SemwidgJS: A Semantic Widget Library for the Rapid Development of User Interfaces for Linked Open Data

Timo Stegemann, Jürgen Ziegler

University of Duisburg-Essen
Lotharstr. 65
47057 Duisburg
timo.stegemann@uni-due.de
juergen.ziegler@uni-due.de

Abstract: In this paper we introduce SemwidgJS, a JavaScript based library for displaying Linked Open Data via *Semantic Widgets*. SemwidgJS can be integrated in almost any website and handles the querying, processing and displaying of semantic data. SemwidgJS facilitates the retrieval of semantic data with its own simplified query language SemwidgQL, a Path Query Language that allows the traversal of the Linked Data graph. For complex queries, standard SPARQL can be used as well. We pursue two objectives with this development: first, we want to provide appropriate widgets for the full range of different data types in semantic databases, second, we aim at enabling Web content publishers who are not experts in semantic technologies to make use of these data in their ordinary Web development activities.

Introduction

The amount of semantically represented data on the open Web as well as within organizations has been growing significantly over the past few years. Semantic databases such as DBpedia[ABK+07] are continuously[1] collecting knowledge as interlinked RDF data, making them publicly available as Linked Open Data[2]. Large corporations, for example Google with its Knowledge Graph [Sin12] or the BBC with data sets for the FIFA World Cup 2010 and the Olympic Games 2012 [Ray12], have recently started to use internal semantic databases. While this data may be visible on standard websites, the technology exploiting the data is not open for external access. Although there is already an enormous amount of valuable information available, the usage of semantic data is still quite limited and has not yet made its way into wide-spread utilization either by the Web developer community or, even more so, by end users. Yet, due to their generic format, global interlinking and standardized query languages, semantic data and services form an excellent basis for building more flexible, generic applications and on-the-fly integrations of distributed data sets. A major issue for exploiting this potential is to significantly alleviate the development of applications based on Semantic Web data.

[1] `http://wiki.dbpedia.org/Changelog`
[2] `http://lod-cloud.net/`

To date, access to Linked (Open) Data usually takes place in two ways: first, through (Semantic) Web Browsers that allow a general, unfiltered display of information, or, second, through tools that are highly specialized for specific problems but limited in their wider applicability. With both types of tools, the interaction typically starts with the user entering a search string or the URI of a known semantic resource as an entry point into the semantic data graph. From there, the end user can then interactively explore related data. Alternatively, an automatic visualization of relevant data is generated.

The current techniques are a major obstacle for average bloggers, website authors, and other content publishers who may want to request Linked Open Data, integrate different sources, mix them with their own editorial content and publish them, similar to what BBC did with the sports datasets mentioned above. When using common blog-publishing services, users typically do not have the necessary access to the server back-end to integrate existing software that supports the usage of Linked Open Data.

To overcome some of these obstacles, we propose in this paper the concept of *Semantic (Data) Widgets* and describe an implementation that shields users from the complexity of the RDF query language (SPARQL[3]) and facilitate the construction of Web applications that use and integrate semantic data. We introduce SemwidgJS, a JavaScript based library for displaying Linked Open Data through Semantic Widgets. SemwidgJS can be integrated in almost any standard HTML webpage and handles the querying, processing and displaying of semantic data. While existing libraries sharing a similar goal only comprise widgets for information visualization purposes, SemwidgJS also features widgets for typical UI elements such as labels, links, and text input fields. To make querying semantic data easy, SemwidgJS supports its own query language – SemwidgQL which is used as an alternative to standard SPARQL (which can still be used additionally). SemwidgQL is a simple Path Query Language that allows the traversal of the Linked Data graph starting with a predefined resource. We aimed at keeping the query language as simple as possible to enable users who are not familiar with the techniques of the Semantic Web to use SemwidgJS.

