ActiveRaUL: Automatically Generated Web Interfaces for Creating RDF Data

Anila Sahar Butt\textsuperscript{a,b}, Armin Haller\textsuperscript{b,*}, Shepherd Liu\textsuperscript{b}, Lexing Xie\textsuperscript{a}
\textsuperscript{a}Australian National University
Canberra, Australia
E-mail: firstname.lastname@anu.edu.au
\textsuperscript{b}CSIRO Computational Informatics
Canberra, Australia
E-mail: firstname.lastname@csiro.au

Abstract. The amount of automatically generated machine-readable data on the Web has significantly increased in recent years. This is in part due to the advent of Linked Data and its publishing tools that allowed the mapping of relational data to RDF. However, the amount of semantic Web data is still many orders of magnitude smaller than the World-Wide-Web, and this limits semantic Web applications. One of the barriers for semantic Web novices to create machine-readable data is the lack of easy-to-use Web publishing tools that separate the schema modelling from the data creation. In this article we present ActiveRaUL, a Web form-based user interface that particularly supports users inexperienced in semantic Web technologies in creating RDF data. These Web form-based user interfaces in ActiveRaUL can be automatically generated from any arbitrary input ontology through a process described in this article. We map the graph-structured input ontology to a tree-structured Web form while still allowing the user to create RDF data typed according to the input ontology. We validate our approach of automatically generating Web interfaces from an ontology in a user study based on use cases developed by the W3C Semantic Sensor Network (SSN) Incubator group. We test the effectiveness, efficiency and the satisfaction of users in creating RDF data based on the SSN ontology with ActiveRaUL generated user interfaces compared to a state-of-the-art ontology editing tool.

Keywords: Semantic Web application, Form-based User Interface, Widget ontology, Read/Write Linked Data application

1. Introduction

The continuous growth of the Linked Data Web brings us closer to the original vision of the semantic Web - as an interconnected network of machine-readable resources. One of the reasons for the growth of Linked Data has been the significant progress on developing ontologies that can be used to define data in a variety of domains, for example, GO [22] in bioinformatics, FOAF [8] and the schema.org initiative in Web engineering or the Sensor Network Ontology (SSN) [10] in the Internet-of-Things. The tools of choice for creating quality-assured ontology instances (the so-called ABox) are still ontology editors such as Protégé [18]. However, creating the ABox in an ontology editor requires some degree of understanding of RDF(s) and OWL since the user has to define to which class an individual belongs to and what are the permissible relationships between individuals. Further, as ontology editors do not separate the schema editing from the data editing, users can, for example, inadvertently make changes to the classes and relations in the ontology (the so-called TBox) while creating data. Addressing this issue, some Web publishing tools on top of Wikis, Microblogs or Content Management systems have been developed (e.g. the work discussed in [16], [7], [19] and [11]) that allow a user to exclusively create ontology instances. However, they are mostly developed for a specific domain (i.e. specific ontologies) and often do not strictly follow OWL se-
mantics and consequently allow the creation of a logically inconsistent ABox.

Due to the shortage of efficient tools for creating data instances, manually created, quality-assured, crowd-sourced semantic Web datasets are still largely missing. Drawing a parallel to data creation on the traditional Web, most of which happens through Web forms, an analogous method to create data is needed on the semantic Web. An abundance of tools exist to support developers in creating such Web forms operating on a relational database scheme. Many of them also support the Model-View-Controller (MVC) [20] pattern where a developer can generate scaffolding code (Web forms) that can be used to create, read, update and delete database entries based on the initial schema. To create such a Web form-based tool that operates based on an ontological schema, a number of challenges have to be addressed:

- Web form data is encoded in a key/value pair model that is not directly compatible to the triple model of RDF. Therefore, a data binding mechanism is needed that binds the user input in Web form elements to an RDF model.

- Whereas a Web form based on a relational table has a fixed set of input fields based on the number of table columns, the RDF model is graph based with potential cycles. Further, RDF(s) properties are propagated from multiple superclasses (including inheritance cycles) and the types of properties for a class are not constrained by the definition (Open World assumption). Consequently, methods are required to decide on the properties to be displayed in a Web form for a given RDF node.

- In contrast to the relational model where tuples are bound to a relation (table), class membership for individuals in RDF(s) is not constrained for a class. Thus, individuals that have been created as a type of a specific class need to be made available for reuse within a different class instance creation process.

- Beyond the standard datatypes in the relational model that can be easily mapped to different form input elements (e.g. String/Integer to text boxes, Boolean to radio buttons, etc.), the OWL model supports object properties that link individuals to other individuals via URIs. Object properties can also span multiple nodes in an RDF graph, forming a property path, i.e. they can refer to a class that is linked to another class through more than one property. To aid users in the creation of object properties who are unaware of the ontology model, methods have to be established to identify and link to existing individuals and to enable the creation of new individuals in the process of creating the object property.

In previous work [14,13], we developed a Web form ontology, the RDFa User Interface Language (RaUL)\(^1\) and a method to use RDFa, a syntactic format that allows machine-readable data to be easily integrated into HTML Web pages, for binding data in traditional Web form elements to concepts and relations in an RDF graph (i.e. ontologies). We developed ActiveRaUL, a Web service that operates on an RDF template RaUL ontology model that describes the structure and data model of a Web form. ActiveRaUL implements a read-write Linked Data architecture that manages the mapping of a Web form record to an RDF graph, its field names to RDF property types and the user input value to instances of RDF properties. Although the functionality previously implemented in ActiveRaUL solves the data binding problem of Web form records to an RDF graph and enables a user to create RDF data (ontology instances) for a given ontology in a Web form, it still requires expertise in defining the Web form template in RDF according to the RaUL ontology. In this article we present extensions to ActiveRaUL that allow the automatic generation of Web form-based user interfaces from arbitrary input ontologies. Addressing the challenges above, our main contributions are:

- a method to generate a concept graph for each concept in the input ontology that accommodates the different types of implementation models in RDFS/OWL for a relation,

- and a mapping procedure for the different types of property paths in the concept graph to different widget elements in a generated Web form.

We argue that the resulting Web form-based user interfaces are easier-to-use for a semantic Web novice to create RDF data, and result in more accurate ontology instances than creating them through traditional ontology engineering tools. We validate our approach in a user study comparing our system with a state-of-the-art ontology modelling tool. The remainder of this article is structured as follows. First, we discuss systems related to ActiveRaUL in Section 2. We then present a motivating example in Section 3 that we are

\(^1\)See http://purl.org/NET/raul#
using throughout the article to describe our Web form generation process. Section 4 gives a brief introduction into the architecture of ActiveRaUL and its execution semantics. We introduce the notion of a concept graph and outline how we use this intermediary concept graph in our mapping procedure in Section 5. In Section 6 we describe the process of constructing the concept graph from the input ontology. In Section 7 we define the mapping from the concept graph to a RaUL Web form model and we outline how the resulting model is displayed in HTML Web forms. In Section 8 we report on the results of our user study, followed by a conclusion and a discussion on future work in Section 9.

