LESS - Template-Based Syndication and Presentation of Linked Data

Sören Auer[1], Raphael Doehring[2], and Sebastian Dietzold[1]

[1] Universität Leipzig, Institut für Informatik,
Postfach 100920, D-04009 Leipzig, Germany,
{auer|dietzold}@informatik.uni-leipzig.de
http://aksw.org
[2] Netresearch GmbH & Co. KG,
Nonnenstrasse 11d, D-04229 Leipzig,
raphael.doehring@netresearch.de
http://netresearch.de

Abstract. Recently, the publishing of structured, semantic information as linked data has gained quite some momentum. For ordinary users on the Internet, however, this information is not yet very visible and (re-) usable. With LESS we present an *end-to-end approach* for the syndication and use of linked data based on the definition of templates for linked data resources and SPARQL query results. Such syndication templates are edited, published and shared by using a collaborative Web platform. Templates for common types of entities can then be combined with specific, linked data resources or SPARQL query results and integrated into a wide range of applications, such as personal homepages, blogs/wikis, mobile widgets etc. In order to *improve reliability and performance* of linked data, LESS caches versions either for a certain time span or for the case of inaccessibility of the original source. LESS supports the integration of information from various sources as well as any text-based output formats. This allows not only to generate HTML, but also diagrams, RSS feeds or even complete data mashups without any programming involved.

1 Introduction

Recently, the publishing of structured, semantic information as linked data has gained much momentum. A large number of linked data providers meanwhile publishes more than 200 interlinked datasets amounting to 13 billion facts[1]. Despite this initial success, there are a number of significant obstacles, which hinder the large-scale deployment and use of the linked data web. These obstacles are primarily related to the *quality and coherence* of linked data as well as to providing *direct benefits to end users*. In particular for ordinary users of the Internet, linked data is not yet sufficiently visible and (re-) usable.

[1] http://esw.w3.org/topic/TaskForces/CommunityProjects/LinkingOpenData/DataSets/Statistics

With the template-based syndication and presentation approach LESS presented in this article, we target primarily the *usability* aspect of linked data for end users. Another important problem of the linked data web tackled by LESS are quality issues, in particular with regard to *performance* and *reliability* of linked data endpoints, the importance of which has, for example, been noted earlier in [4].

LESS represents an end-to-end approach for the syndication and use of linked data based on the definition of templates for the visual presentation of linked data resources and SPARQL query results. LESS allows to edit and publish syndication templates and to share them by using a collaborative Web platform. Templates for common types of entities can then be combined with specific, linked data resources or SPARQL query results and integrated into a wide range of applications, such as personal homepages, blogs/wikis, mobile widgets etc.

As a result, a blogger writing about a recent trip to Berlin can easily integrate a nicely formatted fact box with important information about Berlin obtained from Wikipedia into her blog post. A community of science fiction fans can integrate lists on a recent BBC programming matching their preferences into their community portal. Wikipedia authors can use LESS to generate pages with lists from DBpedia [6] content. Citizen scientists interested in earthquakes can display recent earth crust activity in their region on a map without having to do any programming.

LESS supports the integration of information from various sources as well as any text-based output formats. This allows not only to generate HTML, but also diagrams, RSS feeds or even complete data mashups without any programming involved. In order to improve reliability and performance of linked data, LESS caches versions of the retrieved linked data resource descriptions or SPARQL query results either for a certain time span (thus improving the performance) or for the case of inaccessibility of the original source (thus improving reliability).

LESS represents an *end-to-end solution* by not only focusing on a single aspect, but tackling the whole life-cycle from template definition, processing, integration, authoring to sharing in a coherent way. Concrete usage scenarios tackled by LESS include for example:

- the flexible visualization of linked data resources,
- the creation of views on the linked data web,
- the integration of linked data into existing applications such as Content Management Systems, Wikis, Weblogs etc.
- the integration and compilation of information from various sources,
- the end-user-driven generation of linked data mashups.

The paper is structured as follows: We present the high-level LESS concept and architecture in Section 2. We report about our LESS implementation in Section 3. We present a number of complementary use cases for LESS in Section 4. We review related work in Section 5 and conclude with an outlook on future work in Section 6.

