Efficient Runtime Service Discovery and Consumption with Hyperlinked RESTdesc

Ruben Verborgh*, Thomas Steiner†, Davy Van Deursen*, Rik Van de Walle*, Joaquim Gabarró Vallés†
*Ghent University – IBBT, ELIS – Multimedia Lab, B-9050 Ledeberg-Ghent, Belgium
Email: {ruben.verborgh, davy.vandeursen, rik.vandewalle}@ugent.be
†Universitat Politècnica de Catalunya, Department LSI, 08034 Barcelona, Spain
Email: {tsteiner, gabarro}@lsi.upc.edu

***Abstract*—Hyperlinks and forms let humans navigate with ease through websites they have never seen before. In contrast, automated agents can only perform preprogrammed actions on Web services, reducing their generality and restricting their usefulness to a specialized domain. Many of the employed services call themselves RESTful, although they neglect the hypermedia constraint as defined by Roy T. Fielding, stating that the application state should be driven by hypertext. This lack of link usage on the Web of services severely limits agents in what they can do, while connectedness forms a primary feature of the human Web. An urgent need for more intelligent agents becomes apparent, and in this paper, we demonstrate how the conjunction of functional service descriptions and hypermedia links leads to advanced, interactive agent behavior. We propose a new mode for our previously introduced semantic service description format RESTdesc, providing the mechanisms for agents to consume Web services based on links, similar to human browsing strategies. We illustrate the potential of these descriptions by a use case that shows the enhanced capabilities they offer to automated agents, and explain how this is vital for the future Web.**

***Index Terms*—automated agents; hypermedia links, Semantic Web; service consumption; service discovery**

I. Introduction

Ironically enough, even the end of the Internet[1] has an exit: by means of a form that allows users to refer people to the page, they can leave it. Hyperlinks and forms are the backbone of the way how we navigate the Web. With the REST architectural style, this backbone can also be used for Web services, sometimes synonymously referred to as Web APIs. Unfortunately, many of today's so-called RESTful APIs are little more than just HTTP interfaces because they omit these link relationships in their responses, neglecting the hypermedia constraint as defined by Roy T. Fielding [1]. The absence of such links forces clients to be aware of how the server functions before they can use it, limiting their horizon to what they already know. In contrast, the omnipresence of hyperlinks and forms on the human Web enables us to easily find out where to go next, discovering on-the-fly which of the possible next steps might be the most promising one. Striving for machines to reach the same level of autonomy has been an early idea of Semantic Web and service architects.

Although significant progress was made in different areas, the vision [2] of automated agents consuming Web services has not been fulfilled in the first decade of the Semantic Web. Leading researchers in this area, such as Jim Hendler [3], remark that autonomous agents are still missing, while the infrastructure for intelligent systems is now in place. However, the Achilles' heels of Semantic Web services are service descriptions, as they form the anchor points where automated discovery and consumption start. Several description formats exist, each with specific benefits and—often more cited—drawbacks, but there has never been broad acceptance in the field of service discovery and consumption.

In general, description formats provide detailed characterization of input and output parameters, but the *functionality* of the Web service is only vaguely captured or oftentimes completely missing, forcing us to rely on implicit semantics. This can be explained by the industry's initial concern, which was to provide descriptions that could integrate Web services into traditional software platforms. In practice, this meant that humans selected a specific service, against which the application was compiled. Two obvious problems appear: i) choosing the right service requires *manual intervention*, severely limiting the application's possibilities, and ii) if the service description *changes*, the application has to be recompiled—and possibly altered. We argue that a description focusing on functionality provides a highly efficient way to solve these two problems, and is subsequently the best choice for runtime service discovery and consumption.

This paper takes Web services to the next level and allows agents to act without a fully-defined plan upfront. We explain how hypermedia links can be used in conjunction with service descriptions, enabling agents to discover their options at runtime. We aim to provide possibilities similar to those of humans browsing the Web: given a starting point, people interpret documents and navigate intelligently through pages via hyperlinks, surfing across information as it arrives, following their noses to find what they need. Agents that possess similar capabilities become versatile Web consumers, capable of autonomously finding more information and performing more advanced tasks, without predefined knowledge about a specific server and its modalities.

