# Functional Descriptions as the Bridge between Hypermedia APIs and the Semantic Web

Ruben Verborgh
Ghent University – IBBT
ELIS, Multimedia Lab
Gaston Crommenlaan 8/201
9050 Ghent, Belgium
ruben.verborgh@ugent.be

Thomas Steiner
Universitat Politècnica de Catalunya
Department LSI
08034 Barcelona, Spain
tsteiner@lsi.upc.edu

Davy Van Deursen
Ghent University – IBBT
ELIS, Multimedia Lab
Gaston Crommenlaan 8/201
9050 Ghent, Belgium
davy.vandeursen@ugent.be

Sam Coppens
Ghent University – IBBT
ELIS, Multimedia Lab
sam.coppens@ugent.be

Joaquim Gabarró Vallés
Universitat Politècnica de Catalunya
gabarro@lsi.upc.edu

Rik Van de Walle
Ghent University – IBBT
ELIS, Multimedia Lab
rik.vandewalle@ugent.be

## ABSTRACT

The early visions for the Semantic Web, from the famous 2001 Scientific American article by Berners-Lee *et al.*, feature intelligent agents that can autonomously perform tasks like discovering information, scheduling events, finding execution plans for complex operations, and in general, use reasoning techniques to come up with sense-making and traceable decisions. While today—more than ten years later—the building blocks (1) resource-oriented REST infrastructure, (2) Web APIs, and (3) Linked Data are in place, the envisioned intelligent agents have not landed yet. In this paper, we explain why capturing functionality is the connection between those three building blocks, and introduce the functional API description format RESTdesc that creates this bridge between hypermedia APIs and the Semantic Web. Rather than adding yet another component to the Semantic Web stack, RESTdesc offers instead concise descriptions that reuse existing vocabularies to guide hypermedia-driven agents. Its versatile capabilities are illustrated by a real-life agent use case for Web browsers wherein we demonstrate that RESTdesc functional descriptions are capable of fulfilling the promise of autonomous agents on the Web.

## Categories and Subject Descriptors

H.3.4 [**Information Systems**]: Information Storage and Retrieval—*Semantic Web*; H.3.4 [**Information Systems**]: Information Storage and Retrieval—*World Wide Web*; H.3.5 [**Online Information Services**]: Web-based services

## Keywords

API descriptions, hypermedia, REST, Service descriptions, Semantic Web, Web APIs, Web services

## 1. AGENTS NEED FUNCTIONALITY

### 1.1 We have the APIs—where are the agents?

The Web: one vision, thousands of services, billions of data sources. But how many automated agents are there? In the vision put forward by Tim Berners-Lee [4], intelligent agents would solve a task for humans by consuming services, interpreting information, and delivering the desired result. Now, more than a decade later, all necessary infrastructure for those agents to function seems to be in place. This is why Jim Hendler, co-author of the initial vision article, stepped forward and asked the fundamental question [18]:

“ So where are all the agents? „

The Semantic Web has arrived but its envisioned main consumers, the intelligent agents [17], are still missing. To find causes for this important void on the Web for agents, we should examine the essentials of which this Web consists:

**(A)** a scalable, resource-oriented REST infrastructure [13];

**(B)** an extensive collection of Web APIs or services [9];

**(C)** a large amount of Linked Data and ontologies [5].

Clearly, the above elements together fulfill the required conditions for the existence of agents as originally envisioned, but why are they insufficient? The answer is simple, yet subtle: while the elements are there, we lack the link that combines and integrates the three of them: *functionality*. Currently, there is no straightforward way to capture this.

### 1.2 Functionality binds the Web for agents

A simple answer, however, does not imply a simple solution. We therefore do not claim to have found a definite entrance to the agent-enabled Web. What we do claim is to have identified functionality as the central concept and a way to closely connect the three aforementioned elements. We define functionality as the means that enables agents to use the HTTP REST *infrastructure* **(A)** to learn what a specific *service* **(B)** can do with *Linked Data* **(C)**.

Nowadays, clients have to be programmed against a specific API and vocabulary, because they cannot decide autonomously what services and data they need. To remedy this problem, REST advocates the principle of Hypermedia as the Engine of Application State (HATEOAS, [11]), which demands that a server supplies the possible next steps alongside each resource. That way, an agent does not need to know in advance how to use an API; instead, it can just "follow its nose" at runtime through these supplied hypermedia controls. But how can an automated agent understand what it *means* to follow such a hypermedia link? The goal of our approach is therefore to provide a machine-processable description of the *functionality of hypermedia links*, since functionality is the key differentiating characteristic between Web services and consequently the crucial factor for automated decisions. RESTdesc will thus be a method to capture functionality, integrating REST infrastructure, services, and Linked Data—the three essential elements.

## 2. RESTDESC BUILDS BRIDGES

### 2.1 Connecting the essential elements

Let us first investigate several observations that connect the three elements identified above.

