AAAI Technical Report SS-12-04
Intelligent Web Services Meet Social Computing

A Social Description Revolution – Describing Web APIs' Social Parameters with RESTdesc

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

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

Joaquim Gabarro
Universitat Politècnica de Catalunya
Department LSI
08034 Barcelona, Spain
gabarro@lsi.upc.edu

Erik Mannens and **Rik Van de Walle**
Ghent University – IBBT, ELIS – Multimedia Lab
Gaston Crommenlaan 8 bus 201,
B-9050 Ledeberg-Ghent, Belgium
{*erik.mannens,rik.vandewalle*}*@ugent.be*

Abstract

Functionality makes APIs unique and therefore helps humans and machines decide what service they need. However, if two APIs offer similar functionality, quality attributes such as performance and ease-of-use might become a decisive factor. Several of these quality attributes are inherently subjective, and hence exist within a social context. These social parameters should be taken into account when creating personalized mashups and service compositions. The Web API description format RESTdesc already captures functionality in an elegant way, so in this paper we will demonstrate how it can be extended to include social parameters. We indicate the role these parameters can play in generating functional compositions that fulfill specified quality attributes. Finally, we show how descriptions can be personalized by exploring a user's social graph. This ultimately leads to a more focused, on-demand use of Web APIs, driven by functionality and social parameters.

1 Introduction and Paper Structure

Usually, descriptions of a Web API or service only cover certain aspects. They emphasize its technical aspects, such as parameters, and sometimes formally express its functionality. But how to decide between different API providers that offer similar services? How to express variations in objective (speed, accuracy, reliability…) and subjective (user-friendliness, versatility, popularity…) quality parameters? While one API provider may offer more features, another can be your provider of choice, because people in your social graph give it a high quality rating.

In practice, these objective *and* subjective parameters carry an importance equal to those in traditional API descriptions. This is where social computing opens up new perspectives for Web APIs: while providers supply the technical description for an API, users' social activities can provide the measurements for various quality parameters, from here on referred to as *social parameters*. Some of them, such as speed, can be gathered automatically. Others, such as accuracy and the subjective parameters, can be supplied by users in a variety of ways (tagging, voting, collaborating…). This could even happen independently from the API provider, e.g., on a third party API directory such as Programmable Web (DuVander 2011). The only question that remains is: how to combine those social parameters with the API description, so that they can be incorporated into the creation process of API mashups and compositions?

The API description paradigm RESTdesc (Verborgh et al. 2012), based on Linked Data principles (Bizer, Heath, and Berners-Lee 2009), provides advanced capabilities for capturing the functionality of APIs in a way that enables automated mashups and compositions. Today's social revolution on the Web, however, shows that the power of APIs is not solely defined by their functionality. Therefore, in this paper, we explain how RESTdesc can incorporate external social parameters to differentiate between potentially competing APIs with similar capabilities. This allows for mashups and compositions that reckon with social quality parameters, both objective and subjective, creating on-demand solutions that match more than only the functional needs of a specific context.

We start by explaining the basic principles and strengths of RESTdesc in Section 2, followed by the incorporation of social parameters in descriptions in Section 3. Then, we personalize these social parameters in Section 4 by collecting results from users' social graphs. Section 5 provides an overview of related work, and we conclude this paper in Section 6.

2 RESTdesc: Functional Descriptions

2.1 Focus on Functionality

If we go back to the origins of Web service descriptions, we see that the initial focus was mostly on their technical aspects, such as parameter types and possible exceptions. The central idea of RESTdesc, the description paradigm we proposed previously (2012), was to capture the main differentiating characteristic of Web services, namely *functionality*.

```
@prefix ex: <http://example.org/photos#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix http: <http://www.w3.org/2011/http#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
{
  ?photo ex:faces ?photoFaces. 1
}
=>
{
  _:request http:methodName "GET"; 2
            http:requestURI ?photoFaces;
            http:resp ?faces.

  _:region owl:oneOf ?faces; 3
           ex:regionOf ?photo;
           foaf:depicts [ a foaf:Person ].
}.
```

Listing 1: RESTdesc describes the functionality of APIs using links, illustrated here with a face detection example.