The remainder of this paper is structured as follows: Section II provides an introduction to the description format, its syntax and usage. We continue this paper with related work in Section III. Section IV introduces hypermedia links in service

[1]Located at http://www.endoftheinternet.com/.

```
@prefix dbpedia: <http://dbpedia.org/ontology/>.
@prefix http: <http://www.w3.org/2006/http#>.
@prefix tmpl: <http://purl.org/restdesc/http-template#>.

{
  ?book dbpedia:isbn ?isbn.
}
=>
{
  _:request http:methodName "GET";
            tmpl:requestURI ("/books/" ?isbn);
            http:resp [ tmpl:represents ?book ].
  ?book dbpedia:title ?title.
}.
```

Listing 1. RESTdesc description of a book title service

```
_:request http:methodName "GET";
          http:requestURI "/books/1-57322-245-3";
          http:resp [ tmpl:represents _:mybook ].
_:mybook dbpedia:title _:title1.
```

Listing 2. Combining the book service description with an ISBN code

descriptions, followed by the provisioning of agents that can autonomously consume such services in Section V. Section VI concludes the paper and outlines future work.

II. Introduction to RESTdesc

The description method we use is RESTdesc [4], an RDF-based [5] notation focused on the specific capabilities of a service, instead of its parameters and modalities[2]. The goals for RESTdesc are i) to have a functionality-centered and formally defined service description ii) using Semantic Web technologies iii) that emphasizes simplicity and elegance. The theoretical foundations have been presented previously, so we will limit ourselves to a brief introduction. RESTdesc descriptions are expressed in Notation3 (N3, [6]), an RDF superset that adds support for graphs and quantification, enabling concise and practical descriptions without resorting to additional meta-level constructs for semantics and functionality. As a result, RESTdesc descriptions do not require explicit mentions of services, inputs, nor outputs to express functionality.

Let us consider the example of a service that returns a book title based on its ISBN code. Listing 1 shows a possible RESTdesc description. The identifiers prefixed by a question mark `?` are universally quantified variables, the braces `{}` enclose graphs and the arrow `=>` represents implication. A literal English translation could be "*if* you know a book's ISBN code, *then* an HTTP request to `/books/{ISBN}` will return a representation of this book, including its title". This is the denotational, *logic* meaning of the description.

In addition, Notation3 entails also an *operational* semantics, called N3Logic [7]. This enables agents to *execute* the description as a rule. In this concrete example, if the agent knows the ISBN code of a book:

```
_:mybook dbpedia:isbn "1-57322-245-3". ,
```

it can combine this statement and the description via a reasoner and obtain the knowledge displayed in Listing 2 (namespace declarations omitted for brevity). The reasoner has also turned the URI template into a concrete URL, because all elements were available. The agent can subsequently perform the HTTP request and use the result to complete its knowledge base:

```
GET /books/1-57322-245-3 HTTP/1.1
```

This shows how RESTdesc functions and what it is capable of. Several aspects of RESTdesc descriptions are noteworthy:

- They do not introduce any new terminology, but reuse existing concepts such as N3 and URI templates [8]. As a result, the service's functionality is described directly, *i.e.*, it does not require a meta-level description for parameters, settings, and the like.
- There is no need to specify types, since they can be inferred from ontological knowledge, *e.g.*, in the concrete case from the example before, from the domain and range of `dbpedia:isbn` and `dbpedia:title`.
- Descriptions are self-contained and self-explaining, not requiring additional semantics apart from those delivered by the N3 language.
- They immediately describe the HTTP request required to execute the operation.

While this last point is a benefit in terms of simplicity—which is why we introduced it in the first place—it also couples functional description and execution. What if a client cannot determine beforehand which request and URI it will need? This paper tackles exactly this point and presents an enhanced mode of RESTdesc based on hypermedia links. That way, clients do not need to know in advance what HTTP requests they should issue, but can determine them just in time, adapting to information and links in server responses. This connectedness of services allows clients to be programmed in a generic way without relying on implementation details of servers [9, p. 223–227], making them substantially more powerful and versatile.

III. Related Work

A. RESTful Web Services

As the name indicates, RESTdesc descriptions assume the underlying service has a *RESTful* architectural style, as outlined by Fielding and Taylor [1], referred to as *REST API*. Services that follow these principles employ a resource-oriented model where each resource is uniquely identified by a URI. All operations on these resources use the standard HTTP methods (`GET`, `POST`, `PUT` ...) with their respective correct meaning as originally defined in the HTTP protocol [10]. This contrasts with other techniques and applications that either use HTTP as an envelope protocol (*e.g.*, SOAP [11]) or incorrectly apply HTTP methods.

