Linked Data Reactor: a Framework for Building Reactive Linked Data Applications

Ali Khalili

Knowledge Representation and Reasoning Research Group
Deptartment of Computer Sciences
Vrije Universiteit Amsterdam
a.khalili@vu.nl

Abstract. This paper presents Linked Data Reactor (LD-Reactor or LD-R) as a framework for developing flexible and reusable User Interface components for Linked Data applications. LD-Reactor utilizes Facebook's ReactJS components, Flux architecture and Yahoo's Fluxible framework for isomorphic Web applications. It also exploits Semantic-UI framework for flexible UI themes. LD-R aims to apply the idea of component-based application development into RDF data model hence enhancing current user interfaces to view, browse and edit Linked Data.

Documentation: http://ld-r.org
Demo: http://demo.ld-r.org
Code Repository: https://github.com/ali1k/ld-r

1 Introduction

With the growing number of structured data published, the Web is moving towards becoming a rich ecosystem of machine-understandable Linked Data. Semantically structured data facilitates a number of important aspects of information management such as information retrieval, search, visualization, customization, personalization and integration [1]. 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 current communication gap between Semantic Web developers and User Experience (UX) designers, caused by the need to bear Semantic Web knowledge, prevents the streamlined flow of best practices from the UX community into Linked Data user interface (UI) development. 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.

In [3], We performed an elaborate study on the current pitfalls of LDA UI design and proposed *Adaptive Linked Data-driven Web Components* and its open

source implementation called Linked Data Reactor[1] as a solution to tackle those issues. This paper serves as a more technical description of that idea.

2 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 1, 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:

2.1 LD-R Web Components

As depicted in Figure 2, 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.

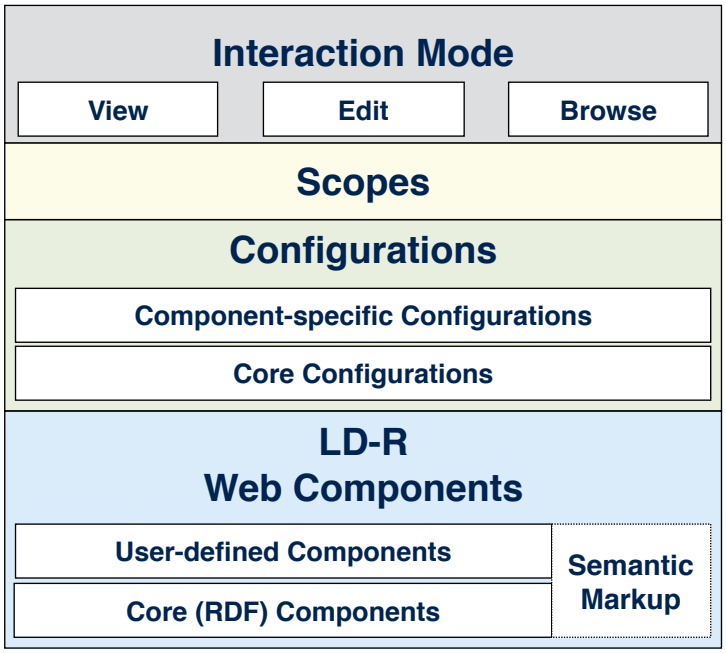

Fig. 1. Main elements of the adaptive LD-R Web components architecture.