**The RESTful Web is resource-oriented.** Most programmers tend to be familiar with method-based thinking. As a consequence, there has been a tradition of Web service techniques that bend HTTP [12] to act as a tunneling protocol for action messages (*e.g.*, SOAP [15]). However, this does not align with the fundamental principles of resource-oriented architectures [13] and rules out the hypermedia mechanism. Designing Web services for intelligent agents requires a different mindset, by thinking in resources instead of in actions. The Semantic Web is also based on those *same* resources (hence, the *Resource* Description Framework or RDF [22]), another reason the Web for agents should focus on resources.

**Links are the Web's vital bridges.** Since the beginning, hyperlinks have been a crucial element of the World Wide Web, and continue to be a substantial success factor. This is certainly true for the human Web, and even more for REST and the Semantic Web, where meaning is created and defined by linking to specific URIs [5]. The possibility to uniquely identify and link to a resource is so crucial that it should apply to all intelligent services. Fielding indeed famously described hypertext as "*the simultaneous presentation of information and controls*" [11].

**Simplicity is preferred over complexity.** The simplicity of the HTTP REST architecture lies in its uniform interface, harnessing an extreme variety in resources. Only a handful of actions provide for the majority of Web interactions. Similarly, the Semantic Web is based on a simple, threefold model, which accommodates for all of its data. We feel it is necessary to continue on this same course of simplicity to maximize the efficiency of our description method.

### 2.2 RESTdesc describes functionality

Now that its needs have been identified, we present a resource-oriented and hyperlink-based method that describes Web services in an elegant way. RESTdesc is entirely created using existing technologies and mechanisms, while its novelty lies in the creative combination of the latter for functional service description. RESTdesc expresses descriptions in Notation3 (N3, [2]), a Semantic Web language put forward by Tim Berners-Lee. N3 builds upon RDF, adding straightforward concepts such as variables and graphs.

As an example, we describe an image thumbnail service: given an image, it generates a smaller version of the image with a fixed height. Listing 1 shows the corresponding RESTdesc description. It is constructed as an implication triple: a precondition graph (between braces **{ }**) is stated to imply (using the arrow symbol **=>**) a postcondition graph (between the second pair of braces **{ }**).

```
@prefix ex: <http://example.org/image#>.
@prefix http: <http://www.w3.org/2011/http#>.
@prefix dbpedia: <http://dbpedia.org/resource/>.
@prefix dbpedia-owl: <http://dbpedia.org/ontology/>.

{ ?image ex:smallThumbnail ?thumbnail. } ❶
=>
{
  _:request http:methodName "GET"; ❷
            http:requestURI ?thumbnail;
            http:resp [ http:body ?thumbnail ].

  ?image dbpedia-owl:thumbnail ?thumbnail. ❸
  ?thumbnail a dbpedia:Image;
             dbpedia-owl:height 80.0.
}.
```

**Listing 1:** RESTdesc describes an image thumbnail service elegantly with hyperlinks and existing vocabulary.

The above description can be interpreted in three parts as: **IF** you have an image with a `smallThumbnail` hyperlink ❶ **THEN** you can make an HTTP `GET` request to that link ❷ to receive a thumbnail of the image with height 80 pixels. ❸ Immediately, the powerful nature of RESTdesc becomes apparent: this description unites *infrastructure* (HTTP), *services* (thumbnail generation), and *data* (vocabulary reuse).

As a concrete instance,[1] suppose an agent has an image located at `http://example.org/images/37`. When it requests this URL, the server returns a representation of this image, accompanied by several hyperlinks (*e.g.*, via Link headers [28]). The description in Listing 1 tells the agent that, in order to obtain a thumbnail of 80 pixels in height, it should find a hyperlink with relation `smallThumbnail`. This hyperlink is amongst the returned links, and points to `/images/37/thumb`. Following the instructions in the description, the agent thus performs a `GET` request to `/images/37/thumb`, in response to which the server generates the desired thumbnail and sends it back.[2]

### 2.3 The RESTdesc approach is different

The example makes clear that we depart from traditional approaches. By design, RESTdesc integrates with existing Semantic Web tools and practices. The important differences with other description methods are listed below.

**Vocabulary reuse** – We do not force description authors to learn and use a new vocabulary. RESTdesc adapts to the application domain of the author, instead of the other way around. Listing 1 uses a service-specific vocabulary (`ex`), DBpedia [6] vocabularies (`dbpedia` and `dbpedia-owl`), and

[1]The instantiation process can be performed automatically by common N3 reasoners, as detailed in Subsection 3.4.

[2]One could argue that many of the semantics of Listing 1 can be expressed by ontological constructs. While this holds for simple examples, and generally for dereferencing (*e.g.*, `GET`), more complex relationships and state-changing operations (*e.g.*, `POST` and `PUT`) go beyond the expressivity of ontologies.

the HTTP vocabulary (`http`, [23]). We do not need additional vocabularies to describe the service, because of the resource-orientedness of our approach: the resources in the description are the resources of the service. Variables, an N3 feature, instantiate the vocabulary for concrete resources. Hypermedia link types are simply predicates in the application's ontology (`ex`), since they indeed express a connection between two resources.