2 Concept and Architecture

From the usage scenarios mentioned in the introduction (and described in more detail in Section 4) we derived the following requirements:

- support for various data access paradigms, in particular direct linked data URI requests and SPARQL queries,
- simple template language, which is, on the one hand, easy to learn and, on the other hand, flexible enough to accommodate a wide range of usage scenarios (in particular the ones described in the introduction),
- mitigation of the current reliability and performance problems of linked data endpoints,
- facile integration of the processed templates into various Web applications, e.g. via a REST API,
- enable the collaborative authoring, sharing and re-use of templates.

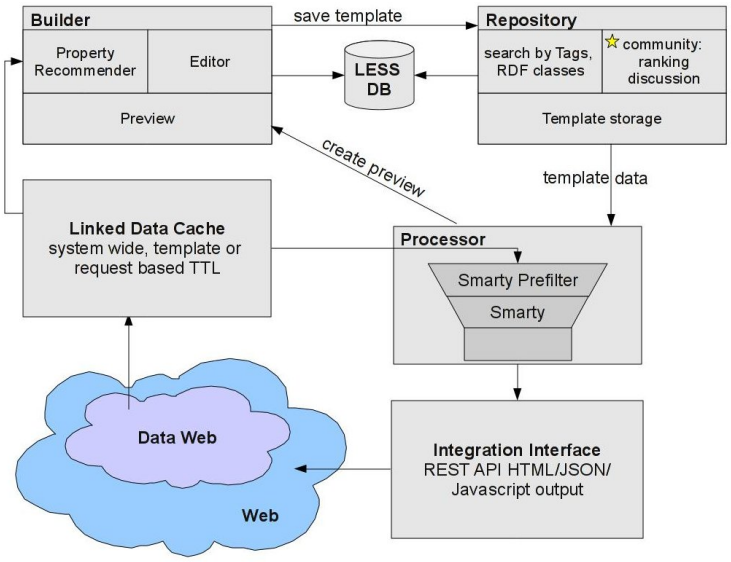


Fig. 1. LESS system architecture.

In order to fulfill these requirements, we developed the LESS architecture as presented in Figure 1. The main LESS components are:

- *template language:* defines a declarative language for creating textual output form RDF resources and SPARQL results,
- *template builder:* allows users to define and edit LESS templates in a collaborative manner,

– *repository:* stores template revisions and metadata, allows to retrieve templates based on various annotations,
– *processor:* combines a template with concrete data retrieved from the data web,
– *linked data cache:* stores retrieved linked data resources and SPARQL queries for reuse.

In the following subsections we describe these components in more depth.

2.1 Template Language

A core part of LESS is the LESS Template Language (LeTL), which allows to define arbitrary text-based output representations. In general, the use of various template languages (such as Fresnel [8] or Tal4RDF [2]) is possible with LESS. However, in order to lower the entrance barrier for end users, we decided to specify a template language tailored for simplicity, convenience and versatility. LeTL is based on the popular Smarty[2] template language, but uses some custom extensions. From Smarty LeTL inherits custom functions, variable modifiers, loops, conditional parts based on the evaluation of complex expressions, cache handling, a plugin architecture and many other features. However, LeTL additionally comprises functionality for direct interaction with RDF data and SPARQL query results.

A LeTL template is applied to each individual resource described in an RDF document or every row in a SPARQL result set (unless the template application is restricted to resources of a certain class). The properties (or columns) of the resource (or SPARQL result row) are made available for reference within the LeTL template by using the syntax showcased in Table 1. The LeTL syntax allows to iterate through multiple property values and to reference recursively (sub-)templates, which are applied to dereferenced object property values. An example of such a recursive template, which iterates through a list of `foaf:Person` resources, dereferences these resources and calls another template for each one of them, is shown in Figure 2.

2.2 Template Builder

The template builder (depicted in Figure 3) enables end users to create LESS templates. The template builder comprises the template editor, a property recommender and a template rendering preview. The template editor supports syntax highlighting for HTML. The property recommender suggests properties based on a given, example linked data resource or SPARQL query. Once a certain property has been selected, the corresponding Smarty code to display the value of the property is added to the current cursor position in the template editor. Once a stable version of a certain template has been created by using the template builder, additional metadata (such as a template name and tags) can be attached to the template and a revision can be stored in the template repository.

