Linked Data and Service Orientation

Erik Wilde

School of Information
UC Berkeley

Abstract. *Linked Data* has become a popular term and method of how to expose structured data on the Web. There currently are two school of thought when it comes to defining what *Linked Data* actually is, with one school of thought defining it more narrowly as a set of principles describing of how to publish data based on *Semantic Web* technologies, whereas the other school more generally defines it as any form of properly linked data that follows the *Representational State Transfer (REST)* architectural style of the Web. In this paper, we describe and compare these two schools of thoughts with a particular emphasis on how well they support principles of service orientation.

1 Introduction

In the recent past, the term of *Linked Data* has become a popular meme for referring to approaches which are publishing structured data on the Web. However, the exact meaning of that term has been contentious, which is not an unusual thing to happen to terms which attract a certain attention and are not rigidly defined by any specific standard or product. This paper is an attempt to explore, explain, and clarify the major world views in this area, and more importantly, to investigate how well they work under the perspective of service orientation.

Unfortunately, the term *service orientation* itself is not all that well-defined, which means that it takes a little bit of explanation in itself. For the purpose of this paper, we refer to service orientation as an approach which allows the providers or information-intensive services to expose services which can be easily used, reused, and recombined using Web technologies. One of the main goals of service orientation should be to achieve *loose coupling* [24] between services, and between service providers and service consumers, so that the service landscape exposed by service providers, and the service landscape used by service consumers, are as agile and as easy to repurpose as possible.

For the purpose of this paper, the main property of service orientation is that it is on a higher level of abstraction and functionality than structured data. In order to implement service-oriented architectures, it is necessary to have some representation for the data that is exchanged between service providers and consumers, but the exact nature of that data, in particular the specific structured data standard used to represent that data, is secondary. Of course, in order to make services easily usable and easily mashable it is advantageous to use standardized and well-established structured data standards, but this is

© Springer-Verlag Berlin Heidelberg 2010

only one of the facets of service orientation. Another important facet is that the exact patterns in which data is exchanged is essential to service design as well, and there are several popular design patterns such as downloads, incremental data transfers, pull-based data feeds, and push-based architectures with various subscription and notification mechanisms using light or fat ping approaches.

The goal of this paper is to provide an overview and a qualitative comparison between the two dominant "world views" of linked data that are currently in use. One of the world views is a more constrained one which is based on Semantic Web technologies and architecture, and this approach is described in Section 2. The other world view is less constrained in terms of technologies and is based on Web technologies and architecture, this approach is described in Section 3. Recently, a third world view has been proposed, seeking the middle ground by not rigidly prescribing the RDF metamodel[1] of the Semantic Web world view, but still mandating that the metamodel used for structured data should be based on triples. This third world view so far has not gained a lot of momentum, though, probably caused by a lack of available candidates for a metamodel which is not RDF, but still based on triples.

The main reason for comparing the different approaches for linked data is that in the end, they are just implementation variants to achieve non-technical goals, which in many cases revolve around ideas of accessibility and usability (of data, and not of user interfaces), openness (non-proprietary ways of accessing and representing data), extensibility (the ability of the environment to adapt to unforeseen needs and use cases), and transparency (giving users the ability to understand where data originates, and easier ways to interact with a larger set of back-end data and service providers). Looking at service orientation is one way of comparing different approaches, so that from the business perspective it becomes easier to decide which technical approach best fits the requirements of the business goals.

2 Narrow Linked Data World View

The *Narrow Linked Data World View* is based on the approach of the *Semantic Web* [6], most importantly mandating the used of Semantic Web standards for structured data models and access. The most important standards in this space are the *Resource Description Framework (RDF)* [19] as the metamodel for any data being published on the Semantic Web, and *SPARQL* [27] as the query language for extracting subgraphs from large RDF graphs. While the original Semantic Web approach focused mainly on the core technologies, the term "linked data" emerged in conjunction with usage patterns around those technologies,

[1] In this paper, the term *metamodel* refers to the model of a model language, i.e. it is the foundation that is provided by a modeling language for building application-specific models. For example, the RDF metamodel is defined by *RDF Concepts and Abstract Syntax* [19], whereas the (most popular) XML metamodel is defined by the *XQuery 1.0 and XPath 2.0 Data Model (XDM)* [4].

and one of the important sources being cited frequently in this context is the *Linked Data* design note,[2] which defines the following rules:

1. Use URIs as names for things.
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL).
4. Include links to other URIs, so that they can discover more things.

This set of rules can be regarded as repeating the pattern that has made the HTML Web successful: Use URIs for Web pages, use HTTP URIs so that people can point their browser at Web pages, make information available in HTML so that a standard browser can render the Web page, and include links to more pages so that people can click on links and follow them to retrieve even more Web pages. Because this approach focuses on the utility of a single metamodel that is consistently used by all information providers, and the utility of a standardized query language for providers of large datasets, for the remainder of this paper we refer to this approach as *Homogeneous Linked Data (HoLD)*.