**Technology reuse** – RESTdesc does not require new models or paradigms. Instead, it adopts existing Semantic Web technologies and is therefore compatible with existing tools. Common N3 reasoners can interpret RESTdesc descriptions and instantiate them for concrete situations (Subsection 3.4). Once instantiated, the descriptions become plain RDF, ensuring compatibility with clients that do not wish to use N3. Importantly, N3 reasoners are by design able to perform *goal-driven service compositions* by combining multiple RESTdesc descriptions, without requiring additional plugins.

**Compactness and simplicity** – Authors and consumers are not faced with long and verbose descriptions that are difficult to understand at sight. RESTdesc descriptions provide at a glance the essence of a service: its functionality, the precise task the service performs. This high level of compactness is possible because RESTdesc only describes what is strictly necessary, namely the functional hypermedia relation between resources and the HTTP request that obtains that result. Details, such as parameter types, reside where they belong: in ontologies. The example in Listing 1 shows for instance that the `thumbnail` and `height` properties reuse the DBpedia ontology, which provides their characteristics.

Similarly, the hypermedia relation types (being RDF predicates) can belong to proprietary or public vocabularies [19, 28]. For instance, `smallThumbnail` represents a 80 pixels high image only in a specific application context. Other applications could have similar or different definitions. Furthermore, it is precisely the meaning of these hyperlinks that is fully defined by the RESTdesc description: in this concrete case, Listing 1 describes the meaning of the application-specific relation `smallThumbnail`.

## 3. RESTDESC IN PRACTICE

### 3.1 Description anatomy: logical grounds

Where Section 2 introduced RESTdesc through an example, this section aims to provide a rigorous description of what RESTdesc descriptions are and how to create them. Basically, the functional description provided by RESTdesc describes the result of an action $\mathbf{A}$ on a resource $r$. More formally, it explains: given a set of preconditions $pre_{\mathbf{A}}$ on this resource $r$, what request $request_{\mathbf{A}}$ is necessary to obtain a set of postconditions $post_{\mathbf{A}}$ for that action $\mathbf{A}$:

$$\mathbf{A}(r) \;\equiv\; pre_{\mathbf{A}}(r) \xrightarrow{request_{\mathbf{A}}(r)} post_{\mathbf{A}}(r) \tag{1}$$

This means that, when the preconditions for the action are fulfilled, executing the request will lead to the postconditions. For instance, having an image and requesting a GET operation on its thumbnail URI implies that we will receive a thumbnail image.

The complexity in Equation 1 lies in the fact that the implication is fulfilled only when the request for the action is successfully carried out. To obtain simple expressions that can be reasoned upon easily, we must express Equation 1 in a

```
@prefix http: <http://www.w3.org/2011/http#>.
{
  Preconditions about a certain resource...
}
=>
{
  ...imply the existence of a certain request...
  _:request http:methodName [...];
            http:requestURI [...];
            http:resp [...].
  ...that effectuates postconditions on this resource.
}.
```

**Listing 2:** The general RESTdesc skeleton is very flexible.

more conventional paradigm. A straightforward conversion to first-order logic would be to see the request as a part of the precondition:

$$\mathbf{A}(r) \;\equiv\; pre_{\mathbf{A}}(r) \wedge request_{\mathbf{A}}(r) \Longrightarrow post_{\mathbf{A}}(r) \tag{2}$$

Equation 2 states that, given the preconditions and the fact that the request has been performed, we obtain the postconditions. However, this equation does not always hold in practice: any actual request could fail to deliver an adequate response for several reasons, invalidating the above implication in the general case. Therefore, a better interpretation of Equation 1 is:

$$\mathbf{A}(r) \;\equiv\; pre_{\mathbf{A}}(r) \Longrightarrow \exists R\big(request_{\mathbf{A}}(r,R) \wedge post_{\mathbf{A}}(r,R)\big) \tag{3}$$

Equation 3 states that when the preconditions are fulfilled, there exists a request that fulfills the postconditions. Of course, it is exactly *this* request that an agent will try to perform when it wants to achieve the postconditions.

### 3.2 Creating descriptions

Since N3 possesses the full power and expressivity of first-order logic [3], we can write Equation 3 in the N3 language, which is the essence of RESTdesc. Listing 1 shows indeed an example of this: the precondition (having a `smallThumbnail` link) means that a request exists (an HTTP `GET` to the `smallThumbnail` link), which entails a postcondition (receiving an image with height 80 pixels in the response body). The general skeleton is displayed in Listing 2.

Often more interesting to agents are state-changing operations using unsafe methods such as `PUT` or `POST`. They can also be described in the RESTdesc skeleton. For example, suppose we want to describe that comments can be added to images, as in Listing 3. First, we add a hyperlink from each image resource to its comments resource 1. For example, the image `http://example.org/images/37` may have its comments at `/images/37/comments`. An additional precondition is that we need to have a comment. We then describe the request needed to add this comment to that image 2. Finally, we explain that the comment is attached to the image as a result 3.

