A Semantic Web Enabled Form Model and RESTful Service Implementation

Armin Haller and Florian Rosenberg
CSIRO ICT Centre
GPO Box 664, Canberra, ACT 2601, Australia
firstname.lastname@csiro.au

***Abstract*—We propose an RDF-based user interface language called RaUL and a RESTful service called ActiveRaUL that interprets the language and produces XHTML+RDFa in return. The RaUL markup language separates the purpose (data publishing) of a Semantic Web form from its presentation. ActiveRaUL operates and generates RaUL markup, that separates the control elements on a Web form from the data that the form controls operate on. The relation between the form controls and the data model is achieved through a data binding mechanism. The form elements include references to an RDF graph defining the data. The rendering of the instances of a RaUL model on the client-side are displayed as XHTML+RDFa elements.**

I. Introduction

Linked Data technologies have rapidly gained momentum in the last years as a way to create typed links between data from different sources [1]. The uptake of Linked Data technologies has lead to the extension of the Web with a public Linked Data space. This data space is increasingly being used for both research efforts and real-world applications. Linked Data can be seen as a step on the road to the Semantic Web [2] by providing a method for publishing data. This encourages reuse, reduces redundancy and maximizes the inter-connectedness of its data. Currently, HTML is the predominant way of publishing information on the Web. However, this information is mostly semi-structured and does not provide a means to publish structured information. To overcome this limitation, RDFa [3] proposes a language that allows the user to embed structured information in the format of RDF (Resource Description Framework) triples within HTML documents. However, there is still a gap between representing structured information using RDFa and processing this information in a Web application. In traditional Web applications this problem is addressed by a number of different frameworks that bind a relational data model or object-oriented data model to templates (e.g. Ruby on Rails[1], GWT[2], Apache Struts[3], Sprin[4] etc.). Another approach for a data binding mechanism is XForms [4], a model that replaces traditional Web forms with an XML-based model. XForms were introduced to separate sections that describe what the form does (data editing/publishing), and how to render the form. It includes a form model in XML that is bound to data described in XML. With the advent of RDFa and Linked Data a similar framework is needed that separates the data from its presentation. Currently, when publishing Linked Data one needs to first define a Web page in HTML and then uses RDF to annotate the data with semantic concepts. Using RDF to present data requires a mechanism to bind data to the presentation. In this paper we present an RDF-based user interface language that separates the purpose (data publishing) of a Semantic Web application from its presentation. This markup language will be used to describe the structure of the model and separates this markup from the presentation layer. This approach allows the definition of the presentation layer independent of the underlying model and as such, allows the user to maintain both, the model and its presentation in the editing tool. For rendering the user interface, we need to develop a processor that generates the presentation logic and the data from the markup language.

In this paper we introduce RaUL, the RDFa User Interface Language, and a generic RESTful Web service (ActiveRaUL) that generates a user interface in form of a XHTML+RDFa page from RaUL. The RaUL model (ontology) itself is described in RDF(s). This rendering allows the created page to be displayed in any Web browser. When the user fills a Web form and submits the data to the server a generic JavaScript processor parses the file, extracts the RDF triples and binds the input values to the RDF model and forwards the data to the ActiveRaUL Web service. The back-end only deals with RDF triples. Since the user input is automatically bound to the respective RDF concepts there is no need to write glue code that binds the data to the form. An RDF triple store manages all application specific RDF data and acts as the main input for dynamically generating the client user interface pages. Although we chose a particular rendering model, RaUL has the capability to work in a variety of ways to display the model in the user interfaces and to transfer the data to the server. A different approach to what we propose in this paper would be to create RDF/XML instead of XHTML+RDFa and include AJAX rendering code on the client that translates it for displaying in the browser.

The remainder of the paper is structured as follows. In Section II, we compare our solution to related work. Section

[1]http://rubyonrails.org/
[2]http://code.google.com/webtoolkit/
[3]http://struts.apache.org/
[4]http://www.springsource.org/

III presents a motivating example which is modeled throughout the paper in our RaUL language. Section IV introduces the RaUL language, including the *form model* and the data binding mechanism. Section V presents ActiveRaUL, its architecture and the interaction model. In Section VI we benchmark the overhead of the semantic annotations introduced by RaUL. We conclude and discuss future work in Section VII.

II. Related Work

Automatically generating annotations from RDF ontologies is a relatively new research topic in the Web engineering realm. We are aware of some earlier attempts concerning form-based editing of RDF data [5] as well as mapping between RDF and forms [6]. None of the approaches proposes a generic RESTful Web service to seamlessly combine data binding with the processing and generation of semantic annotations in Web applications.