The most important implications of the HoLD approach are that it fixes two design options that are left open by the Web itself: It mandates the use of HTTP URIs so that the identities of anything identified can also be used as a way to access it, and it mandates the use of RDF as the metamodel so any structured information made available at those HTTP URIs can be expected to be in a format that is prescribed in advance, instead of being discovered at runtime.

One of the important value propositions of HoLD is that data harvesting and/or aggregation becomes relatively easy because the two most important tasks (how to get access to data about an identified entity, and what to expect when accessing that data) are backed by the constraints defined by this approach. Since RDF is a metamodel with a small unit of atomicity (everything is based on triples, and there is no bigger unit of granularity, such as a concept of a "document," in RDF's metamodel), there is little built-in "bias" in RDF when it comes to mapping existing non-RDF data models to RDF. As discussed in more detail in Section 4, this can be a boon or a bane, depending on the data models in question and the requirements of data publishers and users, but regardless of these individual "data model usability" issues, the overall integrative qualities of the HoLD approach are largely undisputed.

3 Wide Linked Data World View

The *Wide Linked Data World View* is based on general principles of the *Architecture of the World Wilde Web* [18], and more specifically, on the *Representational State Transfer (REST)* [13] architectural style underlying the Web. Whereas HoLD is promoting a specific set of technology choices, this second view is more agnostic in terms of technologies and is operating on the layer of

[2] `http://www.w3.org/DesignIssues/LinkedData.html`

architectural styles, which are design patterns for architectures. In this wider view, the main constraints are derived from the REST architectural style, which underlies the Web and more specifically, the core Web technologies. The REST constraints can be summarized as follows:

1. *Resource Identification:* Anything that should be available for interactions should have an identifier.
2. *Uniform Interface:* Interactions with identified things should be based on a uniform interface, so that anything that is identified is readily available for interactions. Different identification schemes can have different interfaces, and not all identified things have to respond to all interactions defined by their uniform interface.
3. *Self-Describing Messages:* Interactions should be based on exchanges of messages (or documents) which are labeled with their type.[3]
4. *Hypermedia Driving Application State:* Messages should be based on data formats that may contain links (in most cases these are typed links), and interactions with RESTful services essentially means following those links according to the goals of the service consumer, and the semantics of the link types.
5. *Stateless Interactions:* The uniform interface should be usable in a way that interactions are independent of each other and only rely on either client state or resource state. This decouples interactions and allows clients and servers to be more independent of each other (servers do not need to remember clients, and clients can take advantage of scalability optimizations such as caching and replication).

Whereas HoLD to a large part reflects the specific technologies of the human-oriented Web, the wide view reflects the architectural style of the Web and its openness to new technologies, where identification is done by *Uniform Resource Identifier (URI)* [5] but allows a multitude of identification schemes to co-exist and new ones to be invented, where most of the interactions are based on the uniform interface of *Hypertext Transfer Protocol (HTTP)* [12], but other protocols can be used as well, and where structured data standards such as HTML [28], XML [7] or JSON [11] can be used, but new ones can be invented, as long as they are registered and subsequently identified according to the constraint of self-describing messages. Since this wider view thus allows a more open and evolving universe of concrete technologies, it will be referred to as *Heterogeneous Linked Data (HeLD)* for the rest of this paper.

In the same sense as HoLD is established as a set of constraints and patterns for publishing linked data using the Semantic Web's set of technologies, it is

[3] For clarification: REST uses the term "self-describing" in a considerably weaker form than this term is being used in more semantics-oriented research communities. "Self-describing" simply means that the type of the message can be inferred by looking at the message; it does not mean that any higher-level semantics are necessarily made available through advanced semantic descriptions.

possible to come up with some patterns and established best practices for publishing linked data according to HeLD approach. Since this approach is based in Web architecture itself, it can been dubbed the *Plain Web* [31], which is a general attempt to use established Web technologies instead of building entirely new stacks of technology on top of the Web, essentially treating the Web as a transport infrastructure instead of the information system it is.

Plain Web principles and HeLD results in approaches which can also be called *Lightweight Linked Data* [32], using established core Web technologies such as *Atom* [22] and the *Atom Publishing Protocol (AtomPub)* [15]. One of the main motivations of this approach is that HeLD data can be processed using basic Web technology toolsets (instead of the new toolset required for handling HoLD).

