Feed Querying as a Proxy for Querying the Web

Erik Wilde[1] and Alexandros Marinos[2]

[1] School of Information, UC Berkeley, USA, dret@berkeley.edu
[2] Department of Computing, University of Surrey, a.marinos@surrey.ac.uk

Abstract. Managing information, access to information, and updates to relevant information on the Web has become a challenging task because of the volume and the variety of information sources and services available on the Web. This problem will only grow because of the increasing number of potential information resources, and the increasing number of services which could be driven by machine-friendly access to these resources. In this paper, we propose to use the established and simple metamodel of *feeds* as a proxy for information resources on the Web, and to use feed-based methods for producing, aggregating, querying, and publishing information about resources on the Web. We propose an architecture that is flexible and scalable and uses well-established RESTful methods of loose coupling. By using such an architecture, mashups and the repurposing of Web services is encouraged, and the simplicity of the underlying metamodel places no undue restrictions on the possible application areas.

1 Introduction

The Web's most astonishing feature is its magnitude and versatility, and thus the vast amount of information that is available in its resources. However, this feature also is one of the fundamental problems of the Web: the more information there is, the harder it gets to find anything. Following the initial and eventually failed attempts to categorize all Web content in carefully compiled and organized directories, search engines have transformed the Web into a system which is fully indexed[3] and thus allows to quickly search the Web and find resources based on simple (but scalable) keyword search.

Currently, search engines are targeted at human users and thus mostly focus on user interfaces; however, *OpenSearch* is an attempt to specify an API which can be used for any web-based service providing (keyword-oriented) search services. Even so, many aspects of the search service are unspecified or implicit, such as the implicit `ORDER BY` based on methods such as PageRank [14], which is critically required for the typically loosely defined keyword-based searches. So while search engines have certainly transformed the Web and helped it grow to the scale that is has today, their focus on a simple and human-oriented search

[3] With the notable exception of what often is referred to as the "Deep Web" [7], which is all information not accessible to crawling techniques.

service limits their applicability to more focused query access to specific collections of information resources. Going beyond this human-centered approach, the specific question discussed in this paper is: "Is there a way in which various information publishers and information brokers can provide their services for effective and easy reuse?"

In the earlier days of the Web, the concept of "Web Query Languages" [9, 11] looked appealing; the idea was to have a query language that operates on the graph of interconnected Web resources, and retrieves those resources matching a given query. One problem with that approach is the definition of the query language itself (it would have to be general to work for many different application areas, but on the other hand specific enough so that queries could be reasonably optimized), and the other problem is whether it is reasonable to assume that "the Web" is usable as "the input" to such a general query language.[4] Given the ever-increasing size of the Web, the increasing ratio of non-static pages, and the diversity of Web content, it seems unlikely that it will be possible to define a "Web Query Language" that satisfied the wide variety of possible use cases in this area.

Looking at the development of the Web landscape, it can be seen that feeds such as *Atom* [12] (a reformulation of the various RSS versions) have become a popular way in which information is made available on the Web today, mostly by information providers who regularly publish new information resources and want this stream of information items to be easily available. *Syndication* arguably is the most widely available "Web Service" on the Web — however, in most cases, it is used in a rather constrained way, allowing the client no or little control, and focusing on time-ordered collections and results. But for the purpose of this paper, it is important to notice that simple syndication can be regarded as a "query" into a site or service; as a query to get the latest page of results of a permanently updated collection of entries. This is a loose model with, in the most limited case, no query parameters, but it has all the properties of a query model (an input, a query model, and a result), even though many of the essential parts (such as the data model of the dataset and the query language) are implicit and thus not well-defined. Section 2 examines in more detail how such a query model could be defined so that typical Web service scenarios can be mapped to the feed metamodel.