[1] http://ld-r.org

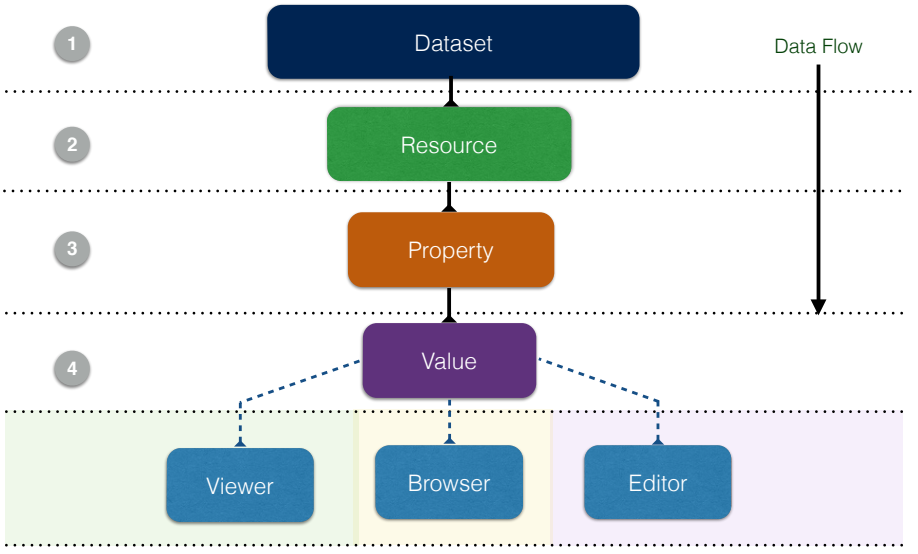

Fig. 2. Core LD-R Web components.

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 by 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 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.

2.2 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 3).

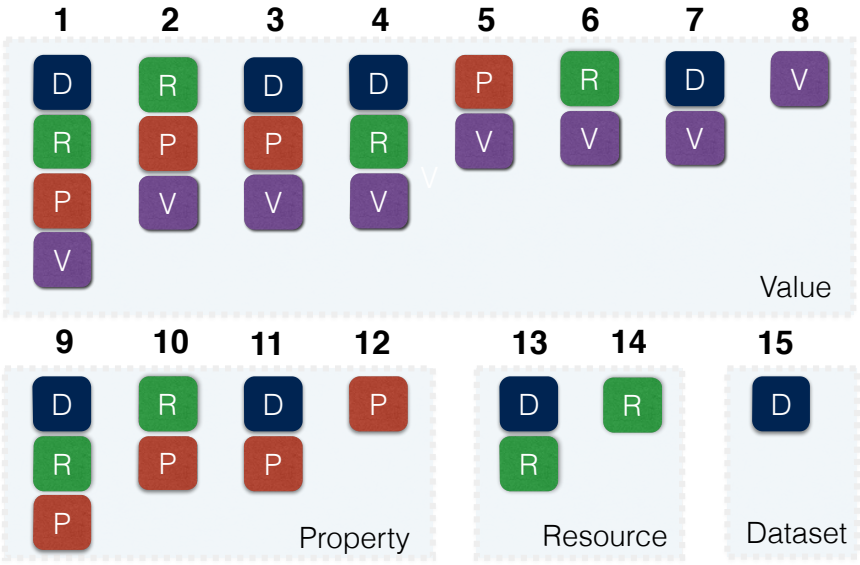

Fig. 3. LD-R scopes based on the permutation of dataset, resource, property and value identifiers.

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

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

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 them when a more specific applicable scope is found. As shown in Code 1 below, in the worst case when the `DRPV` scopes are used and the UI is

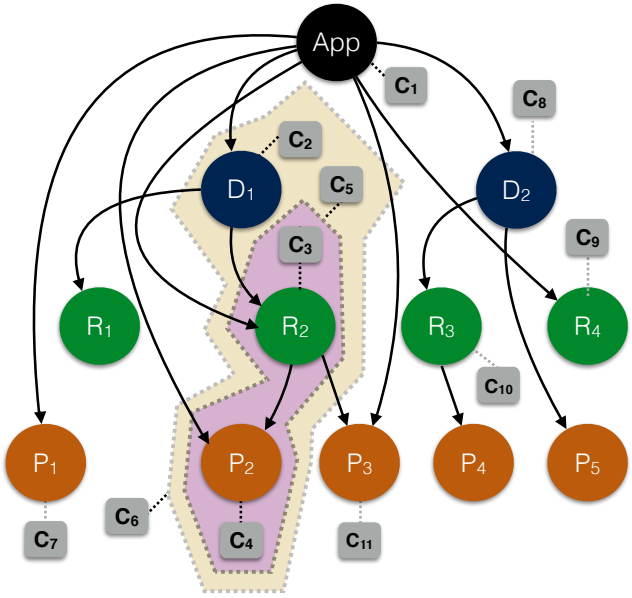

Fig. 4. A sample LD-R configuration hypergraph.

supposed to render the Value components, all 15 scopes need to be traversed for the adaptation:

Figure 4 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.