2. Related Work

Many mature ontology editors such as Protégé [18], TopBraid 2, SWOOP 3, the Neon toolkit [12] etc. exist that offer ways to create individuals based on one or many ontologies. Some of these editors have Web-based version that can be used to allow ordinary Web users to hand-craft ontology instances. However, what they have in common is that a user requires at least a basic understanding of the RDF/OWL semantics to create correct individuals according to a given ontology. We have compared ActiveRaUL to WebProtégé, the Web version of the state-of-the-art ontology editor Protégé, in our user study.

There are other tools that support the user specifically in the creation of a knowledge base without the need of knowing RDF/OWL. An early tool that supports the creation of individuals is SEAL [17]. Although it offers a templating mechanism for a specific ontology, it lacks in providing a domain independent solution for automatic interface creation and data binding that we are presenting in this article. Secondly, as the tool is based on F-Logic and the Ontobroker, it does not support the current semantic Web standards such as RDF and OWL.

The RDF instance creator (RIC) [15] lets a user create ontology instances in simple Web forms. Whilst RIC facilitates the users to create individuals without the need to understand RDF/OWL, it supports only simple OWL features (distinguishing between object and data type properties), but ignores all types of OWL property restrictions and does not offer a convenient way of handling object properties.

OntoWiki [2] is another system that is targeted towards ontology instance level editing, but as a Wiki platform it is more of an annotation tool than a tool to create ontology instances according to an input ontology. A newer version of OntoWiki also includes a visual query builder for easier data access [21]. ActiveRaUL in contrast to Wiki-based systems does not only separate the ontology schema editing from the instance level editing, but it also uses the schema for the instance level editing system’s interface generation to enforce better data quality.

In Callimachus [6] developers can modify sample XHTML+RDFa templates to define new classes and their relationships with nested and composite classes. While it facilitates Web 3.0 developers to easily create Web applications based on an ontology, the developer needs to manually define the template and the resulting Web applications do not fully comply with OWL semantics, do not support global cardinality constraints (owl:FunctionalProperty and owl:InverseFunctionalProperty) and logical characteristics of properties (owl:TransitiveProperty and owl:SymmetricProperty). As a consequence instances generated through these forms may be logically inconsistent to the ontology they are modelled after.

LESS [3] enables users to create templates with SPARQL and apply them to existing RDF documents (e.g., FOAF files) to generate user-friendly HTML pages (e.g., online business cards) which will be persisted in the backend database. The semantic information will, however, be lost due to the lack of proper annotation strategies.

RDFa 2 [4] assists users in generating (X)HTML+RDFa pages from existing RDF documents, but at the time of writing it does not support the persistence of RDF triples.

Summarising, the schema editing for RDF/OWL has drastically improved in recent years and tools such as Protégé [18] and TopBraid have reached a level of maturity that is comparably to relational database schema modelling. However, tools specifically supporting the data modelling such as Callimachus, LESS or OntoWiki [2] still require the user to first define templates based on the schema given by an ontology, before RDF data can be created. In ActiveRaUL, these templates are automatically generated through a process that is described in this article.

3See http://code.google.com/p/swoop/
3. Motivating Example

We introduce a motivating example ontology that we will use to illustrate the workings of the Web form generation algorithm in ActiveRaUL. Our example is motivated by FOAF [8], the Friend-of-a-friend vocabulary, an ontology to describe persons, their activities and their relations to other people and objects.

However, we have designed a small “Person” ontology (see Fig. 1) that includes some more complex relations that FOAF does not provide to showcase the following:

1. OWL property restrictions e.g. owl:inverseOf (i.e. worksFor and hasEmployee), owl:Transitive-Property (i.e. supervises) and owl:Functional-Property (i.e. playRole or gender)

2. Complex relations among concepts that include cycles and multiple relations among the same concepts e.g. the relationship between Persons, Roles and Organizations.

3. The implementation of multiple types of modelling that are supported in RDFS/OWL to express the same logical relation. For example the relation, “A person works for an organization” represents a relationship of a person and an organization. The schema model of this relationship (shown on top in Fig. 2) can be implemented in a formal language such as OWL in two ways as shown in (a) & (b) in Fig. 2. In (a) the RDFS/OWL model explicitly defines a “Person” as the domain and an “Organization” as the range for the property “workFor”. In (b) the RDFS/OWL model uses a class restriction in the form of a subClassOf or equivalentClass of a Person class on the property workFor such that only instances of the Organization class can appear as a range for this property.

ActiveRaUL supports both modelling types, the domain range restriction (a) and the single range existential restriction (b).

4. The ActiveRaUL system

The ActiveRaUL system consists of two main components, (1) the ActiveRaUL Web service (2) and the RaUL JavaScript library.
4.1. The ActiveRaUL Web service

The ActiveRaUL Web service implementation (see Fig. 3) follows the Model-View-Controller (MVC) pattern [20].

Model The definition of a Web form based on an ontology constitutes the model part of the ActiveRaUL system. We developed one such ontology, the RDFa User Interface Language (RaUL)\footnote{See http://purl.org/NET/raul/\#}. Web forms can be handcrafted by a developer according to the RaUL ontology as described previously in [14] or they can be created automatically as described in this article in Sec. 6 and 7. A Web form model based on the RaUL ontology consists of two parts:

I. **Widget elements** describing the structure of a Web form. A Web form model is made up of one or many widget elements that act as a direct point of user interaction and provide access to the triples of the referenced RDF graph (data model). Fig. 4 depicts a high-level overview of the RaUL ontology including a set of classes that define different types of widget elements, such as Textboxes, Radiobuttons, Listboxes etc. All widget elements are a subclass of the Widget class inheriting its standard properties, a label, name etc. The Widget class also defines a value property that is used to associate triples in the data model to a widget element. Widget elements can be grouped together on a Web form with several types of RaUL containers, i.e. a WidgetContainer, a Group or a DynamicGroup. The ordering of the widget elements in one of these containers is defined with an RDF collection.