In a different scenario, the *Semantic Web* has its own view of Web resources, based on resources themselves, and descriptions which are using RDF as the metamodel. The standard query language for the Semantic Web is *SPARQL* [17], which works on an RDF graph. It thus makes the same assumption as some of the early Web query languages: that all the required input data is readily available

[4] Search engine providers will have such a "query language" for the internal representation they have of their Web crawls, but these "query languages" are highly specific to the way the crawl data is being managed and not available for public access. A notable exception to this are features such as Google's `link:` search, which allows users to find pages that link to a given URI, giving them the ability to find incoming links for a given URI.

for querying somehow. In this paper, we do not make this assumption, and instead assume that resources and services are distributed, heterogeneous, and not necessarily published, so that any scenario that is based on a homogeneous and/or complete view of Web resources misses essential parts of the evolving landscape of resources, services, and service systems.

One of the emerging areas that will cause a new wave of many new resources and services and ways to interact with them is the "Web of Things" [6, 20], the idea that increasingly, real-world objects will become available on the Web. The important aspect about the "Web of Things" vision is that not all "things" will necessarily directly be accessible by HTTP, but that the Web's RESTful architecture is perfectly suited to access them through intermediaries which can then use specialized back-end implementations to actually physically interact with the networked "things". *Representational State Transfer (REST)* [4] is the key ingredient in this model, because access to the resources uses a uniform interface which then allows intermediaries to be transparently deployed and used in a scalable manner.

In previous work, we looked at feeds as query result serializations [21], the idea being that the generic and loose structure of feeds can serve as a general abstraction for returning query results in a large variety of scenarios. Currently, the implicit "query" associated with feeds often is "the last 10 entries of a predefined search set in reverse chronological order", but this is only due to the fact that feeds originated in the area of news feeds and content syndication in general, where such a query was the implicit assumption. Another proposal for a general adoption of feeds as a container format for Web service interactions is the idea of *feeds of feeds* [23], where feeds are used as a packaging mechanism to communicate information *about* feeds, and more generally speaking, the observation that feeds satisfy many of the requirements that constitute loose coupling [15] between service providers and consumers. Ongoing research [3] in the area of nested feeds looks at the possibilities to manage collections of feeds by exposing them as feed-represented collections themselves, but this issue is out of the scope of this paper.

2 Mapping to Feeds

While this is a generalization that does not apply in all cases, it is probably safe to say that many services delivered across the Web (some through machine-accessible Web services, the vast majority through human-oriented user interfaces) have some part to them that is about *collections* of *resources*, and providing access to these collections and resources. This generalization can be understood even better in its relation to feeds when looking at *AtomPub* [5], the protocol that extends interactions with feeds to a read/write scenario. In AtomPub, a feed is simply a *view into a collection*, and typically only exposes a part of that collection, and in a specific way (such as in a time-ordered manner).

It is thus important to stress that feeds are not just a wrapper around periodically updated content that has to be delivered to a client. They are a *service* that

makes content available, and this service can be as simple as a time-ordered sequence (in the simplest case with no capability to page back and thus essentially implementing ephemeral access to information resources), or as sophisticated as a service that provides features such as personalization, localization, customization, and the ability to use specific access parameters. This latter and more sophisticated view of feeds as a generalization of Web services of course does not comprise all possible Web service scenarios; there are many scenarios where the simple collection/resource metamodel may be too simple to serve as an adequate foundation for a service. However, we argue that feeds strike a useful balance between simplicity and ease-of-use, and the fraction of Web-oriented scenarios in which they can be used as a foundation for exposing a service.

Based on this *service perspective of feeds*, it is possible to conceptualize feeds as the fundamental metamodel underlying a wide variety of service offerings on the Web.[5] There even is an increasing number of publicly available tools and services that allow to "reverse-engineer" feeds from regularly updated web pages.[6] These approaches, however, suffer from the same drawbacks as all approaches based on reconstructing implicit models from HTML layout structures: they are brittle and can easily break when the Web page changes its formatting, and the scraping of a Web page is unable to reconstruct any information that is contained in the underlying collection, but not in the Web page representation. In particular, feeds provide metadata for feeds and entries that in many cases can be used to publish a rather rich description of the available entries, whereas reverse-engineered feeds often lack this richness in metadata.