2.3 Semantic Markup for Web Components

The innate support of RDF in LD-R Web components enable the automatic creation of semantic markup on 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[2] or `Schema.org` ontologies, can automatically expose the product data as Google Rich Snippets for products[3] which will provide better visibility of the data on Web search results (i.e. SEO).

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.

[2] `http://www.heppnetz.de/projects/goodrelations/`
[3] `https://developers.google.com/structured-data/`

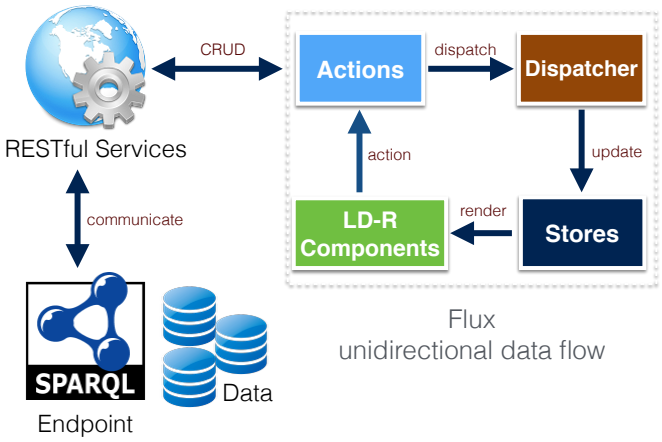

Fig. 5. Data flow in the LD-Reactor framework.

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.

3 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[4] components, the Flux[5] architecture, Yahoo!'s Fluxible[6] framework for isomorphic Web applications (i.e. running the components code both on the server and the client) and the Semantic-UI[7] framework for flexible UI themes. The main reasons we chose *React* components over other Web Components solutions (e.g. Polymer[8], AngularJS[9], EmberJS[10], 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[11].

As shown in Figure 5, LD-Reactor follows the Flux architecture which eschews MVC (Model-View-Controller) in favour of a unidirectional data flow.

[4] `https://facebook.github.io/react/`
[5] `https://facebook.github.io/flux`
[6] `http://fluxible.io/`
[7] `http://semantic-ui.com/`
[8] `http://www.polymer-project.org/`
[9] `https://angularjs.org/`
[10] `http://emberjs.com/`
[11] Elaborating on all these factors is beyond the scope of this paper.

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[12].
- A set of core components called *Reactors* which implement core Linked Data components (see Figure 2) 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* [4] where the existing ReactJS components can be extended by complementary Linked Data services. In contrast to the centralized monolithic architecture, the microservices architecture allows placing 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.

There are three modes of interactions within LD-R components namely *view*, *browse* and *edit*. These modes work with two types of value granularity: individual and aggregate. As shown in Figure 7, components can target individual values or interact with aggregate values when users want to show/update multiple values at once. Figure 6 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 2.3) 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[13] which automatically generates the general

[12] 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.