RESTdesc is based on Semantic Web technologies and emphasizes resource-orientation, hypermedia links, and elegance. It mainly targets RESTful applications (Fielding and Taylor 2002), which is why we employ the term "API" rather than "service". Descriptions are written in Notation3 or N3 (Berners-Lee and Connolly 2009), a small superset of RDF that adds support for variables and quantification. The example in Listing 1 uses a face detection example to show the common elements of RESTdesc descriptions. The displayed description is to be interpreted as follows: **IF** you have a photograph with a `faces` hyperlink 1 **THEN** you can make an HTTP `GET` request to that link 2 to receive regions in this photograph that depict a person. 3 As you can see, this description connects the photograph in a functional way to a list of persons.

Furthermore, the following properties give RESTdesc descriptions a substantial advantage in the Web API and Semantic Web context:

Hypermedia-driven RESTdesc strongly relies on hypermedia links and therefore enables efficient description of REST APIs, which, according to Fielding (2008), should respect the hypermedia constraint by driving their actions with hypermedia controls such as links and forms. In fact, the purpose of the description in Listing 1 is to explain what following a hyperlink of type `ex:faces` means. If an automated agent finds such a link on a photo page, it can expect that the link target will be a list of face regions.

Domain-centered Contrary to many other previous description formats, such as OWL-S (Martin et al. 2007) and WSMO (Feier et al. 2005), RESTdesc puts the application domain into the center of the description. In fact, one of the key points of REST is that the notion of "service" disappears, because the application is modeled in terms of resources instead of endpoints. The domain portion of Listing 1 consists of items 1 and 3, which are glued together by item 2.

Vocabulary reuse As a consequence, RESTdesc does not force description authors to use a certain vocabulary, but rather encourages to use vocabularies that best fit the application domain. This will particularly be an advantage when adding social parameters to descriptions in Section 3, since we will not be limited to a specific vocabulary. In Listing 1, we see the employed vocabularies are either commonly used (`owl`), directly related to the application domain (`foaf` and `ex`), or relevant to the employed protocol (`http`). A RESTdesc-specific vocabulary is not necessary nor desired.

These characteristics make RESTdesc descriptions an excellent candidate to bridge between REST APIs and the Semantic Web and Linked Data worlds.

2.2 Matching and Composition

A feature of RESTdesc especially relevant to social parameters is that is has been specifically designed to allow for straightforward API matching and composition. The *if-then* structure of the N3 rules in which RESTdesc is expressed also carries an operational semantics (Berners-Lee et al. 2008). For matching, this means that we can check if two services can connect by verifying that the consequent of one service entails the antecedent of another in a given context. This process is automated with reasoners such as EYE (De Roo 1999–2012). For composing, the procedure is similar: the current situation (precondition) and desired situation (goal) are fed to a reasoner, combined with a set of API descriptions. The reasoner then arranges the descriptions and chains them together as a path from precondition to goal (Verborgh et al. 2011b).

With regard to social parameters, this matching and composition capability is vital. The static description of social parameters is an interesting start, but what we really want is a method that incorporates those parameters when finding a solution. For instance, users may want a composition chain that organizes photographs based on depicted people, with a satisfaction rate of at least 90%. Scenarios like this one will be explored in Section 3 below. Further aspects of RESTdesc are described in previous works (Verborgh et al. 2011a; 2012), as well as on the website http://restdesc.org/.

3 Social Parameters in Descriptions

3.1 Significance and Role of Social Parameters

By capturing functionality, RESTdesc has a different focus than most previous description methods, several of which are summarized by Harth (2012). However, functionality by itself is still insufficient to obtain a complete picture of any API. For that reason, the measurement of social parameters is important, because it reveals several quality aspects that a functional description cannot convey. Listing 1 tells us that the described API performs face detection—but how fast does it work, how correct are its results, and is it easy to use? The answers to those questions are captured by quality parameters, gathered in a social context.

Figure 1 shows several of those parameters, some of which can be automatically computed, and some of which can only determined by humans. Depending on the application, some of these parameters can fit in one category or the other. The key message of this figure, however, is that

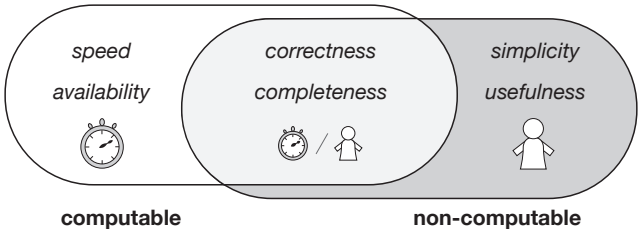

Figure 1: The calculation of several relevant quality metrics will always require human participation.

the *relevant* measurement of *any* parameter, whether computable or not, requires human interactions. For instance, we do not want to know the speed of a hypothetical API query, but the speed of actual queries performed by users. Similarly, 99.9% availability is meaningless if most users want to use the system the other 0.1% of time. Such benchmark disparities for Web applications have been documented before (Ratanaworabhan, Livshits, and Zorn 2010) and make the difference between a server-centric and a user-centric quality model. User-centeredness is, purely by definition, always subjective, but quantity can provide at least a tendency towards objectivity on very individually different aspects such as ease of use.

