Adaptive Linked Data-driven Web Components: Building Flexible and Reusable Semantic Web Interfaces

Ali Khalili
Dept. of Computer Science
VU University Amsterdam
The Netherlands
a.khalili@vu.nl

Antonis Loizou
Dept. of Computer Science
VU University Amsterdam
The Netherlands
a.loizou@vu.nl

Frank van Harmelen
Dept. of Computer Science
VU University Amsterdam
The Netherlands
frank.van.harmelen@vu.nl

ABSTRACT

The amount of published Linked Data on the Web is increasing day by day. As a result, the applications driven by Semantic Web and Linked Data are taking momentum on the Web. One of the major entrance barriers for Web developers to contribute to this wave of Linked Data Applications (LDAs) is the required knowledge of Semantic Web technologies such as the RDF data model and SPARQL query language. This paper presents an adaptive component-based approach together with its open source implementation for creating flexible and reusable Semantic Web interfaces driven by Linked Data. Linked Data-driven (LD-R) Web components abstract the complexity of the underlying Semantic Web technologies in order to allow reuse of existing Web components in LDAs, enabling Web developers who are not experts in Semantic Web to develop interfaces that view, edit and browse Linked Data. In addition to modularity provided by the LD-R components, the proposed RDF-based configuration method allows application assemblers to reshape their user interface for different use cases, by either reusing existing shared configurations or by creating their proprietary configurations.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces; D.2.11 [**Software Engineering**]: Reusable Software: reusable libraries , reuse models

General Terms

Design, Human Factors, Performance, Standardization

1. INTRODUCTION

With the growing number of structured data published on the Web, WWW is moving towards becoming a rich ecosystem of machine-understandable Linked Data[1]. Semantically structured data facilitate a number of important aspects of information management such as information retrieval, search, visualization, customization, personalization and integration [11]. Despite all these benefits, Linked Data Applications (LDAs) are not yet adopted by the large community of Web developers outside the Semantic Web domain and, causally, by the end-users on the Web. The usage of semantic data is still quite limited and most of the currently published Linked Data are generated by a relatively small amount of publishers [6] which indicate some entrance barriers towards wide-spread utilization of Linked Data [2].

The current communication gap between Semantic Web developers and User Experience (UX) designers, driven by the need to bear Semantic Web knowledge, prevents streamline flow of best practices from UX community into the Linked Data user interfaces (UIs). The resulting lack of adoption and standardization often makes current LDAs inconsistent with user expectations and impels more development time and costs on LDA developers. In this situation, more time is spent in re-designing existing UIs rather than focusing on innovation and creation of sophisticated LDAs.

This paper presents adaptive Linked Data-driven Web components as an approach to build flexible and reusable Semantic Web UIs. *Web Components* are a set of W3C standards [5] that enable the creation of custom, reusable user interface widgets or components in Web documents and Web applications. *Resource Description Framework* (RDF), on the other hand, provides a common data model that allows data-driven components to be created, shared and integrated in a structured way across different applications. Linked Data-driven (LD-R) Web components as defined in this paper are a species of Web components that employ the RDF data model for representing their content and specification (i.e. metadata about the component). LD-R components are supported by a set of predefined core Web components, each representing a compartment of the RDF data model on the Web UI. Thus, the Semantic Web nature of an LDA can be encapsulated in LD-R components thereby, allowing UX designers and Web developers outside the Semantic Web community to contribute to LDAs. The components also provide current Semantic Web developers with a mechanism to reuse existing Web components in their LDAs. Furthermore, LD-R components exploit the power and flexibility of RDF data model in describing and sharing resources to provide a mechanism to adapt the Web interfaces based on the meaning of data and user-defined rules.

[1] `lodlaundromat.org` recently (14.10.2015) reported approx. 38.6 billion triples published on the Web.

The LD-R approach offers many benefits that we will describe in the remainder of the paper. Among them are:

Bootstrapping LDA UIs. LD-R components exploit best practices from modern Web application development to bring an exhaustive architecture to perform separation of concerns and thereby bootstrapping an LDA by only selecting a minimal relevant configuration. For example, a developer only needs to set the URL of his in-use SPARQL endpoint and start developing the viewer components without dealing with the underlying connection adapters and data flow mediators in the system.

Standardization and Reusability of LDA UIs. Instead of creating an LDA UI from the scratch, in the component-based development of LDA UIs, application assemblers choose from a set of standard UIs which will reduce the time and costs associated with creation of LDAs. For example, to render DBpedia resources of type 'Location', a standard map can be reused.

Customization and Personalization of LDA UIs. The RDF-based nature of LD-R components allow application assemblers to reshape their user interface based on the meaning of data or user context. For example, for all the resources of type `foaf:Person`, the content can be rendered with a 'ContactCard' component.

Adoption of LDA UIs by non-Semantic Web developers and end-users. Most of the current Linked Data interfaces fall into the *Pathetic Fallacy of RDF* [10] where they display RDF data to the users as a graph because the underlying data model is a graph. Abstracting the complexity of RDF and graph-based data representation provides more *Affordances* [17] for non-Semantic Web users to contribute to Linked Data UIs. Engaging more UX designers and Web developers into LDA UIs will also result in more affordances on the end-user's side to better understand the possible actions and advantages of the LDAs.

2. CONTRIBUTIONS AND OUTLINE

The contributions of this work are the concept of *Adaptive LD-R Web components* and an open source implementation of it available at `http://ld-r.org`. Our primary claim is that adopting a component-based approach that encapsulates the main concerns of a Semantic Web application, paves the way to reusing existing best practices from the UX community within the LDAs hence building more usable and pervasive LDAs. We also present that combining the LD-R components with LD-R scopes and configurations allow application assemblers to provide a high level of flexibility in their LDA UIs.

We explore this claim in stages. First, we collect some data about the current status of Semantic Web UI development. Next, we demonstrate how adaptive LD-R Web components can address the current issues in LDA UI development. Finally, we discuss the implementation of our idea and its usage in real-world scenarios.

3. THE CURRENT STATUS OF LINKED DATA USER INTERFACE DEVELOPMENT

In order to understand the current pitfalls of LDA UI design, we conducted a survey targeting active Semantic Web developers[2]. The participants where selected from the community of Semantic Web developers on Github who have had at least one active Semantic Web-related repository. Github is currently the most popular repository for open source code and its transparent environment implies a suitable basis for evaluating reuse and collaboration among developers [14, 22]. We used Github APIs[3] to search[4] for the potential Semantic Web repositories and then to collect the contact information of the corresponding contributors when available. The search, after removing the organizations and invalid email addresses, resulted in 650 potential Semantic Web developers. We then contacted the potential candidates to ask them about the current pitfalls in developing LDA UIs. In our enquiry, we clearly mentioned to skip the questionnaire if they have not developed any Semantic Web application so far. We used a minimal set of 7 questions to attract more responses and also used inline forms for GMail users to allow filling out the questionnaire in the same time as reading the enquiry email. We collected 79 responses to our questionnaire, which is a considerable number of participants (almost 12% of the potential candidates). Figure 1 shows the main results of our survey.