The remainder of this paper is organized as follows: in the next section we review related work and the current state of research. Then we introduce SemwidgJS, subdivided into the SemwidgQL query language, the widget library, and its capability to act as a framework for custom widgets. Finally, we conclude with a discussion and give an outlook on future work.

Related Work

Several tools and frameworks for exploring and displaying information from the Semantic Web, in particular data from the Linked Open Data (LOD) cloud, have been described in the literature so far. Existing tools typically either allow to browse and visualize semantic data in a generic, standard format, as is the case for Semantic Web browsers [BLCC+06], or are highly specialized for solving a specific problem, for example, finding and visu-

[3] **http://www.w3.org/TR/rdf-sparql-query/**

alizing relations between Linked Data instances [HHL+09]. These tools are mostly very limited in terms of their ability to display semantic data in custom aggregations and layouts or mix them with ordinary Web content, which is the goal of our development.

Projects with similar objectives exist, using both client-side and server-side techniques. Consequently, they differ in the way they access data and how the type of visualization is defined. Sgvizler [Skj12] is a pure JavaScript libraries for information visualization utilizing Google Charts[4]. It queries Linked Open Data endpoints directly via SPARQL. Visualizations can easily be embedded in existing sites or systems (such as wikis or blogs) and are therefore usable for everyone. After including the JavaScript file Sgvizler widgets are declared directly within the HTML code. Chart type, SPARQL endpoint URL, and SPARQL query are defined by HTML5 data-* attributes. The chart is then rendered inside the page. Sgizler is based on RDF-Spark[5], which allows arbitrary visualizations by custom formatting functions programmed in JavaScript.

Exhibit[HKM07] also works on the client side, but does not utilize native semantic data from a SPARQL endpoint or an RDF file. Instead, data has to be serialized as JSON file and be embedded into the website that will contain the Exhibit widget. The widgets themselves are declared within the HTML code.

In contrast, LESS [ADD10] works on the server side. Templates are created in a Web application and saved to a server. Usable data is defined by a SPARQL query or by specification of the URI of a Semantic Web resource. Using the Smarty template engine for PHP, this data can be integrated into HTML code. Templates can be retrieved from the server via a URL and embedded into an existing website. Users hereby depend on both the SPARQL endpoint and the LESS server. The LESS server is no longer available online.

Paggr [Now09] allows the collaborative creation of widgets ("Sparqlets") in a wiki-like dashboard system. The system can query SPARQL endpoints, but also process RDF files directly. SPARQL queries can be generalized with placeholders and queries can be processed by means of a specific script language ("SPARQLScript"). However, using the widgets is restricted to the dashboard system.

Fresnel [PBKL06] is an RDF vocabulary for creating RDF style sheets. These style sheets consist of Lenses and Formats. Lenses select data, while Formats format the selected data. The way the data is displayed is determined by the tool, that utilizes Fresnel. To select data, either SPARQL or the Fresnel Selector Language (FSL)[6] can be used. FSL is a path language for selecting RDF data, similar to XPath[7], which operates on XML documents. However, FSL is not intended for directly querying data from a particular resource. The resource must first be selected by a rather sophisticated filter expression[8]. FSL is restricted to a single variable for its results. Therefore, result sets will always be one-dimensional lists. This is sufficient for simple UI elements, but it can be cumbersome to fill complex elements such as tables or charts that rely on multidimensional data.

[4] **https://developers.google.com/chart**
[5] **http://km.aifb.kit.edu/sites/spark/**
[6] **http://www.w3.org/2005/04/fresnel-info/fsl/**
[7] **http://www.w3.org/TR/xpath20/**
[8] `*[uri(.) = 'http://example.org/foo#bar']`

Another path language for selecting RDF data is LD Path [SBK+12], which became part of the Apache Marmotta project[9]. The alternative RDF serialization Notation3 (N3) already has its own syntax for path querying[10].

Outside the world of the Semantic Web, many JavaScript widget libraries and frameworks[11,12] are already available for normal Web content, which can be utilized to create Web UI elements dynamically and build Rich Internet Applications. Integrating semantic data, however, is mostly difficult or not possible at all.