However, while we believe that the simple and basic metamodel of feeds (unordered collections of resources) is general enough to serve as an abstraction that can represent an interesting share of the resources and services available on the web, there of course are much more sophisticated data models which cannot be simply mapped to feeds without loosing essential information. In some cases, the sophistication might come in terms of more metadata (in which case Atom's extensibility can handle the additional information through extensions, *podcasts* are an example for this kind of metadata sophistication), in others, it might come in terms of more complex relationships than just collections and resources. In theory, of course, such relations could be expressed by extending the Atom format as well, but as with all simplifications, if a simple foundation has to be stretched and extended substantially, the advantages of using such a foundation might be offset by the disadvantages of being restricted by it.

For the purpose of this paper, we propose that for an interesting subset of the resources and services made available on the Web, it is possible to map them to the simple feed metamodel, possibly by adding to it a modest set of extensions to the basic metamodel.

---

[5] As mentioned above already, by extending the reach of the Web beyond information resources alone, this view also comprises services of and for real-world objects [20].

[6] `http://feed43.com/` and `http://www.feedyes.com/` are examples for such services. All these tools work by defining mappings from Web page structures to feeds that should be derived from them.

3 Feed Aggregation

One of the major advantages of using a common model for services around collections of resources is not only the fact that users can use well-known and established tools for interacting with these services, but also, that these services become easily composable, because they are built around the same metamodel and thus there is no need to harmonize their models. When looking at the current landscape of feeds on the Web, there are two major practical problems that need to be addressed for easy aggregation:

- *Feed Formats:* While the feed metamodel of collection and resources is the same across all feed formats, there are a variety of formats in use (a considerable number of often slightly incompatible RSS versions and Atom). Furthermore, many feeds are broken in a variety of ways, so consuming them takes some error-tolerant processing similar to what is required for processing HTML from the Web. There are numerous libraries/packages in various programming languages that make this task easier, but they sometimes come with their own undesirable side-effects such as the inability to process extensions if they are not explicitly supported.
- *Collection/Resource Metadata:* The set of properties that can be associated with collections and/or resources varies across the various feed formats, and this is further complicated if extensions are taken into account. As an example, geolocation information associated with a resource can be expressed in various formats, some of which just use a different syntax (*GeoRSS* [13] vs. the older W3C geolocation format for feeds), whereas others (such as Yahoo!'s WOEID) use a different model of geolocation and thus require non-trivial mapping to be implemented if for example feeds with different geolocation methods should be aggregated and then managed using a single and consistent geolocation model.

The key finding from these observations is that feed aggregation is not necessarily trivial because of varying feed formats and possible problems with varying metadata properties, but that it is facilitated by the fact that the underlying metamodel is structurally similar across all feeds. So based on this ability to combine various feed-based data sources or services reasonably easily (aggregation) the next question then is how to narrow down results so that only a subset of resources in a collection are selected. At this point it might be worth pointing out that the approach taken here has a lot of similarities with the approaches taken in *federated databases* scenarios; in these scenarios the exact metamodel and features of the participating databases is unknown, but they can be dealt with homogeneously by mapping their individual data models to a shared metamodel; this is what feeds allow us to do for the approach presented here.

4 Feed Queries

Starting from the general idea of feeds as the universal metamodel for data sources and services, and thus gaining the ability to aggregate them, the next

logical question is how to be *more specific*, i.e. how to query them. Queries for feeds can be regarded on two levels: first, how to query a feed that is the representation of a single underlying collection. In this case, querying the feed has to be mapped to the data management facilities available for that collection. The second and more challenging scenario is where a feed is aggregated (as described in Section 3), and thus the query, although logically pertaining to the single feed, actually may require distribution across the various aggregated sources. In this second scenario, various challenges have to be met, because in some cases it may be possible to propagate queries upstream, but for join-type queries, there is a limit to this ability. While this second scenario is relevant for the future vision of the Web as a feed-based service system, it is out of the scope of this paper.[7]