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

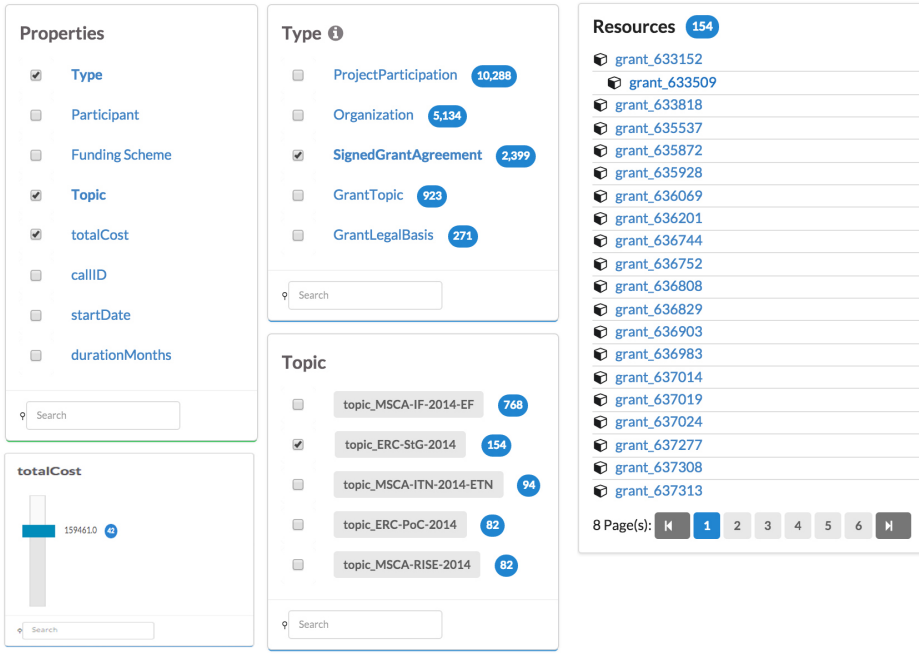

Fig. 6. A screenshot of LD-Reactor browse mode.

metadata about the components in JSON-LD, using `Schema.org`'s SoftwareApplication schema[14].

Code 2 presents a sample LD-R config which is already in-use within the RISIS project[15]:

- The UI should be able 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).
- 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 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).