In [7], [8] the authors proposed a read/write-enabled Web of Data through utilizing RDForms [9]. It provides a way for a Web browser to communicate structured updates to a SPARQL endpoint. RDForms consists of an XHTML form, annotated with the RDForms vocabulary[5] in RDFa [3], and an RDForms processor that gleans the triples from the form to create a SPARQL Update[6] statement, which is then sent to a SPARQL endpoint. The difference to our approach is that RDForms does not propose an ontology for form controls and it is bound to a domain-agnostic model – that is, it describes the fields as key/value pairs – requiring a mapping from the domain ontology (FOAF, DC, SIOC, etc.).

Dietzold [10] propose a JavaScript library, which provides a way for viewing and editing RDFa semantic content independently from the rest of the application. Further, they propose update and synchronization methods based on automatic client requests. Their model is restricted to a fixed environment (the Wiki), and they only present the client in-memory modification of the model, but the execution of these atomic add / delete actions as performed in our case by ActiveRaUL is not discussed.

Further more there are other approaches such as SWEET [11], which deals with semantic annotations of Web APIs. Fresnel [12] provides a vocabulary to customize the rendering of RDF data in specific browser. At time of writing, there are implementations for five browsers available.

Related work in regard to the client-side processing of RDFa are the backplanejs libraries [13]. These JavaScript libraries provide cross-browser code for XForms, RDFa, and SMIL as well as a Fresnel integration and jSPARQL, a JSON serialization of SPARQL. This work is complimentary to ours, since with JQuery we reuse an existing RDFa parser for our client-side implementation. The RDFa API provided by the RdfaDomApi, which is part of the backplanejs project, could be used instead of JQuery to implement the binding between the DOM input values and the RDF triples encoded in RDFa. Other approaches related to the client-side part of the RaUL framework are XForms and XUL which propose a mechanism to bind input data to a model, but the model is expressed in XML rather than in RDF. Like RaUL, XForms relies on a client-side AJAX implementation for interpreting the language to generate the XML instances. Multiple XForms compatible JavaScript libraries such as FormFaces, AJAXForms, XSLTForms, Chiba etc. are available. XUL on the other hand is natively supported by Firefox and interpreted in the browser natively. However, there is no support in the other mainstream browsers.

III. Motivating example

Interaction with forms is ubiquitous on the Web as we experience it every day. The following example is essentially a slightly simplified version of the common user registration forms of social networking sites, such as Facebook or MySpace. Figure 1 shows a Web form on the left side and its encoding in pure XHTML on the right side. This motivating example is used in Section IV-A to present RDF statements according to the RaUL vocabulary that encode the same DOM as the one defined by the XHTML statements shown in Figure 1.

IV. The RDFa User Interface Language (RaUL)

The RDFa user interface language provides a standard set of visual controls to define a form in RDF. The RaUL *form controls* are directly usable to define a form model, such as the "registerAccount" form in our motivating example. The ActiveRaUL Web service implements an interface to load, manipulate and post form models to the server. The service returns a page rendered in XHTML+RDFa from the RDF statements for a *form model* (e.g. "registerAccount") and *data instances* (e.g. a particular user) for a particular form model.

The *form model* and *data instances* make RaUL forms more tractable, since a data binding mechanism in the RDF statements and in the generated XHTML+RDFa rendering ensures that the values submitted in a form are linked with the statements defining the data.

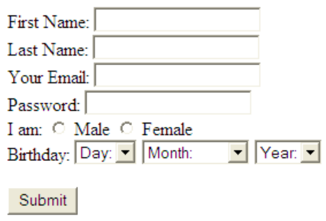
In the following sections we elaborate on the RaUL *form model* that describes how forms are to be presented, and on the *data model*, describing the structure of the data used in the *form model*.

A. Form Controls (RaUL Ontology)

RaUL defines a device-neutral, platform-independent set of *form controls* suitable for general-purpose use. We have implemented a mapping to XHTML form elements. However, the form controls can be bound to other languages

[5]http://rdfs.org/ns/rdforms

[6]http://www.w3.org/TR/sparql11-update/



```
<html>
 <head><title>Registration</title></head>
 <body>
  <form method="post" action="" id="register">
   <span>First Name:</span> <input type="text" class="inputtext" id="firstname" name="firstname" /><br />
   <span>Last Name:</span> <input type="text" class="inputtext" id="lastname" name="lastname" /><br />
   Your Email: <input type="text" class="inputtext" id="reg_email" name="reg_email"/><br />
   Password: <input type="password" class="inputpassword" id="reg_pwd" name="reg_pwd" /><br />
   I am: <input type="radio" name="gender" value="male" /><span>Male</span><input type="radio" name="gender" value="female" /><span>Female</span>
   <br /> Birthday: <select id="birthday_day" name="birthday_day"><option value="0">Day:</option> /* List of days */</select>
   <select id="birthday_month" name="birthday_month"><option value="0">Month:</option> /* List of months */</select>
   <select name="birthday_year" id="birthday_year"><option value="0">Year:</option> /* List of years */</select><br /><br />
   <input type="submit" name="submit" value="Submit" onClick="javascript:parseDom(rdf)" typeof="raul:Button" />
  </form>
 </body>
</html>
```

Figure 1. Example social networking registration form.

than XHTML forms as well. A user interface described in RDF statements according to the RaUL vocabulary is not hard-coded to, for example, display radio buttons as they are in XHTML, but it can be bound to any host language (e.g. proprietary UIs for mobile devices). Our current implementation of the ActiveRaUL processor that is part of the ActiveRaUL Web service supports the rendering of XHTML+RDFa.