The most fundamental question for any query language is the data model it is operating on (which sets the boundaries for possible language features). Apart from simple text search (the underlying assumption of *OpenSearch*) which often only provides simple combinational operators such as AND, OR, NOT, and grouping, most query languages imply a structured data model on which they operate. The most relevant languages for the Web-oriented view we are presenting here are probably *SQL* [8] (operating on the relational metamodel), *XQuery* [2] (operating on XML), and *SPARQL* [17] (operating on RDF). Naturally, another area of interest would be the various (spatio-)temporal languages used in streaming systems [1], because this would allow to extend the query scope and model to also cover spatial and/or temporal dimensions. However, research on how to best expose real-time streams on the Web has only started recently, and so far there is no detailed study which data model and query languages would best fit the Web, and how it should be exposed in an architecturally sound way. To make the step from specialized standalone systems to making them available on the Web level, there are two major research challenges:

– *Web-oriented Exposure:* Many stream-oriented systems have sophisticated and dedicated real-time features that make them ideal for challenging real-time applications. However, some of those properties may not make a lot of sense for direct delivery over the Web, because the Web lacks the highly optimized architecture and engineering that can only be guaranteed in tightly coupled settings. In this case, the question to ask is *what* to expose on the Web. For example, it may not make sense to expose a real-time stream of

[7] It is interesting to note that many approaches and methods of the database field are applicable here. For example, for the problem of how to handle distributed query processing in the presence of multiple and potentially heterogeneous underlying data providers, the database field has come up with *distributed query processing* and with *data warehouses*, two approaches where the first one analytically optimizes query processing, whereas the second one conveniently first collects all relevant data, and then provides query capabilities on this unified dataset. For realtime data, though, data warehousing usually is not a viable option because of inherent offline nature and thus delays of this approach.

high-volume data, but it may make a lot of sense to expose services derived from it such as summaries or notification services.
- *Web-oriented Interaction:* As the second step after deciding *what* to expose, an equally important question is to decide *how* to expose it. In this paper, we argue that feeds are a good foundation for many information services on the Web, but there also are other methods, for example publish/subscribe models such as the *Extensible Messaging and Presence Protocol (XMPP)* [19]. It depends on the scenarios (publish/subscribe ratio, publishing frequency and variance, acceptable delays) which Web-oriented interaction model fits best (Section 5 goes into more detail), and we argue that feeds should be the default solution, if there are no indications that they should not be used.

For the purpose of this paper, we assume that there will be a substantial number of use cases in which the second question will result in the choice of feeds as the interaction model of choice, or as one interaction model of choice. The polling model of feeds does introduce some delays and inefficiencies from the viewpoint of the client, but on the other hand it introduces optimization potential in intermediaries, and more importantly significantly reduces the complexity on the server side.

There currently is no established standard for a feed query language, even though some sites provide services where users can query feeds and thus can retrieve custom results for their queries [21]. Providing query services for a feed has implications for the scalability of feeds, because instead of publishing just one feed, a feed provider supporting queries needs to either publish a number of feeds for pre-computed query results (if there are a small number of possible or likely queries), or it has to actually execute the query on the collection when such a feed query is issued. Thus, while feed query features are a useful extension of the feed model, the performance implications need to be taken into account, and one possible strategy would be to provide feed query features as value-added services, so that clients using these features could be better controlled (and maybe even charged).

5 Pull vs. Push

One of the most important constraints of feeds is that they implement a *pull model*, in which the feed is made available by the provider, and consumers access it by "pulling" it from the provider. This model thus creates a certain amount of overhead, since pulling consumers might pull more often than new information is available.[8] HTTP features such as conditional requests and caching intermediaries help to reduce the overhead, but regardless of these optimizations, there is some inherent inefficiency in the content delivery of the pull model.

