RaUL: RDFa User Interface Language – A data processing model for web applications

Armin Haller[1], Jürgen Umbrich[2], and Michael Hausenblas[2]

[1] CSIRO ICT Centre, Canberra, Australia
armin.haller@csiro.au
[2] DERI Galway, Ireland
firstname.lastname@deri.org

Abstract. In this paper we introduce RaUL, the RDFa User Interface Language, a user interface markup ontology that is used to describe the structure of a web form as RDF statements. RaUL separates the markup of the control elements on a web form, the *form model*, from the *data model* that the form controls operate on. Form controls and the data model are connected via a data binding mechanism. The form elements include references to an RDF graph defining the data model. For the rendering of the instances of a RaUL model on the client-side we propose ActiveRaUL, a processor that generates XHTML+RDFa elements for displaying the model on the client.

1 Introduction

Traditional Web applications and in particular Web forms are the most common way to interact with a server on the Web for data processing. However, traditional web forms do not separate the purpose of the form from its presentation. Any backend application processing the input data needs to process untyped key/value pairs and needs to render HTML or XHTML in return. For XHTML, XForms [3] was introduced to separate the rendering from the purpose of a Web form. With the advent of RDFa [4], a language that allows the user to embed structured information in the format of RDF [12] triples within XHTML documents, the presentation layer and metadata layer are similarly interweaved. Currently, when annotating Web pages with semantic information, the user first needs to define the structure and presentation of the page in XHTML and then use RDFa to annotate parts of the document with semantic concepts. The data processing in the backend requires a method to bind the data input to its presentation. For example, an input field for a first name has to be bound to an RDF triple stating a `foaf:firstname` relation over this property. When returning data from the server, the back-end application also needs to deal with and create XHTML as well as RDFa code.

In this paper we introduce RaUL, the RDFa User Interface Language, a model and language that enables to build semantic Web applications by separating the markup for the presentation layer from the data model that this presentation layer operates on. The RaUL model (ontology) itself is described in RDF(s). For the rendering of the RaUL model instances on the client-side we propose ActiveRaUL, a processor that generates XHTML+RDFa elements for displaying the model on the client.

2 Related Work

Annotating forms with semantics 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 [9].

In [16, 6] we proposed a read/write-enabled Web of Data through utilizing RDForms [11], 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[3] in RDFa [4], and an RDForms processor that gleans the triples from the form to create a SPARQL Update[4] statement, which is then sent to a SPARQL endpoint. However, RDForms 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.) and hence is not able to address all the use cases we have in mind.

Dietzold [10] propose a JavaScript library, which provides a more extensive way for viewing and editing RDFa semantic content independently from the remainder of the application. Further, they propose update and synchronization methods based on automatic client requests. Though their model is rich and addresses many use cases, it is restricted to a fixed environment (the Wiki), and hence not generally applicable.

Further, there are other, remotely related works, such as SWEET [13], which is about semantic annotations of Web APIs, as well as Fresnel [1], providing a vocabulary to customize the rendering of RDF data in specific browser (at time of writing, there are implementations for five browsers available). Eventually, we found backplanejs [7] appealing; this is a JavaScript library that provides cross-browser for XForms, RDFa, and SMIL as well as a Fresnel integration and jSPARQL, a JSON serialization of SPARQL.

3 Motivating example

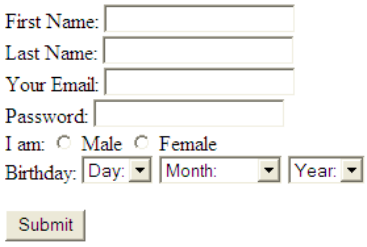
Interaction with forms is ubiquitous on the Web as we experience it every day. From ordering a book at Amazon over updating our Google calendar; from booking a flight via Dohop or filing bugs; from uploading and tagging pictures on Flickr to commenting on blog posts.

The example in Figure 1 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. We will use this example to exemplify the data binding in RaUL in Section 4.1.