### 3.3 Discovering functionality

In order to make the RESTdesc description paradigm work in real-world applications, we need a method to automatically discover descriptions. After all, we aim to make generic agents perform operations with specific services. We envision several possible methods, listed below.

```
@prefix ex: <http://example.org/image#>.
@prefix sioc: <http://rdfs.org/sioc/ns#>.
@prefix http: <http://www.w3.org/2011/http#>.
{
  ?image ex:comments ?comments. ❶
  ?comment sioc:content ?commentText.
}
=>
{
  _:request http:methodName "POST"; ❷
            http:requestURI ?comments;
            http:body ?commentText;
            http:resp [ http:body ?comment ].
  ?image sioc:has_reply ?comment. ❸
  ?comments ex:contains ?comment.
}.
```

**Listing 3:** RESTdesc allows state-changing operations, such as adding comments to an image.

**Dereference link types** – Since we use RDF predicates as the hypermedia link types, the preferred option is to request the description of each link by dereferencing its URI. For instance, the description of the `ex:comments` relation can be found directly at `http://example.org/image#comments`. Using content negotation, agents can indicate whether they are interested in the `image` ontology or the RESTdesc description. This method thus functions in a *link-centric* way.

**HTTP OPTIONS** – The HTTP specification provides an `OPTIONS` method, representing "*a request for information about the communication options available on the request/response chain identified by the Request-URI*" [12]. While the specification does not yet determine the response body, servers could use it to return RESTdesc descriptions via content negotiation. For example, an HTTP `OPTIONS` request to `http://example.org/images/` could return the descriptions in Listings 1 and 3. This method is *resource-centric.*

**Service repository** – As a last resort—for instance, when describing an external a service without access to its server—repositories can provide descriptions for various services. The idea is not to have one central repository but multiple repositories (possibly connected), which a client can use as a starting point. For example, a client can ask to find services for images, and the repository could return Listings 1 and 3. This can happen in link- *and* resource-centric ways. Hypermedia links between the repository and the service can be made, allowing hypermedia-driven discovery.

Further experiments will be necessary to indicate which of the suggested methods will work best in practice. Additionally, all of the above methods can be used in conjunction without interference, giving agents as many options for discovery as they need. In any case, these discovery mechanisms indicate that RESTdesc can fulfill the needs of generic clients without bindings to specific APIs.

## 3.4 Interpreting descriptions

How does a client decide what service to use? Clients start from the *current situation* and work towards a certain *goal.* For example, suppose we have a local image:

```
<myphoto.jpg> a dbpedia:Image.
```

Our goal might be to obtain a thumbnail for that image:

```
<myphoto.jpg> dbpedia-owl:thumbnail _:thumbnail.
```

```
@prefix ex: <http://example.org/image#>.
@prefix http: <http://www.w3.org/2011/http#>.
@prefix dbpedia: <http://dbpedia.org/resource/>.
{
  ?image a dbpedia:Image.
}
=>
{
  _:request http:methodName "POST";
            http:requestURI "/images/";
            http:body ?image;
            http:resp [ http:body ?image ].
  ?image ex:comments _:comments;
         ex:smallThumbnail _:thumb;
         ex:mediumThumbnail _:mediumThumb;
         ex:belongsTo _:album.
}.
```

**Listing 4:** Uploading an image results in links to other resources with possibly relevant functionality for a use case.

First, using one or more of the methods in Subsection 3.3, the agent discovers what functionality it has at its disposal. It retrieves several descriptions, including Listings 1, 3, and 4, which it can obtain by issuing an `OPTIONS` request to the URI `http://example.org/`. Using a reasoner, the agent can find out that Listing 1 describes a way to obtain a thumbnail, on condition that the image has a `smallThumbnail` link. The reasoner also devises that such a link can be obtained by uploading the image, as described by Listing 4.

An important benefit of RESTdesc is that common N3 reasoners, such as cwm [1, 3] or EYE [8], can perform the above deduction in a completely automated way, without requiring any plugin. Indeed, because RESTdesc descriptions are N3 rules by design, they seamlessly integrate with current Semantic Web tools. Reasoners are able to directly instantiate RESTdesc descriptions for a concrete case. For example, the reasoner can combine Listing 4 with the starting situation above to obtain the concrete request, filling in the variables with the actual values. The result is in plain RDF format, without quantifications, and contains all the details the client needs to issue this request. Note that the reasoner does not need access to all data in each step. Instead, it runs initially with the starting context, the goal, and the available descriptions, to generate an *execution plan.* In further steps, only the actual context and this plan are necessary.

Subsequently, the client can execute this request and upload the image to the server as instructed. In response, the client retrieves a representation of this image, containing indeed the links indicated by its description in Listing 4. For example, the actual image may link to comments and thumbnails as follows:

```
<http://example.org/images/37>
     ex:comments </images/37/comments>;
     ex:smallThumbnail </images/37/thumb>;
     ex:mediumThumbnail </images/37/thumb_med>;
     ex:belongsTo </albums/6>.
```

The reasoner then similarly instantiates Listing 1 with the current situation, which now includes the hyperlinks above. This deduction then indicates that the agent needs to issue a `GET` request to the `/images/37/thumb` resource in order to obtain the thumbnail. When the client performs this final request, the initial goal is reached and execution terminates.