Participants. Based on their LDA development experience, we divided the participants into three groups: basic (less than 2 applications), intermediate (3-5 applications) and advanced (more than 5 applications) developers. The result showed that the majority (62%) of participants were intermediate and advanced developers. In addition to the development experience, developers were asked about their knowledge of Semantic Web to compare their practical and conceptual experience. As results revealed, the majority of participants (63%) had proficient (4-5 years) and expert (more than 5 years) knowledge of Semantic Web and Linked Data which makes a good sample for our evaluation.

Questions addressed the following topics:

- *Amount of time spent on bootstrapping LDA UIs.* Before designing the UIs in an LDA, developers need to spend some time on creating the skeleton of their application where querying data and the business logic of the application is handled. The results confirm that developers spend a lot of time (on average more than 2 days) on bootstrapping their LDAs before they can start working on the UI.

- *Reuse of code by Semantic Web developers.* Developers usually reuse sections of code, templates, functions, and objects to save time and resources when developing LDAs. We asked participants about two types of reuse: reuse by copy/pasting code from existing LDAs and reuse by employing current Web components. Reuse by copy/pasting code can be an indicator

[2]results are available at `https://goo.gl/cltqhv`
[3]`https://developer.github.com/v3/`
[4]keywords: "Semantic Web" OR "Linked Data" OR "RDF" OR "SPARQL"

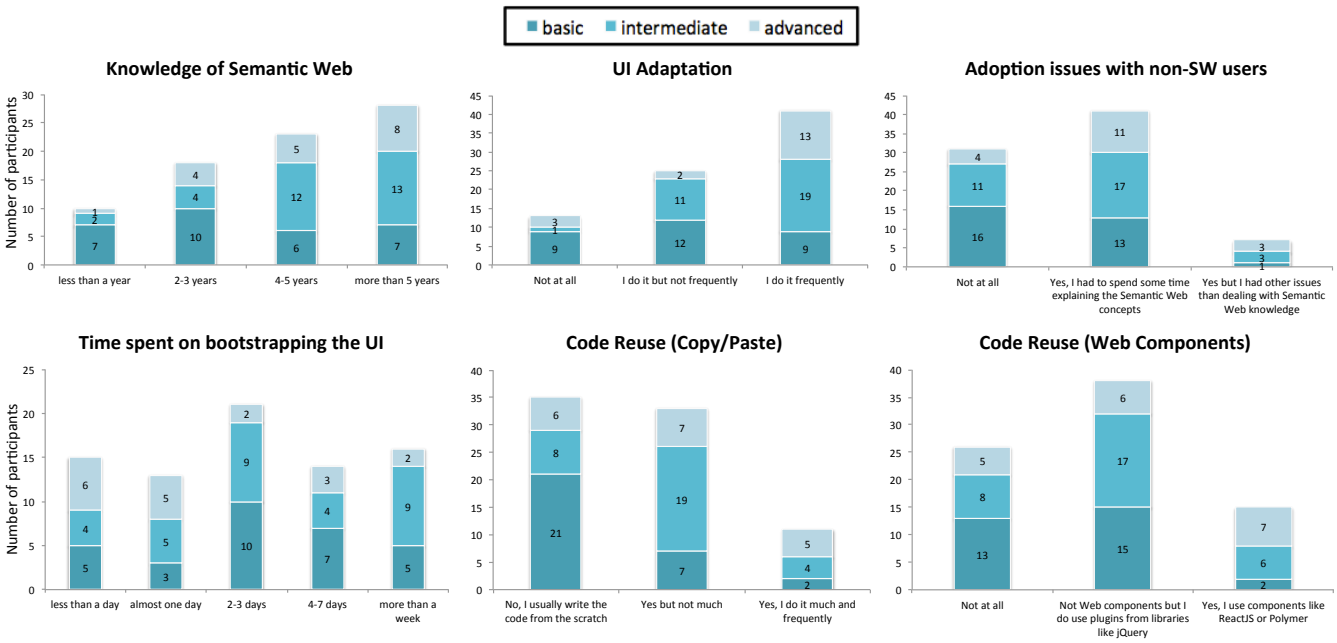

Figure 1: Results of our user study on the current status of LDA UI development.

on the standardization, modularity and reusability of current LDAs. The results indicate that a considerable amount of users (46%), prefer to write the code from the scratch instead of reusing the code from the existing Semantic Web projects. This situation is more pronounced for basic developers who still prefer to write the code from the scratch although they have less experience in programming LDAs. Furthermore, the results on reuse of Web components give an insight on the adoption of current Web Components by Semantic Web developers. The results indicate that despite the prevalence of Web Components solutions, only 19% of the participants (mainly advanced users) were employing them in their applications. Interestingly, the majority of participants (49%) were already reusing other component-like libraries which shows an attitude and capacity towards adopting the Web components.

- *Adaptation of LDA UIs.* Most of the modern Web applications provide a mechanism to customize and personalize their user interfaces based on the type of data and the information needs of their end-users. Proactive user interface adaptation allows the application to act more like a human and consequently, more intelligently [9]. As our study shows, within the current LDA developers, 52% had experience adapting the user interface of their applications frequently. There were also 32% that were doing the UI adaptation but not frequently.

- *Adoption issues with non-Semantic Web developers.* In order to examine if there is a communication gap between UI designers and Semantic Web developers, we asked the participants about their experience when collaborating with a non-SW developer. Among the participants, 51% had communication issues with non-Semantic Web developers to familiarize them with Semantic Web concepts before they can start contributing to the application. The distribution of this issue among more experienced developers (57% of the intermediate and advanced users) further emphasizes the importance of this communication gap.

4. ADAPTIVE LINKED DATA-DRIVEN WEB COMPONENTS

In order to streamline the process of UI development in LDAs, we propose an architecture of adaptive LD-R Web components – Web components enriched by the RDF data model. As shown in Figure 2, the proposed architecture addresses LDA UI reusability and flexibility by incorporating RDF-based Web components and scopes. In the following sections, the main elements of the architecture are described:

4.1 LD-R Web Components

As depicted in Figure 3, there are four core component levels in an LD-R Web application. Each core component abstracts the actions required for retrieving and updating the graph-based data and provides a basis for user-defined components to interact with Linked Data in three modes: view, edit and browse.