3.2 Measuring Social Parameters

Before we can add social parameters to descriptions, we must first explore ways of measuring them. Programmable Web (DuVander 2011) is an example of a portal where people can comment on APIs and assign a *rating* on a 1–5 scale. On other well-known mechanism is *tagging*, where user assign words of their choice to a service. In general, there are three different types of scales in use to measure quality parameters:

Ordinal scale Tags can be seen as an ordinal scale, albeit with a possibly infinite domain that included overlaps. For example, users may tag an API with one or more of *excellent*, *superb*, *average*, and *mediocre*. Clearly, *excellent* and *superb*, just like *average* and *mediocre*, must be grouped together, as they can be considered synonyms. Each quality attribute may have its own specific tags, such as *fast* and *slow* for speed.

Subjective ratio scale People may rate an API using various predefined quality attributes. Ratings can either be related to the entire API by itself (e.g., a ranking of 4 stars in Programmable Web) or concern specific aspects of a service (e.g., ease-of-use, performance). However, the meaning of a certain rating is a personal judgement.

Objective ratio scale Most objective metrics, such as availability and average response time, are performed by automated measurements and calculations. Their results are objective and verifiable.

In this paper, we will focus on values representable in one of the latter two scales, because they can be calculated with.

```
@prefix ex: <http://example.org/photos#>.
@prefix q: <http://example.org/quality#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix http: <http://www.w3.org/2011/http#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
{
  ?photo ex:faces ?photoFaces.
}
=>
{
  _:request http:methodName "GET";
            http:requestURI ?photoFaces;
            http:resp ?faces;
            q:duration 1.3;
            q:speed 0.9.

  ?photoFaces q:uptime 0.99;
              q:availability 0.95.

  ?faces q:correctness 0.9;
         q:precision 0.8;
         q:recall 0.7.

  _:region owl:oneOf ?faces;
           ex:regionOf ?photo;
           foaf:depicts [ a foaf:Person ].
}.
```

Listing 2: RESTdesc seemlessly interleaves quality properties with functional descriptions.

3.3 Adding Social Parameters to RESTdesc

Based on the RESTdesc characteristics outlined in Section 2, we will now present an extensible way to incorporate social parameters in RESTdesc descriptions. Since RESTdesc always adapts to the application domain, there is no technical reason to choose a specific ontology or vocabulary to describe the quality of a Web API. This is an important benefit, since we are not bound to a restricted set of social parameters, but can use as many different vocabularies as demanded by the use case. To keep the example generic, we will use a fictitious quality ontology `q`.

The central question is to which part of the description of Listing 1 we can add quality parameters. Obviously, the antecedent ❶ of the *if-then* rule is not the right place, since it only concerns the contextual precondition. The consequent consists of the request ❷ and response details ❸, both of which will be targets for quality information. Indeed, if we reflect upon the nature of the parameters, we see that some of them are a function of the *request* (e.g., execution time), some of the request's *resource* (e.g., availability), and finally some of the *response* (e.g., correctness). Therefore, they belong at that respective place in the description.

Listing 2 repeats the functional description of Listing 1, complementing it with various quality parameters. We will examine different types of quality parameters, based on their role, in closer detail below.

Request parameters In the example, we want to describe the expected amount of time it takes to execute a request. An objective measurement would be the average request time in seconds, which is supplied by the `duration` parameter. However, this value might not

help users decide—does 1.3 seconds mean the API is fast or not? Therefore, a social parameter `speed` explains user perception of temporal performance on a 0–1 scale, the 0.9 value indicating that 1.3 seconds is considered fast.

Resource parameters The availability measurement has been attached to the resource, because it makes no sense to talk about the availability of requests: a given request is either executed or not, i.e., availability is a probability not for a *specific* request but for the whole of *all* requests. Note the difference between the objective `uptime`, indicating the percent of time the resource is in normal operation, and `availability`, the percentage of demands that are correctly answered. The later is a user-centered metric, as discussed in Subsection 3.1.