A form control in RaUL is an element that acts as a direct point of user interaction and provides write access to the triples in a *data instance*. The controls are bound to the RDF statements defining the *data* via a binding mechanism. Every form control element has a *value* property that is associated with a reified triple or named graph (see section IV-B). Form controls, when rendered by the ActiveRaUL processor, display the underlying data values to which they are bound. While the data presented to the user through a form control must directly correspond to the bound triples describing the *data*, the display representation is not required to match the value of the bound instance data. For example a date can be displayed as 10/10/2008, but bound to a timestamp of type `xsd:string`.

RaUL form controls are defined in an RDF ontology. Figure 2 shows a class diagram like overview of the main classes in the ontology. The full ontology can be found at: http://purl.org/NET/raul#.

Page: The *Page* class defines the main container of a RaUL document. In case of the mapping to XHTML a *Page* instance maps to a `<body>` element. The *Page* class defines some common attributes such as *class*, *id* and *title* which are also inherited by all sub-classes of the *Page* class in RaUL. The *class* and *id* attribute can be used to reference a CSS style identifier to specify the display size of a form control. A *Page* contains one or many *WidgetContainers* which are referenced by a *widgets* property.

Figure 3 shows how a *Page* class is mapped to its XHTML+RDFa representations. This mapping is implemented in our ActiveRaUL processor. The three statements in the RDF graph on the left of the figure (in N3 notation) are mapped to a `head` and `body` element in the XHTML document. The *title* statement is mapped to the `title` element in the `head`, the `raul:id` property is mapped to a

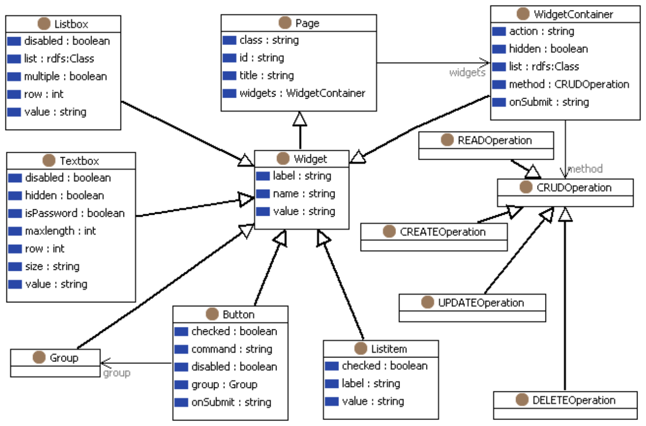


Figure 2. RaUL form model

```
@prefix rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf−schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix raul: <http://purl.org/NET/raul#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/#> .

:registerAccount a raul:Page ;
        raul:title "Registration" ;
        raul:id "maindiv" .
        raul:widgets :registerContainer
```

```
<html xmlns="http://www.w3.org/1999/xhtml"
       xmlns:rdf="http://www.w3.org/1999/02/22−rdf−syntax−ns#"
       xmlns:rdfs="http://www.w3.org/2000/01/rdf−schema#"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
       xmlns:raul="http://purl.org/NET/raul#"
       xmlns:foaf="http://xmlns.com/foaf/0.1/">
<head>
 <meta about="#registerAccount" property="raul:title" content="Registration"
        />
 <title>Registration</title>
</head>
<body>
<div id="maindiv" about="#registerAccount" typeof="raul:Page">
 <span property="raul:id" content="maindiv" />
 <span property="raul:widgets" content="#registerContainer" />
</div>
</body>
```

Figure 3. *Page* RDF triples and the rendering in XHTML+RDFa

`div` container and the `raul:widgets` property references the *#registerContainer*.

***Widget**:* All form controls are a subclass of the *Widget* class inheriting its standard properties, a *label* and a *name*. It also defines the *value* property which is used to associate triples defining the *form model* to the RDF statements defining the *data*.

***WidgetContainer**:* A *WidgetContainer* groups *Widgets* together. The ActiveRaUL processor renders every *WidgetContainer* as a form in XHTML. It defines a *method* and *action* property to define the form submission.

Figure 4 defines the statements for the *WidgetContainer* from our motivating example in RDF N3 notation on the left and the mapping to its XHTML+RDFa rendering on the right side. In the mapping algorithm implemented in the ActiveRaUL processor we make extensive use of the notion of chaining in RDFa which combines statements together to avoid unnecessary repetition of mark-up. For example, we use a `div` container that assigns the subject *#registerContainer* for all subsequent nested statements (ie. *raul:action*, *raul:id*, *raul:method*, *raul:list*).