## 4. RESTDESC FOR TODAY'S AGENTS

### 4.1 Web Intents enable common interactions

Evidently, the Web for agents will not arrive tomorrow, yet. This does not mean that RESTdesc cannot be used *today*. People access Web applications through browsers, which are increasingly adopting characteristics of intelligent agents, either through native support for interactive features, or by an extension mechanism that allows third-party developers to enrich people's browsing experience. Therefore, browsers present interesting RESTdesc use cases.

*Web Intents* [21], a project started by Google Developer Advocate Paul Kinlan, is driven by the observation that users expect Web applications to work together seamlessly. Kinlan states that, for example, a photo gallery Web application should be aware of a user's preferred third party photo editing Web application, rather than enforcing the specific one that the gallery happens to be integrated with. Web Intents therefore proposes client-side service discovery and inter-application communication. Initially, services *register* their *intent*, i.e., their ability to perform a certain action on a certain media type (Listing 5). Thereafter, applications can request to *initiate* such an action (Listing 6). As a result, the Web Intents client then starts the desired service for the user. The developer teams behind the Firefox and Chrome browsers have committed to provide native Web Intents implementations. Meanwhile, full compatibility with all major browsers is provided by a public JavaScript library.

Web Intents build on a long tradition of annotating links with plain words and media types. For instance, *a priori* meaningless Web links can be given a defined meaning using hyperlink relations from the list of IANA link relations [19] and by providing a media type. A common example is annotating the link to a news feed using the `alternate` Link Relation attribute value and the Atom media type:

```
<a href="feed.xml" rel="alternate"
                   type="application/atom+xml">
```

Given such markup, a Web browser can automatically offer the user to subscribe to that news feed. Another example is the `search` Link Relation, which allows user agents to discover the search functionality of a given website. Web Intents carries over this idea to describe various standard interaction patterns between Web applications, such as editing, sharing, and subscribing.

```
<intent
    action="http://webintents.org/share"
    type="image/*"
/>
```

**Listing 5:** Web services can easily register an intent by specifying the action and media type they support.

```
navigator.startActivity(new Intent(
    "http://webintents.org/share", // action
    "image/jpeg", // content type
    "http://example.org/photo.jpg")); // content
```

**Listing 6:** Web applications can offer user-adapted functionality with little effort using Web Intents.

### 4.2 RESTdesc provides new interactions

The standard interaction patterns carry well-defined semantics in Web Intents. However, many of today's popular services [9] offer different interactions that those patterns do not cover. Some services offer simple operations on currently uncovered media types—for instance, services that act on location data, such as a weather forecast service or a map service. Other services offer more complex operations on already covered media types—for instance, a service that recognizes people in an image. Centralized API catalogs like ProgrammableWeb.com let people categorize and document services, much like in the early days of World Wide Web search engines. Still, in order to discover those services, *manual* intervention is necessary, because their actions or media types are currently outside the Web Intents scope.

Adding the power of RESTdesc service descriptions to Web Intents creates an *automated* and more scalable solution. Web Intents is extensible by design: neither the list of actions nor the list of media types are fixed. Nonetheless, service developers cannot simply define new actions, because existing Web Intents clients would not understand how to use them. This is where RESTdesc functional descriptions play an important role, since they can provide semantics for *new* intent actions that can be understood by *existing* clients, even if the described functionality was not available when that client was built. Because the goal of RESTdesc is precisely to support generic clients in their decision making process, its combination with Web Intents is a natural step to make new interaction patterns possible.

### 4.3 Generalizable solutions with Linked Data

Discovering functionality—for example, that a site offers a news feed for feed readers—is possible due to the fact that a set of preconditions is fulfilled. In this news feed example, the preconditions are the existence of the link to a resource (*e.g.*, `feed.xml`), the presence of a certain link relationship (*e.g.*, `alternate`), and the availability of a specific media type (*e.g.*, `application/atom+xml`). Web browsers either already have those preconditions hardcoded, or can be supplemented with browser extensions to offer services for those preconditions. The postcondition in this case is that the user will be subscribed to the feed in her preferred news reader.

Leaving the example and more generally speaking, it is obvious that no user agent (including, but not limited to Web browsers) can be prepared for the functionality of *every* Web API and can have foreseen a mechanism to deal with it. When we think of various APIs—such as currency conversion, weather forecasts, or movie rental—we see that they all have well-defined pre- and postconditions, where the execution of a particular API request is the state transfer from *pre* to *post*. Since RESTdesc is based on the formal description of pre- and postconditions, it is an adequate candidate to capture these interactions.

