MICRO-RESOURCE
A Microformat Framework for Dual Restful Web Services

Li Li and Wu Chou
Avaya Labs Research, 233 Mt. Airy Road, Basking Ridge, New Jersey, U.S.A.

Keywords: Microformat, REST, Web service.

Abstract: RESTful architecture style that underlies the Web has gained rapid adoption as a way to develop web services for machines. But the full potential of REST is hindered by the fact that HTML pages designed for human interactions are not suitable for machine processing. To address this problem, we developed a microformat framework, called micro-resource, to extend web sites into dual RESTful web services for both human and machines alike with minimum changes. This framework avoids the pitfalls of alternative "parallel" web services by keeping the correspondence and duality between human and machine webs. This framework is simple, extensible and also composable with other existing microformats. Initial application of this framework on some RESTful service composition shows that the approach is efficient and feasible.

1 INTRODUCTION

REST stands for REpresentational State Transfer, the architecture style underlying the World Wide Web. The architecture style and design principles of REST are discussed and analyzed extensively in (Fielding 2000, AWWW 2004, Richardson 2007). A key concept in REST is resource, which is addressable by URI and has representations corresponding to its state. Each resource supports a subset of the uniform interface, such as HTTP, that manipulates its state through representations. A resource is typically connected with other resources through URIs. A web service hosted on a web site consists of a collection of resources. A client, typically a browser, interacts with the service by exchanging representations with those resources. In these interactions, the service is stateless as the client maintains the application states. In other words, a representation obtained from a resource contains complete information and instructions on how to interact with the service at that point. This self-describing ability is an important feature of REST and is referred to as the "hypermedia as the engine of application state" axiom of REST.

Due to its proved scalability and robustness over the Web and inherent simplicity, REST has gained rapid recognition as an alternative to the message-oriented web service architecture based on SOAP and WSDL.

In principle, REST can support both human-machine and machine-machine interactions. However, some technologies that implement REST, namly HTML and XHTML, are designed for humans, rather than for machines (For convenience, this paper uses HTML to refer to both HTML and XHTML henceforth). Therefore, it is difficult for a program to extract meaningful data and instructions from these HTML documents. This difficulty creates a gap that prevents us from treating a human web site as a programmable web service for machines, even though such convergence is beneficial and feasible.

To close this gap, we adopt the microformat approach and propose a microformat framework, called *micro-resource*, to annotate HTML documents such that a HTML document becomes suitable for both human and machine processing. A microformat is a set of tags embeded in a HTML document without affecting the visual display of the page, while a program can extract semantic information based on these tags and the predefined rules. The major advantages of microformat are:

1. it normalizes semantic data represented in different formats;
2. it minimizes changes to HTML pages by reusing the information in them;
3. it creates a dual representation for both human and machine;

Li L. and Chou W.
MICRO-RESOURCE - A Microformat Framework for Dual Restful Web Services.
DOI: 10.5220/0002805500560063
In *Proceedings of the 6th International Conference on Web Information Systems and Technology (WEBIST 2010)*, page
ISBN: 978-989-674-025-2

However, most microformats are designed to extract a specific type of semantic data, such as business card. They are not sufficient as a framework for RESTful services. Despite some initiatives to develop microformats for RESTful services, we have not seen a comprehensive technical proposal so far. In current approaches, each microformat requires a special parser that understands the tags and rules of that microformat. But a service framework needs to be extensible to include any type of semantic data from different domains. To address this issue, our framework employes a two-level microformat such that a general parser can extract and compose the semantic information for different services.

Because microformat offers dual representation, it can mitigate the cost of converting a human web site into a machine web service. In this case, we just add microformats to the HTML pages without changing the rest of the service implementation. The cost of maintaing a dual web service is also relatively low especially when the HTML pages are generated automatically at desgin time or runtime. It seems that a dual web service creates overhead as the service has to transmit extra data that are ignored by the machine, but this overhead keeps the correspondence between human and machine webs and eliminates the need to create parallel web services, in which resource representations and service flows are duplicated. Because of the correspondence, dual web services offer a much more user friendly self-describing facility for developers. When designed properly, a dual web service can significantly increase its usability by allowing a developer to interact with the service in the same way a regular user interact with a web application. From a developer's point of view, a web service effectively becomes "What You See Is What You Get," where "What You See" is the HTML pages and "What You Get" is the services embeded in the pages.

