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 RDF data model and SPARQL query language to interact with the triple stores. 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 underlying Semantic Web technologies in order to allow reuse of existing Web components in LDAs and to enable Web developers who are not expert in Semantic Web to develop interfaces to 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

D.2.13 [**Software Engineering**]: Reusable Software

General Terms

Design, Human Factors, 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 (LD) [1]. Semantically structured data facilitate a number of important aspects of information management such as information retrieval, search, visualization, customization, personalization and integration [5]. Despite all these benefits, Linked Data Applications (LDAs) have not yet grasped well 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 [4] which indicate some entrance barriers towards wide-spread utilization of Linked Data [14].

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 makes current LDAs not often function consistent 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 [3] 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 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. LD-R components enable encapsulation of the Semantic Web nature of an LDA thereby allow UX designers and Web developers outside the Semantic Web community to contribute to LDAs. They also allow current Semantic Web developers 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.

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

[1] `lodlaundromat.org` recently (25.09.2015) reported approx. 38.6 billion triples published on the Web.

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. 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. Abstracting the complexity of RDF and graph-based data representation, provides more *Affordances* [11] 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 reuse existing best practices from the UX community within the LDAs hence building more usable and pervasive LDAs.

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 **75** 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 [8, 15]. 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 set of 7 minimal viable questions to attract more people and also used inline forms for GMail users to allow filling out the questionnaire in the same time that reading the enquiry email. The number of participants who filled in our questionnaire at the end was 75 (almost 11.5% of the potential candidates which is still a considerable number of participants). 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 (60%) 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. 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 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: resue by copy/pasting code from existing LDAs and reuse by employing current Web components. Reuse by copy/pasting code can be an indicator 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

[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"

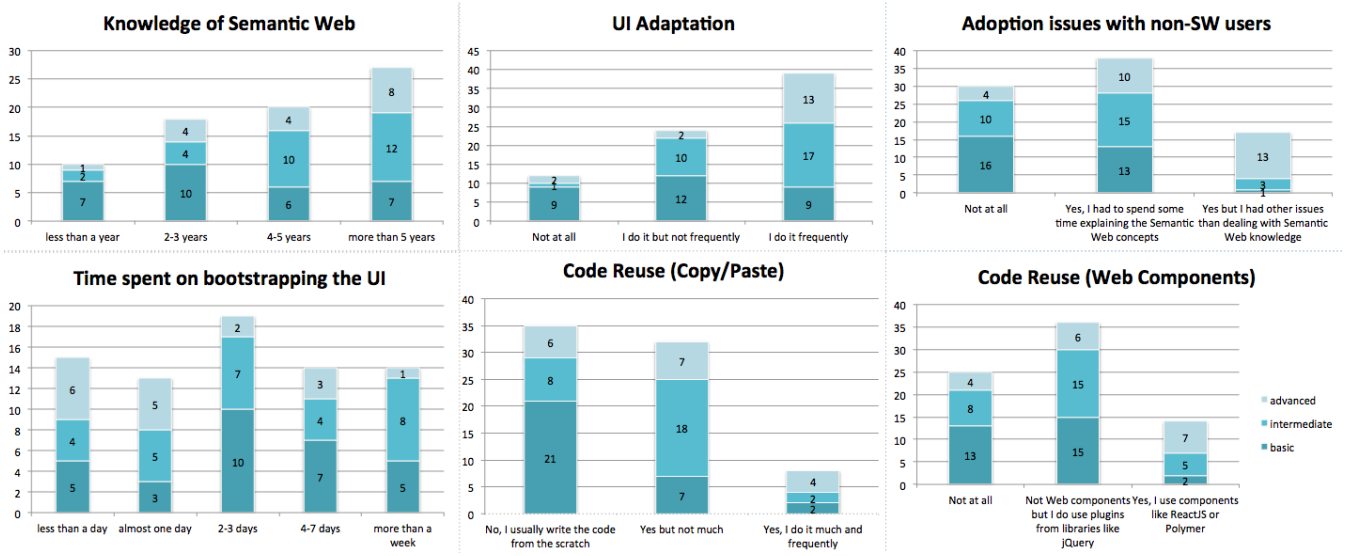

Figure 1: Results of our user study on the current status of LDA UI development.

highlighted for basic developers which still prefer to write the code from the scratch although they have less experience in programming LDAs. Furtheremore, figures 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 provides an attitude and capacity towards adopting the Web components.