[2] `http://www.smarty.net/`

Syntax	Description	Example
`{{property}}`	Refers to the value of the property 'property'	`{{foaf:name}}`
`{{p@lang}}`	Refers to the value with language tag 'lang' of the property 'p'	`{{rdfs:comment@en}}`
`{{p^^dtype}}`	Refers to the value with datatype 'dtype' of the property 'p'	`{{dbp:birth^^xsd:date}}`
`{{p1->p2}}`	Refers to the value of the property 'p2' of the resources referred to by the property 'p1'	`{{foaf:currentProject ->rdfs:label}}`
`{template id="id" uri="uri" instances="" sortBy=""}`	Processes the template with id 'id' and resource 'uri' (using the optional attributes 'instances' iteration can be restricted and sorted with 'sortBy')	`{template id="5" uri="{$var}"}`
`{template id="id" sparql="query" endpoint="srv"}`	Processes the template with id 'id' and SPARQL query 'query' from endpoint 'srv'	`{template id="3" sparql="SELECT * WHERE {}" endpoint="http://dbpedia.org"}`

Table 1. LeTL syntax extensions to the Smarty template language.

2.3 Template Repository

The template repository allows to publish templates, to browse existing templates based on their supported RDF classes and user-defined tags as well as to rate, comment and reuse existing templates. The template repository is implemented on top of a relational database, but made available as RDF by using Triplify [1] at `http://less.aksw.org/triplify`. For each template, a unique template id is assigned. As soon as a template is changed, its revision id is incremented. Each registered LESS user can only change her own templates in order to prevent conflicts. However, a user can create her own copy of any of the templates stored in the template repository. The public availability of templates in the repository has a number of advantages: templates serve as examples for new users, they can be used by other third-party applications and the reuse of templates facilitates a natural modularization.

2.4 Template Processor

The template processor is the actual LESS execution environment. It takes a LESS template and a linked data resource or SPARQL query and renders the respective output. By default the LESS template processor iterates through all the resource descriptions or SPARQL query result items and applies the defined template to each of them. This allows to apply LESS also to RDF documents, which contain more than one resource description. However, in certain cases, the template application should be restricted to resources of a certain type. For these cases, the LESS template can be associated with a certain RDF class. This can be, for example, the class `foaf:Person` for a FOAF profile, thus preventing the

```
<h2>{{foaf:name}}</h2>
{{dc:description}} <br />
<a href="{{foaf:homepage}}">web</a>
<div id="members">
 {foreach {{foaf:member}} as $var}
  {template id="5" uri="{$var}"
    parameter_language="en"}
 {/foreach}
</div>
```

```
<div id="foaf-card-block">
 {if {{foaf:depiction}} !== ''}
  <img class='photo'
    src="{{foaf:depiction}}" />
 {/if}
 <div id="name">{{foaf:name}}</div>
 ...
 {if {{foaf:phone}} !== ''}
  <div id="tel">{{foaf:phone}}</div>
 {/if}
 <div id="email">
  <a href="{{foaf:mbox}}">
   <img src="email.png" />
   {if $language === 'en'}
     email {else} E-Mail
   {/if}
  </a>
 </div>
</div>
```

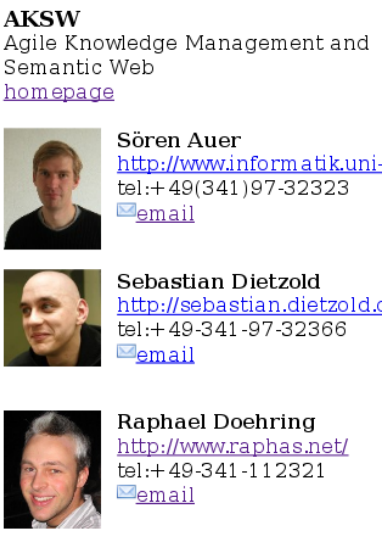


Fig. 2. Left: LESS templates for displaying a group profile including information about group members, obtained from separate FOAF profiles; Right: rendered output

application of the template for a person's projects described in the same FOAF file. When there is more than one resource of the type selected, the `sortBy` parameter can be used to order the resources based on the values of a certain property. The template processing can be additionally influenced by user-defined parameters, which can be accessed from within the template definition. Based on a system, template or request configuration, the template processor can be instructed to use a locally cached version of the linked data resource or SPARQL query. This particularly facilitates situations, where linked data endpoints are temporarily not available or respond slowly.

