The Linked Media Framework

Integrating and Interlinking Enterprise Media Content and Data

Sebastian Schaffert
Knowledge and Media Technologies
Salzburg Research
Salzburg, Austria
sschaffe@salzburgresearch.at

Thomas Kurz
Knowledge and Media Technologies
Salzburg Research
Salzburg, Austria
tkurz@salzburgresearch.at

Dietmar Glachs
Knowledge and Media Technologies
Salzburg Research
Salzburg, Austria
dglachs@salzburgresearch.at

Christoph Bauer
Documentation and Archive
Österreichischer Rundfunk
Vienna, Austria
christoph.bauer@orf.at

Fabian Dorschel
Red Bull Media House
Salzburg, Austria
fabian.dorschel
@at.redbullmediahouse.com

Manuel Fernandez
Red Bull Media House
Salzburg, Austria
manuel.fernandez
@at.redbullmediahouse.com

ABSTRACT

This article presents the Linked Media Framework (LMF), a platform for integrating and interlinking structured data and media content in enterprises and on the Web. The Linked Media Framework is based on the Linked Data principles, but extends these on two important aspects: resource-centric updating and uniform management of resource content and metadata. Both aspects are important for enterprise information integration but not implemented by current Linked Data servers. In addition, the LMF offers the query language LD Path, a path-based language that allows intuitive resource-centric querying and traversal over distributed Linked Data resources and is thus more suitable for querying Linked Data than SPARQL. Finally, we describe two real-world scenarios where the LMF is already used or will be used for interlinking and semantic search: interlinking of multimedia fragments at the Red Bull Content Pool, and interlinking of news archive material at the Austrian Television.

1. INTRODUCTION

Linked Data offers a big potential for enterprises in the media and knowledge management area. On the one hand, more and more datasets are published following Linked Data principles and thus give the opportunity to link enterprise content with background information and also allow disambiguation of concepts. No single enterprise would be capable of managing a standardised vocabulary the size of DBPedia or a location database the size of GeoNames. On the other hand, Linked Data can also offer a comparably simple solution for enterprise information integration *inside* companies.

However, enterprises are still hesitating to use Linked Data in their value chains. From our experience with working with industry partners, one of the main barriers in the adoption of Linked Data is that the technology is still not easy enough to use and does not integrate well with existing information systems. Particularly, accessing data from the Linked Data Cloud is still cumbersome, and Linked Data has so far mostly been seen as a read-only and metadata-only source, while enterprise data usually is highly dynamic and involves both content and metadata. We are therefore focussing in this article on the following aspects:

1. how to extend the Linked Data principles with RESTful principles for addition, modification, and deletion of resources

2. how to extend the Linked Data principles by means to manage content and meta-data alike using MIME to URL mapping

3. how to offer straightforward, path-based querying over the Linked Data Cloud to make Linked Data resources more easily accessible

These issues are motivated by real-world scenarios that we are investigating at Salzburg NewMediaLab in the areas of *enterprise integration* and *media asset management* (see also Section 5). In both scenarios, resources typically describe media or document content (so-called *information resources*). So while most existing Linked Data services are only concerned about *structured data*, our scenarios require treatment of both content and metadata (e.g. the media assets stored in an asset management system and the metadata about these assets). And whereas in most existing cases publishing the datasets as Linked Data is a *secondary* service where the main data source is still a proprietary or non-standardised format (e.g. Wikipedia in the case of DBPedia), in our scenarios Linked Data is the *primary* means of publishing metadata. This immediately gives rise to the question how to update and interact with the Linked Data service in a resource-centred way beyond consuming. Besides the use cases we are working on, both extensions also give rise to new kinds of interactive Linked Data mashups.

I-SEMANTICS 2012 , 8th Int. Conf. on Semantic Systems, Sept. 5-7, 2012, Graz, Austria
Copyright 2012 ACM 978-1-4503-1112-0 ...$10.00.

The main contributions of this article are first a concrete and detailed proposal for the extension of the Linked Data principles for updates and for content access (Section 2), second the proposal of a path-based language for querying data on the Linked Data Cloud (Section 3), and third the implementation of a Linked Data server (called the "Linked Media Framework" or short "LMF") that implements and demonstrates these extensions (Section 4). Our approach is currently validated in several applications, two of which are described in Section 5. Section 6 compares our approach with related work.

2. THE LINKED MEDIA PRINCIPLES

One of the core principles of Linked Data is that it builds strongly on the HTTP protocol for content negotiation and retrieval. As [7] describes, content negotiation in the Linked Data world works as follows:

1. the client sends a HTTP GET request to a resource identified by a URI, together with an `Accept:` header of either `application/rdf+xml` or `text/html`
2. based on the header, the server decides whether to send a human-readable (`text/html`) or machine-readable (`application/rdf+xml`) representation of the resource; it sends a response code of `303 See Other` pointing to the respective resource representation
3. the client performs a second HTTP GET request to the location pointed to by the server response
4. the server responds with a `200 Ok` response code and delivers the requested representation

By applying this content negotiation, Linked Data remains compatible with existing Web browsers and standards. Our *Linked Media Principles* extend these Linked Data principles along two dimensions that are in principle independent from each other but both motivated through our application scenarios. The first extension is concerned with Linked Data updates, while the second is concerned with managing media content and metadata alike.