None of the tools and languages discussed above fully meets the requirements of facilitating the integration of semantic data into conventional Web applications with Semantic Widgets that cover the whole range of data types available in LOD. While Sgvizler is integrateable into any website, it is limited to to the display of charts and users have be able to write SPARQL queries, which is one of the biggest obstacles. Server-side applications such as LESS add a potential single point of failure to the data retrieval process when used as a Web service. Installation of own server software is often impossible for average users or associated with high effort. Fresnel style sheets are written in RDF and therefore reserved for experts. In general, the existing tools require specific knowledge of semantic techologies and set a rather high entry threshold for the average Web developer.

While the different path-oriented query languages are fairly easy to use, they operate on schema level, which complicates their use as a query language for publishing website content. Operation on schema level implies that all data for a class or type is queried. Though typically websites show data for specific instances instead of lists of instances. Using these query languages on instance level requires additional filter expressions. Furthermore, FSL is limited to one-dimensional result lists, which can make it cumbersome to fill complex elements and can lead to a loss of context information. It is not sufficiently defined whether and how these query languages can be translated into SPARQL, which is indispensable for a client-side Web application accessing arbitrary semantic data sources.

SemwidgJS

SemwidgJS is a JavaScript based library for displaying *Semantic Widgets* that enriches websites with UI elements whose data is queried from SPARQL endpoints. SemwidgJS supports creation of simple UI elements such as labels and links, interactive elements for data input and selection that can manipulate other widgets, but also complex elements for displaying charts and maps that can base on further widget libraries.

While the data is queried from publicly available semantic databases, SemwidgJS otherwise works entirely on the client side. Thus no additional software is required. Widgets are declared directly within the HTML code and automatically processed after loading the website. For this purpose, only the SemwidgJS file must be included into the HTML

[9] **http://marmotta.apache.org/**
[10] **http://www.w3.org/TeamSubmission/n3/**
[11] **http://www.sencha.com/products/extjs/**
[12] **http://yuilibrary.com/**

document. To simplify data queries and thus be able to address a larger audience than comparable tools do, we developed SemwidgQL – a simple query language that is translated into SPARQL. If necessary, users can still express more sophisticated queries in SPARQL.

With SemwidgJS website authors can integrate simple data into their website. Furthermore, the internal event system and the ability to manipulate other widgets enables them to create complex Web applications. Changes which are made to widgets or the Document Object Model of a website, either by another widget or an external script will trigger an event the widgets can react on. In the simplest case the widget is reloaded, using the updated data.

In the further course, we present the SemwidgJS widget system in detail. Because the widgets can be used effectively only in combination with the SemwidgQL query language, we start by describing this language first. Then we introduce the widget library and describe how SemwidgJS can be used as a framework for creating own widgets.

SemwidgQL – Semantic Widget Query Language

SemwidgQL is a simple Path Query Language for Linked Data, starting with a predefined resource in the Linked Data graph. Linked Data is based on the RDF[13] data model. Information is represented in statements, called triples composed of subject, predicate and object. The object of a triple can be a literal such as a number or string, but also a separate Linked Data resource that is the subject of other triples.

SemwidgQL is translated into SPARQL – the standard query language for Linked Data – and can thus be used to query almost any public Linked Data endpoint. The intention behind SemwidgQL is not to replace or extend SPARQL. It is merely intended to simplify requesting instance data for users who are not familiar with the techniques of the Semantic Web.

SemwidgQL is limited to the traversal of the Linked Data Graph plus simple filter functions. The traversal is indicated by the dot notation, which is reminiscent of the syntax of object-oriented programming. Filters are enclosed in parentheses and refer to the previously defined property. Filters can be shortened by certain keywords.