2.5 Integration Interface

In order to integrate the output of the template processor into external applications, LESS provides an REST API via the URL `http://less.aksw.org/build`. Required URL parameters are the `id` of the template, the `revision` of the template (can be omitted for accessing the last revision) and either `uri` or `sparql` for defining the linked data resource or the SPARQL query to be used. By default this REST function call returns the rendered output as text (in most cases HTML snippets). By using the optional URL parameter `output`, Javascript or JSON can be alternatively selected as output format. A further optional URL

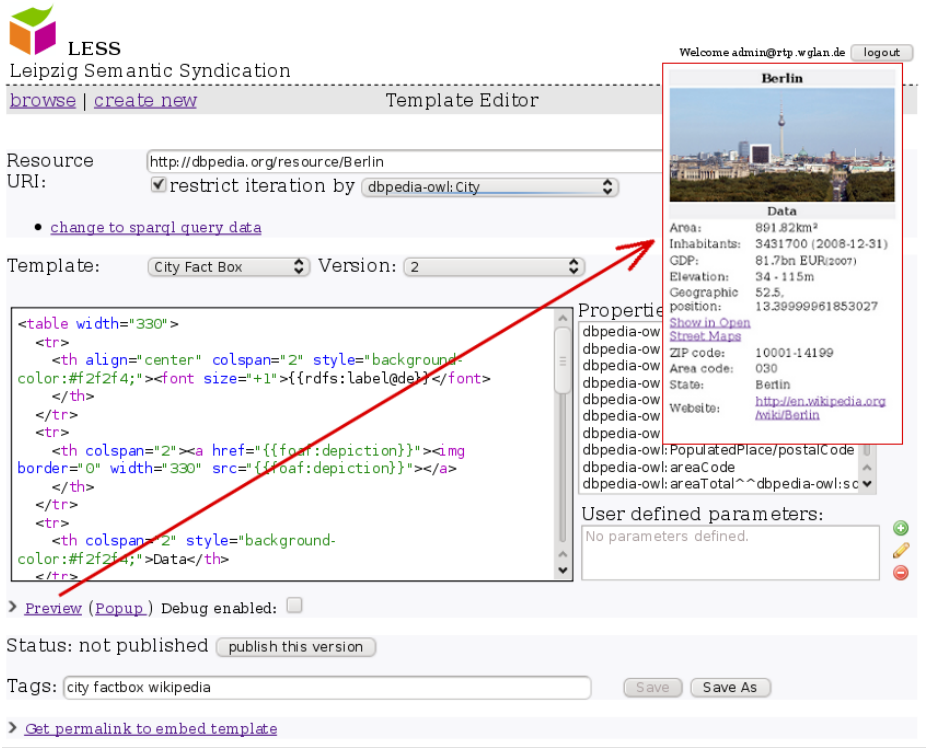


Fig. 3. Template Builder comprising editor, property recommender and rendering preview.

parameter is `ttl`, which specifies the time-to-live of a previously cached version of the linked data resource or SPARQL query. In case this parameter is missing, template- or system-based defaults are used. An example of a REST request (whose result is depicted in Figure 5) would look as follows:

```
http://less.aksw.org/build?id=2&
                    uri=http%3A%2F%2Fdbpedia.org%2Fresource%2FBerlin
```

3 Implementation

The LESS implementation was performed in PHP by using the Zend Framework[3] and the Erfurt framework for the development of semantic web applications [3]. The implementation follows the generic MVC architecture model and the convention-over-configuration paradigm, which assumes reasonable defaults

[3] `http://framework.zend.com`

for all possible configuration parameters. A demonstration platform with the LESS implementation is available at: `http://less.aksw.org`

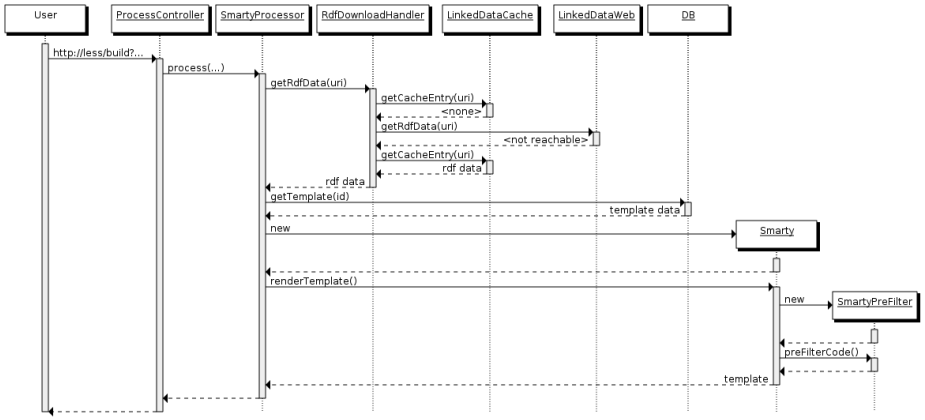


Fig. 4. Template processing UML sequence diagram.

