Description and Interaction of RESTful Multimedia Services for Automatic Discovery and Execution

Abstract

{TODO Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse gravida fringilla ipsum nec tincidunt. Curabitur fermentum tincidunt semper. Suspendisse at orci quam. Ut risus est, mollis nec placerat et, lacinia sed ligula. Nunc varius felis vitae elit mollis laoreet. Nulla congue, sem id ullamcorper convallis, nibh massa scelerisque elit, at congue purus eros vel metus. Nulla purus turpis, egestas eget vehicula et, blandit in nibh. Mauris id sapien velit, in viverra neque. In hac habitasse platea dictumst. Phasellus at ante vitae lectus scelerisque gravida. Curabitur bibendum, libero non interdum rhoncus, diam ante volutpat lorem, a volutpat ante mauris vel nisi. Donec nisi felis, rhoncus et suscipit sit amet, porta eget felis. Maecenas aliquam interdum elit, sed fringilla lorem fringilla non. In hac habitasse platea dictumst. Sed blandit diam sed sem dignissim in aliquet massa consequat. Integer aliquam congue justo a rhoncus. Sed sit amet interdum eros. Nunc aliquam, nisl vitae pulvinar posuere, odio justo malesuada nunc, non sodales tellus diam quis purus. Aenean sed neque quis dui ultricies cursus a vitae orci. Nulla facilisi. Nulla lacus nisl, cursus at vehicula ac, malesuada non est.**}**

1 Introduction

The immense diversity of various multimedia analysis and processing algorithms makes it difficult to integrate them in an automated platform to perform compound tasks. Yet, recent research has indicated the importance of the fusion of different techniques [1]. But how can different algorithms interoperate, if there are no agreements or guidelines on how communication should happen? And how can a coordinating platform select algorithms based on their capabilities, in lack of a formal description detailing their preconditions and postconditions?

In this paper, we show how to lift multimedia algorithms to the level of Semantic Web services with a formal description mechanism that follows a pragmatic approach. Rather than reinventing the existing methodologies, which focus on low-level details, we want to express an algorithm's functionality in way that captures its functionality without requiring lengthy specifications. Our intention is to use established standards and reuse common best practices for implementing multimedia algorithms as true Semantic Web services. The aim is a versatile description and communication model, enabling fully automated service discovery and execution, even under changing conditions. The sole starting point is a Web address of a server—all other information is gathered at runtime.

We will explain our approach by three real-world multimedia use cases, each of which represents challenges that are currently not fully addressed by alternative techniques. They will illustrate the power of our method and hopefully convince the reader that its apparent simplicity accommodates far-reaching possibilities.

2 Related Work

2.1 Web Service Description Language

The description of Web services has a long history. The XML-based Web Service Description Language (WSDL, [3, 4]) provided one of the first models. WSDL focuses on the communicational aspect of Web services, looking from a message-oriented point of view. The details of the message format are written down in a very verbose way and concretized to actual bindings such as SOAP [9] or plain HTTP [5]. Finally, the description can contain endpoints which implement the specified bindings.

For our use case, we spot two major problems with the use of WSDL. First, WSDL only provides the mechanisms to characterize the technical implementation of Web services. It does not provide the means to capture the functionality of a service. For example, a service that counts the number of words in a text will be described by WSDL as an interface which accepts a string and outputs an integer. Clearly, an infinite number of algorithms share those input and output properties, so this information is insufficient to infer any meaning or functionality.

Secondly, in practice, a WSDL description is used to generate module source code automatically, which is then compiled into a larger program. If the description changes, the program no longer works, even if such a change leaves

the functionality intact. {**TODO** Thomas, can we cite a reference here?} So WSDL is not well adapted to real-world circumstantial changes.

The above problems indicate why WSDL cannot offer automatic service discovery at runtime and why we should investigate other possibilities.

2.2 Web Application Description Language

The Web Application Description Language (WADL, [10]), which is also XML-based, is another Web service description format, which advocates proper use of all the aspects of the HTTP protocol, instead of its degradation to a tunneling mechanism as in SOAP. Services that behave in this way are called RESTful [6], the properties of which we will explain in Section 3. While WSDL 2.0 is also capable of specifying bindings to RESTful endpoints, it still requires the abstractions that enable bindings to SOAP and others. WADL, on the other hand, was tailored to the needs of RESTful services, but only exists as a W3C Member Submission and will most likely never reach the W3C Recommendation status of WSDL 2.0 [13].

However, WADL still suffers from the same problem: it does emphasize the technical properties of the underlying service and does not leave any room for the semantics of the task it performs. This also means that there is no way to automatically discover services based on the desired functionality. Therefore, there is no reason why WADL would be used any differently than WSDL, as also argued by Joe Gregorio in [8].