Due to this straightforward syntax, an editor can easily provide assistance during the query creation process. Available properties of a given resource can be requested and be used for an auto-complete/-suggest feature. The same applies to the possible filter functions. We have developed an basic editor for creating SemwidgQL queries that offers the user such assistance[14].

The basic features of SemwidgQL are introduced below. All examples can be used to query data from DBpedia's public SPARQL endpoint.

Named Resources: SemwidgQL supports a naming mechanism for resources. Instead of fully qualified resource URLs or URLs prefixed by a namespace, a user can specify a

[13] **http://www.w3.org/TR/rdf-primer/**
[14] **http://semwidg.org/ql/editor/**

substitute name and use it within all SemwidgQL queries.

```
rome = http://dbpedia.org/resource/Rome
```

Path Traversal: The following SemwidgQL query requests all labels of the resource Rome. Below the corresponding translation into SPARQL is shown. In order to save space and increase readability, we replaced the fully qualified resource name of Rome by its prefixed name (dbpedia:Rome) in the SPARQL query.

```
rome.rdfs:label
```

```
SELECT DISTINCT
  (dbpedia:Rome AS ?uri) ?label
WHERE {
  dbpedia:Rome rdfs:label ?label .
}
```

Inverse Property Selection: Linked Data is represented by a directed graph. Therefore, it is possible that a resource is not aware of all connections that exist between itself and other resources. In order to obtain properties of resources that are linked unidirectionally to the initial resource, it is possible to invert the property selection by a prepended caret symbol (^). For example, the DBpedia database does not contain any links between a city and the people who were born in it. But it contains connections between people and their places of birth. To request a list of people who were born in Rome, the property selection must be inverted. The following query requests the labels of all individuals whose birthplace is Rome.

```
rome.^dbpedia-owl:birthPlace.rdfs:label
```

```
SELECT DISTINCT
  (dbpedia:Rome AS ?uri) ?birthPlace ?label
WHERE {
  ?birthPlace dbpedia-owl:birthPlace dbpedia:Rome .
  ?birthPlace rdfs:label ?label .
}
```

Property Filter: Object properties can be filtered by values of linked properties. The left-hand side of a filter expression specifies a property that references to the previously defined object property or resource in the query path. The rest corresponds to the usual SPARQL syntax for filter expressions. As a multilingual database DBpedia contains the name of Rome in different languages (Rome, Roma, Rom, 罗马 ...). One of the most common actions when querying multilingual texts for displaying purposes is consequently the filtering for a specific language. Therefore, we have a simple language filter integrated directly into our query language. The language filter is prefaced with the keyword `@lang`. The following query contains two filters and requests the English label of all soccer players who were born in Rome.

```
rome.^dbpedia-owl:birthPlace(rdf:type = dbpedia-owl:SoccerPlayer).
  rdfs:label(@lang = 'en')
```

```
SELECT DISTINCT
  (dbpedia:Rome AS ?uri) ?birthPlace ?label
WHERE {
  ?birthPlace dbpedia-owl:birthPlace dbpedia:Rome .
  ?birthPlace rdf:type ?filtertype .
  Filter (?filtertype = dbpedia-owl:SoccerPlayer) .
  ?birthPlace rdfs:label ?label .
  Filter (lang(?label) = '' || langMatches(lang(?label), 'en')) .
}
```

Multiple Property Selection: In some cases, certain properties are inseparable. Hence it is possible to select them simultaneously in a single query. Selection of multiple properties is only allowed as last part of a SemwidgQL query.

```
rome.[geo:lat, geo:long]
```

```
SELECT DISTINCT
  (dbpedia:Rome AS ?uri) ?lat ?long
WHERE {
  dbpedia:Rome geo:lat ?lat ;
               geo:long ?long .
}
```

The complete definition of SemwidgQL's syntax is avaliable online[15].