A UML sequence diagram illustrating the template processing is displayed in Figure 4. After a template instance is requested by a user via the integration interface, the template processor checks the local cache or retrieves the data. Based on the template id, the template data and metadata are obtained from the relational database. The LeTL template is then compiled by the `SmartyPreFilter` into a valid Smarty template and compiled by the Smarty processor into PHP code. The Smarty processor also takes care of caching-compiled templates in order to increase the performance.

4 Usage Scenarios

In this section we present examples for employing LESS for the various usage scenarios identified in the introduction. The examples presented are available in the LESS template repository with tag ESWC[4].

4.1 Flexible Resource Visualization

Although there are some approaches for rendering RDF (e.g. [8, 2]), it is currently quite cumbersome to visualize RDF and linked data in user-defined ways. LESS provides a very flexible and easy-to-learn mechanism for visualizing linked data resources and SPARQL query results, since end users can directly create output fragments with placeholders for RDF data. Figure 5 demonstrates how a LESS

[4] `http://less.aksw.org/browse?tags=\%2Beswc`

template can be used to visualize a linked data resource and to integrate the resulting output into a Wordpress blog. By annotating the generated visual representations with RDFa, the resource descriptions can be easily obtained and reused not only from the original data source, but also from the syndicated content representations.

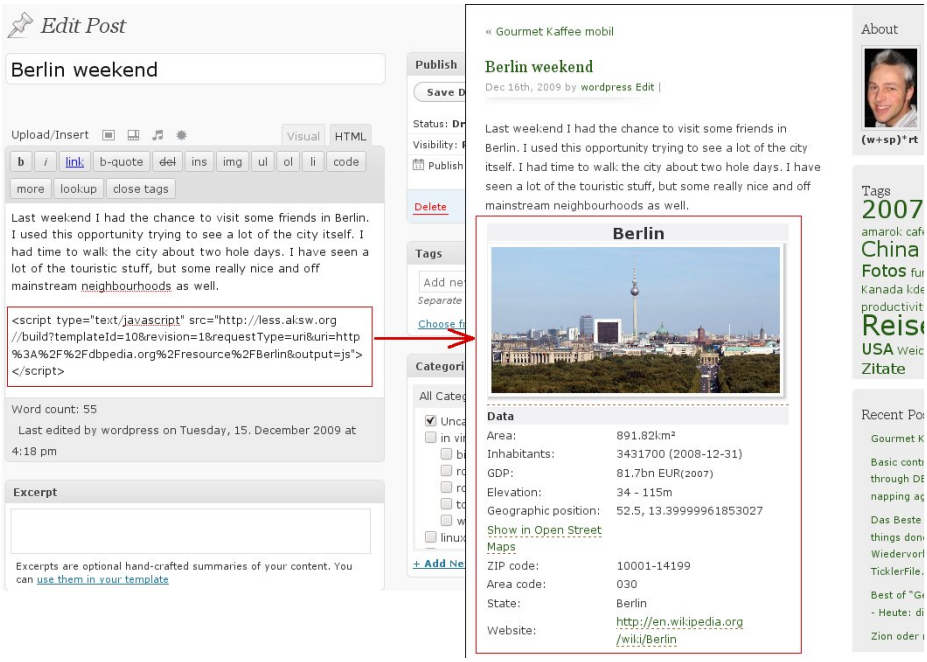


Fig. 5. Integration of a LESS template visualizing data obtained from DBpedia into a Wordpress blog post by using a Javascript snippet.

4.2 Linked Data View Creation and Visualization

In many cases not only a single linked data resource, but information from various interlinked linked data resources or SPARQL queries should be displayed. Figure 6 showcases a rendered template, which obtains information about recent BBC programmings related to the series 'Mountain'. The information is obtained by querying a SPARQL endpoint containing the BBC program information and rendering the query result by employing a LESS template.

4.3 Integration of Information from Various Sources

LESS is not limited to create visual representations of single RDF or SPARQL sources. The possibility to dereference linked data resources or call (sub-)templates

Series: Mountain

Griff Rhys Jones explores the great mountain ranges of Britain, from Scotland southwards. Next show times on BBC One:

- **2009-05-30** 20:00 - 21:00 - Northern Scotland
  Griff Rhys Jones explores the wilderness of the northwest highlands of Scotland.
- **2009-09-25** 20:00 - 21:00 - Northern Scotland
  Griff Rhys Jones explores the wilderness of the northwest highlands of Scotland.
- **2009-10-02** 22:00 - 23:00 - The Lakes
  Griff visits the Lake District.
- **2009-10-02** 20:00 - 21:00 - The Lakes
  Griff visits the Lake District.