2.1 Extending Linked Data for Updates Using REST

The main precondition of our extensions is that they must remain fully backwards-compatible with existing Linked Data implementations so that existing tools can be used. Since Linked Data heavily builds upon HTTP as described above, a natural way of implementing updates in Linked Data is to make use of the HTTP PUT, POST and DELETE commands in addition to the GET command.

While not specifically excluded, Linked Data servers in their current implementations do not make use of these commands. However, they are commonly used to build highly interactive web applications using the REST ("Representational State Transfer") architectural approach to build web services as described in the thesis of Roy Fielding [3]. Like in Linked Data, the central principle of REST is "the existence of resources (sources of specific information), each of which is referenced with a global identifier (e.g., a URI in HTTP)".[1] Combining REST and Linked Data is therefore a natural choice.

Our extension of Linked Data for Updates follows the REST principles and applies them to resources on a Linked Data server. In addition to the HTTP GET command, we make use of POST, PUT and DELETE as follows:

- **POST** creates the resource represented by the URI used in the request; optionally, the request body can contain resource content or metadata, in which case the `Content-Type` header defines the format and kind of data that is sent (see PUT); the server will respond with a `201 Created` in case the resource is created and `200 Ok` in case the resource already exists
- **PUT** replaces the content or metadata of the resource represented by the URI used in the request with the data contained in the request body; the `Content-Type` header indicates the format and kind of the data that is sent; the server will respond with a `300 Multiple Choices` in case the resource exists, rewriting the URI according to the content type as described in the next section; a subsequent PUT to the redirected URI will update the content or metadata on the server; in case the resource does not exist, the server will return a `404 Not Found`
- **DELETE** deletes the resource represented by the URI used in the request and all associated content and meta-data; the response code will be either a `200 Ok` in case the resource exists and is successfully deleted or a `404 Not Found` in case the resource does not exist

Note that current HTTP client implementations often automatically rewrite the request method from PUT to GET when they receive a `303 See Other`, which makes it unsuitable for redirecting a resource update. The solution we chose to this problem is to use the status code `300 Multiple Choices`. In this case the response is not completely consistent with the Linked Data principles. In our implementation, we address this problem by a configuration option that switches between maximum Linked Data compatibility and maximum HTTP conformance.

2.2 Extending Linked Data for Arbitrary Media Types using MIME Mapping

The second extension we propose is capable of handling content and metadata in a uniform way on the same server. This is e.g. required when providing a Linked Data frontend to existing content or media asset management systems so that they can offer both the media content in different formats (e.g. an image in jpeg and png format) and the metadata about the media content (e.g. the EXIF metadata). In addition, our extension gives us the flexibility to deliver content and metadata in formats understood by the respective clients. With the increasing use of Javascript for implementing rich client applications, it is e.g. useful to deliver metadata in RDF/JSON[2] or JSON-LD[3] instead of RDF/XML.

Linked Data is currently only concerned about data and does not take into account media content that is also associated with the resource: when requesting the media type `text/html`, current servers will deliver a (usually tabular) HTML representation of the metadata but not non-metadata Web content. Likewise, it is not really possible to

[1] `http://en.wikipedia.org/wiki/Representational_State_Transfer`

[2] `http://docs.api.talis.com/platform-api/output-types/rdf-json`

[3] `http://json-ld.org/`

distinguish between a document of type RDF/XML *as content* and the metadata about it *as data*. As a consequence, a Linked Data server currently cannot be used as both, a content management system and a Linked Data repository. However, this would be desirable for systems that offer media content as well as meta-data, e.g. media asset management systems or document management systems.

Our extension is based on the content negotiation using `Content-Type` and `Accept:` headers as defined in the HTTP protocol. The Linked Data principles already apply this approach for GET requests using the `Accept:` header, but they leave open how exactly the server generates the redirect URI. In the Linked Media Framework, we implement a uniform mapping of resource URIs to redirect URIs depending on the MIME type passed in the `Content-Type` (PUT/POST commands) and `Accept:` (GET command) headers.

Distinguishing Content and Metadata. In our proposal, we distinguish between metadata and content by extending the media type passed in the `Content-Type:` or `Accept:` header with an additional `rel=...` parameter:

- a media type of `type/subtype; rel=content` indicates that the client requests or sends the *content* associated with the resource in the given format
- a media type of `type/subtype; rel=meta` indicates that the client requests or sends the *metadata* associated with the resource in the given format

For backwards compatibility with existing Linked Data clients, the default behaviour without rel-parameter is `meta`. However, for interaction with users it might be desirable to change this behaviour in certain configurations, as users are mostly interested in the content and not in the metadata.

URI Rewriting. The generation of redirect URIs is based on both the original URI and the extended media type. Resource URIs always have the form

```
http://<host>/<root>/resource/<id>
```

where `<host>` is the host name of the installation (with optional port), `<root>` is the root directory where the Linked Data server is installed and `<id>` is an arbitrary but unique identifier (can also include subpaths separated by '/'). Based on the media type, these URIs are rewritten as follows:

```
http://<host>/<root>/<kind>/<type>/<subtype>/<id>
```

where `<kind>` is either `content` or `meta`, `<type>` is the primary MIME type (e.g. `image` or `text`) and `<subtype>` is the subtype (e.g. `jpeg` or `rdf+n3`).