The rest of this paper is organized as follows. Section 2 surveys related work to our approach. Section 3 describes the general micro-resource framework. Section 4 introduces the rules of micro-resource. Section 5 is dedicated to the aggregation and composition of micro-resource with other microformats. Section 6 presents our experimental study and results. We conclude this paper in Section 7.

2 RELATED WORK

Microformat is an on-line community effort (microformat.org) to develop microformats for different applications. As of today, there are 9 specifications and 15 drafts. Each microformat specifies a set of tags (properties) and rules on how to annotate semantic data with those tags. These tags can be embedded into a HTML document, using HTML attributes `class` and `rel`, to assign semantic meanings to the document without affecting its rendering. For example, the hCard (microformat.org) microformat has 2 required properties and 24 optional ones, collectively describing a business card. Since all HTML elements have these two attributes, we can tag any element in a HTML document,that has semantic data. If necessary, the HTML grouping elements and can be used to create additional tagging points.

To illustrate the power of microformat, a telephone number `5555` that occurs in three different places: plain text, a table cell and an anchor text of a hyperlink, can be normalized with one tag of the hCard microformats:

```
<span class="tel">5555</span>
<td class="tel">5555</td>
<a class="tel" href="…">5555</a>
```

Such microformats in HTML pages can be transformed into semantic representations, such as JSON or RDF, by special parsers or using W3C GRDDL (GRDDL 2007) framework. In this sense, microformat can be regarded as a binding mechanism of abstract Infoset (Li 2009).

There are some initiatives to develop microformats for describing RESTful web services (REST microformat) as well.

REX (REST-Enabled XHTML) (REX) is an effort to develop a dual-use XHTML profile using microformats. Its goal is to cover a subset of REST services that can be represented by XHTML with microformats. The REX project page outlines some interesting questions, challenges and general directions, but we did not find a proposed specification or draft in the official microformat site. For this reason, it is difficult to compare our approach with REX, but REX encourages our pursuit of microformat based approach.

Peter Williams proposes (Williams 2008) a JSON based framework to describe REST services in terms of concepts such as service, resource, service provider and container. In this framework, there is a top-level capability resource from which services can be discovered through HTTP GET. On

the other hand, services can also register with the capability resource. In some sense, the capability resource is a directory service, like a Sitemap (Sitemap). But its service description does not contain any information about how to interact with the services, which is the focus of our approach.

WADL (Web Application Description Language) (Hadley 2006) is a XML dialect that describes REST services, as an equivalent of WSDL for SOAP based web services. The key elements of WADL include <resources>, which is a collection of <resource> elements that describe resources of the web application. Each <resource> element has a URI path attribute, a list of <params> elements and may contain other resources. A <resource> is also associated with a set of <method> elements, each having a HTTP verb, <request> and <response> elements. The <request> and <response> elements are associated with <representation> elements which define the media type and root XML element of the representation. In addition, a <response> element also contains <fault> element that lists the HTTP status codes. While WADL codifies the basic concepts of REST services, it only describes the static aspects of resources within a web service. If a web service creates new types of resources, as it often happens when a web site upgrades, the corresponding WADL has to be updated accordingly. This paradigm therefore misses the dynamic and self-describing facility of REST services.

hRESTS (Kopecky 2008) is an effort to use microformat to describe REST services in HTML. The proposed microformat is based on a functional service model similar to WSDL. The model is defined as RDF classes such as service, address, operation, method, input and output. These class labels are then used to annotate HTML pages, including links and forms, in the microformat fashion. Using the GRDDL framework (GRDDL 2007), XSLT transformations are applied to the HTML with microformat to extract RDF graph about the services, to be consumed by the clients. Although the goal of this approach is very similar to ours, its service model does not have the notion of resource. Without modelling resource explicitly, it is difficult to model the representation of resources and relations between resources, which in our opinion are the corner stones of REST.