4 The RDFa User Interface Language (RaUL)

The RDFa User Interface Language 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. The

[3] `http://rdfs.org/ns/rdforms`
[4] `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="sex" value="male" /><span>Male</span>
  <input type="radio" name="sex" value="male" /><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:
        parseAndSend('body')" typeof="raul:Button" />
 </form>
 </body>
</html>
```

Fig. 1. Example Social Networking Registration Form.

automatically rendered webpage in XHTML+RDFa from RDF statements according to the RaUL vocabulary consists of two parts:

1. *Form model*: One or many form model instances which are rendered as an XHTML page including RDFa annotations describing the structure of the form according to the RaUL vocabulary and
2. a *data model* describing the structure of the exchanged data, also expressed as RDFa annotations which are referenced from the *form model* instance via a data binding mechanism.

The *form model* and *data model* parts make RaUL forms more tractable by referencing the values in the forms with the structure of the data defined by an ontology. It also eases reuse of forms, since the underlying essential part of a form is no longer irretrievably bound to the page it is used in. Further, the data model represented by the form can be accessed by external applications, ie. semantic Web crawlers or software agents supporting users in their daily tasks.

4.1 Form model

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, similar to the design of XForms the form controls can be bound to other languages than XHTML forms as well. A user interface described in RDF triples 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). As mentioned, our current implementation of the ActiveRaUL processor 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 the *data model*. The controls are bound to the *data model* via a binding mechanism. Every form control element has a *value* property that is associated to a reified triple or named graph in the *data model* (see Section 4.2). Form controls, when rendered by the ActiveRaUL processor, display the underlying data values to which they are bound.

In the following we describe the RaUL form controls, including their attributes. They are declared using the RaUL form 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#`.

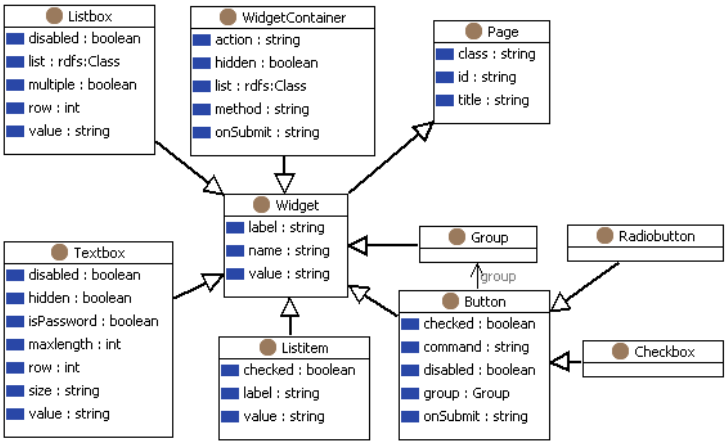


Fig. 2. RaUL form model

Page: The *Page* class acts as the main container for all other content in a RaUL document. In the mapping to XHTML a *Page* instance maps to a <`body`> element.

Widget: All form controls are a subclass of the *Widget* class inheriting its standard properties, a *label* and a *name*. The Widget class includes a *value* property that is used to associate triples in the *data model* to a control element.

WidgetContainer: A *WidgetContainer* groups *Widgets* together. It is similar to a form container in XHTML and in the mapping to XHTML every *WidgetContainer* is rendered as a <`form`> in XHTML. It defines a *method* and *action* property to define the form submission. 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 the ordering in the RDF collection.

Textbox: The *Textbox* form control enables free-form data entry. The constraints on the input type are obtained from the underlying triples associated with the *Textbox* control element. In XHTML a *Textbox* is rendered as 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.

Listbox: This form control allows the user to make one or multiple selections from a set of choices. Special properties include *list*, *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 multi-row selection box (*row*) and with the ability to select more than one value (*multiple*). The *list* 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*.

Button: The *Button* form control is used for actions. 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, input controls of type *radio* or of type *checkbox*, respectively. To group buttons together and determine the selected values, the group property of the *Button* class can be used. For the data binding any *Group* of *Buttons*, ie. either *Radiobuttons* or *Checkboxes*, which values are not literals, must bind to an RDF collection with the same number of node elements as there are *Buttons* in the respective group. After the submission of the *Button* control element the JavaScript processor creates a *checked* relation for all selected *Checkboxes* or for the selected *Radiobutton* denoting the user selection of the respective control element.

4.2 Data model

The purpose of XHTML forms is to collect data and submit it to the server. In contrast to the untyped data in XHTML forms, in RaUL this data is submitted in a structured way as RDF data according to some user defined schema. The data, defined in the backend as an RDF graph, is also encoded in the generated XHTML+RDFa document as statements within a *WidgetContainer*. 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 3) 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.

In our motivating example, the data structure of the *firstname* element is given by the FOAF ontology [8], 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* instance 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 3 shows how the triple *<http://sw.deri.org/ haller/foaf.rdf#ah><foaf:name><"">* which defines the data structure of the #valuefirstname is described as a resource using RDF reification.

This new resource is then associated in the *form model* instance with the `raul:value` property as shown in Listing 4. Initial values for the instance data