While the basic approach of using Atom and XML for exposing services and representing data might seem limiting, it is important to notice that because of the reliance of established and widely-used Web technologies, HoLD can benefit from developments in that rapidly developing space. One example is *PubSubHubbub (PuSH)*, a protocol that is based on Atom's model of exposing services via feeds, but that extends Atom's pull mechanics with a *Publish/Subscribe (Pub/Sub)* mechanism that can be used to implement push mechanics (this is described in more detail in Section 5).

4 Data Model Issues

One of the most striking differences between the two approaches is the fact that HoLD has a fixed metamodel, whereas in HeLD there is no fixed metamodel, and as long as the representation is properly labeled with its type (*self-describing*) and contains links (*hypermedia*), it can be used in a HeLD scenario. From the service point of view, this can be both good and bad. It can be good because it allows services to expose linked data in metamodels that fit their needs, and may be a good fit for the data model they are using (Section 4.1 contains a more in-depth discussion). It can be bad, because it means that service consumers may have to deal with a variety of metamodels, and this requires a variety of toolsets and may make it hard to combine data from different sources.

One of the interesting observations in this area is that the degree of freedom allowed by HeLD allows an "evolution" of metamodels used by services. As a popular example, for a while many Web-based services published structured data in XML, which was the first established Web-wide metamodel for structured data, and was reasonably easy to process in many client-side environments. However, because the majority of service consumers were using JavaScript (and processing XML in JavaScript involved some extra steps and created somehow awkward data structures), many services started providing the data in JSON, an alternative representation. JSON is a bit more limited than XML in its metamodel, but most XML models can be mapped to JSON in a fairly straightforward way. Because many developers preferred JSON's more straightforward mapping of data structures into programming language abstractions, JSON has now replaced XML in popularity for many Web-based services. This evolution has not changed anything about the data being exchanged and the way how it is linked and used, but it has allowed developers to move to a metamodel that better fits their needs.

4.1 Metamodel Bias

The example of XML and JSON highlights the fact that in many cases (at least in reasonably simple cases), data models can be mapped between metamodels. However, there are also many examples where data models, and especially more sophisticated data models, gain a lot of their expressiveness and convenience from the underlying metamodel. A good example for this are markup languages such as SGML or XML, which are metamodels specifically designed for representing human-readable documents. Many document types can be expressed rather conveniently in these metamodels, because of the specific design bias of the metamodel.

Metamodel bias is important in many scenarios where the data is non-trivial in its internal structure, and where there's a certain "natural order" to data, such as in documents which often are quite naturally a sequence of a variety of content objects. Metamodel bias has two main impacts: If there is a good match between a metamodel and a data model, then defining the model and representing and managing data with it are made easier by the metamodel's properties, and maybe the technologies and toolsets that have evolved around that metamodel. If there is a bad match between a metamodel and a data model, then defining the model becomes awkward, and representing and managing data with it become tasks where technologies and tools often feel like they work against the developer.

Metamodels can be roughly classified into three classes (without making any attempt to create a precise classification scheme), which at least for the purpose of a rough classification are sufficient:

- *Hierarchical:* In this case, models are always a *Directed Acyclic Graph (DAG)*, and sometimes they may be trees (having only one root node), sometimes they are allowed to have multiple root nodes. Another possible distinction is whether models can use a built-in ordering, or wether there is no ordering of nodes in the graph. Examples for hierarchical models are IBM's *Information Management System (IMS)* for a rather old model, and more recently the *Extensible Markup Language (XML).*
- *Linked Tables:* Instead of having the inherently hierarchical structure of trees or other kinds of DAGs, the model of linked tables (as formalized by relational algebra) is less concerned with one predefined structure for a model instance, and instead is based on relations and operators on those relations. The most prominent example of this metamodel is the relational model introduced by Codd [10], which nowadays has its most popular representation in the *Structured Query Language (SQL).*
- *Generalized Graphs:* The two previously discussed metamodels have a structural bias, for the hierarchical model this is some variation of DAG, and for the linked table model the bias are relations (i.e., connected n-tuples).

The third approach to metamodel structure is to try to avoid as much bias as possible, and just use generalized graphs. RDF can be seen as such a metamodel, where each triple represents an edge (the predicate) connecting two nodes (the subject and the object).

This categorization of metamodel structures is rough and only an approximation, but it does illustrate that various modeling languages may be based on different classes of metamodel structures, and based on this bias (and on the underlying use cases and derived technologies and tools), they provide differently specialized environments for certain classes of applications. The most important thing to realize is that this specialization is both a constraint in terms of these environment working better for some classes of problems than for others, and an opportunity for optimization, because it is possible to develop more effective and more efficient solutions for those problems that do fit the environments' bias.

As an example, while an ordered tree model (a DAG-based model with additional constraints) such as XML can feel like a good fit and almost natural for scenarios involving structured text, its built-in bias can become inconvenient and hard to adjust to when the scenarios become more advanced and include concepts such as overlapping or concurrent markup, which cannot be easily represented in XML-based data models. However, for the majority of document processing environments, tree-based models have proven to be the most convenient foundations, and metamodels such as SGML and XML and associated processing technologies and tools have produced an environment with a good mix of built-in bias, and freedom for customization and specialization.