The alternative model is the *push model*, sometimes also referred to as *Publish/Subscribe (PubSub)*. In this model, the provider notifies consumers when

[8] Unfortunately, Atom does not even have metadata to indicate what the most effective pulling strategy might be (RSS 1.0 has such a field).

new content is available, eliminating the polling overhead of the pull model. It is important to realize, however, that the push model trades efficiency of content delivery for complexity of subscription. In essence, the price to pay for better optimized content delivery is more complex subscription management, and thus the question of whether push or pull is the "better" model depends on the application scenario in terms of subscriber base, subscriber churn, disappearing subscribers, and the pattern of how often and how regularly new content is made available. The following issues are relevant when making the choice between push or pull models:

- *Ease of Deployment:* Pushing trades its higher efficiency in communications for a more complex management of subscriptions and the need to asynchronously handle incoming notifications. All these issues can be handled in application programming frameworks, but they are establishing a more complex framework than a simple pulling model.
- *Loose Coupling:* Pushing requires subscriptions which need to be managed somewhere, and there are many potential problems such as repeated un- and re-subscriptions, orphaned subscriptions, and the general question of how to address a subscriber when something needs to be pushed. Generally speaking, pushing requires a more complex management framework that couples both the provider and the consumer rather tightly to that specific framework, whereas the more lightweight pulling approach creates a less tight coupling and typically makes it easier for providers and consumers to switch infrastructures respectively services.
- *Anonymity:* Push notifications and anonymity are two conflicting goals. There are various mechanisms which allow anonymous use of push-based services (such as *anycast* services), but these are not widely deployed and again create a rather tight coupling to such a mechanism if a service is using them.

The trend on today's Web is to use push models in the minority of cases, or only in cases where there is a tighter inherent coupling because of the given scenario (for example, when a service is accessed by Ajax applications, users using these applications are more likely to be online and actively working, in which case it makes sense to use a push model for this setting).

6 RESTful Database Queries

Throughout this paper we suggest using query mechanisms for feeds as a proxy for representing query services on the Web. An area where this approach can be applied is direct RESTful access to databases, based on RESTful interactions with the databases' metamodels (instead of designing a specific RESTful service for each data model implemented in the databases' metamodel). This could be tackled for all major database metamodels and the languages associated with them, such as the relational model (SQL), XML (XQuery), RDF

(SPARQL), column stores, and spatio-temporal databases. The feed-based approach has advantages because it assumes a homogenous metamodel and thus collections managed in different types of databases could still be accessed uniformly in such a setting; the feed query would be mapped to the databases' native query language as far upstream as possible, probably at the feed where only one collection is exposed by one specific database. On the other hand, the approach to design RESTful interactions with specific database types would expose more specific features of the underlying metamodel, and could thus allow for more specialized interactions with these collections.

As an application of the ideas expressed so in this paper, we discuss a prototype implementation of a RESTful querying mechanism tailored to the relational model. The principle behind this approach is to make a relational database available as a Web-native data service with no extra configuration. To develop our approach, we incrementally apply the constraints that REST as an architectural style entails to the question of interfacing with a database. At the most fundamental level, lies the question of *resource identification*. We consider our fundamental resources to be a records, tables, and query results. These resources need to be given URIs within our service. This enables usages such as users exchanging query URIs over email or IM protocols, a common use case on the Web which however is rather unusual when it comes to database queries.

Closely connected to the issue of addressability is the constraint of *statelessness*, stating that each request should contain all the information needed for the server to execute it with no assumption of extra server-side state. This, apart from being necessary for the URI exchange scenario, also leads to simpler service design with better scalability properties. In our case, this would require the use of HTTP Authentication for access control, and the creation of a format to serialize SQL-like queries in the URI addressing of each query.

Furthermore, REST mandates the *manipulation of resources through representations*. Our resources should be available in different representations, depending on the context. For instance, a human may want to interface with the data by using an HTML rendering, while an automated client may find plain XML more efficient. A feed reader would certainly find feeds the most suitable formats and it seems that feeds are a good general format for communication of collections of resources, such as a query result, between machines. However REST does not force us to standardize on one representation type for our resources, encouraging the availability of choice.

The *uniform interface* is another constraint of REST, aiding in the scalability and evolvability of its applications. It requires that resources are accessed through a standard set of operations. In the case of HTTP, these operations include GET, PUT, POST, and DELETE. AtomPub uses all these and by mapping our tables to Atom feeds we can utilize these capabilities to give to clients the ability to edit the data in the database, possibly subject to authentication.