Semantic Widget Library

SemwidgJS's widget library covers simple widgets such as widgets for text display (SwLabel), lists (SwSimpleList) and Web links (SwLink), used only for presentation of semantic data, but also interactive elements for user input and manipulation of other widgets. Furthermore, complex elements exist for displaying charts and maps that can base on external widget libraries, for example Leaflet[16] to embed OpenStreetMap material.

Widgets are declared within a website's HTML code. A widget's properties are defined by HTML5 data-* attributes of standard HTML elements. While `div` elements should be the default host element of a widget, all other elements such as `h1` or `p` are also possible, as long as the element supports the widget's HTML code. The HTML code of a widgets is directly injected into its host element and therefore inherits its style properties.

SemwidgJS distinguishes between two classes of elements: Widgets (data-sw-widget) and configuration elements (data-sw-config). All elements are identified by the unique value of its ID attribute (data-swp-id). Configuration elements, just like widgets, are declared directly within the HTML code. There are five configuration elements for 1) resources, 2) endpoints, 3) namespaces, 4) proxies, and 5) mappings.

1 – 4 are used to configure semantic endpoint connections and the corresponding query

[15] **http://semwidg.org/ql/ebnf/**
[16] **http://leafletjs.com/**

in SPARQL or SemwidgQL format. An endpoint element stores, among others, the URL under which the SPARQL endpoint is accessible. A resource element stores a URI, which identifies a resource in a semantic database and the ID of an endpoint from which the data should be queried. When using SemwidgQL it is not necessary to enter the often quite long resource URI and SPARQL endpoint URL for every query but only to enter the ID of a resource element that is linked to an endpoint element.

The mapping element is used to replace data within an attribute of a widget. For example, a search string entered into a text input widget can change the value of a mapping element. Afterward all other widget attributes connected to the mapping element change accordingly.

In the further course, we present some example widgets and demonstrate the interoperation of the widget library and the SemwidgQL query language.

Configuration: Below a minimal example configuration for Semwidg JS is presented. First, the connection to the DBpedia SPARQL endpoint is configured. Second, the resource for the city of Rome is configured using its URI, and then linked to the previously created endpoint "dbpedia". The order and exact position within the HTML code is irrelevant for configuration elements. At the start SemwidgJS searches the entire HTML document and processes all elements in the required order.

```
<div data-sw-config="endpoint"
  data-swp-id="dbpedia"
  data-swp-url="http://dbpedia.org/sparql">
</div>

<div data-sw-config="resource"
  data-swp-id="city"
  data-swp-uri="http://dbpedia.org/resource/Rome"
  data-swp-endpoint="dbpedia">
</div>
```

Single Value Widgets: The configuration of widgets takes place in the same way as for configuration elements. Below a simple label widget is embedded into an `h1` element of a website to act as a heading. For this purpose the widget type (SwLabel) and a query must be specified. The widget is rendered directly within the `h1` element. In contrast to the configuration elements, the position is relevant. SemwidgQL queries are enclosed in single curly braces, while SPARQL queries are enclosed in double curly braces.

```
<h1 data-sw-widget="SwLabel"
  data-swp-value="{city.rdfs:label(@lang = 'en')}">
</h1>
```

Multiple Value Widgets: The link widget (SwLink) accepts several values in order to create a hyperlink. Simple text and queries – whether SemwidgQL or SPARQL – can be combined with each other.

```
<div data-sw-widget="SwLink"
  data-swp-value="{city.foaf:homepage}"
  data-swp-label="Website of {city.rdfs:label(@lang = 'en')}"
```

```
  data-swp-target="_blank">
</div>
```

Interactive Elements: Interactive elements such as text input fields are able to manipulate configuration elements and other widgets. For example, interactive elements can be used to change the URI of a resource element. Thus, the website can be built generically for all cities inside the database and the end users can select the city to which they wish to receive information. Following input widget (SwInputText) is connected with the previously defined resource element "city" and its attribute "(data-swp-) uri". Once the user confirms the input, the text is written to the corresponding attribute and the internal event system ensures that all related elements will be updated.

```
<div data-sw-widget="SwInputText"
  data-swp-id="resourceUriInput"
  data-swp-target="city.uri">
</div>
```