In terms of comparing the two linked data approaches, HeLD allows data to use any media type as long as it is properly declared, and thus services are free to use models that fit their needs. HoLD, on the other hand, prescribes RDF's generalized graph as the only acceptable metamodel and thus does not introduce any particular structural bias (DAGs and relations have considerably more of a structural bias to them), but on the other hand also does not allow services to use the more biased metamodels if they would fit their needs. Many popular services prefer exposing XML or JSON over exposing RDF, because the installed base of both tools and developers is much bigger.

As a side note, it is interesting to see that recently, the W3C has started work on a standardized mapping of relational models to RDF models [26], recognizing the fact that a lot of data today is managed in relational systems. However, since this mapping is intended to be generic (i.e., it maps any relational model to an RDF model), it is likely that working with the RDF "view" of the relational data will be rather awkward, and figuring out the SPARQL queries to retrieve model-level data, and making sure that they can run with similar efficiency than in a fine-tuned RDBMS will likely be a challenge.

4.2 Data Granularity

One of the issues not discussed in the previous discussion of metamodel bias is that of *data granularity*. From the service perspective, in many cases data has a certain natural granularity in the sense that some data only makes sense as a unit, whereas other data is more loosely linked and can exist even if some of the linked resources cease to exist. This idea of data granularity is addressed in *document engineering* [14], and many frameworks have constructs to deal with it.

One example is the *Unified Modeling Language (UML)* [17], which supports various level of how tightly coupled data is. *Association*, *aggregation*, and *composition* are the different levels which are supported, and while the exact difference between these constructs is out of the scope of this paper, it is important to realize that these concepts were deemed important enough to be hardcoded into the metamodel.

The *Extensible Markup Language (XML)* and the *JavaScript Object Notation (JSON)* are two other examples where there is a level of granularity between a model's "atoms" (elements/attributes in XML and fields in JSON) and the global view of all data available. Data granularity is important for issues such as *provenance*, *versioning*, and *security*, where the "natural unit" of the data model (for example an XML document or a JSON object) makes it easier to express document-level semantics such as the origin, a version number, or a digital signature.

RDF does not have the concept of documents in the metamodel, there is no granularity concept beyond the triple. This has been noticed by the HoLD community as a problem, because *reification* as one way to solve this problem has the unfortunate side-effect of increasing data size several-fold, and *named graphs* [8] as the other way to solve this problem are not a part of RDF itself, but have been introduced by SPARQL. This means that as long as all data in a HoLD scenario is treated as merging seamlessly into one graph (an approach which for a while was dubbed the *giant global graph*), RDF works well and provides a good fit for processing this accumulation of all data into one big graph. However, when document boundaries become important because of the provenance, versioning, and security issues mentioned above, there is no support in the metamodel for modeling this, and applications have to come up with their own ways of introducing such an intermediate level of granularity between the "atom" and the "universe" as a whole.

4.3 Data Processing

Maybe the biggest difference between the two linked data approaches are in how data processing is supposed to work. Data processing can be considered on at least two different levels: One is the level of metamodels, where the question is which metamodel processed data is based on, and thus which technologies and tools are required to be able to process this data. The other level is that of understanding the meaning of the data, and being able to relate it to other models or other data. Unsurprisingly, the HoLD approach makes processing and understanding/combining data simpler, because it is based on a single metamodel, whereas the HeLD approach allows data to be more diverse, which has both positive and negative implications.

Processing Mechanics. Because HoLD is based on a single metamodel, processing can be reliably based on technologies and tools supporting this metamodel, and then by definition all structured data can be reliably processed.[4] HeLD mandates that structured data must be labeled with its format (*self-describing*), but if a client encounters data using a metamodel that it does not support, then it cannot process this data. This allows for new metamodels to be introduced and gain in popularity (such as the XML/JSON example mentioned earlier), but it does introduce an element of uncertainty in terms of what clients can encounter when following links.

Semantics and Mashups. The main difference between the basic *Semantic Web* and *Linked Data* (the HoLD variety) is that the latter establishes a set of patterns and best practices that are intended to actually link data, either because entities have well-known URIs, or because data and data models are created using existing data models, allowing data and data models from various sources to be joined. Again, the mandated use of a single metamodel not only make processing of individual resource representations easier as discussed above, they also provide a unified framework within which clients can infer the "meaning" of data (because concepts in RDF are identified by URI), and a simple way to "mash up" all data, because all data uses the same metamodel. This ability to combine data from any source is the biggest strength of HoLD, and is much harder to accomplish in HeLD. HeLD allows different metamodels to co-exist, and has no unified way of representing structured data, or identifying concepts. HeLD clients thus have to deal with heterogeneity on two different levels: models may be hard to relate because there is no standardized way of identifying concepts, and data can hard to combine because it may be based on different metamodels.