The data-flow in the system starts from the *Dataset* component which handles all the events related to a set of resources under a named graph identified using a URI. The next level is the *Resource* component which is identified by a URI and indicates what is described in the application. A resource is described by a set of properties which are handled by the *Property* component. Properties can be either individual or aggregate when combining multiple features of a resource (e.g. a component that combines longitude and latitude properties; start date and end date properties for a date range, etc.). Each property is instantiated by

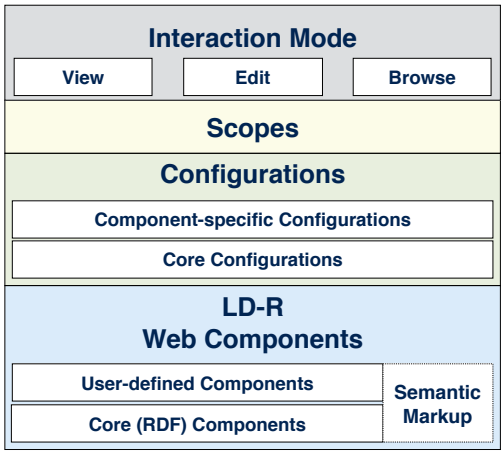

Figure 2: Main elements of the adaptive LD-R Web components architecture.

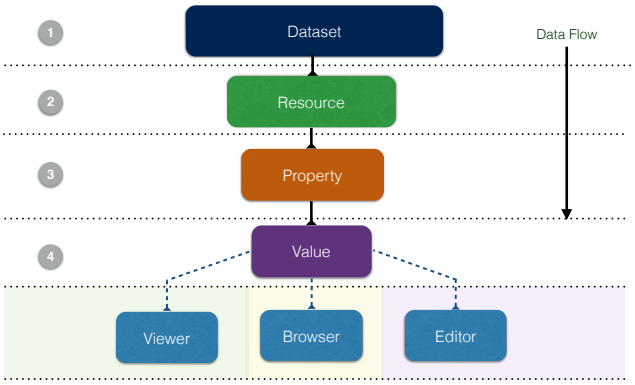

Figure 3: Core LD-R Web components.

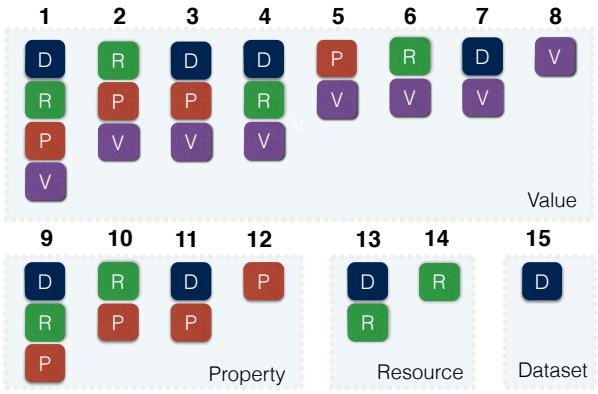

Figure 4: LD-R scopes based on the permutation of dataset, resource, property and value identifiers.

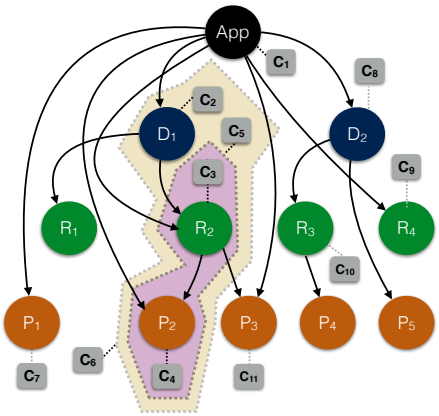

Figure 5: A sample LD-R configuration hypergraph.

an individual value or multiple values in case of an aggregate object. The value(s) of properties are controlled by the *Value* component. In turn, Value components invoke different components to view, edit and browse the property values. *Viewer*, *Editor* and *Browser* components are terminals in the LD-R single directional data flow where customized user-generated components can be plugged into the system. User interactions with the LD-R components are controlled by a set of configurations defined on one or more selected component levels known as scopes.

4.2 Scopes and Configurations

LD-R Web components provide a versatile approach for context adaptation. A context can be a specific domain of interest, a specific user requirement or both. In order to enable customization and personalization, the LD-R approach exploits the concepts of *Scope* and *Configuration*. A scope is defined as a hierarchical permutation of Dataset, Resource, Property and Value components (cf. Figure 4). Each scope conveys a certain level of specificity on a given context ranging from 1 (most specific) to 15 (least specific). Scopes are defined by using either the URIs of named graphs, resources and properties, or by identifying the resource types and data types. A configuration is defined as a setting which affects the way the LDA and Web components are interpreted and rendered (e.g. render a specific component for a specific RDF property or enforce a component to display Wikipedia page URIs for DBpedia resources). UI adaptation is handled by traversing the configurations for scopes, populating the configurations and overwriting the configurations when a more specific applicable scope is found. As shown below, in the worst case, when the `DRPV` scopes are given and the UI is supposed to render the Value components, all the 15 scopes need to be traversed for the adaptation:

```
1  InitialConfig = {initial application
       configuration}
2  Context = [array of scopes with the
       corresponding configuration objects]
3  Config = InitialConfig
4  for (i = 15; i < 1; i--) {
5       Config.compareWith(Context[i]) {
6            Config.addMissingAttributes()
7            Config.overwriteExistingAttributes()
8       }
9  }
```

Code 1: Algorithm for the LD-R UI adaptation.

Figure 5 demonstrates an example of the LD-R configuration hypergraph containing scopes with the maximum depth

of DRP. The graph defines a generic configuration for the application as C_1. There are configurations defined for the dataset scope D_1 as C_2, for the resource scope R_2 as C_3 and for the property scope P_2 as C_4. There are also configurations for the RP scope R_2P_2 as C_5 and for the DRP scope $D_1R_2P_2$ as C_6. Let's suppose we have a setting with the following values for the scopes and configurations:

- D_1= <http://ld-r.org/users>
- R_2= type foaf:Person
- P_2= rdfs:label
- C_1={{viewer:'basic'},{$attr_1$:1},{$attr_2$:3}}
- C_2={{$attr_1$:0},{$attr_3$:2}}
- C_3={{$attr_3$:1},{$attr_4$:4},{$attr_5$:1}}
- C_4={{$attr_5$:2},{$attr_6$:1}}
- C_5={{viewer:'contact'},{$attr_3$:5},{$attr_7$:6}}
- C_6={{$attr_3$:8},{$attr_7$:1},{$attr_8$:3}}

With the above settings, when a property component for rdfs:label is rendered without the dataset and resource context, the configuration will be:

{{viewer:'basic'},{$attr_1$:1},{$attr_2$:3},{$attr_5$:2},{$attr_6$:1}}

When the property component gets rendered within the resource context of type foaf:Person, the settings for viewer and $attr_5$ are overwritten and new settings for $attr_3$, $attr_4$ and $attr_7$ are added:

{{viewer:'contact'},{$attr_1$:1},{$attr_2$:3},{$attr_3$:5},{$attr_4$:4},
{$attr_5$:1},{$attr_6$:1},{$attr_7$:6}}

When the additional context of dataset as <http://ld-r.org/users> is given, $attr_3$ and $attr_7$ get overwritten and a new setting for $attr_8$ is added:

{{viewer:'contact'},{$attr_1$:0},{$attr_2$:3},{$attr_3$:8},{$attr_4$:4},
{$attr_5$:1},{$attr_6$:1},{$attr_7$:1},{$attr_8$:3}}

Scopes can also be defined on a per user basis, facilitating the versioning and reuse of user-specific configurations. User-Specific configurations provide different views on components and thereby data, based on the different personas dealing with them.