The ordering of the form controls in a *WidgetContainer* is defined with an RDF collection. When rendering the XHTML+RDFa code, the ActiveRaUL processor determines the positioning of the control elements based on this RDF collection. For example, in our registration form, as described in Figure 4, the *inputbox* (see below) for the *firstname* would be rendered first, followed by a *lastname* input box etc.

```
:register a raul:WidgetContainer;
 raul:id ”register”;
 raul:method :PostRegistrationDetail;
 raul:list :RegisterContainerList (
  :firstname
  :lastname
  :reg_email
  :reg_pwd
  :gender_1
  :gender_2
  :birthday_day
  :birthday_month
  :birthday_year ) .
```

```
<form method=”post” action=”parseDom(rdf)” id=”register” typeof=”raul:
        WidgetContainer”>
<div about=”#registerContainer”>
 <span property=”raul:method” content=”#PostRegistrationDetail” />
 <span property=”raul:id” content=”register” />
 <span property=”raul:list” content=”#registerContainerList” />
  <ol about=”#registerContainerList” typeof=”rdf:Seq”>
   <li rel=”raul:list” resource=”#firstname”/>
   <li rel=”raul:list” resource=”#lastname”/>
   <li rel=”raul:list” resource=”#reg_email”/>
   <li rel=”raul:list” resource=”#reg_pwd”/>
   <li rel=”raul:list” resource=”#gender_1”/>
   <li rel=”raul:list” resource=”#gender_2”/>
   <li rel=”raul:list” resource=”#birthday_day”/>
   <li rel=”raul:list” resource=”#birthday_month”/>
   <li rel=”raul:list” resource=”#birthday_year”/>
  </ol>
</div>
```

Figure 4. *WidgetContainer* RDF triples and the rendering in XHTML+RDFa

***Textbox**:* The *Textbox* form control enables free-form data entry. The constraints on the input type are obtained from the XSD data type of the associated RDF statement defining the data. However, a type check of the input data in the ActiveRaUL controller is future work. The rendering in XHTML generated by the ActiveRaUL processor maps a *Textbox* to an *input* box of type *text*. Properties of the Textbox are, *disabled*, *hidden*, *isPassword*, *maxlength*, *row* and *size*. These properties are straightforwardly mapped to their equivalents in the XHTML model. Figure 8 shows the rendering of the *firstname* textbox from our motivating example.

***Listbox**:* This form control allows the user to make one or many selections from a set of choices. Special properties include *options*, *multiple*, *row* and *disabled*. In case of the *row* and *multiple* properties the rendering in XHTML is straightforward. In both cases it is rendered as a *select* input box displayed as a multirow selection box (*row*) and with the ability to select more than one value (*multiple*).

The *options* property is used to associate the *Listbox* to a collection of *Listitems*. *Listitems* are required to define a *value* and *label* property and can be defined as *checked*. Figure 5 defines the “birthday_day” *Listbox* from our motivating example (please note that for space considerations only two *Listitems* (ie. #birthday_day_1 and #birthday_day_2) are shown).

```
<span about=”#birthday_day” typeof=”raul:Listbox”>
 <span property=”raul:id” datatype=”xsd:string” content=”birthday_day”/>
 <span property=”raul:label” datatype=”xsd:string” content=”Birthday”/>
 <span property=”raul:value” content=”#valuebirthday_day”/>
 <span property=”raul:options” content=”#birthday_day_options”/>
</span>

<ol about=”#birthday_day_options” typeof=”rdf:Seq”>
 <li style=”display:none;” rel=”raul:options” resource=”#birthday_day_1”/>
 <li style=”display:none;” rel=”raul:options” resource=”#birthday_day_2”/>
</ol>

<span about=”#birthday_day_1” typeof=”raul:Listitem”>
 <span property=”raul:label” content=”Day:”/>
 <span property=”raul:value” content=”0”>
</span>
<span about=”#birthday_day_2” typeof=”raul:Listitem”>
 <span property=”raul:label” content=”Day:”/>
 <span property=”raul:value” content=”1”>
</span>
```

Figure 5. XHTML+RDFa rendering of the “Birthday” *Listbox*.

After the submission of the *Listbox* control element the JavaScript processor creates a *checked* relation for all selected *Listitems*. If the *Listbox* is a multi select one, defined by its multiple property, the referenced reified triple in the value property must be an RDF collection.

***Button**:* The *Button* form control is used for actions, either triggering or non-triggering. Beyond the common attributes inherited from its superclasses it defines the following properties, *checked*, *command*, *disabled* and *group*. The mapping to XHTML creates either a normal push button (ie. an input field of type *button*) or a submit button in case the *command* property is set to “submit”. A submit button is used to trigger the action defined in the form element.