5 Service Orientation

When looking at data processing as discussed in the previous section, it becomes apparent that HoLD's approach allows users to deal with a more unified environment, whereas HeLD may require users to deal with various metamodels, and even for environments where all data is using the same metamodel (for example, XML), each model has its "private" semantics and there is no established way in which different models can be related or mapped [30]. This makes it easier to use data in HoLD, but HeLD does have the advantage of allowing services to use the metamodel that has the bias that makes most sense for their scenario. However, this is still looking at the data model level alone, and the next interesting area to look at is *service orientation*. How well can services be represented and exposed in these two approaches, and how do they compare?

[4] This is not entirely true because RDF supports various representations and the only standardized one, RDF/XML [3], has been on a steady decline in popularity, whereas alternative but not yet standardized syntaxes have become more popular.

On a very abstract level, a service can be defined as a well-defined unit of functionality that is accessible through some well-defined interface.[5] While this definition is probably general enough to not conflict with any other definition of services, it is also too wide to serve as a starting point for deciding on how good or bad specific services are exposed, or even more importantly, for deciding how well a certain architecture is suited for exposing services, and for allowing service innovation.

HeLD allows for service innovation in a variety of places, and most importantly, in the architecture itself. It is possible to introduce new identification methods, new interfaces, and new metamodels, and while this flexibility makes it necessary for clients to cope with the potential of new things being invented, it allows the service ecosystem to evolve over time and to adjust to the varied and unforeseeable needs and evolution of service providers and consumers. One good example on the human Web is Flash: Regardless of its merits as a well-designed or not-so-well-designed container for multimedia content, the open architecture of the Web allowed this new media type to flourish and succeed. All it needed was the `application/x-shockwave-flash` media type to be supported by a substantial share of clients, and to be provided by an increasing share of services. With the recent advances in HTML5 and the problematic support for Flash on mobile devices, combined with the rise of the mobile Web, it may happen that multimedia content will increasingly use HTML again, and move away from Flash. For many content producers, this is not even a major issue, because they produce their content with tools which increasingly will be able to export Flash or HTML5 representations of multimedia content. The important observation is that in this scenario, the service is to provide a multimedia representation of some content at some URI, and whether this is done in Flash or HTML5 is an implementation question that has no "correct" answer, but only a "best" answer given the constraints of content production tools and support for specific media types on the client side.

From this point of view, the "service" in HeLD is on a more abstract level than it is in HoLD, and spelling out the specific "service" definition implicitly asserted by these two approaches is crucial for understanding the differences between them:

– *Service as defined by HoLD:* Homogeneity is a top concern in HoLD, because it allows the seamless joining of both data and models across services. For this reason, a service in HoLD in required to only use RDF's metamodel. HoLD also encourages service providers to reuse existing identifiers, so that data retrieved from a variety of sources is more likely to be joinable on the data and/or the model level. From the interaction perspective, the services currently supported by HoLD are simple resource retrieval (HTTP `GET` of a URI), or SPARQL queries at an endpoint that exposes the SPARQL protocol via HTTP [9].

[5] For comparison, this is the definition of a service as given by the OASIS SOA reference model [20]: "A service is a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description."

- *Service as defined by HeLD:* Heterogeneity is the biggest differentiating factor of HeLD, and it starts with identification, which can use a variety of schemes. Most schemes have well-defined interactions, and the uniform interface constraint of REST should allow clients to interact with a service if it supports the scheme linking to that service. REST's self-description constraint allows the service to consume and/or provide whatever metamodel it wants to, as long as the media type is exposed in the interactions.[6] The semantics of interactions often may be described or at least constrained by the interaction protocol associated with the URI scheme (for example, HTTP's methods describe basic safe/unsafe and idempotent/non-idempotent semantics for the HTTP methods), but this is not required by HeLD itself.

When contrasting these two approaches on how to define services, the differences between the approaches become strikingly apparent: HoLD's homogeneous approach defines an architecture where everything is predictable: data is always based on the same metamodel, and services always expose the same functionality. Because of this homogeneity, HoLD makes it possible basically ignore the underlying machinery of HTTP for resource retrieval and SPARQL for remote query execution: the virtual world view provided by HoLD is that of a seamlessly interconnected graph of data across all HoLD providers.