However, as just noted, not all possible use cases can be foreseen. How can RESTdesc then anticipate on every possible API that will be described with it? And how will an agent, when it sees a RESTdesc pre- and postcondition, be able to know for what cases this API could be useful? The answer is provided by Linked Data. An ever-growing amount of structured content is being published on the Web [5]. Recent Web-scale structured data efforts such as Facebook's Open Graph Protocol [10] or Google, Yahoo!, and Microsoft's schema.org specification [14] are intensifying this trend even more. These initiatives help make content—such as prices, locations, or movies—machine-accessible, closing the circle to interpretation of conditions:

- **IF** your product page contains a *price* (as Linked Data), **THEN** you can convert its value to your local currency with *this* request.
- **IF** your travel info refers to a *location* (with schema.org), **THEN** you obtain its weather forecast with *this* request.
- **IF** you like the summary of a *movie* (via Open Graph), **THEN** you can buy its DVD with *this* request.

Again, reasoning can play an important part here. By incorporating ontological knowledge into the reasoning process, a bridge between different ways of expressing information can be built. As a result, service discovery can work even in cases where the vocabulary of the description is different from the vocabulary of the data. Furthermore, reasoning also allows different clients to obtain the information they need to construct the request that satisfies their goals.

All of the above indicates that the long-envisioned intelligent agents suddenly become within reach, using the tools and technology that are already available on the Web today. In the following section, we discuss the chances, risks, and limitations of our approach.

## 5. DISCUSSION

### 5.1 Justification of employed technologies

We deliberately chose to describe services that employ the resource-oriented and hypermedia principles underpinning the fundamentals of REST. Commonly referred to as REST services or RESTful APIS, they contrast with services that use Remote Procedure Calls, the so-called RPC-style. Unfortunately, several services that label themselves as "REST" lack a hypertext-driven architecture [11], typically by falling back to URI construction rules defined in advance instead of at runtime, which makes them plain HTTP interfaces [30]. However, the REST community is working hard to turn the tide.

The reason we decide to focus on REST services—in the original meaning intended by Fielding and others—is that they provide a beautiful integration on every level of the Web. After all, HTTP was designed with REST principles in mind [12], offering a high degree of scalability. It therefore comes as no surprise that the resource-oriented nature of REST aligns perfectly with the resources on the Semantic Web, where the R*esource* Description Framework (RDF) is the dominant model. This alignment is not a coincidence—it is a feature, which we purposely embrace. Also, even if a service does not fully follow REST principles, RESTdesc can still describe it, albeit with more verbosity (*e.g.*, using string concatenation to construct URIS).

Therefore, we see our choice for REST as an important benefit, because resource-orientation is a key principle of both the Web of Services and the Semantic Web. The REST community's efforts convince many services to move towards a RESTful architecture [9]. Furthermore, when a server offers such a resource- and hypermedia-based API, the concept "service" is gradually fading, since clients then see nothing but resources and the relationships between them. The service exists only behind the scenes, but its *functionality* manifests itself in the resources—which is why we coined RESTdesc as a *functional* description format.

Another important decision is our choice for Notation3, whose simplicity and powerful logic foundation reflect on RESTdesc. This decision was prompted by the observation that, in order to make a statement about generic instead of specific resources, support for quantification is necessary, yet currently not provided by RDF. The same observation was implicitly made by OWL for Services (OWL-S, [26]) and Linked Open Services (LOS, [27]), which both have to resort to richer expression languages inside RDF string literals.

While N3 offers an integrated solution for quantification with a strong logic basis and a minimal extension to RDF, some clients might not be fully compatible with it. Our answer here is twofold: first, the clients themselves do not need to consume N3, as the reasoner is the component carrying out the N3-related transformations and providing the client with instantiated representations in plain RDF format. Secondly, powerful N3 consumers such as cwm [1] and EYE [8] exist for several years now, the latter being compatible with a broad spectrum of other Web logic languages, thanks to its interoperability with the W3C Rule Interchange Format (RIF [20]).

### 5.2 Feasibility of the RESTdesc vision

One of the key questions is of course whether our proposed approach is able to realize the functional descriptions that generic intelligent agents need. Therefore, we investigated how current state-of-the-art reasoners perform on RESTdesc descriptions, based on functionality. The main questions are whether RESTdesc descriptions are sufficient for a generic reasoner without plugins, and whether the reasoner returns sufficient information for a generic client.

To verify this, we have fed the descriptions of Listing 1 and Listing 4 into a reasoner, together with respectively the starting situation (a local image) and the post-upload situation (a hyperlinked resource). We then examined the result and verified that in both cases, the necessary details to construct the HTTP request were correctly present. This experiment and its result details are publicly available at `http://notes.restdesc.org/2011/images/`.

Additionally, we needed to prove that the reasoner can create an execution plan, as indicated in Subsection 3.4. The descriptions of Listings 1 and 4 should be sufficient for the reasoner to, given the starting situation (a local image), create a plan to achieve a goal (obtaining a thumbnail). We verified that it correctly generates the details of the two required requests: uploading the image and retrieving the thumbnail. This experiment and its result details are also available online. Furthermore, from previous experience with reasoner-based composition, we are confident that this approach is scalable to a level that comfortably enables practical use [31].