```
<span about="#valuefirstname" typeof="foaf:Person">
 <span rel="rdf:subject" resource="http://sw.deri.org/~haller/foaf.rdf#ah" />
 <span rel="rdf:predicate" resource="foaf:name" />
 <span rel="rdf:object" resource="" />
</span>
```

Fig. 3. RDFa reified triple for a foaf:firstname object.

```
<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>
```

Fig. 4. Value association in the *firstname* textbox.

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.

5 Architecture & Interaction

The general architecture as shown in Figure 5 follows the Model-View-Controller (MVC) pattern [14]. It uses a Java servlet container as the controller part, RDF as the general model for the domain logic and the ActiveRaUL Processor as the responsible for the view creation (GUI). A client interacts with RaUL by triggering a submit action of a particular XHTML form. Before the form submission to the server, the *parseAndSend(...)* function of the *JavaScript (JS) Processor* handles the *RDFa parsing* on the client side by extracting the embedded RDF content and the HTTP request method from the DOM (Document Object Model) tree. The implementation of the parser and the data binding is a pure Ajax implementation. Once the request is submitted to the server, the *Controller Servlet* handles the RDF input and the related HTTP response message. The input content (the structure/form and instance information) will be validated and processed according to the domain-logic as implemented in the backend application by the service provider. The response is modeled again in RDF and forwarded to the *ActiveRaUL Processor*. It controls the presentation of the response at the server side by generating the XHTML+RDFa representation. Finally, the Controller Servlet streams the generated data 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.

6 Evaluation

The evaluation is split into two parts: 1) a comparison between existing solutions based on features and 2) benchmarks of the overhead added by the RDFa markup to the existing XHTML forms.

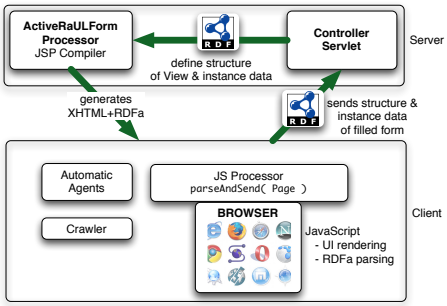


Fig. 5. Architecture diagram.

6.1 Feature comparison

We compare our approach 1) RaUL with 2) the native XHTML form/POST mechanism, 3) RDForms, 4) XForms and 5) XUL.

The first category of features are the *complexity* of the development *on the client and server side*. The complexity on the client-side is only low for XHTML forms as it is supported by all browsers. XForms requires Ajax scripts or browser plug-ins (for Firefox) rendering the data. RaUL and RDForms provide third-party JavaScript libraries which parse and process RDF statements from the RDFa annotations and for the latter transforms the statements into an SPARQL update operation. Processing XUL on the client side requires a high complexity; it is a proprietary language that has to be rendered by client-side code (ie. in Firefox). Looking at the complexity for the server-side handling of the request, XHTML forms have a medium complexity. Although there exists an abundance of CGI scripts that handle the requests and submitted parameters, the data is submitted as untyped key/value pairs only which have to be handled in custom built code. RDForms require a medium complexity since the server has to understand SPARQL update requests. RaUL has a low complexity, since it provides a generic ActiveRaUL service that handles the RDF serialization. The web application developer only needs to deal with triples for which many data abstraction layers exist. The complexity for XForms is low as the data is encoded in XML and for XUL all logic resides on the client-side and thus the server side complexity is low.