II. A **Data model** defining the structure of the exchanged data as RDF statements which are referenced from the widget elements via a data binding mechanism. Thus, the referenced RDF graph gives meaning to the data used in the widget elements by uniquely referencing concepts and properties in Web ontologies. The actual binding of the widget elements to the underlying RDF data is realised via reification. It has to be noted that we use reification only as a data binding mechanism which is particularly needed for maintaining the semantics in the (X)HTML+RDFa rendering. In the RDF database in the backend we store the reified triple and the reification triples, making it easy to use SPARQL without the need to define complicated queries over the reified triple. Reification is used for the data binding as follows: the rdf:subject triple references the URI assigned to the RDF instance graph by the ActiveRaUL service after submission. The rdf:predicate triple is a reference to the URI of a property in a Web ontology, and the rdf:object triple is a reference to the value that can be edited by the form control the reified triple is referenced from. Empty rdf:object fields serve as place-holders and are filled at run-
time by the RaUL JavaScript client library with the user input.

View  The view in MVC is the part that the user interacts with. ActiveRaUL supports different view representations including RDF/XML, RDF/JSON, RDF/N3 and (X)HTML+RDFa. However, only (X)HTML+RDFa will involve a rendering of the model as a Web form. Since the mapping of the model to (X)HTML+RDFa depends on the underlying form model, a GenericViewProcessor Java interface is provided that defines method signatures that have to be implemented for a particular form model. We provide an implementation of this interface in the ActiveRaULProcessor, that performs a view generation based on the RaUL vocabulary.

Controller  The ActiveRaUL RESTful Web service implements a controller that is responsible for creating, retrieving, updating and deleting Web forms and their referenced data. ActiveRaUL supports different data representations such as RDF/XML, RDF/JSON, RDF/N3 and (X)HTML+RDFa. It uses the HTTP Accept header to determine what kind of representation will be sent back to the client. The controller also implements the functionality to automatically generate a Web form if the input file is a domain ontology (see “RaUL Graph Generation” box in Fig. 3). In Table 1 we summarise all supported HTTP resources, describe their functionality and specify their return values. We omit error handling in this table for brevity. In a nutshell, errors are handled by returning an appropriate HTTP status code.

In this table we distinguish between “Deployment endpoints” and “Usage endpoints”.

Deployment endpoints  The first three endpoints in Table 1 describe the RESTful interfaces that can be used to deploy a Web form. All three endpoints, after execution, will assign a URI to the newly deployed Web form resources and return the URI in the Location header of the HTTP response, for example, /public/forms/person. We refer to these identifiers as formid in Table 1. For the deployment of Web forms, we handle duplicate names by appending unique numbers, e.g., person1. For Endpoint I the payload is required to be a RaUL RDF form model. This endpoint can be used to deploy a handcrafted RaUL Web form file with ActiveRaUL and store it in the RDF triple store. For Endpoints II & III the payload can be any arbitrary RDF ontology file. These endpoints will use the RaUL graph generation library (see Fig. 3) to automatically generate a RaUL RDF form model from the input ontology file using the process described in this article. Endpoint II will create a Web
<table>
<thead>
<tr>
<th>Nr</th>
<th>Resource</th>
<th>Parameters</th>
<th>Method</th>
<th>Data</th>
<th>Server Action</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>/forms</td>
<td></td>
<td>POST</td>
<td>RaUL RDF model, defining the form structure (in JSON, XML or N3)</td>
<td>creates a new form</td>
<td>201 (Created); Location header of new form</td>
</tr>
<tr>
<td>II</td>
<td>/forms</td>
<td></td>
<td>POST</td>
<td>arbitrary RDF ontology</td>
<td>creates a new form</td>
<td>201 (Created); Location header of new form</td>
</tr>
<tr>
<td>III</td>
<td>/forms ?conceptId={id}</td>
<td></td>
<td>POST</td>
<td>arbitrary RDF ontology</td>
<td>creates a new form for the given concept</td>
<td>201 (Created); Location header of new form</td>
</tr>
<tr>
<td>IV</td>
<td>/forms</td>
<td></td>
<td>GET</td>
<td>arbitrary RDF ontology</td>
<td>retrieves all concepts in the submitted ontology</td>
<td>200 (Ok); List of all concepts in ontology and rdfs:comment for each</td>
</tr>
<tr>
<td>V</td>
<td>/forms /{formId}</td>
<td></td>
<td>GET</td>
<td></td>
<td>retrieves the form identified by {formId}</td>
<td>200 (Ok); desired representation as defined in the Accept header</td>
</tr>
<tr>
<td>VI</td>
<td>/forms /{formId}</td>
<td></td>
<td>PUT</td>
<td>RDF triples of updated form</td>
<td>updates the form identified by {formId}</td>
<td>200 (Ok)</td>
</tr>
<tr>
<td>VII</td>
<td>/forms /{formId}</td>
<td></td>
<td>POST</td>
<td>RDF triples containing submission data</td>
<td>creates new form data</td>
<td>201 (Created); Location header of new form data</td>
</tr>
<tr>
<td>VIII</td>
<td>/forms /{formId}</td>
<td></td>
<td>DELETE</td>
<td></td>
<td>deletes the form identified by {formId}</td>
<td>204 (No Content)</td>
</tr>
<tr>
<td>IX</td>
<td>/forms /{formId}?instance={query}</td>
<td></td>
<td>GET</td>
<td></td>
<td>retrieves instances matching the query string for the {formId}</td>
<td>200 (Ok); desired representation as defined in the Accept header</td>
</tr>
<tr>
<td>X</td>
<td>/forms /{formId}/{dataId}</td>
<td></td>
<td>GET</td>
<td></td>
<td>retrieves the form {formId} and its data identified by {dataId}</td>
<td>200 (Ok); desired representation as defined in the Accept header</td>
</tr>
<tr>
<td>XI</td>
<td>/forms /{formId}/{dataId}</td>
<td></td>
<td>PUT</td>
<td>RDF triples of form and form data</td>
<td>updates the data identified by {dataId}</td>
<td>200 (Ok)</td>
</tr>
<tr>
<td>XII</td>
<td>/forms /{formId}/{dataId}</td>
<td></td>
<td>DELETE</td>
<td></td>
<td>deletes the form data identified by {dataId}</td>
<td>204 (No Content)</td>
</tr>
</tbody>
</table>

For example, if the Web form resource URI /public-/forms/person created via the deployment endpoints above is accessed in a browser, a GET request with the Accept header set to text/html is issued which will result in ActiveRaUL returning the Web form in (X)HTML+RDFa as defined in Endpoint V. When a user fills out this Web form and submits it through the RaUL JavaScript client library (see Sect. 4.2), the (X)HTML+RDFa form will be parsed and the object triples updated with the user input data. The resulting RDF/XML is sent to Endpoint VII in order to create an instance graph for a specific formId. After submission, the ActiveRaUL service will process the request, insert the data in the RDF triple store and send the HTTP Location header back to the client. This uniquely identified data can then be accessed independently of the form model by send-
A directed graph $G = \langle U, E \rangle$ where $U$ is a finite set of labeled nodes, $E$ is a finite set of directed labeled edges and $U \subseteq R$ and $E$ is a finite set of directed labeled edges and $U \subseteq R$ and $E$

**Definition 1 (RDF Graph)** A directed graph $G = \langle U, E \rangle$ where $U$ is a finite set of labeled nodes, $E$ is a finite set of directed labeled edges and $U \subseteq R$ and $E$

**Definition 2 (Semantic Association)** A directed path $v \overset{\pi}{\rightarrow} u_m$ between two nodes is a sequence $(v, p_0, u_{i_0}), (u_{i_0}, p_1, u_{j_1}), ..., (u_{i_m-1}, p_m, u_{i_m})$, for $i, j, k, l, m \geq 0$, for which $v$ is the concept node and $u_m$ is the last node. The length of the semantic association $l(v \overset{\pi}{\rightarrow} u_m)$ is the maximum number of consecutive connected edges involved in the sequence.