The only constraints imposed by RESTdesc are 1) for agents to understand the HTTP vocabulary in RDF—a reasonable condition, since one of their main tasks is to deal with HTTP requests and responses—and 2) for developers to be able to design and work with descriptions in RESTdesc. Since RESTdesc does not invent a new language but adopts N3, the design should not pose large problems for developers familiar with the Semantic Web. To support description creators, we provide an informational website `http://restdesc.org/` with an interactive online group for questions and advice. Also, developers unfamiliar with reasoning techniques can use the reasoner as a black box, for example through a Web service.

### 5.3 Current limitations

A first issue is the responsibility for creating RESTdesc descriptions. This is indeed a manifestation of the chicken-and-egg problem that many new Semantic Web developments face: widespread support makes a technology used, but support only evolves in case of widespread use. We aim to counter this argument by making the threshold to RESTdesc as low as possible by its innate simplicity and consiseness, adoption of existing technologies, and availability of community support. Additionally, we plan to bootstrap the process by providing descriptions for popular Web APIS.

Secondly, the main benefit of RESTdesc—its solid integration with the Semantic Web—is also its main dependency. Current research questions on several topics, such as ontological alignment, are therefore also relevant for RESTdesc. Just like the Semantic Web, RESTdesc depends on the availability of machine-readable information and the interlinking of that information, which similarly can be a benefit (*e.g.*, in terms of scalability and independence) or a burden (*e.g.*, in case of insufficient availability or absence).

Thirdly, the dependency on Semantic Web technologies means that RESTdesc inherits some of the limitations of RDF. For instance, RDF does not provide a native way to state that a resource does not exist. This is a consequence of the open world assumption: the absence of a certain triple does not necessarily indicate its inexistence. Therefore, there is currently no direct way to explain a DELETE request. Even if there was, it still would carry a danger of contradiction in the first-order logic model, because a resource must exist before you can delete it—but if you delete it, it does not exist anymore. However, we believe that workarounds are possible, such as introducing a “deleted” flag, or the notion that future requests will result in a 410 Gone status code [12]. But most importantly, the semantics of DELETE are already fully defined by HTTP itself, so there is most likely no need to redefine them in RESTdesc in the first place. Still, we have to be well aware of those limiting model properties.

## 6. RELATED WORK

Web service or Web API description has been a topic of intense research research for at least a decade. There are many approaches to service description with different underlying service models. Earlier service description technologies mainly focus on technical aspects like input and output parameters, data types, and exceptions. Prominent examples include the Web Services Description Language (WSDL, [7]) and to some extent also the Web Application Description Language (WADL, [16]). However, recently, a trend towards more functionality-oriented formats is evolving.

Semantic Markup for Web Services (OWL-S, [26]) is a well-known service ontology. OWL-S requires an additional description for the grounding, commonly WSDL, which results in OWL-S inheriting WSDL's issues like verbosity and perceived complexity. The lack of *semantic* description of input and output parameters in WSDL is addressed by Semantic Annotations for WSDL (SAWSDL, [24]). However, no functional parameter relation is established.

Linked Open Services (LOS, [27]) expose functionality on the Web using Linked Data technologies, namely HTTP, RDF, and SPARQL. Input and output parameters are described with SPARQL graph patterns embedded inside RDF string literals to achieve quantification, which RDF does not support natively.

Linked Data Services (LIDS, [29]) define interface conventions that are compatible with Linked Data principles [5] and are supported by a lightweight formal model. This enables automatic creation of LIDS interfaces and integration of links to LIDS in existing data sets.

## 7. CONCLUSIONS AND FUTURE WORK

In this paper, we introduced the functional description format RESTdesc. The obvious question is: why would this method offer what several previous formats could not? And why would RESTdesc help fulfill the long-standing promise of generic intelligent agents?

RESTdesc differs in that it finds a solution *inside* the boundaries of currently available Web technology, instead of introducing new techniques that require new tools and infrastructure. Let us look back to RESTdesc from our three initial perspectives: *infrastructure* **(A)**, *services* **(B)**, and *data* **(C)**.

On the level of infrastructure, a strong link with HTTP can be identified. RESTdesc is based on the fundamental properties of the Web and especially its resource-oriented nature, the ideas of which formed the basis of the current HTTP 1.1 standard [12]. The hypermedia constraint inherent to REST is satisfied by extensively making use of link types, which directly map to RDF predicates. The essence of RESTdesc is indeed to describe the relationships among resources and the concrete HTTP requests instantiating the functionality of those relationships.

For services, RESTdesc uncovers their key differentiating feature, namely functionality, exposing this in a logical way that integrates semantics and the REST architectural modalities. This enables services to involve in new and different interactions in an automated way, making compositions based on functionality instead of input and output parameters.

Finally, Linked Data and vocabularies form an important part of RESTdesc, again because existing work is fully reused. A major strength of RESTdesc is that it functions with the vocabulary of the application domain. This gives service authors the freedom to select the vocabularies that explain their functionality with the greatest expressiveness. Thanks to the links between different vocabularies and data, a wide interoperability is possible.