In addition to the fine-grained component customization, LD-R Web applications provide a fine-grained access control over the data through the component scopes. For example, an application developer can restrict access to a specific property of a specific resource in a certain dataset and on a specific interaction mode.

4.3 Semantic Markup for Web Components

The innate support of RDF in LD-R Web components enable the automatic creation of semantic markup in the UI level. Lower semantic techniques such as *RDFa*, *Mircodata* and *JSON-LD* can be incorporated in the core LD-R components to expose structured data to current search engines which are capable of parsing semantic markup. For example, an LD-R component created based on the Good Relations[5] or Schema.org ontologies, can automatically expose the product data as Google Rich Snippets for products[6] which will provide better visibility of the data on Web search results (i.e. SEO).

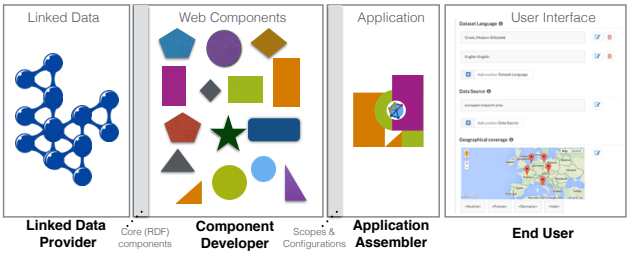

Figure 6: LD-R components life cycle.

In addition to automatic annotation of data provided by the LD-R Web components, the approach offers semi-automatic markup of Web components by creating component metadata. Component metadata consists of two categories of markup:

- Automatic markup generated by parsing component package specification – metadata about the component and its dependencies. It includes general metadata such as name, description, version, homepage, author as well as technical metadata on component source repository and dependencies.
- Manual markup created by component authors which exposes metadata such as component level (dataset, resource, property, value), granularity (individual, aggregate), mode (view, edit, browse) and configuration parameters specification.

Similar to content markup, Component markup can utilize commonly-known ontologies such as Schema.org in order to improve the visibility of LD-R components and enable application assemblers to better understand the intended usage and capabilities of a given component.

4.4 Stackeholders and Life Cycle

As shown in Figure 6, the LD-R components lifecycle encompasses four primary types of stakeholders:

- *Linked Data Provider*. Since the LD-R approach focuses mainly on Linked Data applications, the provision of RDF-compliant data is an essential phase in developing the LD-R components. There are different stages [1] in Linked Data provision, including data extraction, storage, interlinking, enrichment, quality analysis and repair which should be taken into account by data scientists and Linked Data experts. Once the data and schemata are provided to the LD-R component system, the system can bring a reciprocal value to Linked Data providers to better understand and curate the data when needed. For example, in the case of geo-coordinates, a map component can enable data

[5] http://www.heppnetz.de/projects/goodrelations/

[6] https://developers.google.com/structured-data/

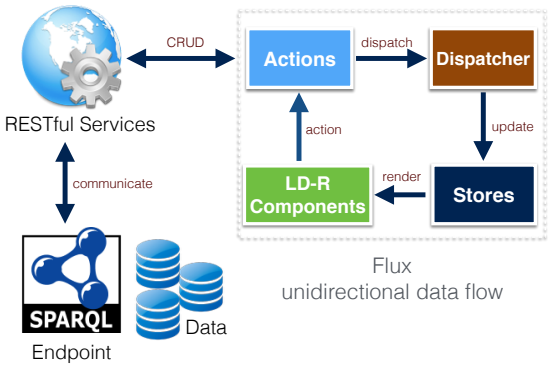

Figure 7: Data flow in the LD-Reactor framework.

providers to easily curate the outlier data (e.g. ambiguous entities) within a certain geo boundary in a visual manner.

- *Component Developer.* Component developers are UX designers and Web programmers who are involved in component fabrication. There are two types of Web components developed in this step: a) *Core components* (cf. Figure 3) which abstract the underlying RDF data model. These components are built-in to the system, however can still be overwritten by developers who have proficiency in Semantic Web and Linked Data. b) *Community-driven components* which exploit the core components. These components are either created from the scratch or by remixing and repurposing existing Web components found on the Web.

- *Application Assembler.* The main task of application assembler is to identify the right components and configurations for the application; and to combine them in a way which fits the application requirement. Within the LD-R component system, the metadata provided by each Web component facilitates the discovery of relevant components. Having shared vocabularies on Linked Open Data allows assemblers to not only reuse components but also reuse the existing configurations and scopes published on the Web. For example, if there is already a suitable configuration for `RP` scope which uses `foaf:Person` as resource type and `dcterms:description` as property URI, the assembler can reuse that configuration within his application.

- *End-User.* End-users experience working with the components to pursue goals in a certain application domain. As such, they may request the development of new components or configurations in order to fulfil their requirements and are expected to provide feedback on existing components.

5. IMPLEMENTATION

In order to realize the idea of adaptive Linked Data-driven Web components, we implemented an open-source software framework called *Linked Data Reactor (LD-Reactor)* which is available online at `http://ld-r.org`. LD-Reactor utilizes Facebook's ReactJS[7] components, the Flux[8] architecture, Yahoo!'s Fluxible[9] framework for isomorphic Web applications (i.e. running the components code both on the server and the client) and the Semantic-UI[10] framework for flexible UI themes. The main reasons we chose *React* components over other Web Components solutions (e.g. Polymer[11], AngularJS[12], EmberJS[13], etc.) were the maturity and maintainability of the technology, the native multi-platform support, the number of developer tools/components/applications, and the efficiency of its underlying virtual DOM approach[14].

As shown in Figure 7, LD-Reactor follows the Flux architecture which eschews MVC (Model-View-Controller) in favour of a unidirectional data flow. When a user interacts with a React component, the component propagates an action through a central dispatcher, to the various stores that hold the application's data and business logic, and updates all affected components. The component interaction with SPARQL endpoints to retrieve and update Linked Data occurs through the invocation of RESTful services in actions.

In order to allow the bootstrapping of LDA UIs, LD-Reactor provides a comprehensive framework that combines the following main elements:

- A set of RESTful Web services that allow basic CRUD operations on Linked Data using SPARQL queries[15].
- A set of core components called *Reactors* which implement core Linked Data components (see Figure 3) together with their corresponding actions and stores.
- A set of default components which allow basic viewing, editing and browsing of Linked Data.
- A set of minimal viable configurations based on the type of data and properties from commonly-used vocabularies on the Semantic Web (e.g. foaf, dcterms and SKOS).
- A basic access control plugin which allows restricting read/write access to data.