RDFa (RDFa 2008) and eRDF (eRDF 2006), both derived from RDF, are more powerful mechanisms to insert semantic information into HTML pages, in a fashion similar to microformats. However, since our primary purpose is to interact with a web service, instead of inferring properties about the service, we choose microformat over RDF for this project. But our approach can use semantic web if necessary, as a microformat can always be converted to RDF triples.

Realizing that HTML is not suitable to encode machine-readable data, many web sites create "parallel" web services where two types of services, HTML for human users and XML for machine APIs are developed. While this approach satisfies both human and machine, as dual web services intend to do, it has some drawbacks. First, it increases development and maintenance cost because the two web services offer different URIs, representation formats, functionalities and access methods (Richardson 2007, page 27). Second, parallel web services tend to drift apart. As pointed out by (Richardson 2007, page 96), most web service APIs are not connected through links and forms, as human web sites do. The loss of connectedness decreases the self-describing facility and usability of the service. Third, the machine APIs cannot be learned and tested in a web browser as easily as surfing a web site. Developers have to read manuals and write code to understand the services. Even though we can create XSLTs to transform XML documents for machines into HTML pages for humans, writing XSLT rules is not a trivial amount of work and it does not put the human needs first. The dual web service that we promote is centred on human users and developers. This is one of the principles of microformat: "human first and machine second." (microformat.org wiki)

3 MICRO-RESOURCE FRAMEWORK

Our goal is to develop a microformat framework to enable dual web services, such that one service implementation is used for both human and machine consumption. The high-level interactions of duality of a web service are illustrated in Figure 1, where the arrows indicate the flow of data between components.

A micro browser is part of an automaton that interacts with web services through microformats and forms. The automaton can be a stand-alone client application or a component within web services. But in the dual web service, these use cases are treated the same.

A human user can interact with a well-designed web application he never visited before without any problem. This is because 1) the web pages clearly describes the interactions using a natural language,

such as English, that is understood by the user; and 2) the user already has some background knowledge about the basic mechanics of web browsers, for example, what happens if a link is clicked, or a submit button is pushed.

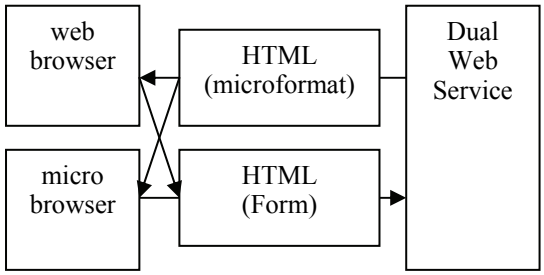

Figure 1: Dual web service architecture.

For the dual web service to work, a micro browser has to "mimic" the human-machine interactions with the same services. For this "mimic" to be feasible, we have to make more stringent assumptions because machines can only understand precise and well-defined data and protocols.

Without losing generality, we can assume an automaton is implemented as a finite-state machine (FSM) that controls the micro browser. Because each web service is different, we assume the finite-state machine is service specific. However, the micro browser is independent of specific services, just as a web browser is independent of web sites.

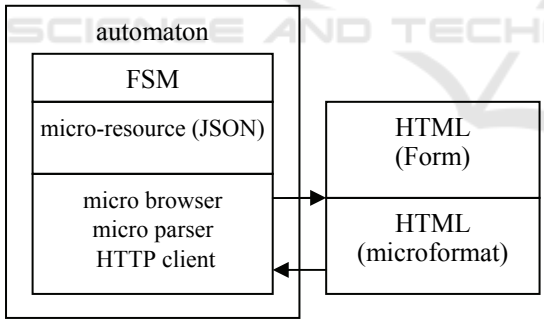

Figure 2: Components of micro browser.

The architecture of automata based on this idea is depicted in Figure 2, where a micro parser parses the microformats into the micro-resource data model proposed in this paper. The JSON (http://json.org/) is used as the intermediate language between the two layers. The reason we choose JSON is that it is a standard, simple and yet powerful data structure that can be serialized, transmitted and manipulated by a variety of programming languages, including Java, C++, C#, Perl and JavaScript. For this reason, a micro browser can be implemented in those languages and executed in various environments, including inside a web browser.