This paper aims to be an important step towards simple functional descriptions. Several important research challenges are still ahead. For example, integration with authentication mechanisms and other real-world applicational concerns will have to be fluently incorporated, especially in pay-per-use scenarios where a lower number of requests is preferred. A study showed that more than 80% of all Web APIS on the Web service catalog ProgrammableWeb.com require some form of authentication [25]. In general, the handling of exceptional situations in a RESTful context, resulting in changes to the initial execution plan, should be investigated. Also, developers must have a clear understanding of the benefits of RESTdesc for their services. This is why we will work on implementing agents that use the power of RESTdesc to accomplish tasks for human needs.

The most important realization, however, is that RESTdesc is not a technology for the future, but for today. Starting from within Web browsers—for instance, with Web Intents—RESTdesc can deliver services on demand, precisely because it captures and exposes functionality. After all, the Web for agents will not introduce a disruptive change, but rather be the result of an evolution that has already started.

## 8. ACKNOWLEDGMENTS

The described research activities 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). Joaquim Gabarró is partially supported by TIN-2007-66523 (FORMALISM), and SGR 2009-2015 (ALCOM).

## 9. REFERENCES

[1] T. Berners-Lee. cwm. Semantic Web Application Platform, 2000–2009. Available at http://www.w3.org/2000/10/swap/doc/cwm.html.

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

[3] 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*, 8(3):249–269, 2008.

[4] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[5] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data – The Story So Far. *International Journal On Semantic Web and Information Systems*, 5(3):1–22, 2009.

[6] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia – a crystallization point for the Web of Data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154–165, 2009.

[7] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. W3C Note 15 March 2001, Mar. 2001. Available at http://www.w3.org/TR/wsdl.

[8] J. De Roo. Euler proof mechanism, 1999–2011. Available at http://eulersharp.sourceforge.net/.

[9] A. DuVander. 4,000 Web APIs: What's hot and what's next?, Oct. 2011. Available at http://blog.programmableweb.com/2011/10/03/4000-web-apis-whats-hot-and-whats-next/.

[10] Facebook. Open Graph Protocol. Specification, Nov. 2011. Available at http://ogp.me/.

[11] R. T. Fielding. REST APIs must be hypertext-driven. Untangled – Musings of Roy T. Fielding, Oct. 2008. Available at http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven.

[12] R. T. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. Request for Comments: 2616, June 1999. Available at http://www.ietf.org/rfc/rfc2616.txt.

[13] R. T. Fielding and R. N. Taylor. Principled design of the modern Web architecture. *ACM Transactions on Internet Technology*, 2(2):115–150, May 2002.

[14] Google, Inc., Yahoo, Inc., and Microsoft Corporation. Schema.org. Specification, June 2011. Available at http://schema.org/docs/schemas.html.

[15] M. Gudgin, M. Hadley, N. Mendelsohn, and J.-J. Moreau. SOAP version 1.2 part 1: Messaging framework (second edition). W3C Recommendation, Apr. 2007. Available at http://www.w3.org/TR/2007/REC-soap12-part1-20070427/.

[16] M. Hadley. Web Application Description Language. W3C Member Submission, Aug. 2009. Available at http://www.w3.org/Submission/wadl/.

[17] J. Hendler. Agents and the Semantic Web. IEEE *Intelligent Systems*, 16(2):30–37, Mar–Apr 2001.

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

[19] IANA. Link relations, Nov. 2011. Available at http://www.iana.org/assignments/link-relations/link-relations.xml.

[20] M. Kifer. Rule Interchange Format: The framework. In D. Calvanese and G. Lausen, editors, *Web Reasoning and Rule Systems*, volume 5341 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2008.

[21] P. Kinlan. Web Intents. Specification, Dec. 2010. Available at http://webintents.org/.

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

[23] J. Koch, C. A. Velasco, and P. Ackermann, *Eds.* HTTP vocabulary in RDF 1.0. W3C Working Draft, May 2011. Available at http://www.w3.org/TR/HTTP-in-RDF10/.

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

[25] M. Maleshkova, C. Pedrinaci, J. Domingue, G. A. Rey, and I. Martinez. Using semantics for automating the authentication of Web APIs. In *International Semantic Web Conference (1)*, pages 534–549, 2010.

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

[27] B. Norton and R. Krummenacher. Consuming dynamic Linked Data. In *1st International Workshop on Consuming Linked Data (November 2010)*, 2010.

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

[29] S. Speiser and A. Harth. Integrating Linked Data and services with Linked Data Services. In G. Antoniou, M. Grobelnik, E. Simperl, B. Parsia, D. Plexousakis, P. De Leenheer, and J. Pan, editors, *The Semantic Web: Research and Applications*, volume 6643 of *Lecture Notes in Computer Science*, pages 170–184. Springer Berlin / Heidelberg, 2011.

[30] T. Steiner and J. Algermissen. Fulfilling the hypermedia constraint via HTTP OPTIONS, the HTTP vocabulary in RDF, and Link Headers. *Proceedings of the 2nd International Workshop on RESTful design*, 2011.

[31] 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*, 2012.