Response parameters Finally, the API's response is scored on its quality. The classical `precision` and `recall` parameters are evaluated, here by respectively counting the number of correct face detections versus the total number of returned regions, and by counting the number of undetected faces versus the total number of faces. These scores can only be exact if rated by humans, due to current visual algorithm limitations. Another quality parameter, `correctness`, indicates how closely the returned regions fit the boundaries of the detected faces.

An important technical remark we should make here, is that the possible difference between *expected* quality values (e.g., a duration of 1.3 seconds) and *actual* values (e.g., the execution took 2.4 seconds) is *not* a contradiction. After all, the logical interpretation RESTdesc description rules is that the satisfaction of the preconditions implies the existence of *some* request and response with the given parameters (as opposed to *all* requests). Whether or not the actual request will have quality attributes similar to those of that hypothetical request is not stated, but only reasonably expected.

By virtue of the parameters above, the question whether a certain Web API can satisfy a functional requirement *and* certain quality constraints, based on a social context, can now be answered with matching. Furthermore, these parameters can propagate into a composition to estimate their value in chains of several API calls.

4 Personalized Descriptions

The previous section leaves an important question open: how do the personal parameter values from one user reach other users of the API? In fact, this also brings up trust issues: how do you know whether an advertised quality rating is truthful, and if it is, did all users that contributed to this rating have sufficient expertise to do so? For example, the API ratings on Programmable Web seem trustworthy, since this website serves as an independent information source. However, there is no guarantee that competing API providers offering a similar service, if economical or other motives are involved, cannot artificially influence ratings. And even if there would be such a guarantee, the significance of an average rating value to an individual user remains unsure, because that user is unaware of the background of people that rated the API.

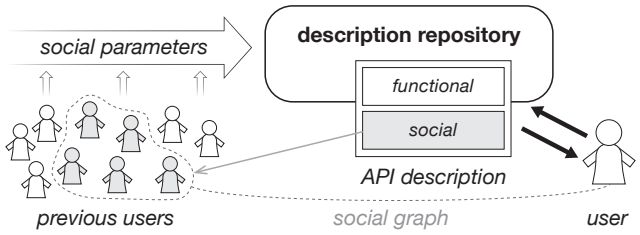

Figure 2: The repository combines functional descriptions with social parameters from a user's social graph.

In essence, this is a question of provenance data (Golbeck and Hendler 2008), telling us what information comes from what source. Based on this provenance data, the trust in the information should be determined (Golbeck, Parsia, and Hendler 2003). This solution thus starts from a large data collection, out of which a selection of trusted information for a user is distilled.

Our approach for personalized descriptions will work the other way round. Based on a user's social graph, we search for people that the user estimates as sufficiently trustworthy and experienced to give an opinion on a certain API, with the obvious fallback option to use the data from people outside of the user's social graph if necessary. In the past, we have suggested several description discovery methods for RESTdesc (Verborgh et al. 2012). They can either be provider-based, when the API provider itself offers the descriptions, or repository-based, when a third party offers the descriptions of various APIs. We will discuss personalized descriptions with repository-based discovery mechanisms here, since this is the most straightforward solution.

The idea is similar to websites such as Programmable Web where people can rate and comment on a service, but the difference is the locus of control. The driver are social networks on which the user has an account, such as Facebook (Joinson 2008) or Twitter (Huberman, Romero, and Wu 2009). Each API is uniquely identified by its base URL, and this identifier is used to group reactions together. When users want to comment on an API, they can create key-value annotations, likely using an assisting tool. The keys come from a fixed set, for instance, from a quality ontology such as the one featured in Section 3. The ratings can either be stored in their social network, e.g., with the Open Graph Protocol (Zuckerberg and Taylor 2010) in the case of Facebook, or along with the descriptions in repository, e.g., in the case of Twitter, which doesn't have a data store.

When a user then requests an API description, the repository will request access to the social graph of one or more networks. Possibly, the user can also indicate which groups within these graphs he or she deems authorized to assess the quality of APIs. The repository then traverses the given (part of) the social graph, looking for quality ratings for the requested API. A weighted average, depending on the position in the graph, then forms the definitive score. If no such data is available in the social graph, or if the user does not wish such personalization, then a global average can still be used as an approximation.