Finally, there is the constraint of *hypermedia as the engine of application state* (HATEOAS), sometimes called connectedness. This constraint goes beyond addressability and requires all resources be linked to from other resources.

requires resources. With HATEOAS in place, a client can discover an API piecemeal by simply following links. Also, any change in the naming of a resource can be compensated on the client side simply by rediscovering the renamed resource. This enhances the loose coupling of RESTful APIs to their clients as it creates a client-side recourse against breakage and a viable alternative to the complexity of managing multiple versions of an API for the sake of backwards compatibility. In our RESTful database access case, HATEOAS can be applied by linking from queries to instances, but also from queries to other queries possibly in such a way that it is possible, starting from a general query that selects all the fields in a table, to navigate to a fine grained query which uses all the features of the querying language to frame a precise result. The consequence of a successful HATEOAS application would be to render the resource naming scheme irrelevant as clients would be able to rediscover the URI of a specific resource, whether query, table or record, if it happened to be renamed, simply by following the links it followed to locate it in the first place, assuming a known entry point.

The discussion of a RESTful Querying service based on the Atom metamodel is based on implementation experience of prototype which is being developed in collaboration with Jiannan Lu of the University of Surrey and Dimitris Michalakos of the University of Piraeus. The principles discussed above have either been implemented or are currently being implemented. Next steps for this research are the capability to create new databases and edit the schema of a database, again based on the feed metamodel. Also, the RESTful database framework can become an excellent testbed for the so-far theoretical work on RESTful Transactions [10].

7 Conclusions

In this paper we have argued that feeds and feed-oriented query mechanisms can serve as a proxy for querying the Web. While we are not claiming that all applications on the Web can or should use feeds as a general mechanism to provide access to resources to provide query services for them, we do argue that the lightweight approach of feeds, and the simplicity of the underlying metamodel turn them into "the HTML of machine-readable data." In the same way as HTML's ability to simply publish sequences of simple text structures has turned the Web into a tremendously successful publishing platform,[9] in the same way the ability of feeds to provide access to collections of resources can replicate this success on the level of machine-oriented access to resources and services.

Increasingly, the Web will see large-scale efforts to provide information to vast data collections such as the reporting information for the *2009 U.S. Recovery Act* [22]. In this case, the goal is to provide simple access to the reporting information of how the $787 billion of the Recovery Act are being spent. Instead

[9] And it is important to note that while Web-based publishing has become widespread, there are still publishing scenarios for which HTML is not a good fit and that use different methods for a good reason.

of providing a specialized API for retrieving reporting information, we proposed to make available feeds which would carry the reporting information in a way that would allow anybody to subscribe to it through a simple feed reader. More sophisticated features would allow interested parties to subscribe to reporting feeds configured to their personal preferences, taking the reporting information (such as funded agency, subcontractors, geolocation, and job creation performance indicators) as input for a "recovery feed query language." The Recovery Act reporting infrastructure is a work in progress, but it is significant in its scale, and in its legislated requirement to make the information available to the general public, not just to a few specialized IT experts with specific toolsets.

Taking the bigger picture into account, it can be observed that the past years have seen an emphasis on moving various activities into more Web-oriented architectures, for example in the Web services area [16]. Further pushing the envelope of how to apply RESTful principles to more advanced scenarios are ongoing research efforts towards support for RESTful transactions [10]. The approach presented in this paper is another contribution to this field, investigating how a REST design pattern (queries on URI-based resources, in this case feeds) can be used as a blueprint for a variety of query-oriented services on the Web.

Both approaches, the unified model based on feeds, and RESTful interactions with specific types of database, have trade-offs in terms of universal usability, and access to the features of the back-end system. Ideally, both approaches should be combined, so that a back-end on the one hand exposes a (possibly limited) feed-oriented service for ease of combination with other services, but on the other hand also exposes a more specialized RESTful service which allows clients to interact with the service in a more specific way, leveraging the specific features of the underlying platform.