HeLD's world view is less homogeneous and thus supports a less virtualized world view. In HeLD, clients have to be prepared for heterogeneity on a variety of levels, which means that by definition they can never have a complete and definitive view across all HeLD providers, because that would require global knowledge across an unrealistic set of variables:

- *Identification:* Identification can use a variety of schemes and new schemes can be introduced at runtime.
- *Interaction:* Schemes in most cases imply interaction through a uniform interface, and this uniform interface must be implemented when interaction with a resource using that scheme is required.
- *Representation:* There is no fixed metamodel and services are free to consume and provide their preferred metamodel. Discovery of metamodels is done through registration and runtime labeling of representations with the metamodel they are based on, but new metamodels can be introduced at runtime.
- *Interpretation:* Even if a metamodel is supported, most metamodels do not have an overarching concept of how models encode and reuse semantics. Thus, often it is necessary to specifically support the interpretation of models, which thus must be detectable in representations.[7]

[6] HTTP's content negotiation adds a dynamic and negotiable pattern to this basic setup, but we will not discuss the specifics of HTTP here.

[7] In XML, for example, this is traditionally done with DTD declarations, XML namespaces, or XSD-specific attributes, but a recent W3C specification [16] proposes to use a unified syntax (based on processing instructions) for all these association mechanisms.

Thus, when comparing HoLD and HeLD, the trade-offs between the approaches become probably best visible on the service level: HoLD standardizes all services into RDF producers (static RDF or SPARQL endpoints), whereas HeLD provides an environment which is open and extensible on a variety of levels. Since HoLD standardizes a lot of this (thus making things more interoperable), extensions to this picture have to published as additional standards. One example is *SPARQL Update* [29], which extends the currently read-only nature of SPARQL to support create, update, and delete operations as well.

Since HeLD is more open, evolution and development can happen in a more informal way. One good example for this is in the area of feeds. Feeds have become the de-facto standard for lightweight data distribution on the Web, either using one of the various RSS formats, or the more recent Atom [22]. Feeds are a good example for HeLD because they provide an evidently working framework for implementing large scale data distribution and aggregation, but they also allow publishers to decide on the actual contents of the feed. Podcasts, for example, are just feeds which happen to carry audio or video content instead of more static media types. The big disadvantage of feeds has always been their *pull* model, which has been a great advantage for achieving loose coupling and scalability, but also produces a lot of polling in time-critical scenarios. Recently, *PubSubHubbub (PuSH)* has achieved some success by layering a push-oriented overlay on top of the pull model of feeds. PuSH allows clients to register callbacks with "hubs" (layered designs of multiple hubs are supported), and whenever a feed is updated, the clients will be notified via their callbacks. The basic information flow remains unaltered (services produce entries which are exposed via feeds), but the reversed control flow allows to eliminate polling. From the HeLD perspective, this was a straightforward innovation, with the only difference being that the media types involved in the scenario now are reversed in the HTTP interactions (the service provider or the hub acting on behalf of it acts as an HTTP client pushing the entry, and the service consumer accepts the entry by running a server at the callback URI).

Interestingly, recent work on combining SPARQL and PuSH in *sparqlPuSH* [23] has replicated this behavior in HoLD, adding a new service to the HoLD picture. How this new capability (which introduces something that could be described as "SPARQL triggers" and uses PuSH feeds carrying RDF as the notification mechanism) fits into the existing picture of HoLD services remains to be seen, but it fits well into the general direction of the HoLD and Semantic Web research community, where there is an increasingly strong push to move past the currently established model of "all RDF data accessible via one SPARQL endpoint," and is moving towards a more distributed scenario, including issues such as provenance and versioning.

The first draft of SPARQL 1.1 Federation Extensions [25] is looking at the fact that the current view of a service in HoLD is limited, because it is either retrieval of a fixed resource from a URI, or submitting a SPARQL query to an endpoint which then returns a subset of the RDF data managed behind that endpoint. SPARQL federation is supposed to work across a variety of RDF-oriented data sources, and introduces the `SERVICE` keyword. Because of the more constrained view of what a service is, this keyword allows a query to contain a query to another SPARQL endpoint, and the results of this query will then become available in the context of the "outer" query. While SPARQL federation will allow interesting new patterns of how to combine multiple RDF stores, it does not move outside of the basic assumption that services always consume and produce RDF, and that the only interactions possible with a service are retrieval and SPARQL-based querying.

6 Which One Is Better?

To a certain extent, research communities both from the HoLD and the HeLD side picture these two approaches as competing. This does not necessarily have to be the case. It also is misleading to picture the HoLD approach (or the Semantic Web in general) as the "next step in Web evolution." It is much more helpful to think of both approaches as being complementary, and of having different strengths and weaknesses.

HoLD shines when it comes to providing an abstraction layer that essentially makes the Web go away, and allows information to be viewed as the proverbial giant global graph. This capability can be very valuable when it comes to making sense of a large dataset, but it also comes at the price of having to do the homogenization of all data and services. Often, many of the most expensive tasks for producing good linked data in HoLD are non-technical, such as when data is aggregated from a wide variety of sources and entity resolution becomes a cumbersome process made expensive by data quality and a lack of transparency [33].