- *Adaptation of LDA UIs.* Among the participant, 52% had experience adapting the user interface of their applications frequently.
- *Issues with non-Semantic Web developers to create LDA UIs.* Among the participant, 51% had communication issues with non-Semantic Web developers to familiarize them with Semantic Web concepts before they can start contributing to the application.

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 components – Web components enriched by the RDF data model. In the following sections, the main elements of the architecture are described:

4.1 LD-R Web Components

As depicted in Figure 2, there are 4 main 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. For example, a viewer in the level of Dataset component can enable visualizing a set of resources based on a certain property value whereas a viewer in the level of Resource component can only allow visualizing properties of a specific resource (or a specific type of resource).

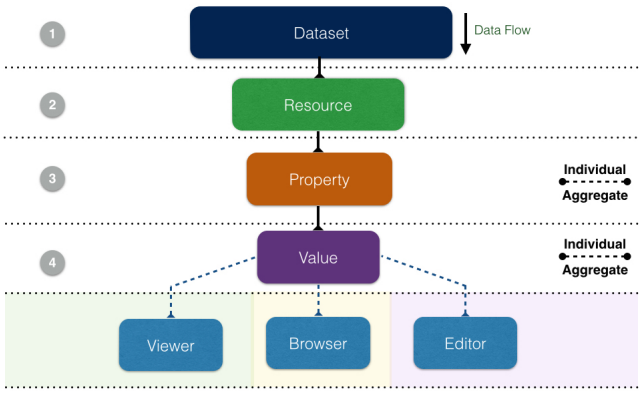

Figure 2: LD-R components architecture.

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 specified 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 data 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. Value component invokes 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. These components apply on individual and aggregate values (e.g. to show mul-

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

tiple coordinates on a map).

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, LD-R approach exploits the concept of *Scope*. A scope is defined as a hierarchical permutation of Dataset, Resource, Property and Value components (cf. Figure 3). Each scope conveys a certain level of specificity on a given context ranging from 1 (most specific) to 15 (least specfic). Scopes are defined by using either the URIs of named graphs, resources and properties; or by identifying the resource types and data types. UI adaptation is handled by traversing the configs for scopes and overwriting the configs when a more specific scope is found. For example, on the property level, we can define a generic configuration for all properties and then for some specific properties (e.g. `dcterms:title` or `rdfs:label`) within a specific resource (e.g. `<http://ld-r.org>`), we can change those configurations using the `RP` scope. Another example would be having a different rendering for all resources of a specific type (e.g. `foaf:Person`).

Scopes can also be defined under a specific user which facilitates versioning and reuse of user-specific configs. User-Specific configs provide different views on components and thereby data, based on the different personas dealing with those components and data.

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.

4.3 Semantic Markup for Web Components

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

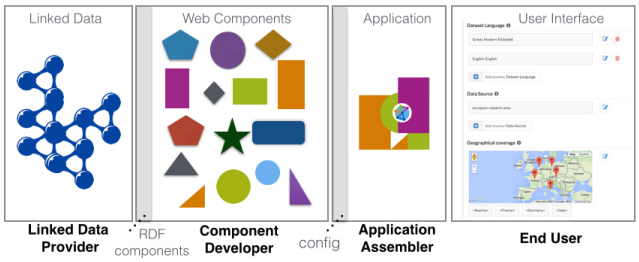

Figure 4: LD-R components life cycle.

LD-R component created based on the Good Relations [5] or `Schema.org` ontology, 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).

In addition to automatic annotation of data provided by the LD-R Web components, LD-R 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 addresses metadata such as component level (dataset, resource, property, value), granularity (individual, aggregate), mode (view, edit, browse) and config parameters specification.

Similar to content markup, Component markup can utilize commonly-known ontologies such as `Schema.org` in order to provide better visibility and understandability of LD-R components for application assemblers.