The simplicity and uniformity of the HTTP architecture makes REST APIs particularly good candidates for automated consumption, in addition to manual browsing. The well-defined properties of the standard HTTP methods, such as (un-)safeness and (non-)idempotence, enable to reason about the results and side-effects of actions.

[2]Online documentation is available at http://restdesc.org/.

Note, however, that adhering to the correct RESTful use of the HTTP protocol is not a hard requirement for RESTdesc. Rather, creating descriptions for services that do otherwise will involve a larger amount of effort.

B. Universal Description, Discovery, and Integration

Universal Description, Discovery, and Integration (UDDI, [12]) was an early attempt at making both public and private Web services available by acting as a service broker. UDDI provides constructs at organizational, service-classifying, and technical level, but never widely found adaption. Its main criticisms are that it does not support truly automated service matching [13], and that it endorses tight coupling, making change problematic [14]. Furthermore, it was designed with the concepts of one service type in mind (SOAP), so it cannot function in a heterogenous service environment.

C. Traditional Service Descriptions

In the early years, service descriptions and the Web Service Description Language (WSDL, [15]) were virtually synonyms, or at least treated that way. However, it has now become apparent that WSDL is not the appropriate format to deal with today's service challenges on the Web. While in theory, version 2.0 could be used to describe REST APIs [16], WSDL has mainly been applied for SOAP services and therefore remains associated with it. Also, it suffers from the same verbosity as version 1.0 and does not contribute to an understanding of the functionality of the service, focusing on the technical and implementational aspects in a non-semantic way.

The Web Application Description Language (WADL, [17]) was designed from the ground up to describe RESTful services. Yet, it also neglects the functional aspect, disabling runtime service discovery, in contradiction to the needs of parts of the REST community [18]. For example, it is insufficient to describe a book author lookup service as *"a service with a book's ISBN number as input and a person name as output"*. Does this person represent a reader of the book, the editor, or maybe even the subject? A human has to verify this in advance and hardwire the service where an author lookup is required.

D. Semantic Service Descriptions

A first step towards semantics was offered by Semantic Annotations for WSDL (SAWSDL, [19]). Later on, an adaptation specifically for RESTful services was proposed [20]. However, SAWSDL is only concerned with giving a semantic definition to the input and output parameters of services, rather than connecting them in a functional way. Being an extension of WSDL, it inherited all other drawbacks.

Meanwhile, the Semantic Web was bridging the gap from the other side and proposed formats such as OWL [21] for Services (OWL-S, [22]), an ontology for modeling Web services. While OWL-S still focuses somewhat on input and output parameters (albeit with semantics), it allows to describe functional relations using a variety of expression languages (SWRL, DRS, and KIF by default, others are possible). Unfortunately, these expressions are unintegrated and form a separate layer which – unless interpreted explicitly by a supported toolset – is ignored.

The Web Service Modeling Ontology (WSMO, [23]) is an alternative to OWL-S. Although they share the same goals, substantial differences between both approaches exist [24]. The most relevant difference for our purpose is the logical expressivity. While OWL-S provides the aforementioned different expression languages, a drawback from the interoperability viewpoint, WSMO employs a single family of layered logic languages [25]. However, when expressed in RDF syntax, WSMO expressions become similarly unintegrated and hence not self-descriptive. The same holds for WSMO-Lite [26], which extends SAWSDL with conditions and effects.

IV. RESTdesc with Hypermedia Links

A. About Hypermedia Links

One of the fundamental properties of REST is the so-called hypermedia constraint, which basically can be summarized as the constraint that each server response should contain the possible next steps the client can take, since the client maintains the application state. As Fielding points out, HTTP APIs must be driven by hypertext, which is *"the simultaneous presentation of information and controls such that the information becomes the affordance through which the user (or automaton) obtains choices and selects actions*" [27]. He clarifies, however, that hypertext and hypermedia links go beyond specific technologies such as HTML. Concretely, this means that a service can provide links in a machine-understandable format for agents to follow.

One interesting possibility is to put hypermedia links in the HTTP response's `Link` headers, following the proposed standard Web Linking [28] by Mark Nottingham. It is equivalent to the `link` element in HTML, allowing to specify various properties, including the relationship type of the link. The added value of `Link` headers is that they allow hypermedia links within other content types such as text, image, or audio.