3. LDPATH: RESOURCE-CENTRIC QUERYING OVER LINKED DATA

Even though resources on Linked Data servers are typically interlinked and thus conceptually integrate data from many different sources, querying such data is still very cumbersome. The main reason is that existing query languages for RDF like SPARQL are rather dataset-centric and do not easily query over distributed or even unknown sources. There are currently three approaches to address this issue:

- a central index harvests the Web for RDF data and stores it in a central repository and offers it for querying, e.g. using SPARQL. This approach is followed e.g. by Sindice,[4] which offers a public SPARQL endpoint.
- a query is distributed over several query endpoints and the results are then combined. This approach is proposed in the SPARQL 1.1 Federation Extension [15].
- accepting the incompleteness of the results returned by the query and trying to improve the recall by different heuristics, as proposed e.g. by Hartig et.al. [6]

The first two approaches have obvious disadvantages: a central repository is not always recent and a single point of failure, while explicit federated queries are cumbersome to write and need exact information on how and where to access the SPARQL endpoint. They also require that all queried servers implement the SPARQL 1.1 Federation Extensions. The third approach is in our opinion not very user friendly, since the user cannot easily determine whether the results he will get are complete or not and important enterprise decisions might depend on that information.

In this section, we therefore propose an alternative approach to querying Linked Data resources based on a path traversal following RDF links between Linked Data resources. Our approach is resource-centric and thus intuitive to users of Linked Data. It also does not need any specific extension beyond the Linked Data principles and the standard HTTP protocol: any existing Linked Data server can be queried without modification. The path language has originally been developed as a means to configure the LMF Semantic Search component, but we believe it is a valuable contribution in itself.

3.1 RDF Path Language

LDPath follows the same ideas and syntax as XPath 1.0 [14], but instead of XML elements it allows traversal over the conceptual RDF graph represented by interlinked Linked Data servers. In its core, LDPath is thus similar to the SPARQL 1.1 Property Paths [18]. However, LDPath is deliberately restricted in its expressiveness to syntactically disallow queries that would be difficult to evaluate completely on the Linked Data Cloud (e.g. following links backwards). The path language supports a number of path selectors that start at the current "context resource" and return a collection of nodes based on the path specification. A full description of LDPath is available as part of the LMF documentation.[5] The following example shows the selection of the foaf:name of all friends of the person represented by the queried resource:

```
friend = foaf:knows/foaf:name :: xsd:string;
```

The remainder of the section outlines the capabilities of the LDPath language by providing a brief overview.

Property Selection. Property selections allow to follow a triple (RDF link) to another node in the RDF graph. The node may be either local or remote. Property selections have the following form:

```
<URI> | PREFIX:LOCAL
```

i.e. either a URI enclosed in `<..>` or a property name in abbreviated notation using namespace prefix and local name (prefix mapping has to be defined, cf. the online documentation). The following two paths both select the name of a person represented by the context resource as a string:

```
name = foaf:name :: xsd:string ;
name = <http://xmlns.com/foaf/0.1/name>::xsd:string;
```

[4] http://sindice.com/

[5] http://code.google.com/p/ldpath/

Path Traversal. RDF links can be followed by separating selectors with `/`. In this case, the left-hand selector is evaluated first to select new context nodes, and for each of these context nodes the right-hand selector is then evaluated. The following LDPath expression first selects the resources representing the persons known to the person represented by the current resource, and then selects their names as string:

```
friend = foaf:knows/foaf:name :: xsd:string;
```

Path traversal transparently "hops over" to other Linked Data servers when needed, because the querying can be performed entirely by issuing Linked Data retrievals.

Path Connectors and Grouping. Several path selectors can be connected with boolean operators, forming a union (connector `|`) or intersection (connector `&`) of the selection results. Parentheses can be used to change the operator precedence. For example, the following LDPath expression selects the names of all friends, regardless whether they are represented using the FOAF or the RDFS vocabulary:

```
friend = foaf:knows/foaf:name
       | foaf:knows/rdfs:label :: xsd:string;
```

The same query could be expressed by grouping the union in the path traversal:

```
friend = foaf:knows/(foaf:name | rdfs:label)
```

Path connectors are a simple but powerful tool for coping with the use of different vocabularies on different servers for the same kind of information.

Path Tests. Path tests allow filtering the result nodes of a selector based on a condition on their properties. Path tests are added at the end of a selection and embraced in `[...]` like in XPath. A path test can either test for the existance or value of a property, in which case it is itself a selector, or on the language of a literal. For example, the following LDPath expression would select RDFS labels if they are either in German or without language specification:

```
title = rdfs:label[@de] | rdfs:label[@none]
```

Similarly, the following RDF Path would only select interests of a person that are a kind of food:

```
food = foaf:interest[rdf:type is ex:Food]
```

Path Functions. Functions can be used inside the path to transform the results of path queries. For example, the following function selector concatenates the FOAF first name and last name of a person and represents it as a string:

```
f = foaf:knows/fn:concat(foaf:given,foaf:surname)
```

Functions can thus also be used to cope with different ways of representing the same kind of information on different servers. Currently, the LMF provides a number of built-in functions, among them string concatenation (`fn:concat`), HTML removal (`fn:removeTags`), and XPath selection within XML literals (`fn:xpath`).