LD-Reactor implementation is compliant with *Microservices Architecture* [13] where the existing ReactJS components can be extended by complementary Linked Data services. In contrast to the centralized monolithic architecture, the

[7] `https://facebook.github.io/react/`
[8] `https://facebook.github.io/flux`
[9] `http://fluxible.io/`
[10] `http://semantic-ui.com/`
[11] `http://www.polymer-project.org/`
[12] `https://angularjs.org/`
[13] `http://emberjs.com/`
[14] Elaborating on all these factors is beyond the scope of this paper.
[15] the framework is compliant with the SPARQL 1.1 standard. However, we have identified certain inconsistencies between OpenRDF Sesame and OpenLink Virtuoso RDF stores, which did not allow the execution of syntactically identical queries across both systems. Thereby, specific adaptors have been implemented for each of these two RDF stores.

microservices architecture, allows to put the main functionalities of the LDA into separate decoupled services and scale by distributing these services across servers, replicating as needed. This architectural style also helps to minimize the redeploying of the entire application when changes in components were requested.

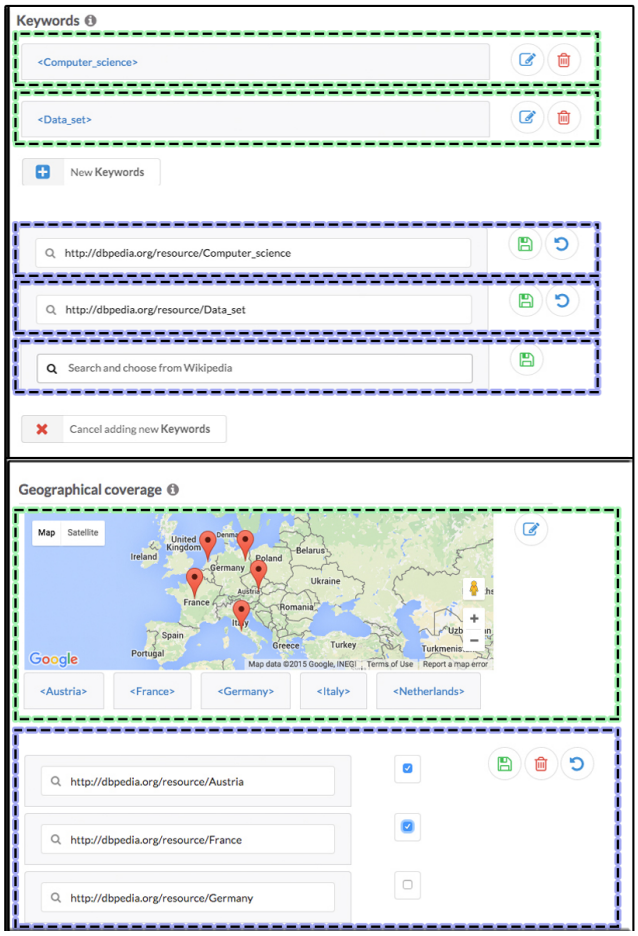

Figure 8: A screenshot of LD-Reactor view and edit mode for individual (top) and aggregate (bottom) values.

There are three modes of interactions within LD-R components namely *view*, *browse* and *edit* mode. These modes work with two types of value granularity: individual and aggregate. As shown in Figure 8, components can target individual values or interact with aggregate values when users want to show/update multiple values at once. Figure 9 depicts the browse mode where individual (e.g. item lists with check boxes) and aggregate data browser (e.g. data sliders or maps) components can be employed.

Semantic markup of data (as discussed in Section 4.3) is supported natively within the framework by embedding Microdata annotations within the LD-R Web components. Additionally, in order to facilitate the creation of component metadata, we developed a tool[16] which automatically generates the general metadata about the components in JSON-LD, using `Schema.org`'s SoftwareApplication schema[17].

[16] `https://github.com/ali1k/ld-r-metadata-generator`

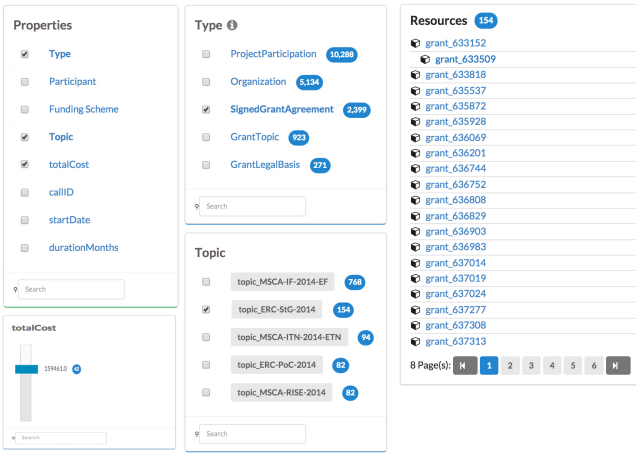

Figure 9: A screenshot of LD-Reactor browse mode.

6. USE CASES

The LD-Reactor framework is already in use within the RISIS[18] and Open PHACTS[19] projects.

6.1 RISIS

The RISIS project aims to provide an infrastructure for research and innovation, targeting researchers from various science and technology domains. The LD-Reactor framework was utilized in RISIS to help data providers with no Linked Data experience to provide RDF metadata about their datasets[20]. This metadata is then used to allow researchers to search the generated metadata to identify and request access to the data they are interested in[21]. In the following, we present the main requirements for configurations and components, together with their representation in the LD-Reactor configuration file[22]:

Configurations:

- The UI should be able to to render metadata properties in different categories (Code 2 line 3, 4).
- The labels for properties should be changeable in the UI especially for technical properties (e.g. RDF dump) that are unknown to researchers outside the Semantic Web domain (Code 2 line 18, 26, 40).
- There should be a hint for properties to help metadata editors to understand the meaning of the property (Code 2 line 20, 28, 41).

[17] `https://schema.org/SoftwareApplication`
[18] `http://risis.eu`
[19] `http://www.openphacts.org`
[20] `http://sms.risis.eu`
[21] `http://datasets.risis.eu`
[22] see the complete configuration file at `http://github.com/risis-eu/sms-platform`