For example, the book resource described in Listing 1 might link to a page with information about the book's authors. When a client accepts `image/*` and performs an HTTP `GET` of `/books/1-57322-245-3`, the server could return a JPEG image of the book cover. To indicate next steps, the server can add accompanying link headers:

```
Link: /authors/khosseini; rel="author"
```

It is apparent that the above URI cannot be determined beforehand by the client. The flexibility of the `Link` header mechanism allows the server to specify any URI, so the agent does not need to know how to construct it. While humans can now decide whether or not to follow the link, the `author` relation does not convey meaningful information for automated decision support. In order to provide semantics for non-human agents as well, the server could choose to return an ontology property instead:

```
Link: /authors/khosseini;↩
      rel="http://dbpedia.org/ontology/author"
```

This relationship can have a meaning to the client, who can now employ it for automated reasoning and decision making.

```
{
  ?book dbpedia:isbn ?isbn.
}
=>
{
  _:request http:methodName "GET";
          tmpl:requestURI ("/books/" ?isbn "/authors");
          http:resp [tmpl:representsMultiple ?author].

  ?book dbpedia:author ?author.
}.
```

Listing 3. Coupled RESTdesc description for book authors

```
{
  ?book a dbpedia:Book.
}
=>
{
  ?book dbpedia:author _:author;
        dbpedia:subject _:subject;
        dbpedia:publisher _:publisher;
        [...].
}.
```

Listing 4. Linked RESTdesc description for book authors

B. Incorporating Links in RESTdesc Descriptions

The initial RESTdesc document already suggests including `Link` headers into the response. It does however not actively use them, because all information required to construct hyperlinks is contained in the description itself—limiting its application to cases where construction in advance is possible. The fact that RESTdesc is a combination of Notation3 and HTTP `OPTIONS` allows its extension with new concepts such as support for hypermedia links to RESTdesc.

As an example, Listing 3 shows how the functionality of the authors resource can be expressed in "traditional" RESTdesc. The descriptions tell us how to take the ISBN code of a book and construct the URI to retrieve the authors. In contrast, the linked RESTdesc description in Listing 4 does not need any properties to construct the URL.

At first sight, it seems unclear why Listing 4 contains a full description, since it appears to be redundant and at the same time lacking necessary information. Redundant, because it *seems* to specify only information already present in the ontology (*e.g.*, we already know that books can have an author). Lacking necessary information, because there is no explicit mention of an HTTP request as in Listing 3. However, Listing 4 does contain all important information without redundancy, as we explain below.

The key to understanding this description is its operational semantics. It does *not* redundantly repeat the ontological assertion that books have an author property. Rather, the description needs to be interpreted as *"if you know a book, than you can also know its author"*[3]. This statement is true, because the service itself is able to fulfill the conclusion. The client can use the operational semantics of the description as a rule in a reasoning process. When it comes to executing this rule, it should get the results from the service.

This brings us to the second aspect: how to construct an HTTP request from the description. It seems that the necessary information is missing, while it is actually *implied*. According to the Linked Data principles [29], RDF URIs need to be *dereferencable*, which means that an HTTP `GET` request to them should return useful information about the concept they identify. Since the server supplies author URIs, it can choose a URI that is dereferencable. So if a client follows the principles, it knows it should execute the following request:

```
GET /authors/khosseini HTTP/1.1
```

This dereferencing implied by Listing 4 can be specified explicitly by the server, as shown in Listing 5. While unnecessary for simple `GET` requests, it is required to express the functionality of other HTTP methods such as `PUT` and `POST`.

```
{
  ?book dbpedia:author ?author.
}
=>
{
  _:request http:methodName "GET";
            http:requestURI ?author;
            http:resp [tmpl:represents ?author].
}.
```

Listing 5. Derefencing: implied semantics of the author relationship

C. Functional Nature of RESTdesc

The two-part description clearly illustrates the *decoupling* of addressing (supplied by the server) and functionality (supplied by the server and interpreted by the client). As a benefit, the descriptions remain valid when the URI scheme changes for any reason. (Although in a good architecture, old URIs remain accessible [30].) Moreover, functional descriptions become possible in situations where the concrete URI is not known in advance, which enables more powerful and flexible service consumption.