Path Types. LDPath expressions have an optional result type that may be used to transform the selected nodes into a different form, e.g. a string, a date, a URI, or a point. Result types are added to a path selection with `::` followed by the type name. The following LDPath expression selects the `foaf:based_near` property and converts the result nodes into a special location type that holds latitude and longitude in a complex object (for geo-indexing in the search index):

```
location = foaf:based_near :: lmf:location ;
```

Currently, the LMF offers the XML Schema base types as well as a number of specialised datatypes used to configure the semantic search index.

3.2 Linked Data Caching

Underlying the LDPath language is an implementation of a Linked Data client that transparently retrieves and caches resources and their triples when needed. The basic operation is as follows: whenever a navigation step of the path language moves the context to a new resource, we check whether the resource is already locally cached and not expired (i.e. the expiry time has not passed). If yes, we perform the next part of the query using the cached version. If no, we issue a HTTP request to retrieve the triples of that resource and cache them locally using the expiry time sent by the server as conforming to the HTTP standard. For retrieving resources, we support the following three types of requests:

Linked Data Requests. The default way of retrieving a resource is to issue a request conforming to the Linked Data principles and rely on content negotiation and redirects to get the triple representation. Our implementation currently supports RDF/XML, Turtle, N3, and RDF/JSON.

SPARQL Requests. There is still much data on the Web of Data that does not conform to the Linked Data principles. To support querying also this data, it is possible to configure that requests to certain resources should be handled using a SPARQL endpoint instead of issuing a Linked Data request. In this case, our implementation sends a simple SPARQL query as follows:

```
SELECT ?p ?o WHERE { <RESOURCE> ?p ?o }
```

This query essentially returns all triples for a fixed resource passed as argument. Another option would be to instead issue a SPARQL DESCRIBE query, but the SPARQL specification is not very specific on what data would be returned.

Cache Requests. The third type of request is sent to a cache server that contains a copy of a resource description originating somewhere else. There are several justifications for this type of requests: (1) requests to a specific server might be very frequent and it is useful to have a local copy, (2) a remote server might be unreliable or slow justifying a local copy, (3) access should be only to a "trusted" or "certified" server where the data will not change unexpectedly, and (4) there is a central repository like Sindice[6] that harvests resource descriptions from the whole web of data and can thus provide additional information about a resource.

Cache requests are simple mappings from resource URI patterns to REST webservices that take as argument the resource URI and return the triple data.

Beyond Linked Data. The Linked Data Caching component goes beyond ordinary Linked Data by providing an API that allows implementing wrappers around other data sources on the Web (e.g. the YouTube or Vimeo API). Such wrappers map proprietary data structures into standardized RDF vocabularies (in the case of YouTube and Vimeo: the Media Ontology) or try to extract structured information out of unstructured content (e.g. EXIF information from images or more sophisticated information extraction). In this way, proprietary data sources can be accessed transparently using the same tools as Linked Data resources.

[6] `http://sindice.com`

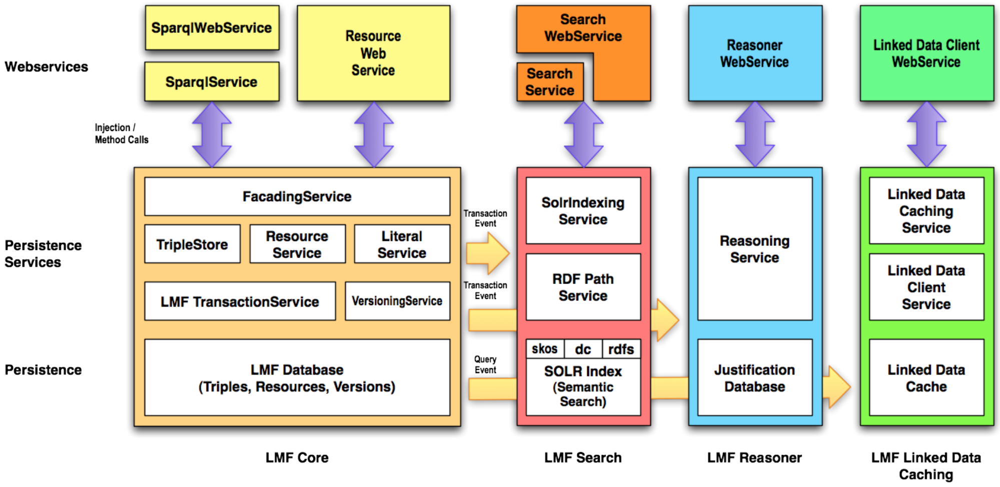


Figure 1: Service-Oriented Architecture of the Linked Media Framework. The ResourceWebService in LMF Core implements the proposed Linked Data extensions. LMF Search provides services for facetted semantic search. LMF Sparql offers querying using a SPARQL endpoint. Services are interacting either via events or via injection (loose coupling).