The *Radiobutton* and the *Checkbox* are subclasses of a *Button* in the RaUL vocabulary. They are mapped to their respective counterparts in XHTML as input controls of type *radio* or of type *checkbox*, respectively. Whereas radiobuttons let a user select only one option, checkboxes let a user select one or more options of a limited number of choices. To group buttons together and determine the selected values, the group property of the *Button* class can be used. Figure 6 shows the “gender selection” *Radiobutton* from our motivating example.

```
<span about="#gender" typeof="raul:Group">
 <span about="#gender_1" typeof="raul:Radiobutton"
       typeof="raul:Radiobutton">
  <input type="radio" name="gender" value="male"/>
  <span property="raul:label">Male</span>
  <span property="raul:value" content="#valuegender"/>
  <span property="raul:name" content="male"/>
  <span property="raul:group" content="#gender"/>
 </span>
 <span about="#gender_2" typeof="raul:Radiobutton"
       typeof="raul:Radiobutton">
  <input type="radio" name="gender" value="male"/>
  <span property="raul:label">Female</span>
  <span property="raul:value" content="#valuegender"/>
  <span property="raul:name" content="female"/>
  <span property="raul:group" content="#gender"/>
 </span>
</span>
```

Figure 6. XHTML+RDFa rendering of the “Select Gender” *Radiobutton*.

After the submission of the *Button* control element the JavaScript processor creates a *checked* relation for all selected *Checkboxes* or for the selected *Radiobutton* and assigns the value defined for the button to the RDF statement referenced in the property value (e.g. #valuegender). If the *Button* is of type *Checkbox*, the referenced reified triple in the value property must be an RDF collection.

***CRUDOperation**:* Each RaUL *WidgetContainer* permits one *CRUDOperation*, such as *adding* the content of a field as an RDF statement to the model. In order to understand which *WidgetContainer* is mapped to which RDF statement, we must flag it somehow. The process of defining which CRUD operation should be applied on which fields is called binding and is described in section IV-B.

The CRUD operations are mapped onto the HTTP verbs (`CREATE` = HTTP POST, `READ` = HTTP GET, `UPDATE` = HTTP PUT, `DELETE` = HTTP DELETE) and processed in the ActiveRaUL Web service.

B. Data model

The purpose of XHTML forms is to collect data and submit it to the server. In contrast to the untyped key/value pairs in XHTML forms, in RaUL this data is submitted in a structured way as RDF data according to some user defined schema and typed with XSD schema types. The data, defined as reified RDF statements, is also encoded in the generated XHTML+RDFa document. This approach gives the user full flexibility in defining the structure of the model. Empty `rdf:object` fields serve as place-holders in the RDF statement describing the data for a control element (see the rdf:object property in Figure 7) and are filled at runtime by the JavaScript processor with the user input. Initial values can be provided in the `rdf:object` field which is used by the ActiveRaUL processor in the initial rendering to fill the value field of the XHTML form. The subject of the reified triple (ie. http://raul.deri.ie/forms/registerAccount#123) was created by the ActiveRaUL Web service at submission time of the form model (i.e. when executing the POST method of the ActiveRaUL service).

In our motivating example, the data in the *firstname* element is of type `foaf:firstName`, defined in the FOAF ontology [14], a vocabulary to describe persons, their activities and their relations to each other. We use RDF reification to associate this triple in the RaUL *form model* in the *firstname textbox*. Reification in RDF describes the ability to treat a statement as a resource, and hence to make assertions about that statement. Listing 7 shows how the triple *<http://raul.deri.ie/forms/registerAccount#123><foaf:name><“”>* is described as a resource itself using RDF reification.

```
<span about="#valuefirstname">
 <span rel="rdf:subject" resource="http://raul.deri.ie/forms/registerAccount
       #123" />
 <span rel="rdf:predicate" resource="foaf:firstName" />
 <span rel="rdf:object" resource="" />
</span>
```

Figure 7. RDFa reified triple stating a foaf:firstName relation for the #valuefirstname.

This new resource is then associated in the *form model* with the `raul:value` property as shown in Listing 8.

```
<span about="#firstname">
 <span property="raul:label">First Name:</span>
 <span property="raul:value" content="#valuefirstname" />
 <span property="raul:class" content="inputtext" />
 <span property="raul:id" content="firstname" />
 <span property="raul:name" content="firstname" />
</span>
```

Figure 8. Value association in the *firstname* textbox.

The JavaScript processor directly submits the data collected as RDF triples. It keeps track of the state of the filled form through this instance data. Initial values for the instance data may be provided or left empty. As such, the data model essentially holds a skeleton RDF document that gets updated as the user fills out the form. It gives the author full control on the structure of the submitted RDF data model, including the reference of external vocabulary. When the form is submitted, the instance data is serialized as RDF triples.