In Listing 3, the functional aspect of RESTdesc is apparent: the description substantiates the `author` relationship, connecting the result of an HTTP request to the author of a book. For Listings 4 and 5, the functional connection might be less obvious because of the decoupling. The true functionality however, lies in the definition of the link itself: Listing 4 describes the expected links, and each link (*e.g.*, `author`) is defined by its ontology.

The functional nature of RESTdesc becomes more explicit if we describe the effects of non-safe HTTP methods. Listing 6 describes the submission of a book review by a user. Through HTTP headers, the client disposes of a link to the book review form. The N3 rule in Listing 6 explains the result of a post action: the book will have a review with the text supplied in the request body. Again, the functionality is expressed by means of a link (*e.g.*, `review`).

[3]The server explicitly states which data it makes available. In Listing 4, the server also exposes subject and publisher information, but not editor or illustrator properties, although these latter two are also part of the ontology.

```
@prefix book: <http://example.org/book#>.
{
  ?book book:reviewForm ?reviewForm.
}
=>
{
  _:request http:methodName "POST";
            http:requestURI ?reviewForm;
            http:body [tmpl:formData ("text=" ?text)];
            http:resp [tmpl:represents ?review].

  ?book book:review ?review.
  ?review book:reviewText ?text.
}.
```

Listing 6. Linked RESTdesc description for submitting a book review

D. Discovery and Composition

Currently, we require the service provider to produce the links and descriptions. From there onwards, clients can start discovering a server by issuing HTTP `OPTIONS` requests. We could, however, imagine a more collaborative environment in which different parties can supply and exchange links and descriptions. This is left as future work.

For service composition, we can rely on existing N3 reasoners ([31], [32]), some of which are known to deliver impressive performance [33]. To solve a composition problem, the input, desired output characterization, and the obtained service descriptions are passed to the reasoner. The reasoning process will subsequently try to find a path from the input to the desired output by triggering the N3 rules of the descriptions. By examining this path, the client can determine which HTTP requests it has to issue on the server [34].

V. Providing for Interactive Agents

A. Operational Modes for RESTdesc Agents

With the coupled RESTdesc variant, there are roughly two modes in which agents can operate:

- *Explorative* — the agent starts with a set of preconditions, which recursively triggers descriptions. In the end, it obtains a set (or finite subset) of all precalculable possible steps it can take. Example: starting with a book's ISBN code, an agent finds how to get its author names, how to look up biographies of those authors *etc*.
- *Responsive* — the agent starts with a set of preconditions and a query (possibly issued by a user) which indicates a desired postcondition. In most cases, this results in a targeted subset of the explorative method for the same precondition, but it can be helpful as a selection step or in case the set of possible steps is infinite. Example: starting with a book's ISBN code, find all positive reviews.

What both of the above techniques have in common is that the actions are determined in advance. Stated otherwise: the agent can only plan actions whose expected results are known beforehand.

`Link` headers work the other way round: possible future actions are communicated when accessing or manipulating a resource. This constitutes the essence of RESTful services: the current application state is maintained by the client. The server communicates this state in terms of possible next steps, and it is up to the client to decide where to go. Our introduction of linked RESTdesc thus enables a third way of operation, the *interactive* mode, which is also goal-driven like the responsive mode, but does not start with a fully determined plan. Furthermore, agents can switch between different modes when solving subproblems.

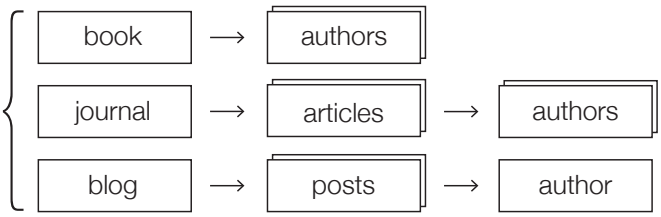

Fig. 1. Structure of the example library Web server

B. Reconciling Linked Descriptions with Planning in Advance

We will first introduce an example query for an agent. Suppose a digital library Web server exists which contains information about books, journals and blogs; the structure of which is shown in Fig. 1. We are interested to find the names of all authors that have written about the Semantic Web.

As humans, we could start this task by using the search engine of the server for the topic "Semantic Web". We expect to find books, journals, and blogs in the result list. For the books, we just take the authors. For the journals, we open the individual articles and collect their author names. Finally, we visit the blogs and gather the author name of each post. We end up with the required list of names.