Semantic associations exhibit multiple distinct property paths based on the structure of the property sequence, and the position of the source concept and the target concept. We distinguish several types of property paths that commonly occur in a semantic association of a concept graph in regards to their mapping to Web forms. The property paths we distinguish are shown in Figure 5.

- **Single-length property path** ($P_{sl}$): We refer to a single-length property path as shown in Fig. 5(a) when a resource $v$ is linked to another concept $u_i$ through property $p$, the length of the path is 1 and $u_i$ is not equal to $source(p)$.
  
  $l(v \overset{p}{\rightarrow} u_i) = 1, u_i = I, \delta(v) = 0$

- **Datatype property paths** ($P_{dt}$): A single length property path where $u_i$ is a literal, as shown in Fig. 5(b).
  
  $u_i = L$

---

3See http://code.google.com/p/rdfquery/
– Multi-length property path ($P_{ml}$) refer to a $\nu^\rightarrow_{\pi}$ where a concept $\nu$ is linked to another concept $u_i$ through more than one property. In Fig. 5(c) $\nu$ is linked to $u_j$ through $pI$, $u_j$ is then linked to $u_i$ through $p2$.

$$I(\nu^\rightarrow_{\pi}) \geq 2$$
$$u_i = (I \cup L), u_j = (I)$$

2. Branched property path ($P_{br}$) a resource $\nu$ is linked to two or more different concepts through different properties as shown in Fig. 5(d).

$$I(\nu^\rightarrow_{\pi}) \geq 1, I(\nu^\rightarrow_{\pi'}) \geq 1$$
$$u_j = (I \cup L), u_{m_1} = (I \cup L)$$

$$\delta -(u_i) = 0$$

– Multi-range property path ($P_{mr}$) A property path in a semantic association where for the property $p$ the $\text{source}(p)$ is a single node $\nu$ but target($p$) are $u_1, u_2, \ldots, u_n$ and $u_1 \neq u_2 \neq \ldots \neq u_n$ for some $n \geq 2$ as shown in Fig. 5(e).

$$|\text{source}(p)| > |\text{target}(p)|$$

– Cyclic property path ($P_{cy}$) A semantic association where one of the concepts or the nodes appears more than once in the path, as shown in Fig. 5(f)(g)(h).

$$I(\nu^\rightarrow_{\pi}) \geq 1$$

$$u_i = \nu, u_j = (I \cup B)$$

Definition 3 (Sub-association) A semantic association $\pi^\rightarrow_{\nu_i}$ is a sub-association of an other association $\pi$ (i.e. $\text{subAssociation}(\pi^\rightarrow_{\nu_1}, \pi^\rightarrow_{\nu_2})$), if and only if all the nodes of $\pi^\rightarrow_{\nu_1}$ are also the nodes of $\pi^\rightarrow_{\nu_2}$, i.e.

$$U(\pi^\rightarrow_{\nu_1}) \subseteq U(\pi^\rightarrow_{\nu_2})$$

Since all the nodes of $\pi^\rightarrow_{\nu_1}$ are a subset of the nodes of $\pi^\rightarrow_{\nu_2}$ we can infer the relationships among nodes of $\pi^\rightarrow_{\nu_2}$ from the association $\pi^\rightarrow_{\nu_1}$.

Definition 4 (Concept graph) A directed rooted graph $G' = \langle U', E', \nu \rangle$, denoted by $CG(\nu)$, is an RDF graph where $\nu \in U'$ is the root - concept node - for the graph. For all other nodes $u_i \in U'$ there exists a semantic association $\nu^\rightarrow_{\pi}$ of length $l$ between $\nu$ and $u_i$.

5.2. Using the concept graph

The process of generating a Web form from a given schema graph $G$ including a set of concept nodes $\nu \in U$ representing the input ontology involves the following steps as shown in Fig. 6:

1. Constructing the concept graph from an ontology: All related properties of a concept node are extracted from the domain ontology to construct the concept graph for $\nu$ such that $CG(\nu) \subseteq G$.

2. Mapping the concept graph to RaUL Web Forms:

– Association set extraction: For $CG(\nu)$ first an association set $\mathcal{P}(\nu)$ is identified composed of only distinct semantic associations for $\nu$ such that,

$$\forall \nu^\rightarrow_{\pi}, \forall \nu^\rightarrow_{\pi'} \in \mathcal{P}(\nu): \text{subAssociation}(\nu^\rightarrow_{\pi}, \nu^\rightarrow_{\pi'}) = \text{false} \text{ if } u_i \neq u_j$$

– Mapping the association set to RaUL Web forms: The mapping $\mu : \mathcal{P}(\nu) \rightarrow \text{RaUL}$ is defined, such that each path $\pi^\rightarrow_{\nu}$ is mapped to one or more valid RaUL widget elements. Once the mapping for all semantic associations is defined, a complete RaUL model for $CG(\nu)$ is returned. This model in turn can be rendered in HTML using ActiveRaUL which implements functionality to map the RaUL model $\mu$ to corresponding HTML elements of a Web form.

In the following sections we describe the individual steps in this process in more detail.

6. Constructing the concept graph from an ontology

The first step to generate a form for a concept node of an arbitrary ontology is to construct a concept graph $CG(\nu)$ from the RDF graph (ontology) $G$. A brief algorithm for the concept graph construction is presented in Algorithm 1. The algorithm corresponds to the function that is implemented for Endpoint III in Table 1. For the concept graph of the whole ontology this algorithm is called repeatedly for each concept node.