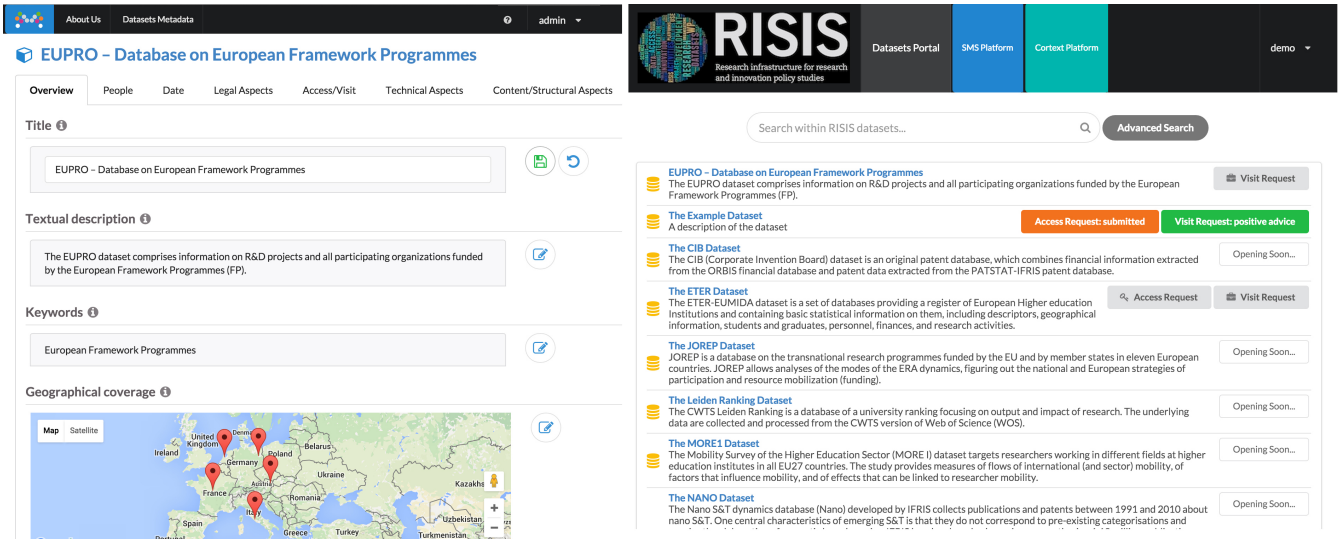

Figure 10: A screenshot of RISIS metadata editor and datasets portal powered by the LD-Reactor framework.

- Instead of showing the full URIs, the output UI should render either a shortened URI or a meaningful string linked to the original URI (Code 2 line 6).
- Whenever a DBpedia URI is provided, display the corresponding Wikipedia URI in the UI enabling users to retrieve human readable information (Code 2 line 33, 45).
- When a dropdown menu is provided, there should be the ability to accommodate user-defined values which are not listed in the menu (Code 2 line 57).

Components:

- A component for `dcterms:spatial` values to allow searching and inserting resources from DBpedia based on the entity type (e.g. Place, Person, Organization, etc).
- A component for `dcterms:subject` values to allow inserting and viewing DBpedia URIs as subject.
- A component for `dcterms:language` values to allow inserting and viewing languages formatted in ISO 639-1 using standard URIs (e.g. `http://id.loc.gov/vocabulary/iso639-1/en`).
- A component for `dcat:byteSize` values to allow inserting and viewing file size specified by a unit.
- A component for `dcterms:format` values to allow inserting and viewing mime types.

In accordance to the LD-Reactor microservices architecture (cf. Section 5), we built a *DBpediaGMap* viewer component where we reused the current react-google-maps[23] together with DBpedia lookup and query services to retrieve the coordinates for the recongnized DBpedia resource values.

[23] `http://github.com/tomchentw/react-google-maps`

Figure 10 shows a screenshot of the generated UIs for metadata and search.

```
1  resource: {
2      'generic': {
3          usePropertyCategories: 1,
4          propertyCategories: ['overview', '
               legalAspects', 'technicalAspects'],
5          resourceReactor: ['Resource'],
6          shortenURI: 1
7      }
8  },
9  property: {
10     'generic': {
11         propertyReactor: ['IndividualProperty'],
12         objectReactor: ['IndividualObject'],
13         objectIViewer: ['BasicIndividualView'],
14         objectIEditor: ['BasicIndividualInput']
15     },
16     'http://purl.org/dc/terms/language': {
17         allowNewValue: 1,
18         label: ['Dataset Language'],
19         category: ['overview'],
20         hint: ['The language of the dataset.
               Resources defined by the Library of
               Congress (http://id.loc.gov/vocabulary/
               iso639-1.html, http://id.loc.gov/
               vocabulary/iso639-2.html) SHOULD be
               used.'],
21         objectIViewer: ['LanguageView'],
22         objectIEditor: ['LanguageInput'],
23         defaultValue: ['http://id.loc.gov/vocabulary
               /iso639-1/en']
24     },
25     'http://purl.org/dc/terms/spatial': {
26          label: ['Geographical coverage'],
27          category: ['overview'],
28          hint: ['The geographical area covered by
                the dataset.'],
29          allowNewValue: 1,
30          objectReactor: ['AggregateObject'],
31          objectAViewer: ['DBpediaGoogleMapView'],
32          objectIViewer: ['BasicDBpediaView'],
33          asWikipedia: 1,
34          objectAEditor: ['BasicAggregateInput'],
35          objectIEditor: ['DBpediaInput'],
36          lookupClass: ['Place']
37      },
38     'http://purl.org/dc/terms/subject': {
39         category: ['overview'],
40         label: ['Keywords'],
```

```
        hint: ['Tags a dataset with a topic.'],
        allowNewValue: 1,
        objectIEditor: ['DBpediaInput'],
        objectIViewer: ['BasicDBpediaView'],
        asWikipedia: 1
    },
    'http://purl.org/dc/terms/license': {
        category: ['legalAspects'],
        label: ['License'],
        allowNewValue: 1,
        objectIViewer: ['BasicOptionView'],
        objectIEditor: ['BasicOptionInput'],
        options: [
            {label: 'Open Data Commons Attribution
                License', value: 'http://www.
                opendatacommons.org/licenses/by/'},
            {label: 'Creative Commons Attribution-
                ShareAlike', value: 'http://
                creativecommons.org/licenses/by-sa/
                3.0/'}
        ],
        allowUserDefinedValue: 1
    }
}
```

Code 2: An excerpt of the LD-Reactor configuration for the RISIS metadata editor.

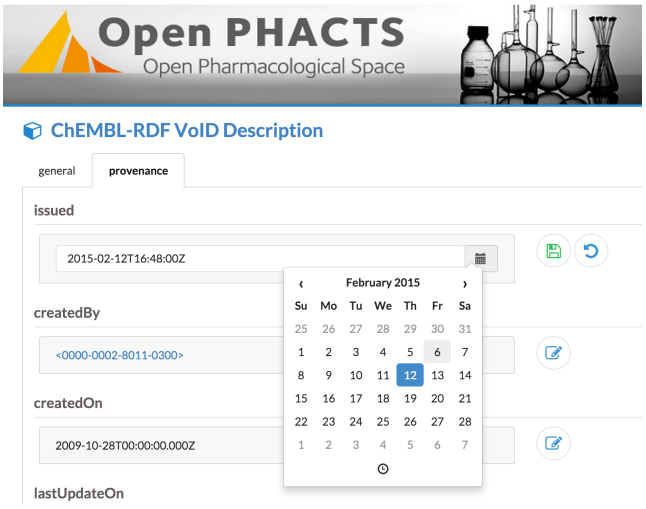

Figure 11: A screenshot of the BasicCalendarInput LD-R component created to allow the editing of datetimes in Open PHACTS VoID descriptions.