To support this architecture, the microformat has to be service independent yet powerful enough to convey the useful semantic information about the service. Otherwise, the micro browser cannot be service independent and the FSM layer does not have sufficient information to act on.

Based on our analysis using REST principles, HTML documents contain the following three types of information that are useful to interact with the resources in a service:

1. properties: the state and schema of a resource;
2. composition: the subordinate resources of a resource;
3. interface: the methods supported by a resource;

To capture this information, we propose a resource-oriented microformat, called micro-resource. *A micro-resource is a microformat representation of a resource*. The data model of micro-resource can be defined by the following UML class diagram (Figure 3).

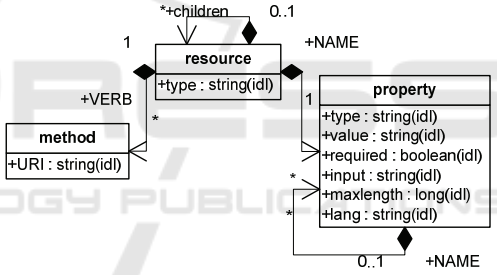

Figure 3: Data model of micro-resource microformat.

This data model specifies that each micro-resource has a type, a collection of properties and some methods. Each property has a set of attributes, modelled after HTML field elements (input, textarea and select). Each method is associated with a URI and is a subset of the uniform interface (GET, POST, PUT, DELETE, etc.). Each micro-resource can recursively contain other subordinate micro-resources. Each property can have recursive child properties as well.

Unlike most current microformats that has a closed set of tags, micro-resource is a generic microformat with open ended tags. This is necessary because we want to use micro-resource to different web services, each with its own set of resources. Consequently, you can treat a hCard and a hCalendar both as micro-resource. We need to tell which tags are for micro-resources, since micro-resource allows open ended tags, and tags from

different microformats can be mixed in one HTML page. With this piece of information, a generic micro parser can be used to instantiate the data model in Figure 3. Otherwise, we have to 1) know all the resource types of all services; 2) build a special micro parser for each service; or 3) disallow mixture of microformats. But none of these choices is practical or necessary.

The micro-resource therefore has a two-level structure: the first level uses the HTML `<meta>` element to list the micro-resources present in the document and the second level are actual micro-resources tagged by using the HTML `class` and `rel` attributes. Figure 4 is an example HTML file with embedded micro-resources.

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8"/>
    <meta name="resource" content="subscriptions
subscription"></meta>
    <title>Subscriptions</title>
  </head>
  <body>
    <p class="subscriptions">
      The subscriptions:
      <table border="1" >
        <thead>
          <tr>
            <th>Title</th>
            <th>Direction</th><th>Author</th><th>Updated</th>
          </tr>
        </thead>
        <tbody>
          <tr class="subscription">
            <td class="title"><a href="..."
rel="GET">title</a></td>
            <td class="direction">1</td><td
class="author">author</td><td class="updated">2009-09-
22T17:05:35.979-0400</td>
          </tr>
        </tbody>
      </table>
      <a href="..." rel="POST">Post New Subscription</a>
    </p>
  </body>
</html>
```

Figure 4: A HTML page with micro-resource microformat.

In this HTML document, the micro-resource tag `resource` in the second `<meta>` element (in bold font) declares there are two micro-resources in this document: `subscriptions` and `subscription`. The `subscriptions` micro-resource has a subordinate micro-resource `subscription`, and supports `POST` method to a URI. The `subscription` micro-resource has properties `title`, `direction`, `author` and `updated`, and accepts `GET` method to a URI.

The above semantic information about resources is extracted by the micro parser into a JSON representation (Figure 5):