To construct a concept graph for the concept node $\nu$ from $G$, we first create a trivial graph $CG_0(\nu)$ composed of vertex $\nu$. $CG_0(\nu)$ represents a graph where
Algorithm 1. Concept graph Construction

Require: RDF Graph G(U, E), Node ν ∈ G
Ensure: Set CG0(ν) ← ν, i.e. CG0(ν) is initial concept graph for node ν */
Set n ← 0 /* n is length of the association */
1: do
2: n ← n + 1;
3: Construct the concept graph CGn(ν) ← CGn−1(ν) ; i.e. a concept graph for node ν where the maximum length of associations in the graph is n */
4: for all ui ∈ CGn−1(ν) where δ+(ui) = 0 and
5: l(ν − ui) = n − l do
6: CGn(ν) ← CGn(ν) ∪ j≤δ+(ν) (ui, Pi(j), ui(j+1));
7: end for;
8: return CGn(ν)

Fig. 6. A concept graph mapping to RaUL to Web Form

Mapping domain range restrictions to semantic associations: For domain range restrictions, the source and the target are explicitly defined for a property using the rdfs:domain and rdfs:range properties of RDFS as shown in Fig. 7.

Since a property can have multiple domains and/or multiple ranges it is difficult to determine for a specific domain and property which range concepts are relevant. Thus, our current implementation creates a semantic association for ui through property pi for each range concept.

Fig. 7. (a) Domain range restriction

Mapping single range existential restrictions to semantic associations: For each concept ui the associated property p+1 and concept u+i are defined as a necessary restriction (i.e. rdfs:subClassOf), or necessary and sufficient restriction (i.e. owl:equivalentClass) on ui as shown in Fig. 8. For a single range existential restriction a property p+1 can have values only from one concept u+i (i.e. p+1 has a single target con-

Fig. 8. (b) Single range existential restriction
cept). The restriction implies on the property $p_{i+1}$ such that it can have all values or some values from the concept $u_{i+1}$. These single range existential restrictions are defined in OWL with the $owl:Restriction$ concept and its properties such as $owl:onProperty$, $owl:allValuesFrom$ or $owl:someValuesFrom$.

To extract the related properties and concepts (range) of a concept $u_i$ from the modeling, we select $target(owl:onProperty)$ as associated property (i.e. $p_{i+1}$) and $target(owl:allValuesFrom)$ as associated concept (i.e. $u_{i+1}$) as range for $u_i$. The target for the properties $owl:allValuesFrom$ and $owl:someValuesFrom$ is a concept. However, the targets of cardinality restrictions are only the integers defining the cardinality for the property (i.e. missing the range of property). For such properties where the range is not defined, we declare $owl:Thing$ as a range for object properties and $xsd: String$ as range for datatype properties.

Mapping multiple ranges existential restrictions to semantic associations: This type of mapping is introduced for relationships where each concept $u_i$ is associated to more than one other concept $u_{i+1}$ through the same property $p_{i+1}$. In other words, property $p_{i+1}$ has multiple target concepts. This type of restriction is modelled in OWL similarly to the single range existential restriction with the only difference that $target(owl:allValuesFrom)$ and $target(owl:someValuesFrom)$ is a blank node of type $owl:Collection$. The $owl:Collection$ is composed of an $rdf:list$ where each $rdf:list$ element is a concept or another list modeled through $rdf:first$ and $rdf:rest$ properties, as shown in Fig. 9.

For a multiple ranges existential restriction of a concept $u_i$, a $target(owl:onProperty)$ is selected as the associated property $p_{i+1}$ and all $target(rdf:first)$ values of the recursive $rdf:list$ are considered as the possible associated concepts of $u_i$ through property $p_{i+1}$.

Mapping multiple properties existential restrictions to semantic associations: In a multiple properties existential restriction a concept $u_i$ is defined as an equivalent class of a restriction that involves more than one property as shown in Fig. 10. A multiple property restriction is a collection and within this collection each property is defined as a restriction similar to single and multiple ranges existential restrictions as described above. Therefore, in this model a single restriction describes many related properties and corresponding concepts.

For all such concepts whose relationships with other concepts are defined using this model, for each member of the collection we select $target(owl:onProperty)$ as a property and the associated concept (i.e. range) from $target(owl:allValuesFrom)$ or $target(owl:someValuesFrom)$ as a property and the associated concept (i.e. range) as a property and the associated concept (i.e. range) as a property and the associated concept (i.e. range). Therefore, the number of semantic associations (property $p_{i+1}$ - concept $u_{i+1}$) for a given concept $u_i$ is equal to the number of members in the $owl:Collection$.

7. Mapping the concept graph to RaUL Web Forms

The mapping of the concept graph to RaUL and then to Web forms has to particularly address the challenges identified in the introduction of the mismatch between the graph nature of the input ontology and the tree structure of Web forms and consequently the tree structure the RaUL RDF Web form model is emulating. To address this challenge, a pre-processing is required before we map a concept graph to a RaUL graph. In this pre-processing step, the semantic associations for a concept graph are modified to match the tree structure of the RaUL Web form model and redundant associations are eliminated.

Algorithm 2 defines the overall procedure for generating a RaUL graph from a concept graph. The algorithm takes a concept graph $CG_{n}(\nu)$ and a concept
node ν as an input and returns a RaUL graph. A semantic association set π(ν) is extracted for CGn(ν) (line 1) by following the procedure described in Sec. 7.2. Next, a RaUL widget container RaULwC is created for the concept node ν (line 2). Then a function SA_RaUL_Mapping (described in Sec. 7.3) is called that takes as input the association set π(ν) and the created RaUL WidgetContainer RaULwC and recursively builds a RaUL graph in RaULwC by adding mappings for each property and its corresponding concepts as defined in Sec. 7.2.

**Algorithm 2. RaUL Graph**
**Require:** CGn(ν), concept node ν
**Ensure:** Set π(ν) ← { } /* The association set for concept graph CGn(ν) */ Set RaULwC ← empty /* WidgetContainer for ν */
1: π(ν) ← getSemanticAssociations(CGn(ν), ν) /* Algorithm 3 */
2: RaULwC ← RaULwC ′ /* A RaUL Widget container with a text box that will hold all associations for this concept node */
3: SA_RaUL_Mapping (π(ν), r, RaULwC) /* Algorithm 4 */
4: return RaULwC

In the following sections we describe the individual steps of this algorithm in more detail.

### 7.1. Association set extraction

The process of extracting an association set is shown in Algorithm 3. The algorithm takes a concept graph CGn(ν) and concept node ν as input and returns an association set π(ν). For each association $\nu \pi \nu$ in CGn(ν) we first determine if it is a base property path, i.e. a property path that has a defined mapping in ActiveRaUL, or a cyclic property path. Cyclic property path do not have a defined mapping in ActiveRaUL and are first converted to one of the base property paths.