6.2 Open PHACTS

The Open PHACTS Discovery Platform has been developed to reduce barriers to drug discovery, by collecting and integrating a large number of prominent RDF datasets in the pharmacology domain. The platform provides a uniform RESTful API for application developers to access the integrated data. In collaboration with the data providers, the Open PHACTS consortium has created a comprehensive dataset description specification[24] based on the Vocabulary of Interlinked Datasets (VoID)[25]. The metadata provided in this context enables (among others) exposing the detailed provenance for each result produced by the platform, the location of source files for each dataset, and example resources.

The provision of VoID dataset descriptors that adhere to the specification proved to be a non-trivial challenge, even for data providers that are well versed in providing RDF distributions of their core data. A series of UIs were therefore created to facilitate the creation of the VoID descriptors. However, as the specification evolved over the first 2 years of the project, any changes or additions made had to be reflected in the UI source code as well; a cumbersome process. Due to the inevitable delay between specification changes and UI development, users often found themselves having to edit large RDF files using text editors, which resulted in frequent syntax errors being made.

A new version of the VoID editor implemented using the LD-Reactor framework is now available online[26]. Though the import/export capabilities of the editor are still not implemented at the time of writing, we have received very positive feedback from the community for a number of reasons:

- UI updates: As the UI is generated based on the underlying data, the process of staying up to date with the current specification becomes trivial. The RDF example provided by the specification can be simply loaded into the RDF store and the changes are immediately visible in the UI through the default core components. Users are then able to adapt the example VoID description to their dataset. The resulting VoID file can be downloaded by exporting all triples in the named graph corresponding to the dataset.

- Dataset releases: A large number of property values remain the same across releases. Similarly with using the example from the Dataset Description Specification, users are able to upload their old VoID description and only edit the outdated values.

- Access control: Using the built–in user authentication mechanism of LD-Reactor we were able to ensure that only the owner(s) of a particular dataset are able to edit its metadata.

- Non-standard properties: Some data providers elect to include additional property that are not prescribed by the specification. The visualisation of such properties is supported easily using the core LD-R components.

- Intuitive navigation: Typically, each dataset consists of a number of subsets, which are also of type `void:Dataset`, and may have further subsets themselves. Displaying all the information together can easily become confusing for the user; instead the LD-Reactor framework was used to provide navigation through the subset links, displaying only a single dataset (or subset) at a time.

- Datetime component: Manually typing datetimes in the required format (e.g. '2015-02-12T16:48:00Z') can be an error prone process. Instead, we have been able to reuse the `react-bootstrap-datetimepicker`[27], to

[24] `http://www.openphacts.org/specs/2013/WD-datadesc-20130912/`
[25] `http://www.w3.org/TR/void/`
[26] `http://void.ops.labs.vu.nl/` Source: `http://github.com/openphacts/ld-r`
[27] `https://github.com/quri/react-bootstrap-datetimepicker`

create a new LD-R value editor for datetimes (BasicCalendarInput) with a graphical interface as shown in Figure 11.

In addition to the significant improvements over previous versions of the VoID editor outlined above, we were able to develop the LD-Reactor version in a fraction of the time that was required for earlier versions.

7. RELATED WORK

Component-based software engineering (CBSE) has been an active research area since 1987 with numerous results published in the research literature [23]. Within the Semantic Web community, the main focus has been on enriching current service-oriented architectures (SOAs) with semantic formalisms and thereby providing Semantic Web services as reusable and scalable software components [24]. There have also been a few attempts to create Semantic Web Components by integrating existing Web-based components with Semantic Web technology [3, 7].

When it comes to component-based development of LDAs, the works typically fall into software application frameworks that address building scalable LDAs in a modular way. The survey conducted by [8] identified the main issues in current Semantic Web applications and suggested the provision of component-based software frameworks as a potential solution to the identified issues. The Semantic Web Framework [4] was one of the first attempts in that direction to decompose the LDA development requirements into an architecture of reusable software components. In most of the current full-stack LDA frameworks such as Callimachus[28] and LDIF[29] the focus is mainly on the backend side of LDAs and less attention is paid on how Linked Data is consumed by the end-user. There are also more flexible application frameworks such as OntoWiki [6] which provide UI widgets and extensions to expose Linked Data to non-Semantic Web end-users.

Besides these generic LDA frameworks, there are also approaches that focus on the development of user interfaces for LDAs. WYSIWYM (What You See Is What You Mean) [12] is a generic semantics-based UI model to allow integrated visualization, exploration and authoring of structured and unstructured data. Our proposed approach utilizes the WYSIWYM model for binding RDF-based data to viewer, editor and browser UIs. Uduvudu [15] is another approach to make an adaptive RDF-based UI engine to render Linked Data. Instead of adopting Web components, Uduvudu employs a set of flexible UI templates that can be combined to create complex UIs. Even though the static templates do not provide enough interactions for editing and browsing data (in contrast to Web components), we believe that algorithms for automatic selection of templates employed in Uduvudu can be reused in the LD-Reactor framework for automatic generation of configurations. Another similar approach is SemwidgJS [21] which brings a semantic Widget library for the rapid development of LDA UIs. SemwidgJS offers a simplified query language to allow the navigation of graph-based data by ordinary Web developers. The main difference between LD-R and SemwidgJS is that LD-Reactor suggests a more interactive model which is not only for displaying Linked Data but also for providing user adaptations based on the meaning of data. LD-Viewer [16] is another related Linked Data presentation framework particularly tailored for the presentation of DBpedia resources. In contrast to LD-Reactor, LD-Viewer builds on top of traditional MVC architecture and its extensions rely heavily on the knowledge of RDF which is a burden for developers unfamiliar with Semantic Web technologies.

In addition to the LDA UI frameworks, there are several ad-hoc tools for Linked Data visualization and exploration such as Balloon Synopsis [19] and Sgvizler [20] which can be utilized as Web components within the LD-Reactor framework. [18] provides an extensive list of these tools aiming to make Linked Data accessible for common end-users who are not familiar with Semantic Web.

Overall, what distinguishes LD-Reactor from the existing frameworks and tools is its modern isomorphic component-based architecture that addresses reactive and reusable UIs as its first class citizen.

8. CONCLUSION AND FUTURE WORK

This paper presented adaptive Linked Data-driven Web components as a solution to increase the usability of current Linked Data applications. The proposed component-based solution emphasizes the reusability and separation of concerns in respect to developing Linked Data applications. The RDF-based UI adaptation mechanism aims to provide better customization and personalization based on the meaning of data. Furthermore, employing standard Web components aspires to bring a better communication between UX designers and Semantic Web developers in order to reuse best UI practices within Linked Data applications.