Fig. 6. Rendered LESS template representing a view on recent BBC programmings.

from within a LESS template (as described in Table 1) allows to integrate information from various sources into a coherent visual representation. In Figure 2 we illustrate this usage scenario with a template, which iterates through members of a group obtained from a group FOAF file and processes an individual FOAF template for each member with data from their personal FOAF profiles.

4.4 Template Integration into Existing Applications

With LESS' REST style integration interface it is very easy to integrate rendered templates into existing Web applications, be it Weblogs, CMS, Wikis, traditional Web pages or any other Web application. LESS offers the template integration into existing application via employing HTML snippets, Javascripts, JSON or IFRAMEs. The integration of linked data content can be even further simplified, if Web applications offer to integrate LESS templates directly, i.e. without the detour to the LESS homepage. In order to showcase how such a LESS integration can be performed, we developed a Typo3 extension, which allows to define Typo3 page content objects based on LESS templates directly from within the Typo3 user interface (cf. Figure 7).

4.5 Mashups

LESS is not limited to employing RDF and solely creating HTML output. Due to the flexible template language, arbitrary text-based output can also be generated, including Javascript, various XML formats (such as RSS), CSV etc. In particular, LESS enables the combination of RDF data with WebAPIs in order to create data mashups. Figure 8 shows a LESS template processing data from Eurostats and combining it with the Google Charts API in order to create a chart with births per year over time. The Google Charts API is accessed by creating a special URL, which returns the generated chart as an image. In the example the URL is constructed by means of a LESS template. Using LESS, users do not have to perform any programing in order to construct the URL for accessing the API. Likewise, the LESS output (i.e. the diagram in this example) dynamically changes as the underlying data changes.