```
{
  "POST": {"_uri": "..."},
  "_children": [
    {
      "GET": {"_uri": "..."},
      "_type": "subscription",
      "author": {"_value": "author"},
      "direction": {"_value": "1"},
      "title": {"_value": "title"},
      "updated": {"_value": "2009-09-22T17:05:35.979-0400"}
    },
  ],
  "_type": "subscriptions"
}
```

Figure 5: JSON representation of a micro-resource data model.

In the JSON representation, micro-resource built-in attributes have an underscore prefix (_type, _children, for example) whereas others are service specific tags outside the micro-resource microformat.

The micro-resource microformat normalizes the resource representations embedded in the HTML documents, making these representations accessible to the micro browser while keeping the correspondence between HTML and micro-resource data model.

4 MICRO-RESOURCE RULES

The micro-resource microformat does not specify what tags can be used for a micro-resource, but only constrain the relations between the tags in a HTML document according to the data model depicted in Figure 3. By default, it assumes all resource and property tags in the `class` attribute and all method tags in the `rel` attribute, because almost all microformats use these two tags exclusively. Due to space limit, we only list the main rules of micro-resource tagging below:

Definition 1: Main Rules of micro-resource.

---

declaration: the micro-resources to occur in the document are declared in a HTML `<meta>` header element:
`<meta name="resource" content="list of tags" />`

resource: a tag `R` in a `class` attribute of element `P` is treated as a micro-resource if `R` is declared so.

children: any child element of `P` tagged as a

micro-resource is treated as a child resource of R.

properties: any child element of P has a class tag but is not a declared micro-resource is treated as a property of R or a child property. The value of a property can be 1) href attribute of <a> element; 2) value attribute of an <input> element; or 3) content of an element.

method: an element, P or its child, with rel attribute is treated as the methods of micro-resource R. The URI of the element, including the <a href=URI> and <form action=URI> elements, becomes the URI of the methods.

form: a form can be tagged as a micro-resource and its fields as the properties of the micro-resource. Only information useful for filling the fields and submitting the form is collected based on the following template:

```
<form class={resource}
method={method} action={uri} >
   <input|textarea|select
name={input} type={type}
value={value} class="{property}
required?" lang={lang}
maxlength={maxlength} />+
</form>
```

These rules essentially map a HTML page with micro-resource tags into a tree, where each node is a micro-resource decorated with properties and methods. We believe majority of HTML pages can be mapped to such micro-resource trees. For example, a Google search result page, if tagged with micro-resource, can be rendered into a micro-resource tree like the following, where each directory /R represents a micro-resource R whose properties and methods are highlighted inside the square brackets:

