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 resuable 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 [8].

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 deveopers. 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 reusable 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 defiend 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 encapsualtion 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 relavant 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* [7] 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 encapsualtes 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 **69** 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. We used Github APIs [3] to search [4] for the potential Semantic Web repositories and then 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 developers 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 out our questionnaire at the end was 69 (10.4% of the potential candidates which is still a considerable amount of participants). Questions addressed the following topics:

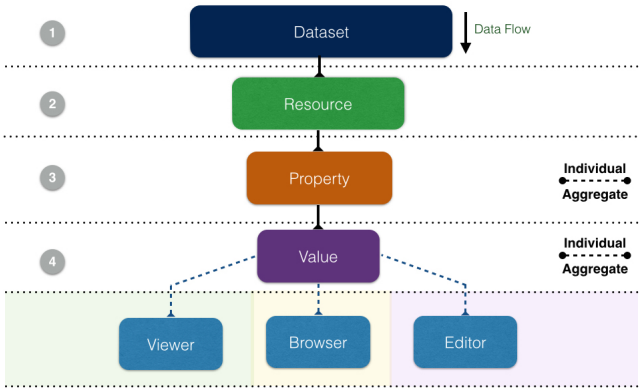

Figure 2: LD-R components architecture.

Knowldge of Semantic Web and Experience in developing LDAs. mainly proficient and expert in SW mainly intermidiate and professional developers

Amount of time spent on bootstraping LDA UIs. a lot of time spent

Reuse of code by Semantic Web developers. not much reuse, mainly writing the code from the scratch

Adoption of current Web Components by Semantic Web developers. less but there is a capacity here because they already reuse libraries

Issues with non-Semantic Web developers to create LDA UIs. 55% had issues which shows the need to mediate

[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"

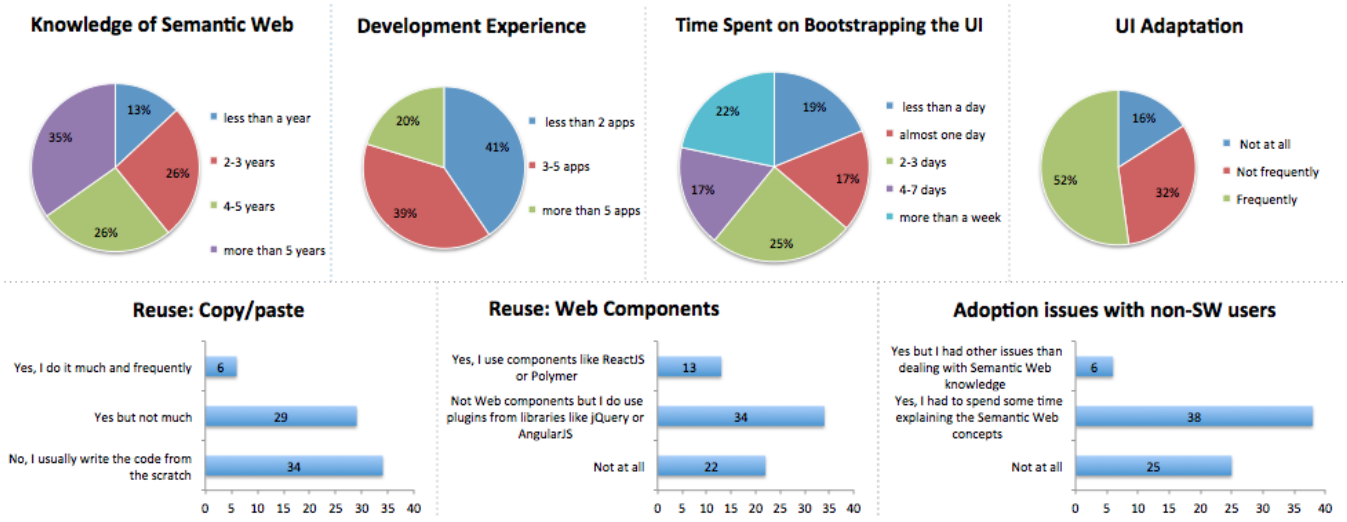

Figure 1: Results of our user study on the current status of LDA UI development.

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).

The dataflow in the application 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 mutiple values in case of an aggreagte object. The value(s) of properties are controled 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 multiple coordinates on a the map).

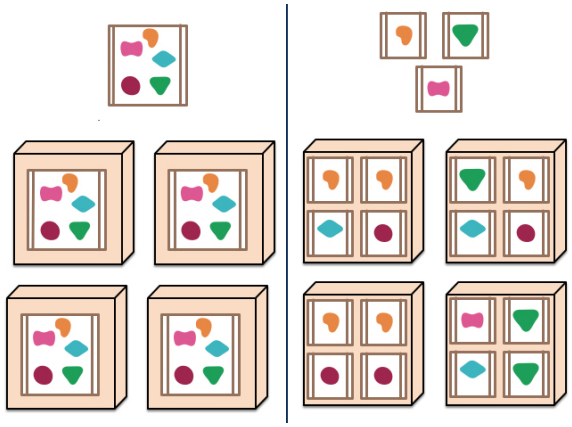

Figure 3: Monoliths vs. Microservices [6]

In contrast to the centalized monolithic architecture, LD-R components comply with *Microservices Architecture* [6]. As shown in Figure 3, microservices architecture puts the main functionalities of a component into separete services (instead of in-memory function calls) and scales by distributing these services across servers, replicating as needed. This architectural style minimizes the redeploying of the entire application when changes in components occur.