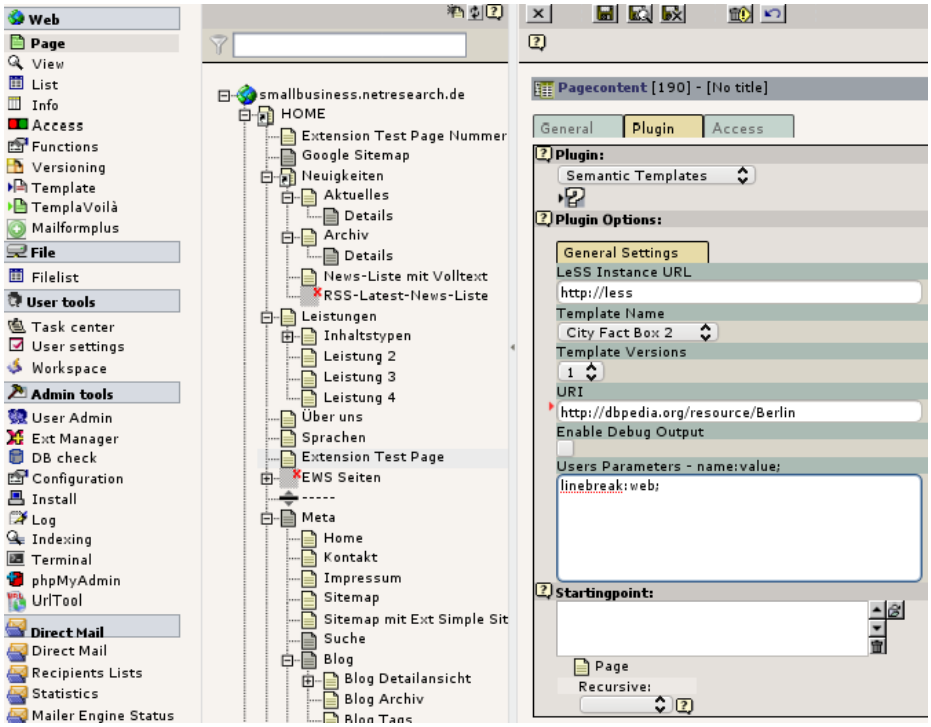


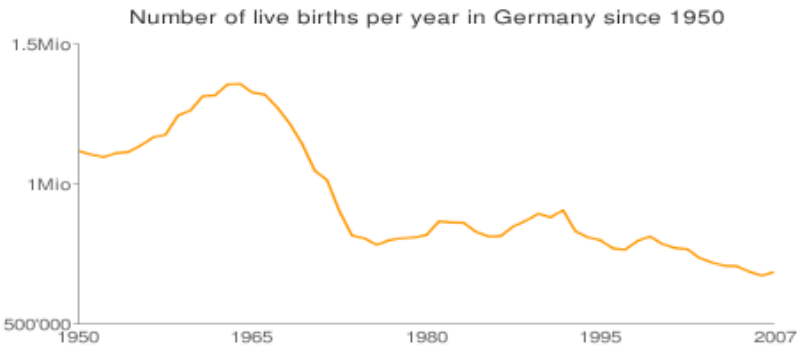
Fig. 7. LESS integration into Typo3: After specifying the LESS instance to use, the user is provided with a list of possible templates. The user is then able to specify the necessary template parameters through a user interface rather than by manually creating the REST URI. This simplifies integrating a LESS template into Typo3 even further.

5 Related Work

Related work dealing with the visual presentation and (re-)use of RDF data can be divided into two main groups. On the one hand, there are interactive, user-interface-centric approaches, which enable the user to select and combine data interactively. The entrance barrier of these approaches is relatively low, since user interfaces are intuitive and easy to understand. On the other hand, there are programmatic-oriented approaches that specify a transformation or template language for RDF data. Although more effort is necessary to get acquainted with them, the latter provide more versatility and flexibility.

Sig.ma[5] is part of the first group and offers a fixed, table-oriented presentation of linked data. The user is enabled to select a certain data source, to edit the ordering of data and to integrate the result into an existing web page. Apart from the order the user does not have any influence on the layout of the information.

[5] http://sig.ma/




```
<img src="http://chart.apis.google.com/chart?chs=450x250&chd=t:
{$data_de}&cht=lc&chds=500000,1500000&chtt=Number+of+live+births
+per+year+in+Germany+since+1950&chxt=x,y&chxl=0:|1950|1965|1980|
1995|2007|1:|500'000|1Mio|1.5Mio">
```

Fig. 8. Mashup template combining the Google Charts API with Eurostats data. The code in the lower part shows the image link template used to access the API.

Marbles[6] is also part of the user-interface-centric group of approaches. Marbles is a web application that allows to aggregate and present RDF data from different sources by using Fresnel *lenses* and *formats* (see also below). Its focus lies on the aggregation and selection of new sources for an RDF resource as well as the selection of data. Marbles is limited to the presentation of data for XHTML clients through Fresnel-based views. There are only three views currently in use. The produced format is limited to the views.

With regard to the first group of related user-interface-centric approaches there is also a large body of work into mashups. This includes Yahoo! Pipes[7] or the (now discontinued) Google Mashup editor. Semantic Pipes [7] is a mashup technology which can make use of Linked Data. However, its focus lies more on the choreography of processing information from different sources, than on provisioning of output for direct integration into Web pages.

With regard to the second group of programmatic-oriented approaches, Fresnel [8] offers a very generic way to solve the problem of presentation and transformation of RDF data. Fresnel is a vocabulary for describing the basic concepts of RDF data presentation. It offers two main components: *lenses* and *formats*. While *lenses* are a filter to select the data to use, *formats* describe the way a set of data is presented. The result of a Fresnel transformation is not a final presentation of the data but a tree structure of data annotated with presentation-related information. The actual interpretation of this structure is delegated to the browser software using Fresnel. In addition to Marbles, other RDF browsers

[6] http://marbles.sourceforge.net/
[7] `http://pipes.yahoo.com`

implementing Fresnel are IsaViz[8] (W3C/INRIA) and Longwell[9] / Piggy Bank [5]. Although very related, Fresnel is quite different from LESS, since it defines its own mechanisms for data selection (LESS uses simple projections and SPARQL), its formatting approach is more difficult to learn (due to the RDF representation of formatters) and it is less aligned with the linked data paradigm.

Tal4RDF [2] is a programming library written in Python based on Zope's Template Attribute Language. The goal of this approach is to create a light-weight template language for RDF data. The library allows to transform RDF data into any text-based format. Similar to LESS, a web service to render the templates is provided. However, Tal4RDF does not offer any means to publish and share templates, there is no support for SPARQL and dynamic linked data dereferencing.