References

1. Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and Issues in Data Stream Systems. In Lucian Popa, editor, *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 1–16, Madison, Wisconsin, June 2002. ACM Press.
2. Scott Boag, Donald D. Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. XQuery 1.0: An XML Query Language. World Wide Web Consortium, Recommendation REC-xquery-20070123, January 2007.
3. Colm Divilly and Nikunj R. Mehta. Hierarchy Extensions to Atom Feeds. Internet Draft draft-divilly-atompub-hierarchy-00, May 2009.
4. Roy Thomas Fielding and Richard N. Taylor. Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology*, 2(2):115–150, May 2002.
5. Joe Gregorio and Bill de Hóra. The Atom Publishing Protocol. Internet RFC 5023, October 2007.
6. Dominique Guinard and Vlad Trifa. Towards the Web of Things: Web Mashups for Embedded Devices. In *Proceedings of the Second Workshop*

on Mashups, Enterprise Mashups and Lightweight Composition on the Web, Madrid, Spain, April 2009.
7. Bin He, Mitesh Patel, Zhen Zhang, and Kevin Chen-Chuan Chang. Accessing the Deep Web. *Communications of the ACM*, 50(5):94–101, May 2007.
8. International Organization for Standardization. Information Technology — Database Languages — SQL — Part 1: Framework (SQL/Framework). ISO/IEC 9075-1:2008, January 2009.
9. Mengchi Liu and Tok Wang Ling. A Conceptual Model and Rule-Based Query Language for HTML. *World Wide Web*, 1(1-2):49–77, March 2001.
10. Alexandros Marinos, Amir Razavi, Sotiris Moschoyiannis, and Paul Krause. RETRO: A (hopefully) RESTful Transaction Model. Technical Report CS-09-01, University of Surrey, Guildford, Surrey, August 2009.
11. Alberto O. Mendelzon, George A. Mihaila, and Tova Milo. Querying the World Wide Web. *International Journal on Digital Libraries*, 1(1):54–67, 1997.
12. Mark Nottingham and Robert Sayre. The Atom Syndication Format. Internet RFC 4287, December 2005.
13. Open Geospatial Consortium. An Introduction to GeoRSS: A Standards Based Approach for Geo-enabling RSS feeds. OGC 06-050r3, July 2006.
14. Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report SIDL-WP-1999-0120, Stanford University, November 1999.
15. Cesare Pautasso and Erik Wilde. Why is the Web Loosely Coupled? A Multi-Faceted Metric for Service Design. In Quemada et al. [18], pages 911–920.
16. Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. RESTful Web Services vs. "Big" Web Services: Making the Right Architectural Decision. In Jinpeng Huai, Robin Chen, Hsiao-Wuen Hon, Yunhao Liu, Wei-Ying Ma, Andrew Tomkins, and Xiaodong Zhang, editors, *Proceedings of the 17th International World Wide Web Conference*, pages 805–814, Beijing, China, April 2008. ACM Press.
17. Eric Prud'Hommeaux and Andy Seaborne. SPARQL Query Language for RDF. World Wide Web Consortium, Recommendation REC-rdf-sparql-query-20080115, January 2008.
18. Juan Quemada, Gonzalo León, Yoëlle S. Maarek, and Wolfgang Nejdl, editors. *Proceedings of the 18th International World Wide Web Conference*, Madrid, Spain, April 2009. ACM Press.
19. Peter Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. Internet RFC 3920, October 2004.
20. Erik Wilde. Putting Things to REST. Technical Report 2007-015, School of Information, UC Berkeley, Berkeley, California, November 2007.
21. Erik Wilde. Feeds as Query Result Serializations. Technical Report 2009-030, School of Information, UC Berkeley, Berkeley, California, April 2009.
22. Erik Wilde, Eric Kansa, and Raymond Yee. Proposed Guideline Clarifications for American Recovery and Reinvestment Act of 2009. Technical Report 2009-029, School of Information, UC Berkeley, Berkeley, California, March 2009.
23. Erik Wilde and Igor Pesenson. Feed Feeds: Managing Feeds Using Feeds. Technical Report 2008-025, School of Information, UC Berkeley, Berkeley, California, May 2008.

This work was partly supported by the (INFSO-IST) OPAALS Project, funded by the European Commission (Project number:034824)