HeLD allows a more heterogeneous perspective on linked data and thus allows a greater variety of data sources and services to be used and possibly combined [2]. One possible use for this available data and the combination of available data sources is to map it to a HoLD view of these sources [1], available in some RDF mapping of the underlying sources and a SPARQL endpoint for using this mapped data.

It is probably unrealistic to assume that all data and service providers will subscribe to the HoLD set of technologies. Thus, it is likely that both approaches will co-exist for the foreseeable future, and both will have application areas where they are good fits, or not so good fits. In the HoLD world, the most exiting development for the near future is probably the inclusion of write features into the general architecture, and a more decentralized view of how HoLD data can be used. In the HeLD world, it is necessary to continuously improve the ways in which fundamental pieces of the infrastructure can use agreed-upon semantics, such as will be made possible for link relations [21] with a new registry of link types on the Web.

One way of benefiting from the differences in approach and strengths in both HoLD and HeLD can be to encourage the use of the less harmonized but still useful and accessible HeLD style in scenarios where the added expense of harmonizing models and metamodels is not justified, and to layer the more harmonized HoLD style on top of those HeLD services, if a more harmonized view is required, and the expenses for it are justified. Converting a set of HeLD services into a HoLD service (or, more accurately speaking, providing a HoLD perspective of a set of HeLD services) can be a value-added service in itself, but it also can be a costly service to implement. In many cases, most of the costs of this service will be caused by the expensive and often manual or semi-automatic tasks of data cleansing and entity resolution.

7 Conclusions

The goal of this paper is to compare the two approaches to *Linked Data* that are currently under discussion in the research and developer communities. For the purpose of this paper, we refer to the approach based on Semantic Web technologies as *Homogeneous Linked Data (HoLD)*, and to the approach based on Web architecture and REST as *Heterogeneous Linked Data (HeLD)*. The main goal of the comparison is to understand how well these approaches work in the context of service-orientation, and how open they are to service innovation on a variety of levels. The goal of this paper is a qualitative comparison, pointing out the strengths and weaknesses of both approaches. The service level turns out to be a very good comparison between those two approaches, because HoLD's more homogenous approach allows clients to work in a very predictable landscape of data and services, whereas HeLD's heterogeneous approach requires clients to deal with heterogeneity on at least four different levels (identification, interaction, representation, and interpretation).

In summary, HoLD can be described as defining more constraints, thus providing a more predictable environment, but also providing less potential for innovation, whereas HeLD with its more open approach has less constraints, thus provides a less predictable environment, but on the other hand has more potential for innovation. We don't think that these two approaches have to be mutually exclusive. In tightly coupled and cooperating environments, the HoLD approach has obvious benefits by providing a more integrated view of the available data and services, and allowing developers to better abstract from the underlying fabric of the Web. In loosely coupled and decentralized environments, the HeLD approach provides a more flexible and open solution that still establishes patterns and practices for data and services to be linked, but allows a more open ecosystem that can change over time, introducing new tools and technologies as service providers and consumers as well as their needs evolve.

References

1. Alarcòn, R., Wilde, E.: From RESTful Services to RDF: Connecting the Web and the Semantic Web. Tech. Rep. 2010-041, School of Information, UC Berkeley, Berkeley, California (June 2010)