4.4 Stackeholders and Life Cycle

As shown in Figure 4, 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, provision of RDF-compliant data is an essential phase in developing the LD-R components. There are different stages [1] in Linked Data provision such as 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

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

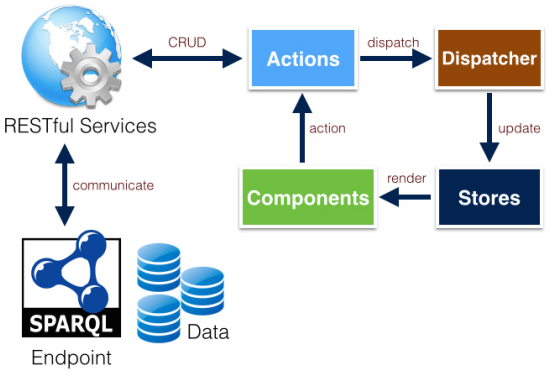

Figure 5: Data flow in the LD-R framework.

curate the data when needed. For example, in case of geo-coordinates, by looking at a map component, data provider can easily curate the outlier data (e.g. ambiguous entities) within a certain geo boundary.

- *Component Developer.* It includes 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 2) 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 `dc-terms:description` as property URI, the assembler can reuse that config within his application.

- *End User.* It is the user who experiences working with components to pursue his goals on a certain application domain. The end user is the one who requests developing a component and the one who sends feedback on the 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-R)* which is available online at `http://ld-r.org`. LD-R utilizes Facebook's ReactJS[7] components, 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 Semantic-UI[10] framework for flexible UI themes. The main reasons we chose *React* components over other existing Web Components solutions (e.g. Polymer[11], AngularJS[12], EmberJS[13], etc.) were the maturity of the technology, maintainability, native multi-platform support, number of developer tools/components/applications, and efficiency of its underlying virtual DOM approach [14]

As shown in Figure 5, LD-R 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, which updates all of the components that are affected. The component interaction with SPARQL endpoints to retrieve and update Linked Data occurs via invoking RESTful services in actions.

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

- A set of RESTful Web services that allow basic CRUD operations on Linked Data by employing SPARQL queries [15].
- 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.

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 6, components can target individual values or interact with aggregate values when users want to show/update multiple values at once. Figure 7 depicts the browse mode where individual (e.g. item lists with

[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 SPARQL 1.1 standard. However, we noticed some incompatibility between Sesame and Virtuoso triple stores. thereby had to implement separate adaptors for them!

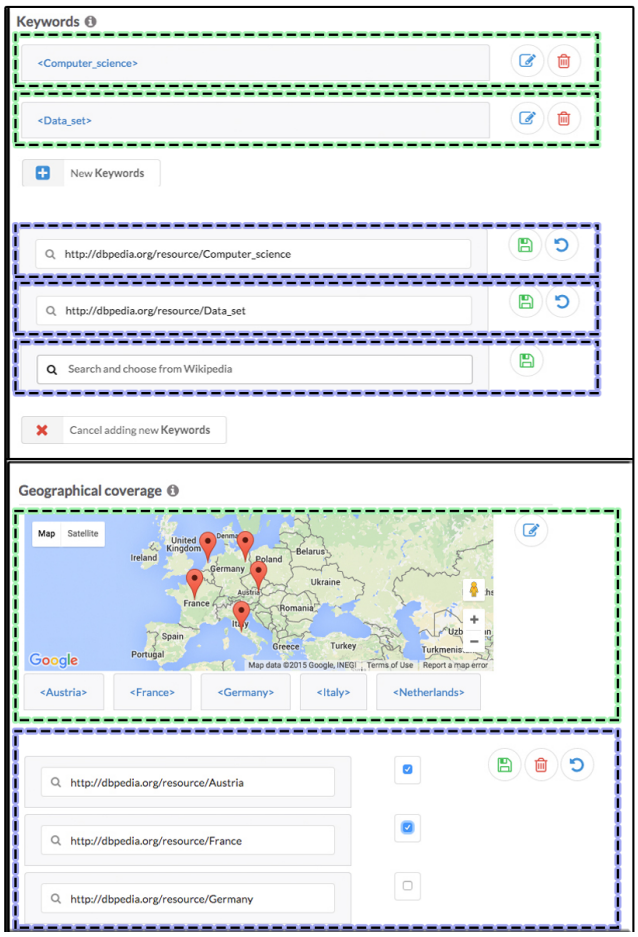

Figure 6: An screenshot of LD-R view and edit mode for individual (top) and aggregate (bottom) values.