Cyclic paths, caused by owl:inverseOf properties or cyclic structures of concept relationships, are converted to the base property paths by removing the edge that introduces the cycle in the path (referred as reverse edge). More formally, if there exist any two nodes uᵢ and uⱼ (uᵢ ≠ uⱼ) in an association, for which besides a direct path between uᵢ and uⱼ there exists an edge $p \in E(\pi)$ such that $(uⱼ, p, uᵢ)$, then p is removed from the association $\nu \pi \nu$ (line 2-5). Since the relationship between nodes connected by reverse edges can be inferred from the rest of the semantic associations using the reasoner used in the implementation of ActiveRaUL, the removal of these reverse edges does not result in any loss of information about the concept relationships.

However, cycles caused by owl:Transitive-Properties are not inferable from the rest of the associations in the semantic association set. Such cyclic paths are implicitly broken during the mapping of a semantic association to a RaUL graph by mapping a transitive property to a single length property path.

Once the cycles are removed for the association, candidacy of the association to the association set is checked in a second step (line 6-11). By default every association of the concept graph is a candidate association. If an association $\nu \pi \nu$ is a sub-association of any association $\nu \pi' \nu$ of the association set (line 7), then association $\nu \pi \nu$ can be inferred from an existing association $\nu \pi' \nu$ of the association set. Therefore, $\nu \pi \nu$ is not a candidate association (line 8). If any association $\nu \pi' \nu$ of the association set is a sub-association of the association $\nu \pi \nu$ (line 9), then the association $\nu \pi' \nu$ can be inferred from the new association $\nu \pi \nu$. Therefore, we remove association $\pi$ from the association set (line 10), and association $\nu \pi' \nu$ will remain a candidate association. After candidacy check, if the association is still a candidate association then it is added to the association set, otherwise we skip the association. Once the whole process is completed for every association in the concept graph, an association set is returned.

**Algorithm 3. Semantic Association Set Extraction**
**Require:** CGn(ν), Node ν of CGn(ν)
**Ensure:** Set π(ν) ← { } /* The association set for concept graph CGn(ν) */
1: Set candidate ← true /* candidate is boolean variable */
2: for each $\nu \pi \nu$ ∈ CGn(ν) do
3: for all uᵢ & uⱼ ∈ U( $\nu \pi \nu$ ) do
4: if $\nu \pi \nu \cap \exists p = \{ p | (uᵢ, p, uⱼ) \cap \exists p \in E(\nu \pi \nu) \} \) do
5: $\pi = \nu \pi \nu - p$
6: end for;
7: for each $\nu \pi' \nu'$ ∈ π(ν) do
8: if subAssociation( $\nu \pi \nu$, $\nu \pi' \nu'$ ) do
9: candidate ← false;
10: else if subAssociation( $\nu \pi' \nu$, $\nu \pi \nu$ ) do
11: $\pi(\nu) = \pi(\nu) \cup \nu \pi \nu$
12: end for;
13: if candidate == true do
14: $\pi(\nu) = \pi(\nu) \cup \nu \pi \nu$
15: else skip $\nu \pi$ */ $\nu \pi$ is not a candidate SA */
16: end for;
17: return π(ν)
Example: Applying the association set extraction process on our motivating example, we first need to identify a semantic association set \( \Pi(Person) \) for the Concept Graph \( CG(Person) \). Table 2 shows all possible semantic associations and corresponding property paths involved in each semantic association for the \( CG(Person) \). \( \Pi(Person) \) will contain only the candidate associations selected from all possible semantic associations for the \( CG(Person) \) after removing avoidable cycles.

During the process of extracting the association set for the concept “Person”, all cycles in the association other than the ones caused by \( owl:Transitive-\)Property are detected and removed. Next, the candidate associations are identified and only candidate associations become part of the association set \( \Pi(Person) \). Starting with \( SA-1 \), all associations in Table 2 are checked one by one. Though \( SA-1 \) has a cyclic path for which \( source(supervises) = target(supervises) = Person \), it is not removed because of the transitive property (i.e. \( supervise \)). Next, a sub-association check is made for \( SA-1 \). At the time of adding the first association (i.e. \( SA-1 \)) the association set is empty and \( SA-1 \) is not a sub-association or any association in \( \Pi(Person) \), so it is added to \( \Pi(Person) \).

\( SA-2, SA-3 \) and \( SA-4 \) are associations with no cyclic paths. Also neither of these associations is a sub-association of another association in the association set, nor is any association of the association set a sub-association of these associations. Therefore these associations are added to \( \Pi(Person) \) without removing any edge or skipping any association.

\( SA-5 \) involves a cyclic property path because of the relationship (Organization, \( locatedIn \), City). The edge (or property) that causes the cycle in this semantic association is removed which results in \( SA-5 \) to be a multi-length property path. Neither \( SA-5 \) nor any association from the association set are sub-associations of each other so \( SA-5 \) is added to \( \Pi(Person) \).

\( SA-6 \) does not involve any cyclic property path but it is a sub-association of \( SA-5 \) (i.e. relationships of \( SA-6 \) can be inferred from \( SA-5 \) ) therefore, it is not a candidate association and we skip this association from \( \Pi(Person) \).

Summarising, we have a final association set that is:

\[
\Pi(Person) = \{ (Person, supervises, Person), (Person, address, String), (Person, gender, Gender), (Person, publication, Publication), (Publication, publisher, Publisher), (Publication, year, Int) \}
\]

### 7.2. Mapping the association set to RaUL Web forms

The semantic association set for a concept node is mapped to a Web Form that is used to create new instances or update existing instances for the concept node. The concept node relationships with other concepts can be implemented by creating new or by using existing instances of the related concepts to the concept node. To use or update the existing instance, the Web elements for object properties have a Lookup Existing link. A user can search the existing instances from the RaUL repository by clicking the link. Every multi-length property path, multi-range property path and single length property path (other than datatype property path) have a Lookup Existing link on the Web form as shown in example Fig. 21.

The decisions on how to map the semantic associations in the concept graph to RaUL and then to Web forms are made on the basis of (1) the position of a node in a property path and (2) the type of the property path involved in a semantic association. In the following we first describe the mapping of the nodes to RaUL depending on their position in the semantic association followed by a description of the different property path mappings. For each mapping a diagram (see Fig. 12-20) presents (1) the type of the property path based on its position in a semantic association (left side of the diagram) (2) the corresponding RaUL graph (center part of the diagram) (3) the Web form field/s corresponding to each RaUL graph (right part of the diagram). The mapping to the HTML Web form fields represents the third step in our mapping process as described in Fig. 6. The solid lines in a diagram show the property, the RaUL construct and the form field under consideration in the respective mapping, whereas the dashed lines present related concepts in semantic associations for which a mapping is already defined in a previous RaUL mapping. Every diagram also includes a mapping for the respective property path type from our motivating example Person ontology (indicated with solid fills in the diagram).