4. IMPLEMENTATION: THE LINKED MEDIA FRAMEWORK

To evaluate the practicality of the suggested extensions, we have developed a prototypical implementation in our Linked Media Framework (LMF). We are implementing several real-world scenarios based on this system (cf. Section 5).

The general architecture of the Linked Media Framework is depicted in Figure 1. The architecture is service oriented and strictly follows the principle of loose coupling of components, making the system very extensible and configurable. Most core components of the LMF are developed using Java and were originally implemented in the KiWi System [12, 9]. The current implementation of the Linked Media Framework consists of the following modules:

- *LMF Core* implements the Linked Data server including the extensions we propose. Linked Data and extensions are handled by the ResourceWebService. The ResourceWebService is backed by our own persistence implementation that in addition to triple management provides transactions and versioning, and is prepared for storing provenance information and reason maintenance information in conjunction with the reasoner.
- *LMF Search* offers semantic search over resources based on Apache SOLR.[7] The search component is highly customizable and allows mapping "LD Paths" to index fields; by default we include rulesets for RDF, SKOS, and Dublin Core. The LMF Search component can be accessed using a REST API conforming to the OpenSearch standard and is compatible with the Apache SOLR search API. Existing SOLR clients can thus be used for accessing the search functionality.
- *LMF Sparql* provides a SPARQL endpoint for querying and updating the data contained in the LMF installation. The implementation makes use of Sesame to offer SPARQL 1.1 Query and Update support.
- *LMF Linked Data Caching* provides an implementation of transparent Linked Data Caching as described in Section 3. Linked Data Caching is used for LDPath queries as well as (in limited cases) for SPARQL.
- *LMF Reasoner* offers a subset of the rule-based reasoner sKWRL [8] developed in the KiWi project for reasoning over triples contained in the LMF. Rules can be added and updated by the user during runtime and are evaluated using a forward chaining fixpoint algorithm. In addition to the inferred triples, the LMF Reasoner also computes so-called *justifications* that give explanations (base triples and rules used in the inference) why certain triples have been inferred. Justifications are used for efficient updating (so-called *reason maintenance* or *truth maintenance*) and for displaying to the user.

Components of the LMF typically interact by sending events. For example, upon completion of a transaction, all transaction data is handed over to the search component for indexing and to the reasoner for incremental inferencing. Additional modules are currently under development. Particularly, we are working on an implementation of WebACL[8] for managing access to resources, integration of content analysis and semantic enhancement based on Apache Stanbol[9], and on extending the reasoner to support full sKWRL as well as event-condition-action rules, which are useful in our application scenarios. With respect to performance and stability, we have tested the system with the GeoNames data set, about 143 million triples. The response times were accurate and the system stable.

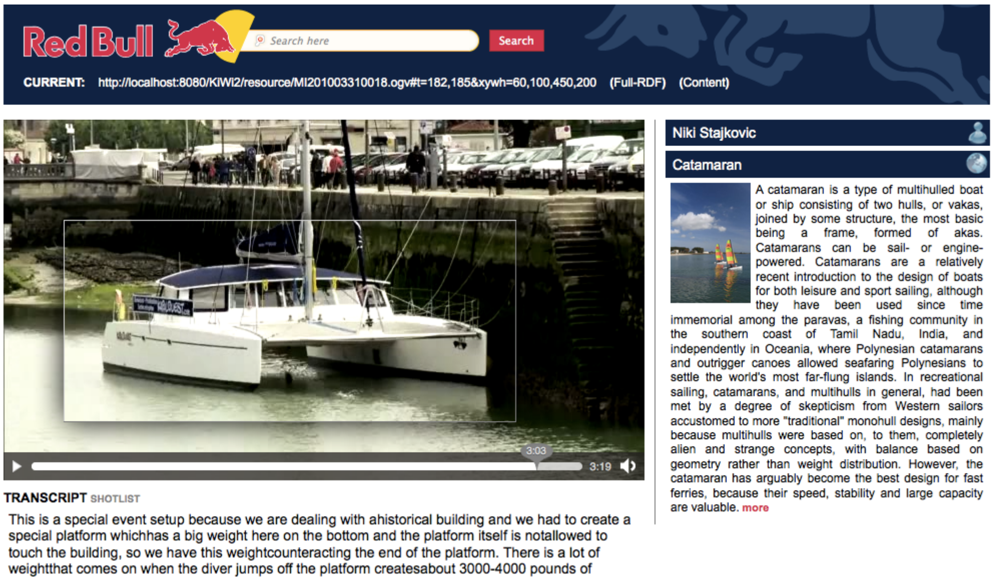
5. APPLICATION SCENARIOS

We are evaluating our technology in a number of application scenarios that are currently implemented together

[7] http://lucene.apache.org/solr/

[8] http://www.w3.org/wiki/WebAccessControl

[9] http://incubator.apache.org/stanbol/



CURRENT: http://localhost:8080/KiWi2/resource/MI201003310018.ogv#t=182,185&xywh=60,100,450,200 (Full-RDF) (Content)