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 requirment or both. In order to enable customization and personalization, LD-R approach exploits the concept of *Scope*. A scope is defiened 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 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

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

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-specifc 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 provided by the components. RDF-based access control in LD-R applications operates at four different granularities provided by the Dataset, Resource, Property and Value component levels and scopes. For example, 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 an LD-R component 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` ontology, can automatically expose the product data as a 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:

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

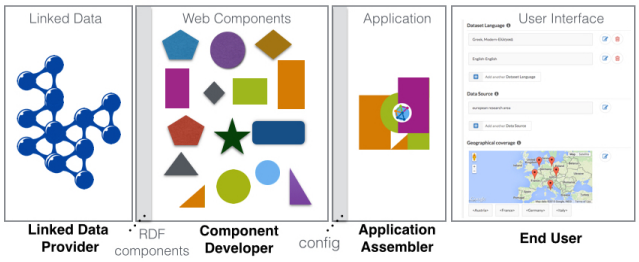

Figure 5: LD-R components life cycle.

- Automatic markup generated by parsing component package specification – metadata about the component and its dependencies. It inlcudes 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 author which addresses metadata such as component level (dataset, resource, property, value), granularity (individual, aggregate), mode (view, edit, browse) and config parameters.

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 5, 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 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. ambigious 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 profficiency 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.

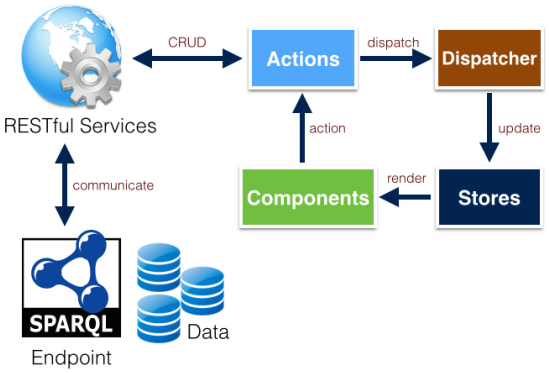

Figure 6: Data flow in the Linked Data Reactor framework.

- *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 faciliates the discovery of relavant 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 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 Linked Data-driven Web components, we implemented a 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 and Flux[8] architecture, Yahoo!'s Fluxible[9] framework for isomorphic Web applications and Semantic-UI[10] framework for flexible UI themes.

The main reasons we chose *React* components over other existing solutions (e.g. Polymer[11], AngularJS[12], EmberJS[13], etc.) were the maturity of the technology, maintainablity, number of developer tools/components/applications, and efficiency[14]. As shown in Figure 6, LD-R follows the Flux architecture which eschews MVC (Model-View-Controller)

[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.

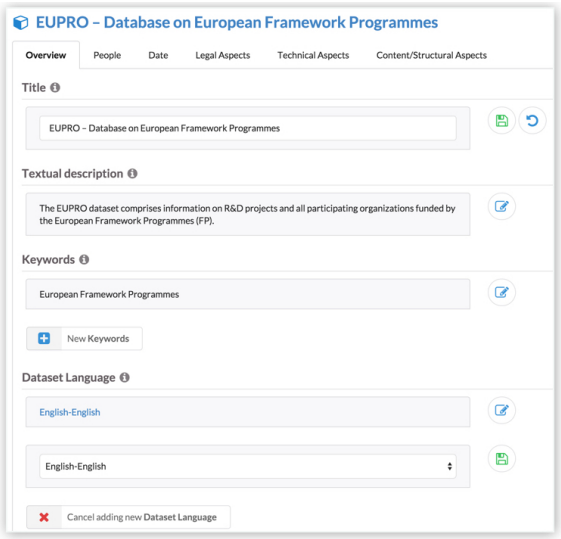

Figure 7: Screenshot

in favor 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.

metadata about components (https://github.com/ali1k/ld-r-metadata-generator) in JSON-LD

6. EVALUATION

two types of evaluations:

2. evaluating the solution provided by us: RISIS OpenPhacts

7. DISCUSSION

8. RELATED WORK

Web Components and the Semantic Web [2]

Semantic Web Services

Existing tools to view/edit and browse LD e.g. OntoWiki, Saha

9. CONCLUSION

Web components aim to bring *Component-Based Software Development* (CBSD) to the World Wide Web. Some advantages of CBSD approach are reusibility, replacability, extensibility, encapsulation and independence. LD-R approach not only facilitates the discovery and reuse of Web components but also makes the creation of Linked Data application easier.

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

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

[8] 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.