**Single Node to RaUL Mapping** (RaUL\(_{sn}\)): The mapping for any single node \( u \) of a property path other than the last node is presented in Fig. 11. Every single node \( u \) is mapped to a widget element of type raul:Textbox. For the raul:Textbox a
### Table 2

<table>
<thead>
<tr>
<th>Semantic Associations</th>
<th>Property Paths Involved</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SA-1</strong></td>
<td><code>Person supervises −−−−−−−−→ Person</code></td>
</tr>
<tr>
<td><strong>SA-2</strong></td>
<td><code>Person address −−−−−→ String</code></td>
</tr>
<tr>
<td><strong>SA-3</strong></td>
<td><code>Person gender −−−−−→ Gender</code></td>
</tr>
<tr>
<td><strong>SA-4</strong></td>
<td><code>Person publication −−−−−−−−→ Publication</code></td>
</tr>
<tr>
<td><strong>SA-5</strong></td>
<td><code>Person playRole definedBy −−−−−−−−→ Organization</code></td>
</tr>
<tr>
<td><strong>SA-6</strong></td>
<td><code>Person worksFor Organization locatedIn −−−−−−−→ City</code></td>
</tr>
</tbody>
</table>

---

**raul:label** property is defined with its value set to the label of node `u`. The *raul:range* of the *raul:Textbox* is set to `u` and the value of the *rdf:predicate* of the reified triple is set to *rdfs:label*. This information encodes the data binding and defines that data submitted through this textbox is an instance of type `u` and it becomes the object value for the *rdfs:label* predicate. The RaUL graph resulting from this mapping encodes that a node `u` appears as a textbox with a label `u` on the Web form.

**Example:** An example single node in our Person ontology is the Publication class. The Publication node will be mapped to a textbox according to the *RaULsn* described above. The object values in the *RaULsn* graph for *raul:label* and *raul:range* are Publication and *URIPublication* respectively. The corresponding Web form element is a textbox with a label as shown in the right part of Fig. 11.

**Concept Node to RaUL Mapping (*RaULcn*):** The concept node `ν` (i.e. the first node) of a property path is mapped to a *raul:Textbox* according to the mapping defined in *RaULsn*. Additionally, for the concept node, a container is required that encodes the relationship of `ν` to all its related properties and concepts. Therefore, a *raul:WidgetContainer* is created that references the widget elements that are created for related properties and concepts of `ν` (see Fig. 12). The first member of this collection *rdf:_1* is always set to the URI of the *raul:Textbox* of `ν`, while *rdf:_2*, …, *rdf:_n* are mapped depending on the type of the *target(p)* as shown in later mappings. For the *raul:WidgetContainer* a *raul:range* property is defined that is set to `ν`. This information encodes the data binding and defines that the domain of all members of the *WidgetContainer* collection is `ν`. In the mapping to a Web form the *raul:WidgetContainer* becomes a `<div>` container.

**Example:** In our motivating example a Web form is to be generated for the Person class of the Person ontology. Therefore, the Person is the concept node and thus mapped according to *RaULcn*. A *raul:WidgetContainer* holds the *raul:Textbox* that was created for the Person class as described
by mapping \textit{RaUL}_sn through a membership property \texttt{rdf:1}. The textbox has the object value of the \textit{raul:label} property set to \textit{Person} and the object value of the \textit{raul:range} set to \texttt{URI}_{\text{Person}}. All the associations for the \textit{Person} node are linked to the \textit{raul:WidgetContainer} through membership properties (i.e. \texttt{rdf:2}, \texttt{rdf:3} etc.). In a corresponding Web form, the textbox for the \textit{concept node} always appears as the first textbox on the Web form as shown in Fig. 12.

**Last Node to RaUL Mapping (RaUL_{ln})**: Every last node \texttt{u} in a semantic association, i.e. every node where there exists no \textit{source}(p) for \texttt{u}_i, and its \textit{target}(p)\texttt{i} are mapped to a \textit{raul:Textbox} for which the \textit{raul:label} is set to \texttt{u}_1 as shown in Fig. 13. The \texttt{rdf:predicate} value is set to the URI of \texttt{p}_1 and the \texttt{raul:range} value for the \textit{raul:Textbox} is \texttt{u}_1. Consequently, values submitted through this textbox are instances of class \texttt{u}_1 and are linked to the instance of \textit{source}(p) through property \texttt{p}_1.

**Example**: The \textit{City} is a last node of the \texttt{π}(\textit{Person}) class (denoted as SA-5 in Tab. 2). Therefore, according to its type, the \textit{City} node is mapped to a textbox as defined in \textit{RaUL}_{ln} and described above. The object values in the \textit{RaUL}_{ln} graph for \textit{raul:label}, \textit{raul:range} and \texttt{rdf:predicate} are \texttt{locatedIn}, \texttt{URL}_{\text{City}} and \texttt{URL}_{\text{locatedIn}}, respectively. This RaUL graph encodes that for this node a textbox appears on the Web form with a label \texttt{locatedIn}. Values submitted through this textbox are instances of \textit{City} and are linked to the instance of \textit{source}(\texttt{locatedIn}), i.e., \textit{Organization} through the property \texttt{locatedIn}.

**Single-length property path to RaUL Mapping (RaUL_{sl})**: The mapping for the \textit{Single-length property path} is already largely covered by the \textit{RaUL}_{sn} and \textit{RaUL}_{ln} mapping. In a \textit{single-length property path} the \textit{source}(p) can either be the \textit{concept node} or any single node, but the \textit{target}(p) is always a \textit{last node}. To map the relation of \texttt{p}_1 and \texttt{u}_1 to the \textit{source}(p), \textit{RaUL}_{ln} is set as a value of the \textit{membership property} of the \textit{raul:WidgetContainer} (i.e. \texttt{rdf:2}) that holds the mapping for \textit{source}(p) as shown in Fig. 14. A special case of \textit{single-length property} is the \textit{datatype property path} with the only difference that the \textit{raul:range} value is an \texttt{xsd datatagetype} corresponding to the \textit{datatype value} for the property \texttt{p}_1.