check boxes) and aggregate data browser (e.g. data sliders or maps) component 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 creation of components metadata, we developed a tool [16] which automatically generates the general metadata about the components in the JSON-LD format using `Schema.org` SoftwareApplication schema [17].

6. USE CASES

LD-R framework is already in use within the RISIS [18] and OpenPhacts [19] projects.

RISIS project with the aim of providing an infrastructure for research and innovation, targets researchers from various science and technology domains. LD-R approach was utilized in RISIS to help non-Linked Data experts to provide RDF-based metadata about their datasets [20] and then allowing researchers to search and request data based on the generated metadata [21]. In order to expose a user-friendly interface to researchers, a set of configurations and components were devised. In the following, we present the list of main main requirements together with their representation in the LD-R config file [22]:

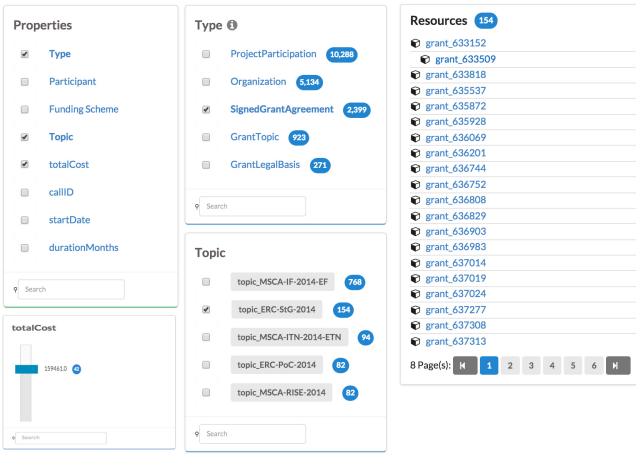

Figure 7: An screenshot of LD-R browse mode.

Configurations

- The UI should be able to to render metadata properties in different categories (Code 1 line 3,4).
- The labels for properties should be changeable in the UI esp. in case of technical properties (e.g. RDF dump) unknown to researchers (Code 1 line 18,26,40).
- There should be a hint for properties to help metadata editors to understand the meaning of the property (Code 1 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 1 line 6).
- When a DBpedia URI is provided, use the Wikipedia URI of the resource in the UI (Code 1 line 33,45).
- If a dropdown menu is provided, there should be the ability to accommodate user-defined values which are not listed in the menu (Code 1 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

[16] `https://github.com/ali1k/ld-r-metadata-generator`
[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 config file at `http://github.com/risis-eu/sms-platform`