V. Architecture & Interaction

In this section we elaborate in detail on the architecture and interaction model of our proposed solution. The general overview of the proposed architecture is depicted in Figure 9. The architecture follows the Model-View-Controller (MVC) pattern [15] with the following components:

- **RDF** as the general *model*, whereby the RaUL ontology defines the form model and arbitrary RDF statements defining the data that the form controls operate on. The model is stored and managed in an RDF triple store and can be accessed via an API;
- the **ActiveRaUL processor** that controls the *view* part generating the XHTML+RDFa rendering that is forwarded to the
- **ActiveRaUL Web service** acting as the *controller* that operates upon the CRUD operations specified in the request and instructs the action on the model.

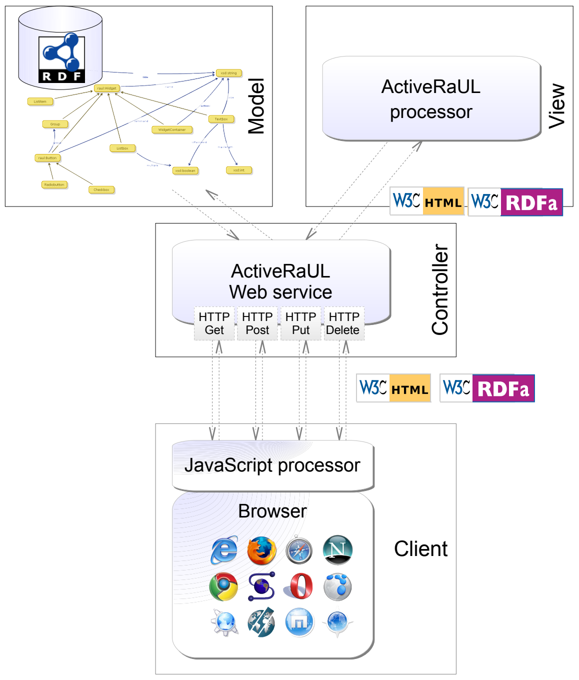


Figure 9. Architectural approach

The general sequence of interactions starts with an HTTP request from the client. The request submit action invokes the *parseDom(rdf)* function of the AJAX client which handles the *RDFa parsing*. The request is processed by a generic JavaScript RDFa parser on the client that parses the XHTML DOM and extracts the RDF content together with the specified HTTP request method. The HTTP requests corresponding to the CRUD operation on a widget element are executed, ie. one of the following: `CREATE`, `READ`, `UPDATE` or `DELETE` as defined in the RaUL vocabulary, along with its binding data model. The content is then encoded in a CRUD operation in a RESTful manner and submitted to the ActiveRaUL Web service. The mapping of the four HTTP terms to its corresponding Web service operations and the sequence of interactions are shown in the diagram in Figure 10.

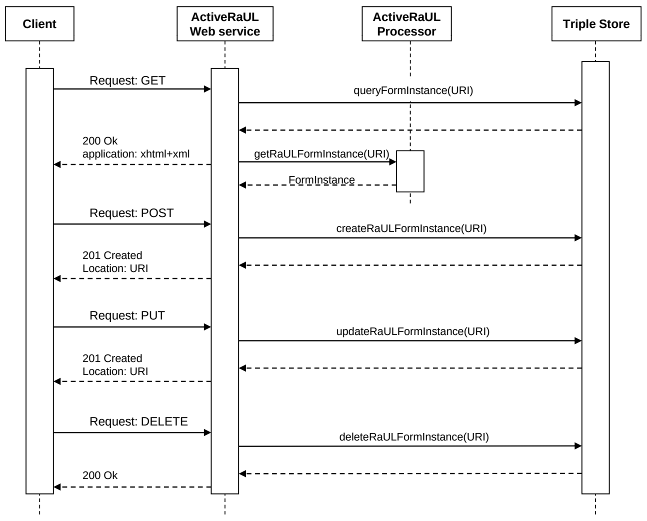


Figure 10. CRUD operations and sequence of interaction

The **ActiveRaUL Web service** handles the *RDF input* and the related HTTP response message. The input content, which contains the structure/form and instance information which are necessary, will be validated and processed according to the domain-logic (implemented in the back-end application by the service provider). The response of this process is modeled again in RDF and forwarded to the ActiveRaUL processor which controls the rendering of the response view.

The **ActiveRaUL processor** controls the presentation of the response at the server side. The output RDF of the controller is rendered by the ActiveRaUL processor and the generated XHTML+RDFa representation is streamed back to the client. This allows not only browsers to interact with the server but also Web crawlers or automated agents to ingest the data since they receive the full XHTML+RDFa content. A client-side implementation with JavaScript is also possible, but would not enable this feature out-of-the box.

VI. Evaluation

To evaluate the performance of our approach it is of interest how much overhead in the data size is added by the RDFa annotated forms. The processing and parsing speed of the page content depends on the file size wrt. to two factors. 1) The resulting upload and download times (file size / available bandwidth) and 2) how efficient an RDFa parser can handle the content; the RDFa parser used in our implementation – as with most other RDFa parsers – needs to parse the whole DOM structure into memory to recreate the RDF structure.

