in the API, or in a particular operation input or output. In the example in Listing 4, the data format markup is shown in line 4. In this case, the hotel service uses JSON.

The **`p-lang-binding`** **class** on textual markup specifies the programming language frameworks for which client libraries are available. To indicate the availability of client libraries in Java and PHP, the developer would use the markup shown on lines 6 and 7.

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

Listing 4. SA-REST example

With SA-REST, which is only sketched here, it will be possible to find and browse Web APIs based on various facets important to mashup developers.

7. MicroWSMO: Towards SWS Automation

In this section, we briefly describe how hRESTS can be used to support automation of the use of RESTful Web services. Such automation has been researched under the name Semantic Web Services (SWS), 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; *composition* puts together multiple services when no single service can fulfill the whole goal; *ranking* orders the discovered or composed services based on user requirements and preferences so the best service can be selected; *invocation* then communicates with the service to execute its functionality; and *mediation* resolves any arising heterogeneities.

```
@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 wsmolite:  <http://www.wsmo.org/ns/wsmo-lite#> .
@prefix sawsdl:   <http://www.w3.org/ns/sawsdl#> .

wsmolite:Ontology a rdfs:Class ; rdfs:subClassOf owl:Ontology .
wsmolite:ClassificationRoot rdfs:subClassOf rdfs:Class .
wsmolite:NonfunctionalParameter a rdfs:Class .
wsmolite:Condition a rdfs:Class .
wsmolite:Effect a rdfs:Class .

sawsdl:modelReference a rdf:Property .
sawsdl:liftingSchemaMapping a rdf:Property .
sawsdl:loweringSchemaMapping a rdf:Property .
```

Listing 5. WSMO-Lite and SAWSDL ontology

To support automation of these tasks, we need to capture four aspects of service semantics: *information model* (a domain ontology) represents data, especially in input and output messages; *functional 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.

WSMO-Lite [12] proposes a lightweight ontology for the four kinds of semantics, shown in Listing 5, and uses SAWSDL (Semantic Annotations for WSDL and XML Schema [7]) to annotate WSDL documents with instances of that ontology. This makes WSDL-based Web services amenable to SWS automation.

Because the hRESTS service model (Section 4) is so similar to that of WSDL, we can adopt SAWSDL properties as an extension of hRESTS and use them to add semantic descriptions conforming to the WSMO-Lite service ontology. As also shown in Listing 5, SAWSDL defines three properties: *model reference* is used to link any part of a service description with its semantic properties, and *lifting* and *lowering schema mappings* point from message descriptions to transformations between the ontological data and the on-the-wire message serialization.

Listing 6 shows MicroWSMO, our SAWSDL-based ex-

```
<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>
```

Listing 6. MicroWSMO example

tension of hRESTS. Line 3 specifies that the service does hotel reservations (this would be a category in some classification of services), line 7 defines the input of the operation to be an instance of the class Hotel, and the lowering schema mapping on line 8 would then map a given instance of Hotel into the ID that the service expects as a parameter.

MicroWSMO is only sketched here; finalizing it is future work. When completed, it will let RESTful Web services be used seamlessly along with WSDL-based services in WSMO-Lite based semantic automation frameworks.

8. Related Work

There are several alternatives to hRESTS for machine-readable description of Web APIs, e.g. WADL [5] and even WSDL 2.0. Probably due to their perceived complexity, they do not seem to be gaining traction with API providers; service descriptions remain mostly in unstructured text. Therefore we propose hRESTS as a simpler approach.

There is also an alternative to using microformats for machine-readable annotations of HTML pages: RDFa [9] is a general-purpose approach for embedding RDF data in HTML. While it could, in principle, be used for hRESTS, RDFa markup tends to be more complex than microformat markup. In addition to the simplicity, the humans-first approach of microformats also allows hRESTS to help users comprehend APIs better.

In the area of WSDL-based Web services, the annotation standard SAWSDL has given rise to a systematic approach to data mediation, utilizing data schemas with semantic information. Web APIs can also enable easier data mediation, for instance in context of service composition or mashups, by using hRESTS and MicroWSMO to describe their messages. Additionally, [4] demonstrates that semantic annotations of data schemas can also be used to estimate the level of effort required for a user to perform mediation manually.

9. Conclusions

The programmable Web needs machine-readable descriptions of the available Web services. With such descriptions, search engines can gather better information about the services, and developers can easier use it. Tools enabled by the existence of such descriptions can support the developer in using the Web APIs and mashing them up with others.

In this paper, we have defined a model of RESTful Web services, and we used that model to create the hRESTS microformat, which can be used to make the crucial parts of existing Web API documentation machine-readable. We have further outlined SA-REST and MicroWSMO, two extensions that build on top of hRESTS.

To foster wider adoption of hRESTS, we intend to follow the `microformats.org` process and to build community consensus on machine-readable descriptions of Web APIs. The authors were a part of the SAWSDL standardization process and hope to use their experience in working with the community of providers and users of public Web APIs. Currently, the work on hRESTS is supported by the W3C SWS Testbed incubator group, information regarding this can be found at `http://www.w3.org/2005/Incubator/swsc/`.

References

[1] R. T. 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] K. Gomadam, A. Ranabahu, M. Nagarajan, A. P. Sheth, and K. Verma. A Faceted Classification Based Approach to Search and Rank Web APIs. In *ICWS*, 2008. To appear.

[4] K. Gomadam, A. Ranabahu, L. Ramaswamy, K. Verma, and A. P. Sheth. Mediatability: Estimating the Degree of Human Involvement in XML Schema Mediation. In *ICSC*, 2008.

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

[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] J. Kopecký, T. Vitvar, C. Bournez, and J. Farrell. SAWSDL: Semantic Annotations for WSDL and XML Schema. *IEEE Internet Computing*, 11(6):60–67, 2007.

[8] M. Maximilen, A. Ranabahu, and K. Gomadam. IBM Sharable Code: A Domain-Specific Language and Online Platform for Web APIs and Services Mashups. *IEEE Internet Computing, to appear*, 2008.

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

[10] L. Richardson and S. Ruby. *RESTful Web Services*. O'Reilly Media, May 2007.

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

[12] T. Vitvar, J. Kopecký, J. Viskova, and D. Fensel. WSMO-Lite Annotations for Web Services. In *The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008*. Springer, 2008.

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