TRANSCRIPT SHOTLIST

This is a special event setup because we are dealing with ahistorical building and we had to create a special platform whichhas a big weight here on the bottom and the platform itself is notallowed to touch the building, so we have this weightcounteracting the end of the platform. There is a lot of weightthat comes on when the diver jumps off the platform createsabout 3000-4000 pounds of

Niki Stajkovic

Catamaran

A catamaran is a type of multihulled boat or ship consisting of two hulls, or vakas, joined by some structure, the most basic being a frame, formed of akas. Catamarans can be sail- or engine-powered. Catamarans are a relatively recent introduction to the design of boats for both leisure and sport sailing, although they have been used since time immemorial among the paravas, a fishing community in the southern coast of Tamil Nadu, India, and independently in Oceania, where Polynesian catamarans and outrigger canoes allowed seafaring Polynesians to settle the world's most far-flung islands. In recreational sailing, catamarans, and multihulls in general, had been met by a degree of skepticism from Western sailors accustomed to more "traditional" monohull designs, mainly because multihulls were based on, to them, completely alien and strange concepts, with balance based on geometry rather than weight distribution. However, the catamaran has arguably become the best design for fast ferries, because their speed, stability and large capacity are valuable. more

Figure 2: Video player interface of Red Bull scenario, displaying time-based as well as region-based media fragment annotations based on the Media Fragments URI specification and the Media Ontology. Additional information from internal and external Linked Data sources is displayed while the video is playing

with partner companies. The aim of the Linked Media Framework is to integrate information in enterprises by using Linked Media. The first scenario shows the interlinking of multimedia content with background information from the Linked Data Cloud, whereas the second scenario outlines the seamless integration of the technology with existing systems and the annotation of human-readable content with Linked Data concepts. In all applications, the Linked Media Framework is used for provisioning a semantic search service. Demonstrators of both scenarios are available at `http://labs.newmedialab.at/`.

5.1 Red Bull: Interlinking Media Fragments

The Red Bull Content Pool[10] designed and operated by Red Bull Media House is the central repository of media content related to sports events organised by Red Bull, e.g. the Air Race, the Cliff Diving Competition, or the Red Bull Rampage mountainbike race. Media content is mostly raw or processed video material that Red Bull offers to other media providers in different formats and quality for further use. Typically, the content is also annotated with the event, year, location, and athletes that are shown. The following persona (Markus, the sports journalist) describes a typical intended use of the Red Bull Content Pool:

Markus works as a sports journalist for a small regional TV station. Since the cliff diving world championship also tours through the region, he has to prepare a short TV report on cliff diving in general, famous athletes and spectacular scenes in known places. As a small TV station does not have its own video material, he visits the Red Bull Content Pool to acquire the content he needs from there.

Linked Media helps Markus in the following tasks:

- in finding videos and video fragments in the Red Bull Content Pool ("Fort Boyard", "Helicopter Scene"), because videos and video fragments are annotated and indexed for semantic search
- in finding relevant background information for videos, e.g. about athletes and locations, because videos are connected with background content and data from the Linked Data Cloud and from Red Bull
- in finding the persons behind videos, because videos are connected with the people that are related to them

The Red Bull Content Pool scenario makes use of Linked Data for three purposes: (1) enriching own content with higher-quality metadata from the Linked Data Cloud, e.g. about athletes or locations, (2) publishing content and metadata as Linked Data for others, and (3) making multimedia content available for cross-referencing and interlinking.

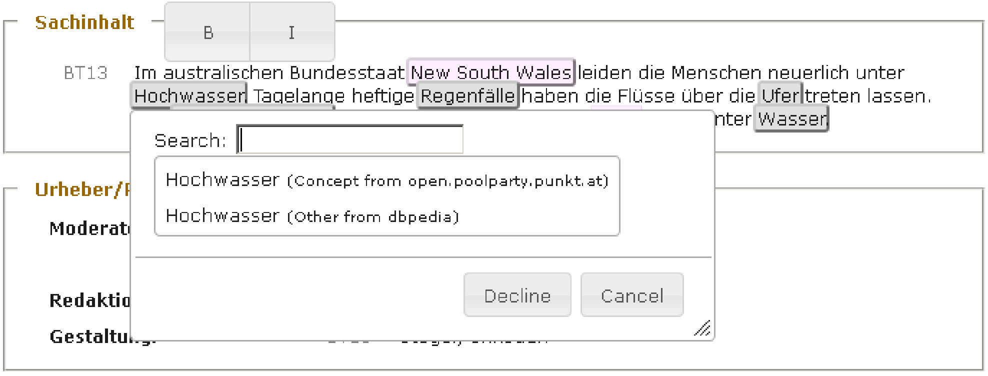
All three functionalities are implemented based on the Linked Media Framework using sample media content from the Cliff Diving events (see Figure 2). Media fragments are identified by making use of the Media Fragments URI 1.0 specification [16]. Annotations of media resources and fragments are represented in the LMF using the W3C Media Ontology [17].Media resources are annotated based on (manually created) metadata sheets with basic information (keywords, time, location), textual transcripts of spoken language, and shotlists briefly describing scene settings. These textual transcripts serve both, as textual annotation for multimedia fragments and as input for named entity recognition and semi-automatic interlinking with Linked Data concepts.