```
/result
  /search [input: q, POST: uri]
  /matches
       /match [preview: text, GET: uri]
            /cached [GET: uri]
            /similar [GET: uri]
       …
  /more
       /page [index: 1, GET: uri]
       …
       /next [GET: uri]
  /related
       /term [term: text, GET: uri]
       …
```

Figure 6: Micro-resource tree for Google search results.

5 AGGREGATION AND COMPOSITION

Microformats can be aggregated and composed. Aggregation is the process that merges the semantic data from the same or different HTML documents. For example, we can aggregate XFN microformats (microformat.org) in a person's home page and work page to infer his social relationships. Composition is the process that builds a microformat from other ones. For example, hReview microformat is built upon three microformats: 1) hCard that designates the person who reviews an item, 2) rel-tag that assigns a tag to the item being reviewed; and 3) rel-license the associates a license with the review.

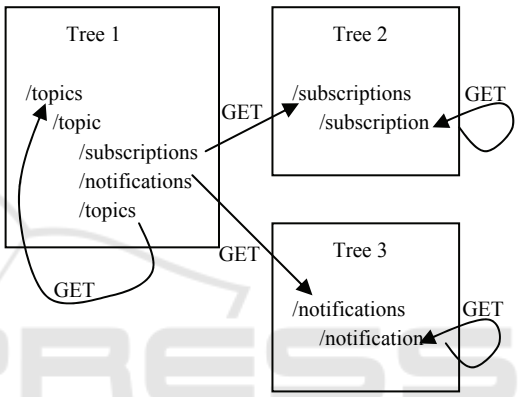

Figure 7: Part of the aggregated micro-resource map for a dual web service.

Micro-resource can be aggregated and composed in the same way. The micro-resource trees obtained from the HTML pages of a web site can be aggregated into a *micro-resource map*, where the nodes in individual micro-resource trees are linked based on methods of the micro-resources. For example, Figure 7 shows a portion of the cross-linked micro-resource map for a web service we developed. For clarity, only the GET links are displayed for each micro-resource. This micro-resource map provides a snapshot of the web services that can help developers to find entry points to the services and how to consume them accordingly.

Aggregation can be achieved in several ways. For example, a web crawler that recognizes the micro-resource and other microformats can search the HTML documents and parses the microformats in them. A plug-in to the browser can also be used to parse the retrieved microformats while a developer interacts with the service.

Micro-resource can also be composed with other existing microformats. We can declare that hCard,

hCalendar, hAtom, hReview, or any proper microformat as micro-resource, using the declaration rule in Definition 1. In such composition, each composed microformat data model is treated as a "super class" of the micro-resource data model in Figure 3. In other words, the micro-resource data model extends the composed microformat (hCard for instance) with additional data, including property attributes and methods. This inheritance relation is illustrated in Figure 8.

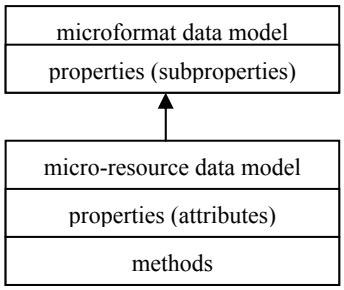

Figure 8: Composition of micro-resource as inheritance.

Such composed data model can be built by coordinating multiple microformat parsers. Given a HTML document, we can activate the parsers for declared micro-resources. Once the root class of a microformat is detected (vcard for hCard, for example), the parser for that microformat is used to process the corresponding HTML section to build one data model. The micro-resource parser is also used to process the same section to build another data model. The two models are then merged into one.

```
<html>
<head>
   <meta name="resource" content="vcard" />
</head>
<body>
<form class="vcard" action="..." rel="PUT">
 <ol>
 <li>Your name: <span class="fn">Li Li</span></li>
 <li class="email">
    Your <span class="type">work</span> email: <input
type="text" name="work-email" class="value"
value="lli5@avaya.com" />
 </li>
 <li class="tel">
   Your <span class="type">home</span> phone number:
   <input type="text" name="home-phone" value="5555"
class="value required"/>
 </li>  </ol></form></body></html>
```

Figure 9: A form tagged as vCard micro-resource.

To illustrate this process, Figure 9 is a small HTML document in which a form used to update a person's contact is tagged as a hCard which is also declared as a micro-resource. For clarity, micro-resource tags are in bold font and vCard tags are in italics. Notice that the `<input>` element is tagged by both hCard property `tel` and micro-resource property `required`. The merged micro-resource JSON representation is shown in Figure 10.