When making the server accessible to machines, the first step is to express link relationships. This process is straightforward: books link to authors, journals link to articles, which link to authors, *etc*. Furthermore, a search results page should link to each result.

Then, we proceed by adding RESTdesc descriptions (similar to the one of Listing 4) for books, journals, and blogs. These descriptions convey the *expectations* humans have about the website: that a book will have authors, that a journal will have articles. Another important notion is that the result page returns books, journals, or blogs. It is exactly this expectation that directed us to the search function in the first place, together with the expectation that each of the results would lead to authors. We can translate this in an ontological property: we state that a search result contains works, and (using `owl:oneOf`) that a work can either be a book, journal, or author. When the agent receives a concrete result link, it is able to determine the actual type of the work and take the appropriate action depending on this type.

C. Behavior of Reactive Agents

We will now explain how reactive agents that use linked RESTdesc can be implemented, continuing the library query

example. At the start, the client disposes of a list of entry points (URIs of servers) and a query, for instance[4]:

```
@prefix db: <http://dbpedia.org/resource/>.
@prefix dbpedia: <http://dbpedia.org/ontology/>.
{
  _:work dbpedia:subject db:Semantic_Web;
         dbpedia:author ?author.
}
=>
{
  ?author a <#SemanticWebAuthor>.
}.
```

The agent first tries to extract as much information as possible from this query. In this case, it can dereference the resource `dbpedia:Semantic_Web` to look up more information, such as its English label *"Semantic Web"*.

The next step for the agent is to discover what functionality the servers offer. Note that the agent can actually perform this step in advance, before it answers any query. Every known server receives an HTTP `OPTIONS /` request, to which it responds with RESTdesc descriptions and/or link headers to other pages that can subsequently be requested with HTTP `OPTIONS`. For example, the root `/` could link to `/books`, and an HTTP `OPTIONS` to `/books` could return the description from Listing 4. That way, the agent recursively builds up an understanding of each resource's capabilities.

Then, the agent composes an execution plan using a reasoner. The input consists of the information derived from the query and the RESTdesc descriptions. When the query is issued, the reasoner activates the operational semantics of the RESTdesc descriptions, which are in fact N3 rules. Using backwards-chaining, it determines which rules it should activate to satisfy the query clause. This activation pattern corresponds to the HTTP requests that should be executed to obtain the desired result.

The execution plan actually forms a directed graph whose edges indicate dependencies, governing the execution order. For this example, the reasoner would devise a preliminary plan that starts with a search, and for each result proceeds to find the author. How this author is found, depends on the result type (book, journal, blog). Since there are no inter-dependencies, the result explorations can be executed in parallel.

The agent then starts consumption by issuing a search. The server returns the search results, placing typed link relationships either in the HTTP headers, the (HTML or RDF) response, or both. If the result list spans multiple pages, the agent can follow links to navigate them and continue similarly. Each of the results is stored in a list.

Meanwhile, for every result, the agent issues an HTTP request to look up more information. For a book, it simply extracts the authors information. A journal requires finding the individual articles first, which also lead to authors. Similarly, blogs contain posts which have an author. All this information is put in a second list.

[4]While expressed as a so-called Notation3 filter or transformation rule here, the query could also be a SPARQL query or similar.

Finally, the agent goes through all authors on this list and executes the query, delivering the requested results: a graph of `SemanticWebAuthor`s. Remarkable here is that the agent did not have any preprogrammed knowledge about authors, books, or this specific server. It is a general-purpose client that, given RESTdesc descriptions, was able to fulfill this task similar to how humans would browse the Web: by having a high-level plan and following low-level hyperlinks.

VI. Conclusion and Future Work

In this paper, we proposed an extension to RESTdesc that allows automated agents to react on hypermedia links. We explained how services can be described and how these descriptions can be linked together. Furthermore, we detailed how agents can answer queries by following their nose, similar to the way humans browse the Web for information.

Essential in our approach is that the client does not need any specific knowledge about the server or even the topic. All this information is discovered at runtime through standard use of the HTTP protocol. The information contained in the RESTdesc descriptions can be extended with ontological and instance knowledge, obtained by dereferencing resource URIs.