In addition to the Linked Data functionalities described above, the LMF also offers semantic search over the media content and metadata, and an interactive video player that displays additional related information to videos while they are playing (see Figure 2).

5.2 ORF Archive: Interlinking News

The ORF Archive is the central repository for all video and audio material created by the Austrian Television in the last 60 years and contains a vast amount of media content in

[10] `http://www.redbullcontentpool.com`



Im australischen Bundesstaat New South Wales leiden die Menschen neuerlich unter Hochwasser Tagelange heftige Regenfälle haben die Flüsse über die Ufer treten lassen.

nter Wasser

Search:

Hochwasser (Concept from open.poolparty.punkt.at)

Hochwasser (Other from dbpedia)

Figure 3: Annotation and interlinking interface for the ORF archive. Suggestions for interlinking are highlighted in the text; when clicking, a dialogue is opened recommending Linked Data resources for interlinking. When confirmed, annotations are stored in the LMF for semantic search.

different formats. Its primary objective is to preserve video material for potential future use and make it accessible to editors. The main task is therefore to properly annotate and organise video material so that it can be retrieved easily later and the related context information is readily available. In this scenario, we work with two types of content: news material of the daily news magazine and sports content. The news material typically is devided into short news reports with a brief textual description of the content, the journalist, and the location. The following persona describes a prototypical archivist working in the ORF archive:

Monika works as an archivist in the ORF archive. When she receives video material from the editors, it already contains base metadata as well as a textual description. Her task is to annotate the content with metadata of consistently high quality and connect it with the appropriate background information for editors to later quickly get an overview of the context. Of particular interest are locations and persons, because they are often ambiguous or written differently. The time Monika has available for performing the annotation is typically very limited – a news article of 3 minutes has to be fully annotated in at most 15 minutes.

The LMF addresses Monika's needs as follows:

- it performs automatic analysis of textual descriptions of the video material using NLP methods and suggests appropriate concepts from pre-configured Linked Data servers (either internal or public)
- it provides quick access to background information by transparently retrieving content from the Linked Data Cloud as needed
- it allows to directly store the textual content as well as the metadata for a news resource and make it available to semantic search
- it does not require exhaustive adaptations of the proven processes within the company. The annotation and search facilities are optional addons for the archivists and journalists.

This scenario is implemented by combining the LMF with Apache Stanbol[11], also developed at Salzburg Research. A typical use case is shown in Figure 3 above: an archivist can open a page describing a news report in his web browser, which has a special annotation plugin installed. Clicking on "annotate" sends the textual content to a special installation of Apache Stanbol, configured with an index of both, trusted Linked Data sources and an internal SKOS thesaurus. Stanbol analyses the text by applying natural language processing and suggests annotations from the configured sources, e.g. a concept from DBPedia or from the internal company thesaurus. By clicking on a highlighted word, the archivist can review the suggestion and select one or more of them for annotation. Annotations are then stored in the LMF using the resource-based updating mechanism or SPARQL Update and available to semantic search.

6. RELATED WORK

There have been some proposals for updating Linked Data and comparing and combining Linked Data with RESTful architectures, as well as for uniform access to content and data. In the following we summarise the most recent state-of-the-art and describe how our approach differs from it.

Combining Linked Data and REST. The combination of Linked Data and REST is quite natural and has been implicitly present even in the original Linked Data principles proposed by Tim Berners-Lee [1]. Most articles are however concerned only with retrieval of resources, not with updating. The idea of also applying REST for updating has gained more interest only recently: [2] e.g. mentions an *information resource modification mechanism* based on REST, and [4] described an actual implementation of an approach called *Rhizomer* that is similar to ours, but the authors do not go into further detail. The article [10] provides a very interesting comparison of similarities and differences between Linked Data and REST. Finally, the article [5] also describes the use of REST for updating Linked Data resources, albeit based on SPARQL Updates.

From these approaches our proposal differs as follows: (1) it treats Linked Data *updates* in a way that is analogous to Linked Data *retrieval*; (2) it provides a detailed and complete specification of the content negotiation and protocol, (3) it has been implemented and evaluated in real-world scenarios, and (4) it addresses both updates and uniform content and metadata management.

Updating Linked Data. Most of the research on Linked Data so far is concerned with publishing and consuming Linked Data, and there are only few proposals on how to

[11] http://incubator.apache.org/stanbol/

update a Linked Data server. Updating currently typically means re-exporting the original dataset into RDF or performing RDF triple store operations programmatically. There are only few approaches that aim to provide a webservice API for this purpose. For example, OpenLink Virtuoso offers a SPARQL endpoint that is capable of executing updates using the SPARQL Update extension[12] currently under development. Most noteworthy is sparqlPuSH [11], which is based on SPARQL Update and the PubSubHubbub protocol[13]. A very interesting aspect of this approach is that it allows to proactively distribute updates using a publish-subscribe mechanism.

SPARQL updates are well suited when updates are carried out programmatically or distributed from a main data source to secondary data sources. However, in contrast to our approach, both the Virtuoso approach and the sparqlPuSH approach are not resource-oriented in the same way as the retrieval part of Linked Data.