[14] `https://schema.org/SoftwareApplication`
[15] `http://datasets.risis.eu`

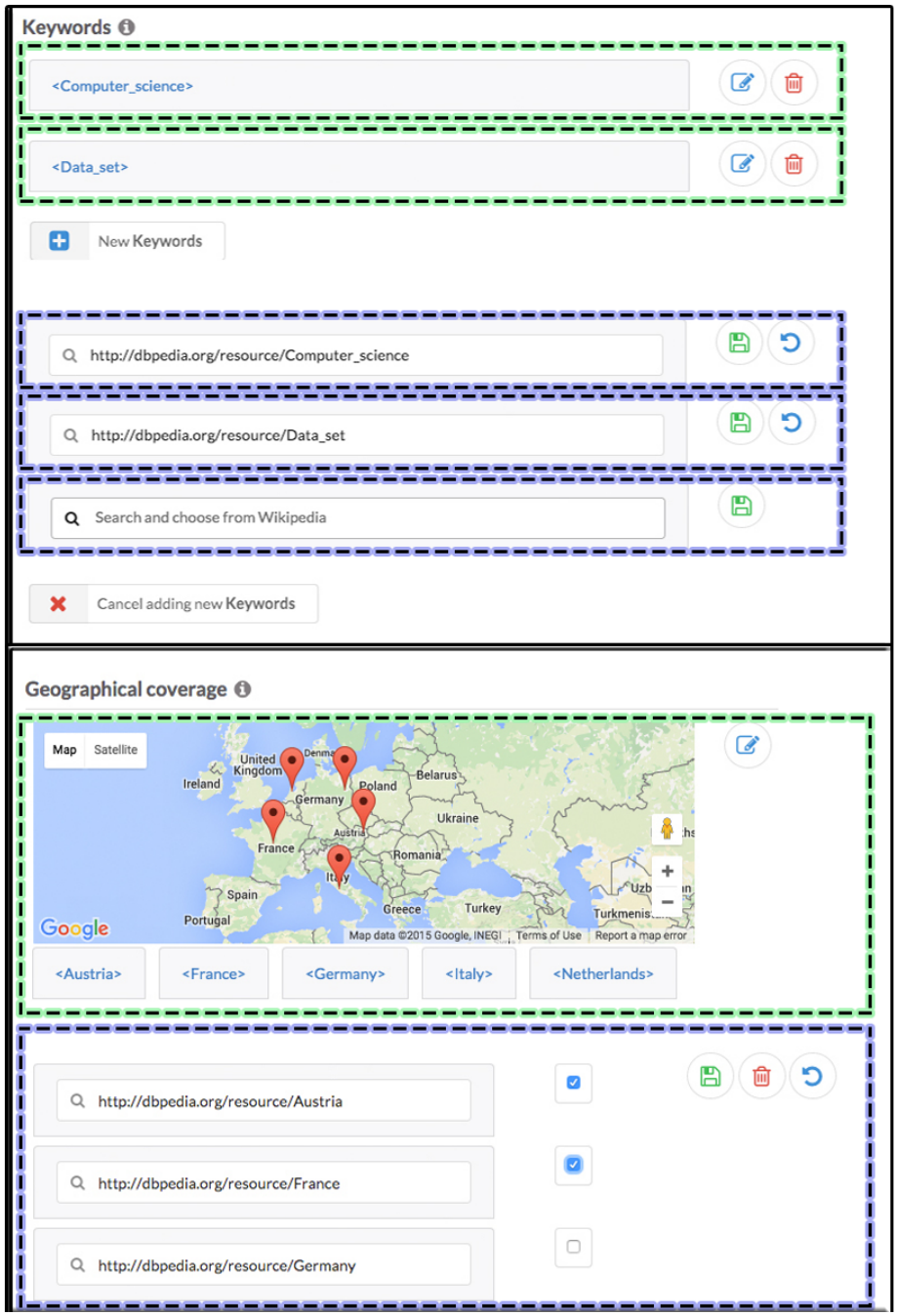

Fig. 7. A screenshot of LD-Reactor view and edit mode for individual (top) and aggregate (bottom) values.

```
resource: {
    'generic': {
        usePropertyCategories: 1,
        propertyCategories: ['overview', 'legalAspects', 'technicalAspects'],
        resourceReactor: ['Resource'],
        shortenURI: 1
    }
},
property: {
    'generic': {
        propertyReactor: ['IndividualProperty'],
        objectReactor: ['IndividualObject'],
        objectIViewer: ['BasicIndividualView'],
        objectIEditor: ['BasicIndividualInput']
    },
    'http://purl.org/dc/terms/language': {
        allowNewValue: 1,
        label: ['Dataset Language'],
        category: ['overview'],
        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.'],
        objectIViewer: ['LanguageView'],
        objectIEditor: ['LanguageInput'],
        defaultValue: ['http://id.loc.gov/vocabulary/iso639-1/en']
    },
    'http://purl.org/dc/terms/spatial': {
         label: ['Geographical coverage'],
         category: ['overview'],
         hint: ['The geographical area covered by the dataset.'],
         allowNewValue: 1,
         objectReactor: ['AggregateObject'],
         objectAViewer: ['DBpediaGoogleMapView'],
         objectIViewer: ['BasicDBpediaView'],
         asWikipedia: 1,
         objectAEditor: ['BasicAggregateInput'],
         objectIEditor: ['DBpediaInput'],
         lookupClass: ['Place']
     },
    'http://purl.org/dc/terms/subject': {
        category: ['overview'],
        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 file.

4 Related Work

We have brought an elaborate analysis of the related work in [3]. In this section, we only summarize the main related work.

UI Frameworks. WYSIWYM (What You See Is What You Mean) [2] 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 [5] is another approach to making 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 [10] 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 [6] 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 the traditional MVC architecture and its extensions rely heavily on the knowledge of RDF which is a burden for developers unfamiliar with Semantic Web technologies.

Tools and Applications. In addition to the LDA UI frameworks, there are several ad-hoc tools for Linked Data visualization and exploration such as Balloon Synopsis [8] and Sgvizler [9] which can be utilized as Web components within the LD-Reactor framework. [7] 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 Linked-Data-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.

5 Conclusion

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

This paper presented Linked Data Reactor as a component-based LDA development framework which aims to bring a better communication between UX designers and Semantic Web developers in order to reuse best UI practices within Linked Data applications.

References

1. 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.
2. A. Khalili and S. Auer. Wysiwym – integrated visualization, exploration and authoring of semantically enriched un-structured content. *Semantic Web Journal*, 2014.
3. A. Khalili, A. Loizou, and F. van Harmelen. Adaptive linked data-driven web components: Building flexible and reusable semantic web interfaces. *Semantic Web Conference (ESWC) 2016*, 2016.
4. J. Lewis and M. Fowler. Microservices, 2014. `http://martinfowler.com/articles/microservices.html`.
5. 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, *Workshop on Linked Data on the Web (LDOW)*, number 1409 in CEUR Workshop Proceedings, Aachen, 2015.
6. 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.
7. 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*, volume 9298 of *Lecture Notes in Computer Science*, pages 149–166. Springer, 2015.
8. 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, *ISWC Developers Workshop*, volume 1268 of *CEUR Workshop Proceedings*, pages 19–24. CEUR-WS.org, 2014.
9. M. G. Skjveland. Sgvizler: A javascript wrapper for easy visualization of sparql result sets. In *9th Extended Semantic Web Conference (ESWC2012)*, May 2012.
10. 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.