Mapping: In addition, another configuration element exists that can be used for the further generalization of widgets. The mapping element stores an arbitrary string that replaces the corresponding placeholders in attribute values. Again, the internal event system ensures that all related elements will be updated, after the value of a mapping element changed.

```
<div data-sw-config="mapping"
  data-swp-id="lang"
  data-swp-value="en">
</div>

<h3 data-sw-widget="SwLabel"
  data-swp-value="{city.rdfs:label(@lang = '$$lang$$')}">
</h3>
```

Example: Figure 1 show the composition of several Semantic Widgets in a website some of which have been described in the previous subsections. A complete running example plus its source code is available online[17].

Semantic Widget Framework

SemwidgJS does not only serve as a widget library, but also as a framework for building custom widgets. In most cases, widget authors only need to define the properties a widget should support (in addition to the standard properties inherited from the base widget class) and override the function that generates the HTML code. Getter and setter function for these properties are created automatically by the framework and contain all necessary event listener and dispatcher for proper cross-widget interoperation. Additional getter and setter functions for explicit SPARQL endpoint definition, selector functions, and query

[17] **http://semwidg.org/js/**

Figure 1: Page composed of several Semantic Widgets

results are also created automatically according to predetermined naming conventions. These functions can be utilized by widget authors at any time.

Although JavaScript can react to DOM events of a website, it does not have its own event system that can be utilized directly. Therefore, we implemented a simple event model that allows widgets to react to events of other objects, or conversely fire own events.

The automatic generation of functions ensures that all required events are fired (e.g. when setting a SPARQL result) and in return received by the respective controllers that are responsible for managing a widget's lifecycle. If necessary, these functions can be overwritten within the widget's code, for example, to check values before setting them or inject custom events.

All predefined SemwidgJS widgets are also constructed in this way. When defining custom properties, in addition to the property name the type and a property description is specified. Possible types are "rigid", "query" and "selector". The value of a rigid property is considered to be a simple text and will not be further processed by SemwidgJS, while query properties can contain simple text, a SemwidgQL query and a SPARQL query. Selector properties reference another SemwidgJS element by its ID and property name (id.property). This information should be used in the future by a widget editor to provide users information about these properties but also to be able to make assistance such as auto-complete/-suggest.

Discussion

The ongoing growth of the Semantic Web and Linked Open Data endpoints poses the problem of enabling non-expert users to interact with and make use of semantic data in an intuitive, yet flexible and powerful manner. Although there are already numerous tools and libraries that enable the use of semantic data within own websites, these tools and libraries are often cumbersome or highly limited in their functionality and reusability.

In this paper we have presented SemwidgJS, a library for Semantic Widgets that is easy to

extend and supports its own simple query language. With SemwidgJS, web design skills and fundamental knowledge of the Semantic Web are already sufficient to take advantage of Linked Open Data in own websites. Previous solutions required typically three different kind of roles or skills. RDF or SPARQL skills to query the data, programming skill to parse and process this data, and web design skill to integrate this processed data into the website. SemwidgJS supports the entire range of Web UI elements, from simple text output, through interactive input elements to complex charts and maps.

When building Semantic Widgets with SemwidgJS, the integration of semantic data can be defined on the instance level. A Semantic Web resource (instance) is defined explicitly and data is queried for this resource, for example the city of Rome. Included mapping and templating mechanisms also allow schema level integration.

According to a categorization of Semantic Web application [MA10] websites and Web applications created with SemwidgJS can presumably be categorized as visualization oriented. They are using extrinsic retrieval, are having a consuming information flow direction, and are semantically integrated.

Currently, we are planing an editor for the creation of complex templates or even complete Web applications based on SemwidgJS. The final result could then be exported as HTML code and be embedded into an existing website, a technique which is already known from many other websites and services (e.g. the embedding of Youtube videos[18]).