The second feature category are general characteristics of the approaches. The first characteristic is that the solution should be *representation agnostic* with respect to the RDF input/output, meaning whatever RDF serialization is supplied (RDF/XML, Turtle, RDFa, microformats+GRDDL), the system should be able to handle it. This is not possible with any language but RDForms. The current implementation of ActiveRaUL only supports RDFa, but in future work we intend to support other formats as well. The second characteristic is the support for an explicit *data model*: XHTML does not define a data model, the data is send as key/value pairs. RaUL and RDForms explicitly define the data model in RDF. XUL and XForms define it in XML. The third characteristic is the *form model* - which is, if the client request contains the form model as structured data. Only RaUL and XUL encode the form model explicitly. XHTML forms do not

	RaUL	XHTML form	RDForms	XForms	XUL
1. Complexity					
1.1 Client	medium	low	medium	medium	high
1.2 Server	low	medium	medium	low	low
2. Characteristics					
2.1 Agnostic	no	no	yes	no	no
2.2 Data Model	yes	no	yes	yes	yes
2.3 Form Model	yes	limited	limited	limited	yes
2.4 Browser support	all	all	all	some	one

Table 1. Comparison of different update interfaces

Form Element	XHTML		XHTML+RDFa		# triples		Overhead in %	
	min	max	min	max	min	max	min	max
Page	39	89	121	279	1	4	210%	213%
WidgetContainer	72	115	251	376	3	6	248%	226%
Textbox	41	100	88	377	1	7	114%	277%
ListBox	49	125	228	459	4	8	365%	367%
Button	48	81	98	415	1	8	104%	412%

Table 2. Comparison of added overhead by RDFa markup. Min and max values in the XHTML and XHTML+RDF columns are the content size in Bytes.

define the form model as structured data, but offer DOM manipulation. RDForms do not define the form structure explicitly either, but the mapping logic between the RDF triples defining the form and the XHTML rendering is hard-coded in the transformation tool. The last characteristic is the *browser support* of the approaches. The first four solutions are browser independent approaches; RaUL, RDForms and XForms also require that the browser supports and enables JavaScript. XUL is only supported by Firefox.

6.2 Overhead Benchmark

From a performance point of view 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 download time (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. Thus, we measured the additional overhead in bytes for the form elements in Table 2.

Minimal (min) in the table denotes the minimal RaUL model to generate 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 *Page* element is considerably bigger than its pure XHTML counterpart (header and body) when rendered in RDFa because of the namespace definition (at least the RaUL namespace has to be defined). However, it is only required once for every page and as such the bytes in the table are the absolute overhead per page. The *WidgetContainer* element is also adding about $2\ ^1/_2$ times the overhead to a pure form container in XHTML. Again, only one *WidgetContainer* for every form is required (typically one per page) and as such the added bytes in the table are in most cases only added once per page. An annotated *Textbox* takes about twice the size of the pure XHTML form and only requires one triple in the backend. Adding all properties (in total 7 RDF statements) to a *Textbox* causes an overhead of about 277%. The *Listbox* control rendered in RDFa adds more than $3\ ^1/_2$ times the size of the pure XHTML form. This is due to the fact that there is at least one Listitem associated to a *Listbox*, which is modeled as a class in RaUL. As such, a Listbox needs at least four statements (RDF triples). Similar to the *Textbox* an annotated *Button* takes about twice the size of the pure XHTML control element and only requires one triple in the backend. Again, adding all properties to a *Button* adds a considerable amount of space (more than four times the pure XHTML element) to the page due to the *Group* class which can be used to associate multiple buttons together (see Section 4.1). However, it also includes 8 triples in the annotation.