{**TODO** Thomas, given the experience you have with WADL, you can probably ameliorate this section and/or add your views and/or correct mine :) }

{**TODO** However, I would think that WADL can be interesting when a certain amount of detail is required. So why don't we output add a `Link` header to a WADL description (or WSDL, or whatever, just to be generic) on HTTP `OPTIONS`?

Could it be possible *in theory* to generate a WADL description from RESTdesc (given the description and the use of RESTful practices)?}

2.3 Semantic Markup for Web Services

OWL-S [14] is a an OWL [16] ontology for describing Semantic Web services in RDF [12]. A service description consists of three parts: a profile, a model and a grounding. Some aspects of profile and model are very similar, in the sense that they both describe input, output, preconditions and effects. The difference is that the profile is used for discovery, while the model is used to control the interaction.

Here, for the first time, we have an actual focus on the functionality of a service which is separate from how the interaction happens. However, whether this separation was successful is debatable, since there is no way to enforce the consistency of profile and model of a single service. Finally, the grounding part specifies the implementation of the service. The OWL-S submission defines a grounding to WSDL, but other groundings are possible (e.g., [17]). This means that the OWL-S description describes the functionality, whereas its grounding describes the communication.

At least, this is what it is supposed to do. OWL-S input and output types provide more or less the equivalent of what a WSDL message format contains, albeit with RDF types, so there is only a minimal added semantic value on that level. The real possibilities lie in the use of preconditions and postconditions (the latter under the form of result effects), which allow to express complex relationships between input and output values, finally capturing the semantics and functionality of the service.

Unfortunately, there is no obligation to use these conditions and the OWL-S submission only mentions the rule languages KIF [7], DRS [15] and SWRL [11], in order of increasing verbosity. Extensions to more light-weight rule languages, such as Notation3 Logic [2], are possible [17]. The real problem here is that that none of those languages are integrated into the main service description, but rather form subdocuments within it, which require a separate interpretation. The conditions thus live in another context, which are solely linked by identifiers but not by semantics. As a result, agents lacking support for the used rule language could parse the OWL-S constructs in a service description document and skip the parts they fail to understand—effectively ignoring important conditions they should reckon with[1].

Furthermore, while OWL-S offers functional descriptions capable of automatic discovery of the capabilities of a single service, it does not provide mechanisms to express its relation to other services. Also, descriptions contain redundancies and require a fair amount of vocabulary, even to express conceptually simple services, and rely on external groundings for technical implementations.

3 RESTful Multimedia Services

{**TODO** Describing what a RESTful way would be to access multimedia algorithms. By treating their inputs and outputs as resources, instead of invoking commands.}

4 RESTdesc Semantic Description

4.1 Simple yet functional description

{**TODO** Sketch the RESTdesc format, what it does, why it is simple yet fully functional.}

{**TODO** Emphasize that RESTdesc = REST + description, e.g., that RESTful practices are a requirement.}

{**TODO** Avoid critique on the use of Notation3 Logic. Mention Tim BL etc.}

[1] This behavior resembles that of Web browsers without JavaScript support: they parse and render HTML but ignore any `script` tags. Service descriptions, in contrast, *always* require a full interpretation for correct functionality.

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

{ ?photo :photoId ?id. }
=>
{
  _:request http:methodName "GET";
            tmpl:absoluteURITemplate
                "http://example.org/photos/{photoId}";
            tmpl:uriTemplateValues (?id);
            http:resp [ tmpl:represents ?photo ].
}.
```

Listing 1: RESTdesc description of photo retrieval

```
@prefix log: <http://www.w3.org/2000/10/swap/log#>.

@forall :photo, :id.            # ∀ photo, id :
@forsome :request, :response.
{ :photo :photoId :id. }        # photoId(photo,id)
log:implies                     # ⇒ ∃ request,r : resp(request,r)
{                               #     ∧ represents(r,photo) [...]
  :request [...] http:resp :response.
  :response tmpl:represents :photo.
}.
```

Listing 2: Listing 1 with explicit quantifiers

```
{ ?photo a foaf:Image. }
=>
{
  _:request http:methodName "POST";
            http:absoluteURI "http://example.org/photos";
            http:body [ tmpl:formValue ("photo" ?photo) ];
            http:resp [ tmpl:represents ?photo ].

  ?photo :photoId _:photoId.
}.
```

Listing 3: RESTdesc description of photo upload

```
{ ?photo :photoId ?photoId. }
=>
{
  _:request http:methodName "GET";
            tmpl:absoluteURITemplate
  "http://example.org/photos/{photoId}/faces";
            tmpl:uriTemplateValues (?photoId);
            http:resp [ tmpl:representsMultiple ?region ].