In [16] we have evaluated the overhead caused by the semantic annotations of form elements with the RaUL vocabulary. The overhead we measured is relevant for the response message, as it contains the XHTML+RDFa content in the body of the HTTP response and it influences the parsing time as the client-side RDFa parser is required to parse the whole DOM tree. In the following we focus on the additional data transferred over the wire for HTTP requests. We need to distinguish between the different HTTP methods with respect to the data transferred. DELETE actions do not add any data overhead, since the message body is empty in the request as well as in the response message. GET requests are unchanged as well, only a URI is send, similar to DELETE requests. The additional overhead for GET responses corresponds to the size of the annotations as described in [16], since the message body returned contains XHTML plus the RDFa annotations according to the RaUL vocabulary.

PUT and POST actions cause the same overhead since on the client-side it is not known which form elements have been updated in the case of a PUT request and thus all elements have to be resend as with the POST request. For PUT and POST actions the overhead depends on the method called and we can distinguish two cases: 1) Creating or updating a new RaUL form model instance (e.g. creating the “registerAccount” form), and 2) creating or updating data of a RaUL form model instance (e.g. submitting or editing a user account). We compare the data transferred over the wire for those two methods with an implementation in pure XHTML. Table I shows the overhead for the upload of form elements in XHTML compared with their serialization in RDF/XML when calling the “createRaULFormInstance” or “updateRaULFormInstance” method of the ActiveRaUL Web service.

Minimal (min) in the table denotes the minimal RaUL model to upload and create the respective form element (or page). The maximal (max) value denotes a model that uses all properties of the respective form element in its annotation. The number of triples column denotes how many RDF triples are required in the backend and are encoded in the resulting Web page as annotations. Whenever instance identifiers are required in the RDFa annotations we assumed a two digit identifier (which allows a page to include at least 3844 identifiers if we consider case sensitive alphanumeric combinations).

Results: The serialization of form elements in RDF/XML causes, depending on the element, an overhead of minimal 69% and maximal 159%. The RDF/XML send over the wire to the ActiveRaUL Web service is generated from the RDFa annotations by the client-side JavaScript processor. Although the overhead seems to be significant in size when all properties of a form control element are used, for the minimally required annotations, for all elements, but the listbox, the size of the HTTP POST or PUT request does not even double. In comparison, stylesheet annotations in HTML also cause significant overhead, but still the majority of website’s use CSS.

As there are potentially many form controls in a Web form, the user has the trade-off between the depth of the annotations and the size they consume. The more triples are used for a form control the richer its annotations. However, adding the structure of the page as semantic relations (RDF statements) yields the benefits we described earlier which are:

- Support for full machine understandable structured form data;
- Structured data is encoded directly in the Web page and usable to any Semantic Web application;
- No XHTML manipulation in the backend required;
- Full manipulation freedom of form controls in the backend in RDF;
- Browser agnostic approach via rendering in XHTML + RDFa.

Further, once a form model is created or updated, when adding data, ie. when a user or agent is filling out a form, only the instance data and the reference to its corresponding form model is transferred. This case corresponds to point 2) of our evaluation criteria as described above. Table II shows the total size in bytes of a POST or PUT request including one filled form element encoded as “application/x-www-form-urlencoded” data for HTML forms and as RDF/XML or JSON for RaUL models. For the evaluation we assumed the values to be a three digit alphanumeric string. Further, as seen in Figure 11 we used a `foaf#firstName` for the property of the triple and defined the subject to be a URI in the http://raul.deri.ie domain. Although the size of the RDF/XML and JSON encoding is considerably bigger, as there are typically only a few fields to fill in a form, the absolute size of the HTTP request in kilobyte will be in most cases in a low two digit range. Further, instead of untyped key/value pairs the submitted data is typed through an ontological model and unambiguously defined with URIs.