Path Languages for RDF/Linked Data. A number of path languages have been proposed for RDF,[14] most of them now dormant. Most prominent of the path languages is SPARQL Property Paths, included in the SPARQL 1.1 Query working draft [18]. The path language proposed in this article resembles SPARQL Property Paths in the way basic navigation is carried out. LDPath differs from SPARQL Property Paths in several aspects:

- it is a query language with restricted expressiveness; Property Paths are part of full SPARQL and as such not well suited for querying Linked Data, because SPARQL allows queries that require complete knowledge of the Linked Data Cloud to be carried out correctly; in LDPath, only queries can be expressed that can be evaluated properly over Linked Data
- it extends Property Paths by some concepts from XPath [14], particularly node tests for restricting result sets, functions for carrying out transformations, and XML types for representing type information
- it is backed by a Linked Data client service that carries out queries using only the Linked Data principles; no knowledge of endpoint locations or existance is necessary (but can be exploited if available)

Another very similar approach allowing a path-based traversal through the Linked Data Cloud is Ripple, a "Semantic Web Scripting Language" [13]. While Ripple aims to be a complete functional programming environment for Linked Data, LDPath concentrates on the single task of querying Linked Data. It can thus be embedded in many different host environments easily, requires less learning effort for developers, and its implementation is very lightweight.

7. ACKNOWLEDGMENTS

This research has been funded by the Republic of Austria within the COMET project Salzburg NewMediaLab and by the the European Commission within the 7th Framework Programme project KiWi - Knowledge in a Wiki (No. 211932). The latest version of the Linked Media Framework source code is available with a permissive Open Source license at `http://lmf.googlecode.com`.

8. REFERENCES

[1] T. Berners-Lee. Linked Data, 2006.

[2] B. Ferris. A generalisation of the Linked Data publishing guideline (blog post), 2011.

[3] R. Fielding. Architectural styles and the design of network-based software architectures. 2000.

[4] R. García, J. M. Brunetti, A. López-Muzás, J. M. Gimeno, and R. Gil. Publishing and Interacting with Linked Data Categories and Subject Descriptors. In *1st Int. Conference on Web Intelligence, Mining and Semantics, WIMS'11*, Sogndal, Norway, 2011.

[5] A. Garrote and M. Moreno García. RESTful writable APIs for the web of Linked Data using relational storage solutions. In *WWW2011 Workshop: Linked Data on the Web (LDOW2011)*, Hyderabad, 2011.

[6] O. Hartig, C. Bizer, and J.-C. Freytag. Executing SPARQL Queries over the Web of Linked Data. In *Proc. 8th International Semantic Web Conference (ESWC20099)*, Washington DC, USA, 2009.

[7] T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space. Synthesis Lectures on the Semantic Web: Theory and Technology, 1:1.* Morgan & Claypool, 1st edition, 2011.

[8] J. Kotowski and F. Bry. A Perfect Match for Reasoning, Explanation, and Reason Maintenance: OWL 2 RL and Semantic Wikis. *Proc. of the 5th Semantic Wiki Workshop (SemWiki 2010) at ESWC 2010*, 2010.

[9] T. Kurz, S. Schaffert, T. Bürger, S. Stroka, and R. Sint. KiWi - A Platform for building Semantic Social Media Applications. In *9th Int. Semantic Web Conference (ISWC2010)*, 2010.

[10] K. R. Page, D. C. De Roure, and K. Martinez. *REST and Linked Data.* ACM Press, New York, New York, USA, Mar. 2011.

[11] A. Passant and P. N. Mendes. sparqlPuSH : Proactive notification of data updates in RDF stores using PubSubHubbub. In *Scripting for the Semantic Web Workshop SFSW2010 at ESWC2010*, 2010.

[12] S. Schaffert, J. Eder, S. Grünwald, T. Kurz, M. Radulescu, R. Sint, and S. Stroka. KiWi - A Platform for Semantic Social Software. In *Proc. of the 4th Semantic Wiki Workshop (SemWiki 2009), at ESWC 2009*, 2009.

[13] J. Shinavier. Ripple: Functional programs as linked data. In *3rd Workshop on Scripting for the Semantic Web*, Innsbruck, Austria, 2007.

[14] W3 Consortium. *XML Path Language (XPath)*, November 1999. `http://www.w3.org/TR/xpath`.

[15] W3 Consortium. *SPARQL 1.1 Federation Extensions (W3C Working Draft)*, June 2010. `http://www.w3.org/TR/sparql11-federated-query/`.

[16] W3 Consortium. *Media Fragments URI 1.0 (W3C Working Draft)*, March 2011. `http://www.w3.org/TR/media-frags/`.

[17] W3 Consortium. *Ontology for Media Resources 1.0 (W3C Candidate Recommendation)*, July 2011. `http://www.w3.org/TR/mediaont-10/`.

[18] W3 Consortium. *SPARQL 1.1 Query (W3C Working Draft)*, May 2011. `http://www.w3.org/TR/sparql11-query/`.

[12] `http://www.w3.org/TR/sparql11-update/`

[13] `http://code.google.com/p/pubsubhubbub/`

[14] `http://www.w3.org/wiki/RdfPath`