It should be noted that this type of targeted active information discovery is substantially different from, for example, SPARQL querying. While the example query could indeed be posed to a regular SPARQL endpoint, such an endpoint is not always available, and even if it is, it lives separated from the human Web consisting of pages. Also, the proposed approach of link browsing works across different domains and is far more flexible than queries, which have a rigid structure. This is demonstrated with the blogs in the example where the agent has to look up posts on an external website to find the author. Furthermore, agents are not limited to pure information retrieval. In previous work, we demonstrated how RESTdesc agents can also execute tasks with side effects (*e.g.*, image uploading), something that clearly goes beyond SPARQL functionality [4].

Concluding, we can state that RESTdesc has a strong potential in the field of service description, automatic discovery, and consumption. Based on RESTful principles and in particular the hypermedia constraint, it targets modern, resource-oriented websites and focuses on the resources and their functional relationships instead of technical properties.

Future work includes the application of RESTdesc technologies to different fields and applications, and versatile error handling and recovery based on HTTP status codes. We plan to provide a public implementation of the reasoning framework for use as a black box, so intelligent agents can employ RESTdesc composition techniques transparently. Another interesting area is the collaboration and integration of different services, for example using ontology matching. Furthermore, the development of even more intelligent agents will offer exciting challenges, such as decision optimization when multiple alternative solution paths exist.

ACKNOWLEDGMENT

The research activities as described in this paper were funded by Ghent University, the Interdisciplinary Institute for Broadband Technology (IBBT), the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT), the Fund for Scientific Research Flanders (FWO Flanders), and the European Union. This work was partially supported by the European Commission under Grant No. 248296 FP7 I-SEARCH project. J. Gabarró is partially supported by TIN-2007-66523 (FORMALISM), and SGR 2009-2015 (ALBCOM).

REFERENCES

[1] R. T. Fielding and R. N. Taylor, "Principled design of the modern Web architecture," *ACM Transactions on Internet Technology*, vol. 2, no. 2, pp. 115–150, May 2002.

[2] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, vol. 284, no. 5, pp. 34–43, May 2001.

[3] J. Hendler, "Why the Semantic Web will Never Work." Presented at the 7th Extended Semantic Web Conference (ESWC 2011), Crete, Greece, May 2011. [Online]. Available: http://www.slideshare.net/jahendler/why-the-semantic-web-will-never-work

[4] R. Verborgh, T. Steiner, D. Van Deursen, R. Van de Walle, and J. Gabarró Vallés, "Description and Interaction of RESTful Services for Automatic Discovery and Execution," in *Proc. FTRA International Workshop on Advanced Future Multimedia Services*, submitted.

[5] G. Klyne and J. J. Carrol. (2004, Feb.) Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation. [Online]. Available: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

[6] T. Berners-Lee and D. Connolly. (2011, Mar.) Notation3 (N3): A readable RDF syntax. W3C Team Submission. [Online]. Available: http://www.w3.org/TeamSubmission/n3/

[7] T. Berners-Lee, D. Connolly, L. Kagal, Y. Scharf, and J. Hendler, "N3Logic: A logical framework for the World Wide Web," *Theory and Practice of Logic Programming*, vol. 8, no. 3, pp. 249–269, 2008.

[8] J. Gregorio, R. Fielding, M. Hadley, and M. Notthingham. (2010, Mar.) URI Template. [Online]. Available: http://tools.ietf.org/html/draft-gregorio-uritemplate-04

[9] L. Richardson and S. Ruby, *RESTful Web services*. O'Reilly, 2007.

[10] R. T. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. (1999, Jun.) Hypertext Transfer Protocol – HTTP/1.1. [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt

[11] M. Gudgin, M. Hadley, N. Mendelsohn, and J.-J. Moreau. (2007, Apr.) SOAP Version 1.2. W3C Recommendation. [Online]. Available: http://www.w3.org/TR/2007/REC-soap12-part1-20070427/

[12] T. Bellwood, S. Capell, L. Clement, J. Colgrave, M. J. Dovey, D. Feygin, A. Hately, R. Kochman, P. Macias, M. Novotny, M. Paolucci, C. von Riegen, T. Rogers, K. Sycara, P. Wenzel, and Z. Wu. (2004, Oct.) UDDI Version 3.0.2. OASIS. [Online]. Available: http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm

[13] D. Kourtesis and I. Paraskakis, "Combining SAWSDL, OWL-DL and UDDI for semantically enhanced Web service discovery," in *Proceedings of the 5th European Semantic Web Conference (ESWC 2008)*, ser. Lecture Notes in Computer Science, S. Bechhofer, M. Hauswirth, J. Hoffmann, and M. Koubarakis, Eds., vol. 5021. Springer-Verlag, pp. 614–628.