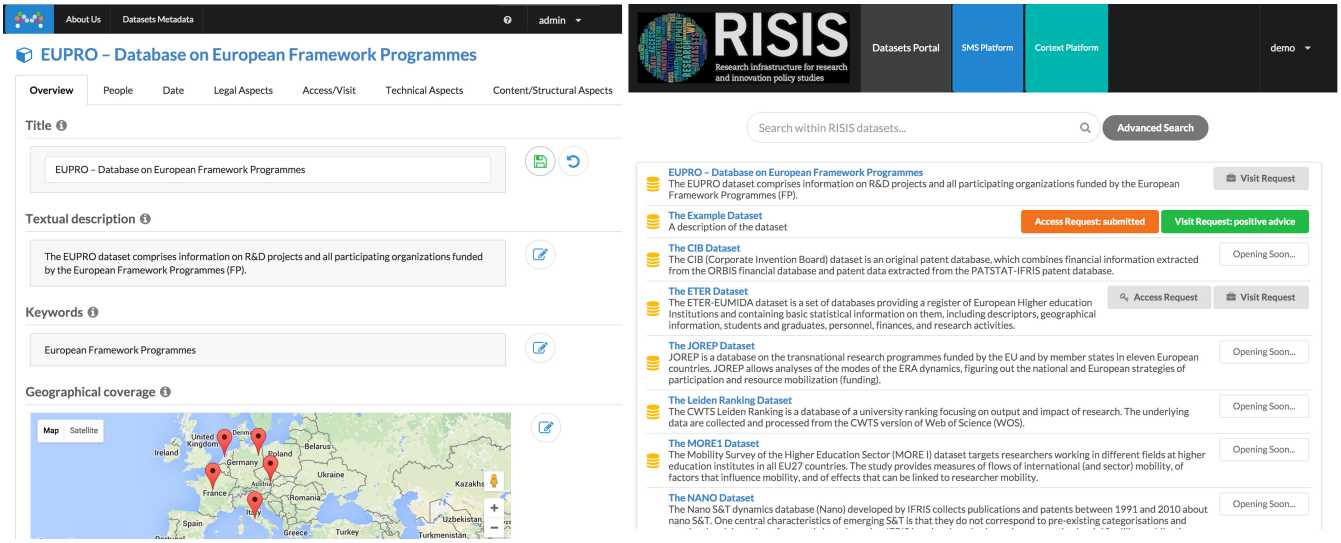

Figure 8: An screenshot of RISIS metadata editor and datasets portal powered by the LD-R framework.

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.

```
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 1: An excerpt of LD-R configs for RISIS.

The required components were built on the mircoservice architecture where we could reuse existing ReactJS components extended by complementary Linked Data services. For example, we built a *DBpediaGMap* viewer component where we reused the current react-google-maps [23] together with a DBpedia query service to retrieve the coordinates for the DBpedia resource values. Figure 8 shows an screenshot of the generated UIs for metadata and search.

Todo: add a short description for OpenPhacts LD-R implementation. link is `ldr.antonis.ops.labs.vu.nl/` now but maybe we can have a more concise link.

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

7. DISCUSSION

8. RELATED WORK

The idea of component-based software engineering (CBSE) has attracted the attention of many researchers since 28 years ago and has led to many results being published in the research literature [16]. Within the Semantic Web community, the main focus has been on enriching current service-oriented architectures (SOAs) by semantic formalisms and thereby providing Semantic Web services as reusable and scalable software components [17]. There has been also a few attempts to create Semantic Web Components by integrating existing Web-based components with Semantic Web technology [2]. 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. In most of the current full-stack LDA frameworks such as Callimachus [24] and LDIF [25] 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 [4] 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) [6] is a generic semantics-based UI model to allow integrated visualization, exploration and authoring of structured and unstructured data. Our proposed LD-R approach utilizes the WYSIWYM model for binding RDF-based data to viewer, editor and browser UIs. Uduvudu [9] 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 in contrast to Web components, the static templates do not provide enough interactions for editing and browsing data, we believe that algorithms for automatic selection of templates employed in Uduvudu can be reused in the LD-R framework for automatic generation of configurations. Another similar approach is SemwidgJS [14] 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-R 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 [10] is another related Linked Data presentation framework particularly tailored for the presentation of DBpedia resources. In contrast to LD-R, LD-Viewer builds on top of traditional MVC architecture and its extensions rely heavily on the knowledge of RDF which puts a burden on non-Semantic Web developers.

In addition to the LDA UI frameworks, there are several ad-hoc tools for Linked Data visualization and exploration such as Balloon Synopsis [12] and Sgvizler [13] which can still be utilized as Web components within the LD-R framework. In overall, what distinguishes LD-R from the existing frameworks and tools is its modern isomorphic component-based architecture that addresses reactive and reusable UIs as its first class citizen.

[24] `http://callimachusproject.org/`

[25] `http://ldif.wbsg.de/`

9. CONCLUSION

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 a 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.

As our future plan, we envisage to create a cloud infrastructure to share and reuse LD-R configurations and 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 automatic composition of Web components based on the provided semantic markup.

10. 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).

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

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

[4] 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.

[5] 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.

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

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

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.