We believe that making a bridge between Semantic Web and Web Components worlds brings mutual benefits for both sides. On one hand, the Semantic Web technologies provide support for richer component discovery, interoperability, integration and adaptation on the Web. On the other hand, Web Components bring the advantages of UI standardization, reusability, replaceability and encapsulation to the current Semantic Web applications.

As our future plan, we envisage to create a cloud infrastructure to share and reuse LD-R scopes and configurations as well as LD-R Web components without the need to install the framework. We also plan to make a user interface to facilitate creation of the LD-R scopes and configurations. Another direction for future research is developing mechanisms for the automatic configuration and composition of Web components based on the semantic markup provided.

9. AKNOWLEDGEMENT

We would like to thank our colleagues from the KRR research group at VU University Amsterdam for their helpful comments during the development of the LD-R framework. This work was supported by a grant from the European Union's 7th Framework Programme provided for the project RISIS (GA no. 313082).

[28] http://callimachusproject.org/
[29] http://ldif.wbsg.de/

10. REFERENCES

[1] S. Auer, J. Lehmann, A.-C. Ngonga Ngomo, and A. Zaveri. Introduction to linked data and its lifecycle on the web. In *Proceedings of the 9th International Conference on Reasoning Web: Semantic Technologies for Intelligent Data Access*, RW'13, pages 1–90, Berlin, Heidelberg, 2013. Springer-Verlag.

[2] E. Benson and D. R. Karger. End-users publishing structured information on the web: An observational study of what, why, and how. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, pages 1265–1274, New York, NY, USA, 2014. ACM.

[3] M. Casey and C. Pahl. Web components and the semantic web. *Electr. Notes Theor. Comput. Sci.*, 82(5):156–163, 2003.

[4] R. G. Castro, A. G. Pérez, and M. n.-G. Óscar. The Semantic Web Framework: A Component-Based Framework for the Development of Semantic Web Applications. In *DEXA '08: Proceedings of the 2008 19th International Conference on Database and Expert Systems Application*, pages 185–189, Washington, DC, USA, 2008. IEEE Computer Society.

[5] D. Cooney. Introduction to web components, 2014. `http://www.w3.org/TR/components-intro/`.

[6] P. Frischmuth, M. Martin, S. Tramp, T. Riechert, and S. Auer. OntoWiki—An Authoring, Publication and Visualization Interface for the Data Web. *Semantic Web Journal*, 2014.

[7] O. Hartig, M. Kost, and J. C. Freytag. Designing component-based semantic web applications with DESWAP. In C. Bizer and A. Joshi, editors, *Proceedings of the Poster and Demonstration Session at the 7th International Semantic Web Conference (ISWC2008), Karlsruhe, Germany, October 28, 2008*, volume 401 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[8] B. Heitmann, S. Kinsella, C. Hayes, and S. Decker. Implementing semantic web applications: reference architecture and challenges. In *5th International Workshop on Semantic Web-Enabled Software Engineering*, 2009.

[9] R. Hervás and J. Bravo. Towards the ubiquitous visualization: Adaptive user-interfaces based on the semantic web. *Interacting with Computers*, 23(1):40–56, 2011.

[10] D. Karger and M. Schraefel. The pathetic fallacy of rdf. Position Paper for SWUI06, 2006.

[11] A. Khalili and S. Auer. User interfaces for semantic authoring of textual content: A systematic literature review. *Web Semantics: Science, Services and Agents on the World Wide Web*, 22(0):1 – 18, 2013.

[12] A. Khalili and S. Auer. Wysiwym – integrated visualization, exploration and authoring of semantically enriched un-structured content. *Semantic Web Journal*, 2014.

[13] J. Lewis and M. Fowler. Microservices, 2014. `http://martinfowler.com/articles/microservices.html`.

[14] A. Lima, L. Rossi, and M. Musolesi. Coding Together at Scale: GitHub as a Collaborative Social Network. In *Proceedings of the 8th AAAI International Conference on Weblogs and Social Media (ICWSM'14)*, Ann Arbor, Michigan, USA, June 2014.

[15] M. Luggen, A. Gschwend, A. Bernhard, and P. Cudre-Mauroux. Uduvudu: a graph-aware and adaptive ui engine for linked data. In C. Bizer, S. Auer, T. Berners-Lee, and T. Heath, editors, *Proceedings of the Workshop on Linked Data on the Web (LDOW)*, number 1409 in CEUR Workshop Proceedings, Aachen, 2015.

[16] D. Lukovnikov, C. Stadler, and J. Lehmann. Ld viewer - linked data presentation framework. In *Proceedings of the 10th International Conference on Semantic Systems*, SEM '14, pages 124–131, New York, NY, USA, 2014. ACM.

[17] D. A. Norman. *The Design of Everyday Things: Revised and Expanded Edition*. Basic Books, Inc., New York, NY, USA, 2013.

[18] S. Ojha, M. Jovanovic, and F. Giunchiglia. Entity-centric visualization of open data. In J. Abascal, S. Barbosa, M. Fetter, T. Gross, P. Palanque, and M. Winckler, editors, *Human-Computer Interaction INTERACT 2015*, volume 9298 of *Lecture Notes in Computer Science*, pages 149–166. Springer International Publishing, 2015.

[19] K. Schlegel, T. Weißgerber, F. Stegmaier, M. Granitzer, and H. Kosch. Balloon synopsis: A jquery plugin to easily integrate the semantic web in a website. In R. Verborgh and E. Mannens, editors, *Proceedings of the ISWC Developers Workshop 2014, co-located with the 13th International Semantic Web Conference (ISWC 2014), Riva del Garda, Italy, October 19, 2014.*, volume 1268 of *CEUR Workshop Proceedings*, pages 19–24. CEUR-WS.org, 2014.

[20] M. G. SkjÃęveland. Sgvizler: A javascript wrapper for easy visualization of sparql result sets. In *9th Extended Semantic Web Conference (ESWC2012)*, May 2012.

[21] T. Stegemann and J. Ziegler. Semwidgjs: A semantic widget library for the rapid development of user interfaces for linked open data. In *44. Jahrestagung der Gesellschaft für Informatik, Informatik 2014, Big Data - Komplexität meistern, 22.-26. September 2014 in Stuttgart, Deutschland*, pages 479–490, 2014.

[22] J. Tsay, L. Dabbish, and J. Herbsleb. Let's talk about it: Evaluating contributions through discussion in github. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 144–154, New York, NY, USA, 2014. ACM.

[23] T. Vale, I. Crnkovic, E. S. de Almeida, P. A. da Mota Silveira Neto, Y. a Cerqueira Cavalcanti, and S. R. de Lemos Meira. Twenty-eight years of component-based software engineering. *Journal of Systems and Software*, 2015.

[24] H. H. Wang, N. Gibbins, T. Payne, A. Patelli, and Y. Wang. A survey of semantic web services formalisms. *Concurrency and Computation: Practice and Experience*, 27(15):4053–4072, 2015. 10.1002cpe.3481.