Finally, the repository combines the static, functional version of the description (such as Listing 1) with the retrieved social parameters to obtain a personalized description (Listing 2), as can be seen in Figure 2. The decision where each parameter has to be placed (request, resource, or response) can be documented in the quality ontology, since this placement depends only on the property and not on the assigned value. In the end, this results in descriptions that not only indicate the functional properties of an API, but also the quality parameters as perceived by members of the user's social graph.

5 Related Work

Related work can be split in two parts: (i) research on describing Web APIs in general and (ii) research on judging and documenting Web API quality. We have treated Web API description efforts in detail in (Verborgh et al. 2012) and therefore focus exclusively on Web API quality in the current paper. A famous quote by Robert M. Pirsig in his book *Zen and the Art of Motorcycle Maintenance* goes "even though quality cannot be defined, you know what it is". Common terms in the context of Web API quality are *Quality of Service (QoS)*, usually referring to network-related quality factors such as service response time, latency, or loss; and *Service Level Agreements (SLAs)*, documents in plain language terms defining among others the mean time between failures, the mean time to repair, and the mean time to recovery. An exemplary SLA document guaranteeing a 99.9% uptime for a concrete Web API can be seen in Google, Inc. (2012).

The ISO/IEC 9126 standard (2001), partly superseded by the standard ISO/IEC 25000 (2005), defines quality metrics for software that, to some extent, can be applied to Web APIs as well. Concretely, these quality metrics are *functionality*, *reliability*, *usability*, *efficiency*, *maintainability*, and *portability*.

In Erradi, Padmanabhuni, and Varadharajan (2006), the authors motivate research to extend Web services management platforms with more sophisticated control mechanisms to cater for differentiated service offerings given the variation of contexts in which a Web service could be used and the resulting variation in QoS expectations. Where most Web services platforms are based on a best-effort model, which treats all requests uniformly without any type of service differentiation or prioritization, the authors present *WS-DiffServ*, a service differentiation middleware based on prioritization, which leverages service requestor profiles to classify service requests.

Tosic describes in his Ph.D. thesis (2004) so-called *classes of service*, a mechanism for differentiation of service and QoS that, according to the author, incurs less overhead than custom-made SLAs, user profiles, and other alternatives. For the concrete description of service classes, Tosic developed the Web Service Offerings Language (WSOL), which is compatible with the Web Service Definition Language (WSDL).

Ludwig et al. from IBM Research describe the Web Service Level Agreement (WSLA) language (2003). WSLAs are agreements between a service provider and a customer and as such define the obligations of the parties involved, primarily the obligation of a service provider to perform a service according to agreed-upon guarantees for service parameters, such as availability, response time and throughput. Guarantees are defined in detail, including the quality measurement algorithms.

An OASIS specification (Kim et al. 2011) provides a standard for quality factors of Web services in their development, usage and management. Web services usually have distinguished characteristics and, as a result, require their own quality factors. For instance, as the quality of Web services can be altered in real-time according to changes by the service provider, considering real-time properties of Web services is very meaningful. The specification presents quality factors of Web services with definition, classification, and sub-factors case by case.

6 Conclusion

Functionality is an essential element in API descriptions, but it does not tell the whole story. To compare APIs with similar functionality from different providers, quality parameters need to be taken into account. These parameters are all the more relevant if they are are assessed by people in one's social graph, effectively turning them into social parameters. RESTdesc is able to integrate functional elements and social parameters elegantly into a single description that allows autonomous agents, on behalf of their owners and reckoning with their owners' social connections, to decide what API is the best to use in a given context.

In more than 20 years of Web development, a lot of research efforts have been put into describing Web services. Fundamental questions in service matching and composing under quality-of-service constraints have been tackled. However, the recent revival of resource-driven REST APIs, as opposed to the heavy WS-* services, combined with the flourishing of social networks, leads us to the question whether the old solutions are fit for today's Web.

RESTdesc offers descriptions that prioritize finding the right API for the job, and strives to do this with the elegance that is ubiquitous in modern resource-oriented design. We believe that descriptions should focus on a user's perception of the application instead of on technical details. Therefore, embracing social parameters has been a logical step in the RESTdesc vision, which is evident from the seamless integration of these parameters into existing descriptions. You can follow the evolution of RESTdesc at http://restdesc.org/.

Acknowledgments

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. Joaquim Gabarro is partially supported by TIN-2007-66523 (FORMALISM), and SGR 2009-2015 (ALBCOM).

References

Berners-Lee, T., and Connolly, D. 2009. Notation3 (N3): A readable RDF syntax. W3C Recommendation. Available at http://www.w3.org/TeamSubmission/n3/.