2. Alarcón, R., Wilde, E.: Linking Data from RESTful Services. In: Third Workshop on Linked Data on the Web, Raleigh, North Carolina (April 2010)
3. Beckett, D.: RDF/XML Syntax Specification (Revised). World Wide Web Consortium, Recommendation REC-rdf-syntax-grammar-20040210 (February 2004)
4. Berglund, A., Fernández, M.F., Malhotra, A., Marsh, J., Nagy, M., Walsh, N.: XQuery 1.0 and XPath 2.0 Data Model (X DM) (2 eds.) World Wide Web Consortium, Proposed Edited Recommendation PER-xpath-datamodel-20090421 April 2009)
5. Berners-Lee, T., Fielding, R.T., Masinter, L.: Uniform Resource Identifier (URI): Generic Syntax. Internet RFC 3986 (January 2005)
6. Berners-Lee, T., Hendler, J.A., Lassila, O.: The SemanticWeb. Scientific American 284(5), 34–43 (2001)
7. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F.: Extensible Markup Language (XML) 1.0 (Fifth Edition). World Wide Web Consortium, Recommendation REC-xml- 20081126 (November 2008)
8. Carroll, J.J., Bizer, C., Hayes, P., Stickler, P.: Named Graphs, Provenance and Trust. In: Ellis, A., Hagino, T. (eds.) 14th International World Wide Web Conference, pp. 613–622. ACM Press, Chiba (May2005)
9. Clark, K.G., Feigenbaum, L., Torres, E.: SPARQL Protocol for RDF. World Wide Web Consortium, Recommendation REC-rdf-sparql-protocol-20080115 (January 2008)
10. Codd, E.F.: A Relational Model of Data for Large Shared Data Banks. Communications of the ACM 13(6), 377–387 (1970)
11. Crockford, D.: The application/json Media Type for JavaScript Object Notation (JSON). Internet RFC 4627 (July 2006)
12. Fielding, R.T., Gettys, J., Mogul, J.C., Frystyk Nielsen, H., Masinter, L., Leach, P.J., Berners- Lee, T.: Hypertext Transfer Protocol | HTTP/1.1. Internet RFC 2616 (June 1999)
13. Fielding, R.T., Taylor, R.N.: Principled Design of the Modern Web Architecture. ACM Transactions on Internet Technology 2(2), 115–150 (2002)
14. Glushko, R.J., McGrath, T.: Document Engineering, The MIT Press, Cambridge (August 2005)
15. Gregorio, J., de Hóra, B.: The Atom Publishing Protocol. Internet RFC 5023 (October 2007)
16. Grosso, P., Kosek, J.: Associating Schemas with XML documents 1.0 (1eds.) World Wide Web Consortium, Note NOTE-xml-model-20100415 (April 2010)
17. International Organization for Standardization: Information Technology– Open Distributed Processing – Unified Modeling Language (UML) Version 1.4.2. ISO/IEC 19501 (April 2005)
18. Jacobs, I., Walsh, N.: Architecture of the World Wide Web, Volume One. World Wide Web Consortium, Recommendation REC-webarch-20041215 (December 2004)
19. Klyne, G., Carroll, J.J.: Resource Description Framework (RDF): Concepts and Abstract Syntax. World Wide Web Consortium, Recommendation REC-rdf-concepts-20040210 (February 2004)
20. MacKenzie, C.M., Laskey, K., McCabe, F., Brown, P.F., Metz, R.: Reference Model for Service Oriented Architecture 1.0. Organization for the Advancement of Structured Information Standards, OASIS Standard (October 2006)
21. Nottingham, M.: Web Linking. Internet Draft draft-nottingham-http-link-header-10 (May 2010)

22. Nottingham, M., Sayre, R.: The Atom Syndication Format. Internet RFC 4287 (December 2005)
23. Passant, A., Mendes, P.N.: sparqlPuSH: Proactive notification of data updates in RDF stores using PubSubHubbub. In: 6th Workshop on Scripting and Development for the Semantic Web, Crete, Greece (May 2010)
24. Pautasso, C., Wilde, E.: Why is the Web Loosely Coupled? A Multi-Faceted Metric for Service Design. In: Quemada, J., León, G., Maarek, Y.S., Nejdl, W. (eds.) 18th International World Wide Web Conference, pp. 911–920. ACM Press, Madrid (April 2009)
25. Prud'hommeaux, E.: SPARQL 1.1 Federation Extensions. World Wide Web Consortium, Working Draft WD-sparql11-federated-query-20100601 (June 2010)
26. Prud'hommeaux, E., Hausenblas, M.: Use Cases and Requirements for Mapping Relational Databases to RDF. World Wide Web Consortium, Working Draft WD-rdb2rdf-ucr-20100608 (June 2010)
27. Prud'Hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. World Wide Web Consortium, Recommendation REC-rdf-sparql-query-20080115 (January 2008)
28. Raggett, D., Le Hors, A., Jacobs, I.: HTML 4.01 Speci_cation. World Wide Web Consortium, Recommendation REC-html401-19991224 (December 1999)
29. Schenk, S., Gearon, P., Passant, A.: SPARQL 1.1 Update. World Wide Web Consortium, Working Draft WD-sparql11-update-20100601 (June 2010)
30. Wilde, E.: Model Mapping in XML-Oriented Environments. Tech. Rep. TIK Report 257, Co mputer Engineering and Networks Laboratory, ETH Zürich, Zürich, Switzerland (July 2006)
31. Schenk, S., Gearon, P., Passant, A.: SPARQL 1.1 Update. World Wide Web Consortium, Working Draft WD-sparql11-update-20100601 (June 2010)
32. Wilde, E., Liu, Y.: Lightweight Linked Data. In: 2008 IEEE International Conference on Information Reuse and Integration, Las Vegas, Nevada (July 2008)
33. Yee, R., Kansa, E.C., Wilde, E.: Improving Federal Spending Transparency: Lessons Drawn from Recovery.gov. Tech. Rep. 2010-040, School of Information, UC Berkeley, Berkeley, California (May 2010)