For users, it should be possible to drag and drop selected widgets into their workspace and configure them. Due to the explicit description of each widget's properties and the linear structure of the SemwidgQL query language, an editor can provide ideal support to the users in the creation and integration of *Semantic Widgets*.

References

[ABK+07] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A Nucleus for a Web of Open Data. In *Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference*, ISWC'07/ASWC'07, pages 722–735, Berlin, Heidelberg, 2007. Springer-Verlag.

[ADD10] Sören Auer, Raphael Doehring, and Sebastian Dietzold. LESS - Template-Based Syndication and Presentation of Linked Data. In Lora Aroyo, Grigoris Antoniou, Eero Hyvönen, Annette ten Teije, Heiner Stuckenschmidt, Liliana Cabral, and Tania Tudorache, editors, *ESWC (2)*, volume 6089 of *Lecture Notes in Computer Science*, pages 211–224. Springer, 2010.

[BLCC+06] Tim Berners-Lee, Yuhsin Chen, Lydia Chilton, Dan Connolly, Ruth Dhanaraj, James Hollenbach, Adam Lerer, and David Sheets. Tabulator: Exploring and analyzing linked data on the semantic web. In *In Proceedings of the 3rd International Semantic Web User Interaction Workshop*, 2006.

[HHL+09] Philipp Heim, Sebastian Hellmann, Jens Lehmann, Steffen Lohmann, and Timo Stegemann. RelFinder: Revealing Relationships in RDF Knowledge Bases. In *Proceed-*

[18] **https://developers.google.com/youtube/iframe_api_reference**

ings of the 4th International Conference on Semantic and Digital Media Technologies (SAMT 2009), pages 182–187, Berlin/Heidelberg, 2009. Springer.

[HKM07] David F. Huynh, David R. Karger, and Robert C. Miller. Exhibit: Lightweight Structured Data Publishing. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 737–746, New York, NY, USA, 2007. ACM.

[MA10] Michael Martin and Sören Auer. Categorisation of Semantic Web Applications. In *proceedings of the 4th International Conference on Advances in Semantic Processing (SEMAPRO2010) 25 October – 30 October, Florence, Italy*, October 2010.

[Now09] Benjamin Nowack. Paggr: Linked Data widgets and dashboards. *J. Web Sem.*, 7(4):272–277, 2009.

[PBKL06] Emmanuel Pietriga, Chris Bizer, David Karger, and Ryan Lee. Fresnel - A Browser-Independent Presentation Vocabulary for RDF. In *Lecture Notes in Computer Science (LNCS 4273), Proceedings of the 5th International Semantic Web Conference (ISWC 2006)*, pages 158–171. Springer, November 2006.

[Ray12] Jem Rayfield. Sports Refresh: Dynamic Semantic Publishing. Website, April 2012. Available online at **`http://www.bbc.co.uk/blogs/bbcinternet/2012/04/sports_dynamic_semantic.html`**; visited on March 24th 2014.

[SBK+12] Sebastian Schaffert, Christoph Bauer, Thomas Kurz, Fabian Dorschel, Dietmar Glachs, and Manuel Fernandez. The Linked Media Framework: Integrating and Interlinking Enterprise Media Content and Data. In *Proceedings of the 8th International Conference on Semantic Systems*, I-SEMANTICS '12, pages 25–32, New York, NY, USA, 2012. ACM.

[Sin12] Amit Singhal. Introducing the Knowledge Graph: things, not strings. Website, May 2012. Available online at **`http://googleblog.blogspot.de/2012/05/introducing-knowledge-graph-things-not.html`**; visited on March 26th 2014.

[Skj12] Martin G. Skjæveland. Sgvizler: A JavaScript Wrapper for Easy Visualization of SPARQL Result Sets. In *9th Extended Semantic Web Conference (ESWC2012)*, May 2012.