```
{
  "PUT": {"_uri": "..."},
  "_type": "vcard",
  "email":  {
       "type": {"_value": "work"},
       "value": {
          "_input": "work-email",
          "_type": "text",
          "_value": "lli5@avaya.com"
       }
  "fn": {"_value": "Li Li"},
  "tel": {
    "type": {"_value": "home"},
    "value": {
      "_input": "home-phone",
      "_required": true,
      "_type": "text",
      "_value": "5555"
    }
  }
```

Figure 10: The micro-resource composed with vCard.

6 APPLICATION AND EXPERIMENTAL RESULTS

We have tagged a web service we developed for event notifications with the micro-resource framework. The tagging was made easy because it is done on the HTML templates that generate all the HTML pages. We also developed a prototype general-purpose micro-resource parser based on JAVA SAX parser. The micro-resource framework was used to implement more complex web services based on the primitive ones.

One such service is to move the subscriptions of one topic to another topic such that the notifications can be redirected to the new topic. This composed service is hosted on a redirect resource that mediates interactions between three types of resources distributed across different web sites. A highlight view of these interactions is illustrated in Figure 11, where micro-resource is used to retrieve data from HTML pages to update the involved resources.

We tested the performance of the micro-resource parser on several sample HTML files, and the experimental results, on a Windows 2003 Server with dual 3 Ghz CPU and 2 GB RAM, are summarized in Table 1. The results show that the performance is acceptable and that parsing time increases slowly compared to the file size and tag number increases.

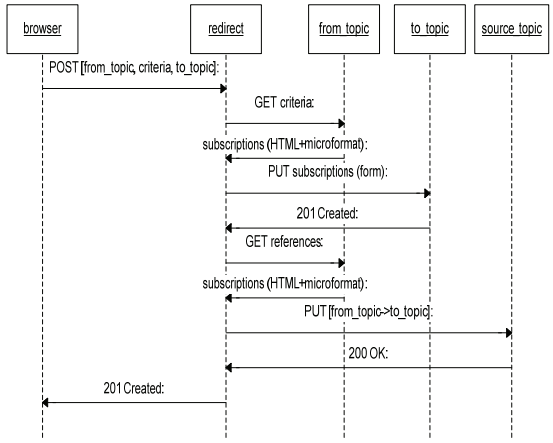

Figure 11: Service composition with micro-resources.

Table 1: Performance of micro-resource parser.

File size (B)	No. of tags	Avg. Time (ms)
349	6	7
1943	15	10
2041	11	14
3662	19	14

7 CONCLUSIONS

This paper presented a microformat framework, micro-resource, based on the fundamental principles of REST, to support the development of dual web services. For this purpose, we compared this approach against others, including alternative parallel web services, and described the framework, rules, aggregation and composition of micro-resources with illustrative examples. We implemented a prototype parser and demonstrated its performance for service composition. Our study showed that micro-resource framework has the potential to narrow or even close the gap between human and machine web services such that the full power of RESTful web services can be reached out to applications in both environments with minimum investment from service consumers and providers.

REFERENCES

AWWW 2004: Architecture of the World Wide Web, Volume One, W3C Recommendation 15 December 2004, http://www.w3.org/TR/webarch/

eRDF 2006: Embeddable RDF, http://research.talis.com/2005/erdf/wiki/Main/RdfInHtml

Fielding, Roy, *Architectural Styles and the Design of Network-based Software Architectures*, Ph.D. Dissertation, 2000, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

GRDDL 2007: Gleaning Resource Descriptions from Dialects of Languages (GRDDL), W3C Recommendation 11 September 2007, http://www.w3.org/TR/grddl/

Hadley, Marc, Web Application Description Language (WADL), http://research.sun.com/techrep/2006/abstract-153.html

Khare, Rohit, Microformats, the Next (Small) Thing on the Semantic Web? *IEEE Internet Computing*, vol. 10, no. 1, pp. 68-75, Jan./Feb. 2006.

Kopecky, J., Gomadam, K., Vitvar, T., hRESTS: an HTML Microformat for Describing RESTful Web Services, 2008 *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, pp. 619-625, 2008.

Khare, R., Celik, T., Microformats: a Pragmatic Path to the Semantic Web, *WWW 2006*, pp. 865-866, 2006.

Li, L., Chou, W., Infoset for Service Abstraction and Lightweight Message Processing, *ICWS 2009*, pages 703-710, Los Angeles, July 2009.

microformat.org: http://www.microformats.org

RDFa 2008: RDFa in XHTML: Syntax and Processing, A collection of attributes and processing rules for extending XHTML to support RDF, W3C Recommendation 14 October 2008, http://www.w3.org/TR/rdfa-syntax/.

REST microformat: http:/www.micrformats.org/wiki/rest

REX: http://microformats.org/wiki/rex-proposal-preso

Richardson, L. Ruby, S., *RESTful Web Services*, O'Reilly Media, Inc. 2007.

Sitemap 0.90, http://www.sitemaps.org/.

Stolley, Karl, Using Microformats: Gateway to the Semantic Web Tutorial, *IEEE Transactions on Professional Communications*, Vol 52, No. 3, pp. 291-302, 2009.

Williams, Peter: RESTful Service Discovery and Description, http://barelyenough.org/blog/2008/01/restful-service-discovery-and-description/, 2008.