	XHTML	RDF/XML	JSON
Size (in byte)	13	559	547

Table II
HTTP POST/PUT REQUEST SIZES.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22−rdf−syntax−ns#">
 <rdf:Description rdf:nodeID="123">
  <rdf:subject rdf:resource="http://raul.deri.ie/forms/123#123"/>
  <rdf:predicate rdf:resource="http://xmlns.com/foaf/0.1/firstName"/>
  <rdf:object />
  <rdf:type resource="http://www.w3.org/1999/02/22−rdf−syntax−ns#
        Statement"/>
 </rdf:Description>
</rdf:RDF>
```

Figure 11. RDF/XML submission data for a form field.

VII. Conclusion and Future Work

In this paper we introduced ActiveRaUL, a generic RESTful Web service for managing Semantic Web enabled forms and their processing. ActiveRaUL operates on RaUL, an

Form Element	XHTML		RDF/XML		# triples		Overhead in %	
	min	max	min	max	min	max	min	max
Page	234	284	397	542	1	5	%69.65	%90.84
WidgetContainer	278	321	525	617	3	6	%88.84	%92.21
Textbox	239	298	454	613	2	7	%89.95	%105.7
ListBox	274	323	656	758	5	9	%139.41	%134.67
Button	245	278	459	720	2	8	%87.34	%158.99

Table I
ADDED OVERHEAD BY RDFA MARKUP IN HTTP POST/PUT REQUESTS FOR A FORM SUBMISSION/UPDATE.

RDFa user interface language, which provides a standard set of visual controls that are replacing the XHTML form controls. The RaUL form controls are directly usable to define a web page in the back-end with RDF statements according to the RaUL ontology. RaUL form controls separate the functional aspects of the underlying control from the presentational aspects. The data expressed as RDFa triples is referenced from the *form model* via a data binding mechanism. For the rendering of the instances of a RaUL model we propose ActiveRaUL, a Web service that generates XHTML+RDFa elements for displaying the model on the client. When data is submitted to the server a a client-side Javascript function parses RDFa and submits the extracted RDF triples to the ActiveRaUL Web service. Summarized, the advantages of RaUL in comparison to standard XHTML forms are:

1) **Non-ambiguous model:** Instead of untyped key value pairs the submitted data is typed through an ontological model.
2) **RDF data submission:** The data submitted to the ActiveRaUL service is encoded in RDF.
3) **Explicit form structure:** The form elements are explicitly modeled as RDF statements. The back-end can manipulate and create forms by editing and creating RDF statements only.
4) **External schema augmentation:** This enables the RaUL web application author to reuse existing schemas in the modeling of the input data.

Future Work. Our current RaUL vocabulary (ontology) covers form controls only. As a future work we intend to provide a widget language that encompasses the entire DOM tree of XHTML. We also plan to build the ActiveRaUL Web service in a way that it consumes and produces different RDF serializations.

REFERENCES

[1] C. Bizer, T. Heath, and T. Berners-Lee, "Linked Data – The Story So Far," *International Journal on Semantic Web and Information Systems (IJSWIS)*, 2009.

[2] N. Shadbolt, T. Berners-Lee, and W. Hall, "The Semantic Web Revisited," *IEEE Intelligent Systems*, vol. 21, no. 3, pp. 96–101, 2006.

[3] B. Adida, M. Birbeck, S. McCarron, and S. Pemberton, "RDFa in XHTML: Syntax and Processing," http://www.w3.org/TR/rdfa-syntax/, W3C Semantic Web Deployment Working Group, W3C Recommendation 14 October 2008, 2008.

[4] "XForms Working Groups," http://www.w3.org/MarkUp/Forms/, 2010.

[5] M. Baker, "RDF Forms," http://www.markbaker.ca/2003/05/RDF-Forms/, 2003.

[6] B. de hOra, "Automated mapping between RDF and forms," http://www.dehora.net/journal/2005/08/automated_mapping_between_rdf_and_forms_part_i.html, 2005.

[7] O. Ureche, A. Iqbal, R. Cyganiak, and M. Hausenblas, "On Integration Issues of Site-Specific APIs into the Web of Data," in *Semantics for the Rest of Us Workshop (SemRUs) at ISWC09*, Washington DC, USA, 2009.

[8] T. Berners-Lee, R. Cyganiak, M. Hausenblas, J. Presbrey, O. Sneviratne, and O.-E. Ureche, "On Integration Issues of Site-specific APIs into the Web Of Data," DERI, NUI Galway, Ireland, Tech. Rep., 2009.

[9] M. Hausenblas, "RDForms Vocabulary," http://rdfs.org/ns/rdforms/html, 2010.

[10] S. Dietzold, S. Hellmann, and M. Peklo, "Using javascript rdfa widgets for model/view separation inside read/write websites," in *Proceedings of the 4th Workshop on Scripting for the Semantic Web*, 2008. [Online]. Available: http://www.semanticscripting.org/SFSW2008/papers/15.pdf

[11] M. Maleshkova, C. Pedrinaci, and J. Domingue, "Semantic Annotation of Web APIs with SWEET," in *Proceedings of the 6th Workshop on Scripting and Development for the Semantic Web*, 2010.

[12] "Fresnel, Display Vocabulary for RDF," http://www.w3.org/2005/04/fresnel-info/, 2005.

[13] M. Birbeck, "backplanejs," http://code.google.com/p/backplanejs/, 2010.

[14] D. Brickley and L. Miller, "FOAF Vocabulary Specification 0.91," Namespace Document, Nov. 2007. [Online]. Available: http://xmlns.com/foaf/spec/

[15] T. Reenskaug, *The original MVC reports*, February 2007.

[16] A. Haller, J. Umbrich, and M. Hausenblas, "RaUL: RDFa User Interface Language – A data processing model for web applications," in *Proceedings of the 10th International Conference on Web Information Systems Engineering*, 2010.