Berners-Lee, T.; Connolly, D.; Kagal, L.; Scharf, Y.; and Hendler, J. 2008. N3Logic: A logical framework for the World Wide Web. *Theory and Practice of Logic Programming* 8(3):249–269.

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

De Roo, J. 1999–2012. Euler proof mechanism. Available at http://eulersharp.sourceforge.net/.

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

Erradi, A.; Padmanabhuni, S.; and Varadharajan, N. 2006. Differential QoS support in Web Services Management. In *Proceedings of the IEEE International Conference on Web Services*, 781–788. Washington, DC, USA: IEEE Computer Society.

Feier, C.; Polleres, A.; Dumitru, R.; Domingue, J.; Stollberg, M.; and Fensel, D. 2005. Towards intelligent Web services: the Web Service Modeling Ontology (WSMO). In *2005 International Conference on Intelligent Computing (ICIC'05)*.

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

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

Golbeck, J., and Hendler, J. 2008. A Semantic Web approach to the provenance challenge. *Concurrency and Computation: Practice and Experience* 20(5):431–439.

Golbeck, J.; Parsia, B.; and Hendler, J. 2003. Trust networks on the Semantic Web. In *Cooperative Information Agents VII*, volume 2782 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg. 238–249.

Google, Inc. 2012. Google Maps API Premier Service Level Agreement. Available at http://www.google.com/enterprise/earthmaps/maps˙sla.html.

Harth, A. 2012. Linked Services. Available at http://linkedservices.org/.

Huberman, B.; Romero, D.; and Wu, F. 2009. Social networks that matter: Twitter under the microscope. *First Monday* 14(1):8.

International Organization for Standardization. 2001. ISO/IEC 9126-1:2001. Available for purchase at http://www.iso.org/iso/iso˙catalogue/catalogue˙tc/catalogue˙detail.htm?csnumber=22749.

International Organization for Standardization. 2005. ISO/IEC 25000:2005(E). Available for purchase at http://www.iso.org/iso/iso˙catalogue/catalogue˙tc/catalogue˙detail.htm?csnumber=35683.

Joinson, A. N. 2008. Looking at, looking up or keeping up with people? – motives and use of Facebook. In *Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, CHI '08, 1027–1036. New York, NY, USA: ACM.

Kim, E.; Lee, Y.; Kim, Y.; Park, H.; Kim, J.; Moon, B.; Yun, J.; and Kang, G. 2011. Web Services Quality Factors Version 1.0. 22 July 2011. OASIS Committee Specification 01. Available at http://docs.oasis-open.org/wsqm/WS-Quality-Factors/v1.0/cs01/WS-Quality-Factors-v1.0-cs01.html.

Ludwig, H.; Keller, A.; Dan, A.; King, R. P.; and Franck, R. 2003. Web Service Level Agreement (WSLA) – Language Specification. Available at http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf.

Martin, D.; Burstein, M.; McDermott, D.; McIlraith, S.; Paolucci, M.; Sycara, K.; McGuinness, D.; Sirin, E.; and Srinivasan, N. 2007. Bringing semantics to Web services with OWL-S. *World Wide Web* 10:243–277.

Ratanaworabhan, P.; Livshits, B.; and Zorn, B. G. 2010. JSMeter: Comparing the behavior of JavaScript benchmarks with real Web applications. In *Proceedings of the USENIX Conference on Web Application Development*.

Tosic, V. 2004. *Service Offerings for XML Web Services and Their Management Applications*. Ph.D. Dissertation, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada.

Verborgh, R.; Steiner, T.; Van Deursen, D.; Van de Walle, R.; and Gabarro, J. 2011a. Efficient Runtime Service Discovery and Consumption with Hyperlinked RESTdesc. In *Proceedings of the 7th International Conference on Next Generation Web Services Practices*, 373–379.

Verborgh, R.; Van Deursen, D.; Mannens, E.; Poppe, C.; and Van de Walle, R. 2011b. 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*.

Verborgh, R.; Steiner, T.; Van Deursen, D.; De Roo, J.; Van de Walle, R.; and Gabarro, J. 2012. Description and Interaction of RESTful Services for Automatic Discovery and Execution. *Multimedia Tools and Applications*. Accepted for publication.

Zuckerberg, M., and Taylor, B. 2010. The open graph protocol. Available at http://www.opengraphprotocol.org/.