  ?region foaf:depicts [ a foaf:Person ];
          :regionId _:regionId;
          :belongsTo ?photo.
}.
```

Listing 4: RESTdesc description of face detection

```
{
  _:region foaf:depicts ?person;
           :regionId ?regionId;
           :belongsTo [:photoId ?photoId].
}
=>
{
  _:request http:methodName "GET";
            tmpl:absoluteURITemplate
  "http://example.org/photos/{photoId}/people/{regionId}";
            tmpl:uriTemplateValues (?photoId ?regionId);
            http:resp [ tmpl:represents ?person ].

  ?person foaf:name _:personName.
}.
```

Listing 5: RESTdesc description of face recognition

4.2 Automated interpretation and composition

{**TODO** Tell why RESTdesc is ideal for composition.}

4.3 Compatibility and automatic translation

{**TODO** Explain compatibility with CoIN, OWL-S, maybe WSDL etc., and how the translation can happen automatically.}

5 RESTdesc Service Discovery

{**TODO** While the previous section introduced the RESTdesc description format, this section details how we can start with a single URI and learn everything about the server.}

6 Real-world Use Cases

6.1 Automated discovery, composition, and execution

This use case presents an overview of the general architecture of RESTdesc for multimedia algorithms.
{**TODO** See Google Docs.}

6.2 Linking to external services

{**TODO** See Google Docs.}

6.3 Adapting to change and errors

{**TODO** See Google Docs.}

7 Conclusion

{**TODO** See Google Docs.}

References

1. Atrey, P., Hossain, M., El Saddik, A., S Kankanhalli, M.: Multimodal fusion for multimedia analysis: a survey. Multimedia Systems (2010), http://www.springerlink.com/index/E31M71152774R630.pdf
2. Berners-Lee, T., Connolly, D., Kagal, L., Scharf, Y., Hendler, J.: N3Logic: A logical framework for the World Wide Web. Theory and Practice of Logic Programming 8(3), 249–269 (2008)
3. Chinnici, R., Moreau, J.J., Ryman, A., Weerawarana, S.: Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C Recommendation (Jun 2007), http://xml.coverpages.org/wsdl20000929.html
4. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.0 (Sep 2000), http://xml.coverpages.org/wsdl20000929.html
5. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext Transfer Protocol – HTTP/1.1. Request for Comments: 2616 (Jun 1999), http://www.ietf.org/rfc/rfc2616.txt

6. Fielding, R.T., Taylor, R.N.: Principled design of the modern Web architecture. ACM Transactions on Internet Technology 2(2), 115–150 (May 2002), http://dx.doi.org/10.1145/514183.514185
7. Genereth, M.R.: Knowledge interchange format. Draft Proposed American National Standard, http://logic.stanford.edu/kif/dpans.html
8. Gregorio, J.: Do we need WADL? (Jun 2007), http://bitworking.org/news/193/Do-we-need-WADL
9. Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.J.: Soap version 1.2 part 1: Messaging framework (second edition). W3C Recommendation (Apr 2007), http://www.w3.org/TR/2007/REC-soap12-part1-20070427/
10. Hadley, M.: Web Application Description Language. W3C Member Submission (Aug 2009), http://www.w3.org/Submission/wadl/
11. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S.: Swrl: A Semantic Web rule language combining OWL and RuleML. W3C Member Submission (May 2004), http://www.w3.org/Submission/SWRL/
12. Klyne, G., Carrol, J.J.: Resource description framework (RDF): Concepts and abstract syntax. W3C Recommendation (Feb 2004), http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/
13. Lafon, Y.: Team Comment on the "Web Application Description Language" Submission (Oct 2009), http://www.w3.org/Submission/2009/03/Comment
14. Martin, D., Burstein, M., Hobbs, J., Lassila, O.: OWL-S: Semantic Markup for Web Services. W3C Member Submission (Nov 2004), http://www.w3.org/Submission/OWL-S/
15. McDermott, D.: DRS: A Set of Conventions for Representing Logical Languages in RDF (Jan 2004), http://cs-www.cs.yale.edu/homes/dvm/daml/DRSguide.pdf
16. McGuinness, D.L., van Harmelen, F.: Owl web ontology language overview. W3C Recommendation (Feb 2004), http://www.w3.org/TR/2004/REC-owl-features-20040210/
17. Verborgh, R., Van Deursen, D., De Roo, J., Mannens, E., Van de Walle, R.: SPARQL Endpoints as Front-end for Multimedia Processing Algorithms. Proceedings of the Fourth International Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web (SMR2 2010) (nov 2010)