Discussion: The overhead seems to be significant in size, especially if all properties of a form control element are used. Whereas a *Textbox* and a *Button* only about double the size of the pure XHTML form element in its minimal configuration, a *Listbox* and a *Button* with all properties defined add about four times the size. Similar to adding div containers and CSS styles to XHTML, adding RDFa increases the size of the file. Since the RDFa annotations are additions to the XHTML code the size of the encoding is influenced by the verbose syntax of the RDFa model. 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 bigger size, though, does not necessarily cause more packages delivered over the wire. However, if the data transfer volume is pivotal, the rendering of the document can be achieved by a JavaScript DOM generation algorithm that operates on pure XML/RDF. Another option is to install a client-side code (similar to the Firefox extensions for XUL) that does the rendering of pure XML/RDF.

7 Conclusion

In this paper we introduced RaUL, the 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 backend with RDF statements according to the RaUL ontology. RaUL form controls separate the functional aspects of the underlying control (the *data model*) from the presentational aspects (the *form model*). The *data model*, which describes the structure of the exchanged data (expressed as RDFa annotations) is referenced from the *form model* via a data binding mechanism. For the rendering

of the instances of a RaUL model on the client-side we propose ActiveRaUL, a processor that generates XHTML+RDFa elements for displaying the model on the client. Only when data is submitted to the server a JSP servlet creates the RDF triples. Summarized, the advantages of RaUL in comparison to standard XHTML forms are:

1. **Data typing:** Submitted data is typed through the ontological model.
2. **RDF data submission:** The received RDF instance document can be directly validated and processed by the application back-end.
3. **Explicit form structure:** The form elements are explicitly modeled as RDF statements. The backend 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 model*.

Acknowledgements This work is part of the water information research and development alliance between CSIRO's Water for a Healthy Country Flagship and the Bureau of Meteorology.

References

1. Fresnel, Display Vocabulary for RDF. `http://www.w3.org/2005/04/fresnel-info/`, 2005.
2. Cross-Origin Resource Sharing. `http://www.w3.org/TR/cors/`, 2010.
3. XForms Working Groups. `http://www.w3.org/MarkUp/Forms/`, 2010.
4. B. Adida, M. Birbeck, S. McCarron, and S. Pemberton. RDFa in XHTML: Syntax and Processing. W3C Recommendation 14 October 2008, W3C Semantic Web Deployment Working Group, 2008.
5. M. Baker. RDF Forms. `http://www.markbaker.ca/2003/05/RDF-Forms/`, 2003.
6. 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. Technical report, 2009.
7. M. Birbeck. backplanejs. `http://code.google.com/p/backplanejs/`, 2010.
8. D. Brickley and L. Miller. FOAF Vocabulary Specification 0.91. Namespace document, Nov. 2007.
9. 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.
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.
11. M. Hausenblas. RDForms Vocabulary. `http://rdfs.org/ns/rdforms/html`, 2010.
12. G. Klyne, J. J. Carroll, and B. McBride. RDF/XML Syntax Specification (Revised). W3C Recommendation, RDF Core Working Group, 2004.
13. 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.
14. T. Reenskaug. *The original MVC reports*, February 2007.
15. RFC2818. HTTP Over TLS. `http://www.ietf.org/rfc/rfc2818.txt`, 2000.
16. 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.
17. XMLHttpRequest. W3C Working Draft. `http://www.w3.org/TR/XMLHttpRequest/`, 2009.