[14] R. Levin. (2008, Oct.) Why UDDI Sucks. [Online]. Available: http://www.cio.com/article/451966/Why_UDDI_Sucks

[15] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. (2000, Sep.) Web Services Description Language (WSDL) 1.0. [Online]. Available: http://xml.coverpages.org/wsdl20000929.html

[16] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana. (2007, Jun.) Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C Recommendation. [Online]. Available: http://xml.coverpages.org/wsdl20000929.html

[17] M. Hadley. (2009, Aug.) Web Application Description Language. W3C Member Submission. [Online]. Available: http://www.w3.org/Submission/wadl/

[18] J. Gregorio. (2007, Jun.) Do we need WADL? [Online]. Available: http://bitworking.org/news/193/Do-we-need-WADL

[19] J. Kopecký, T. Vitvar, C. Bournez, and J. Farrell, "Semantic Annotations for WSDL," *IEEE Internet Computing*, vol. 11, pp. 60–67, 2007.

[20] M. Maleshkova, J. Kopecký, and C. Pedrinaci, "Adapting SAWSDL for Semantic Annotations of RESTful Services," in *Proceedings of the Confederated International Workshops and Posters on On the Move to Meaningful Internet Systems*, ser. OTM '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 917–926.

[21] D. L. McGuinness and F. van Harmelen. (2004, Feb.) OWL Web Ontology Language Overview. W3C Recommendation. [Online]. Available: http://www.w3.org/TR/2004/REC-owl-features-20040210/

[22] D. Martin, M. Burstein, J. Hobbs, and O. Lassila. (2004, Nov.) OWL-S: Semantic Markup for Web Services. W3C Member Submission. [Online]. Available: http://www.w3.org/Submission/OWL-S/

[23] H. Lausen, A. Polleres, and D. Roman. (2005, Jun.) Web Service Modeling Ontology (WSMO). W3C Member Submission. [Online]. Available: http://www.w3.org/Submission/WSMO/

[24] R. Lara, A. Polleres, H. Lausen, D. Roman, J. de Bruijn, and D. Fensel. (2005, Jan.) A Conceptual Comparison between WSMO and OWL-S. [Online]. Available: http://www.wsmo.org/2004/d4/d4.1/v0.1/20050106/

[25] J. de Bruijn, D. Fensel, U. Keller, M. Kifer, H. Lausen, R. Krummenacher, A. Polleres, and L. Predoiu. (2005, Jun.) Web Service Modeling Language (WSML). W3C Member Submission. [Online]. Available: http://www.w3.org/Submission/2005/SUBM-WSML-20050603/

[26] T. Vitvar, J. Kopecký, J. Viskova, and D. Fensel, "WSMO-Lite Annotations for Web Services In The Semantic Web: Research and Applications," in *Proceedings of the 5th European Semantic Web Conference (ESWC 2008)*, ser. Lecture Notes in Computer Science, S. Bechhofer, M. Hauswirth, J. Hoffmann, and M. Koubarakis, Eds., vol. 5021. Springer-Verlag, pp. 230–244.

[27] R. T. Fielding. (2008, Oct.) REST APIs must be hypertext-driven. [Online]. Available: http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

[28] M. Nottingham. (2010, Oct.) Web linking. [Online]. Available: http://tools.ietf.org/html/rfc5988

[29] T. Berners-Lee. (2006) Linked Data. [Online]. Available: http://www.w3.org/DesignIssues/LinkedData

[30] ——. (1998) Cool URIs don't change. [Online]. Available: http://www.w3.org/Provider/Style/URI

[31] ——. Cwm. [Online]. Available: http://www.w3.org/2000/10/swap/doc/cwm.html

[32] J. De Roo. Euler proof mechanism. [Online]. Available: http://eulersharp.sourceforge.net/

[33] T. Osmun. Euler Eye installation, demo, and deep taxonomy benchmark. [Online]. Available: http://ruleml.org/WellnessRules/files/WellnessRulesN3-2009-11-10.pdf

[34] R. Verborgh, D. Van Deursen, E. Mannens, C. Poppe, and R. Van de Walle, "Enabling context-aware multimedia annotation by a novel generic semantic problem-solving platform," *Multimedia Tools and Applications special issue on Multimedia and Semantic Technologies for Future Computing Environments*, 2011.