The Visualization Providers for Ontology ElemenTs (VPOET) is a tool developed for the presentation of ontology elements [10]. It is based on the Fortunata library [9]. The user is enabled to create a template presenting RDF data, and a web service for rendering the templates is provided. Different to LESS, VPOET is limited to the presentation of resources belonging to a single ontology. VPOET's template language appears to be slightly cumbersome (e.g. with regard to the data referencing), as it does not support loops for iterating through resources, and the conditional evaluation of template parts cannot be based on complex conditions such as the comparisons.

6 Conclusions and Future Work

With LESS we aimed at contributing to mitigate the largest obstacles for a large-scale deployment of the linked data web: the demonstration of direct benefits to end users and improving the reliability and performance of linked data endpoints. LESS represents an end-to-end solution by not only focusing on a single aspect, but tackling template definition, processing, authoring and sharing in a coherent and pragmatic way. As a result, end users are empowered to make use of linked data without the need to program or gain deep understanding of the technologies involved. Despite its simplicity, LESS is very flexible and versatile. It supports a wide-range of application scenarios ranging from simple resource visualization to complex data mashup generation.

We see LESS as a first step towards bringing the data web closer to potential end users. As a continuing effort, we aim, in particular, to tackle the following enhancements and improvements in future work:

- usability improvements, such as a visual template designer supporting drag-and-drop of template elements and the integration of OntoWiki's SPARQL query builder for a more user-friendly construction of complex queries,
- development of plugins for standard Web applications, which enable users to define and integrate LESS templates from a single environment,

[8] http://www.w3.org/2001/11/IsaViz/
[9] http://simile.mit.edu/longwell/

– better integration with data web services and search indexes such as Sindice,
– template language extensions, such as a template-in-template mechanism, support for property groups representing common information and support for different template languages such as Fresnel and Tal4RDF.

Acknowledgments

We would like to thank our colleague Christian Weiske for the helpful comments and inspiring discussions during the development of LESS. This work was supported by a grant from the German Federal Ministry of Education and Research (BMBF), provided for the project LE4SW.

References

1. Sören Auer, Sebastian Dietzold, Jens Lehmann, Sebastian Hellmann, and David Aumueller. Triplify: light-weight linked data publication from relational databases. In *18th International Conference on World Wide Web, WWW 2009*, pages 621–630. ACM, 2009.
2. Pierre-Antoine Champin. T4R: Lightweight presentation for the Semantic Web. In Gunnar Aastrand Grimnes Chris Bizer, Sören Auer, editor, *Scripting for the Semantic Web, workshop at ESWC 2009*, June 2009.
3. Norman Heino, Sebastian Dietzold, Michael Martin, and Sören Auer. Developing semantic web applications with the ontowiki framework. In *Networked Knowledge - Networked Media*, volume 221 of *Studies in Computational Intelligence*, pages 61–77. Springer, Berlin / Heidelberg, 2009.
4. Sebastian Hellmann, Jens Lehmann, and Sören Auer. Learning of OWL class descriptions on very large knowledge bases. *International Journal on Semantic Web and Information Systems*, 2009.
5. David Huynh, Stefano Mazzocchi, and David Karger. Piggy Bank: Experience the Semantic Web inside your web browser. In *The Semantic Web ISWC 2005*, pages 413–430. Springer Berlin / Heidelberg, March 2005.
6. Jens Lehmann, Chris Bizer, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia - a crystallization point for the web of data. *Journal of Web Semantics*, 2009.
7. Danh Le Phuoc, Axel Polleres, Manfred Hauswirth, Giovanni Tummarello, and Christian Morbidoni. Rapid prototyping of semantic mash-ups through semantic web pipes. In Juan Quemada, Gonzalo Len, Yolle S. Maarek, and Wolfgang Nejdl, editors, *WWW*, pages 581–590. ACM, 2009.
8. Emmanuel Pietriga, Christian Bizer, David Karger, and Ryan Lee. *Fresnel: A browser-independent presentation vocabulary for RDF*, volume 4273, pages 158–171. Springer Berlin / Heidelberg, 2006.
9. Mariano Rico, David Camacho, and Óscar Corcho. A contribution-based framework for the creation of semantically-enabled web applications. *Information Sciences*, 2009.
10. Mariano Rico, David Camacho, and Óscar Corcho. VPOET Templates to Handle the Presentation of Semantic Data Sources in Wikis. In *Proceedings of the Fourth Workshop on Semantic Wikis The Semantic Wiki Web*, pages 186–190, Hersonissos, Crete, Greece, 2009.