**Example**: The address property in our \textit{Person ontology} is a \textit{single-length property} for which \textit{target(address)} is a literal. To map the property \textit{address} and \textit{String Literal} a \textit{RaUL}_{ln} graph is generated as described above. The object values in the \textit{RaUL}_{ln} graph for \textit{raul:label}, \textit{raul:range} and \texttt{rdf:predicate} are \texttt{address}, \texttt{xsd: String} and \texttt{URL}_{\text{address}} respectively. To encode its relationship with the \textit{target(address)}, i.e., \textit{Person}, the \textit{RaUL}_{ln} created above is set as a value of the \textit{membership property} of the \textit{raul:WidgetContainer} (i.e. \texttt{rdf:2}) of the \textit{Person}. This \textit{single-length property path} appears on the Web form as a textbox with its label set to \textit{address} as shown on the right side of Fig. 14. The values submitted through this textbox are literals of \textit{String} which are linked to the corresponding instances of \textit{source(address)}, i.e., \textit{Person} through its \textit{address} property.

**Multi-length property path to RaUL Mapping (RaUL_{ml})**: In a \textit{multi-length property path}, shown in Fig. 15, the \textit{source}(p)\texttt{2} is linked to another concept (i.e. \texttt{u}_1) through more than one property (i.e. \texttt{p}_2 and \texttt{p}_1). To express this relation, RaUL offers the \textit{raul:Group container} class. In the mapping of a \textit{multi-length property path} to RaUL, the length of the property path determines the type of mapping. Two different views are adopted, one for a property path of length \texttt{l} = 2 and one for a property path of \texttt{l} > 2. A \textit{raul:level} property is set for the \textit{raul:Group class} to track the levels.
within the multi-length property path. The two different mappings based on the length are shown in Fig. 15 and Fig. 16.

If the length \( l \) of the property path is 2 then the group level for \( p_2 \) is 1. In this case a `raul:Group` is referenced from the `source(p_2)` through a membership property in the container class of `source(p_2)`. For the `raul:Group` the `raul:range` property is set to \( u_2 \). A `raul:list` property is defined for the `raul:Group` with an RDF list as the object value. The list holds the RaUL mappings for the other properties and concepts of `target(p_2)` (i.e. \( u_2, p_1 \) and \( u_1 \)). The two concepts in this list, \( u_2 \) and \( u_1 \) are mapped according to the mappings above, \( u_3 \) as a single node and \( u_2 \) connected through property \( p_1 \) to \( u_1 \) as a single-length property path and linked to the `raul:Group` through a membership property of the RDF list shown with the dashed lines in Fig. 15.
In the mapping to a Web form each raul:Group is mapped to a fieldset with a legend showing the property $u_2$ and the referenced concept $u_2$ as shown in the right side of Fig. 15. This fieldset indicates that within this box a new resource of type $u_2$ is created that is linked to $u_3$ via property $u_2$. All widget elements enclosed by this fieldset are defining properties for the instance of $u_2$.

Example: An example multi-length property path in the Person ontology which is part of the semantic association SA-4 is the property path (Person, publication, Publication). It is mapped to the RaUL mt graph as shown in Fig. 15. For the RaUL mt graph the raul:label, raul:range and rdf:predicate are set to publication, URI of publication and URI respectively. The raul:list object value is an rdf:list which refers to a Publication as the single node (RaUL sn) and to the relation (publisher, Publisher) as the RaUL mt. This raul:Group is referenced from the Person container through a membership relation. On the Web form, the raul:Group appears as a fieldset within the Person container, with a legend publication, Publication. This fieldset contains two textboxes one for the Publication as a single node and the other for the publisher property as the single length property mapping.

For multi-length property paths where the length of the property path is greater than 2 (as shown in Fig. 16), the raul:Group is referenced from the source(p3) through a membership property in the container class of source(p3) and the raul:level for this group is ‘2’. The raul:range property for the raul:Group is set to $u_3$. $u_3$ is the first node to map for this path, therefore, $u_3$ is mapped to RaUL sn and linked to the raul:Group through a membership property of the RDF list of the group. For $p_2$, as it is part of a multi-length property path of length 2, it is mapped according to RaUL mt, in a recursive process and linked to the raul:Group through a membership property as shown in Fig. 16.

The mapping of the RaUL graph to the Web form is similar to the one for the multi-length property path of length 2, but a button within the fieldset is created (see right side of Fig. 16) that opens a pop-up that displays a Web form similar to the one shown in Fig. 15 that allows the user to create instances of $u_2$ that are linked to $u_3$ through the property $p_2$.

Example: An example multi-length property path in our Person ontology where the length of path is greater than 2 is the SA-5. The raul:Group of the RaUL mt is referenced from the source(playRole) i.e. Person through a membership property and the raul:level for this group is set to ‘2’. For the RaUL mt graph the raul:label, raul:range and rdf:predicate are set to playRole, URI of playRole and URI respectively. The raul:list is set to an rdf:list which refers to a RaUL sn graph for Role as single node and a RaUL ml for relation (definedBy, Organization). On the Web form, the raul:Group appears as a fieldset, inside the Person container and textbox, with a legend playRole, Role. This fieldset contains a textbox for the Role and a button to add an instance of Organization that plays this role. This instance of an Organization can be created via another RaUL mt generated Web form that appears in a pop-up window when clicking on the button.

Multi-range property path RaUL Mapping (RaUL mr): For a multi-range property path the property $p$ is mapped to a raul:Textbox similar to a last node to RaUL mapping with the raul:label set to the label of $p$ and the rdf:predicate is set to the URI of $p$. However, the raul:range is not a single concept URI, but since $p$ can have multiple ranges, the range is set to a raul:Listbox URI. For this list the raul:list property is set to a list of raul:ListItems, where the raul:value of every raul:ListItem is one of the types of target($p$). As shown on the right side of Fig. 17, in a Web form target($p$) is mapped to a textbox that allows a user to add object values for the property $p$ followed by a listbox, where $u_1$, $u_3$ are the list items, that lets the user select the type of object values from the list. Once the value for the field is submitted, the property ‘selected’ is set to ‘true’ for the selected raul:ListItem.

Example: In the example ontology, the range of the publication property is a Publication. Since the concept Publication has two sub-concepts Conference and Journal, publication is a property for Conference and Journal as well. Consequently, publication is modelled as a multi-range property path. A RaUL mr graph is referenced from the raul:WidgetContainer of the Person concept through a membership property. In the RaUL ln graph raul:label is set to publication and the object value for rdf:predicate is URI of publication. The raul:listbox of the RaUL mr contains three raul:ListItems. The raul:value for the list items are set to Publication, Conference and
Journal. On a Web form, the RaUL$_{mr}$ for publication appears as a textbox followed by a listbox. A user enters the value for a publication in textbox and selects the type of publication from listbox.

**Branched property RaUL Mapping (RaUL$_{br}$):** In a branched property path, as shown in Fig. 18, there exists more than one semantic association for a concept $u$ so that $u = source(p_1) = source(p_2)$. The properties $p_1$ and $p_2$ are mapped separately according to the type of property paths and linked to the parent container, i.e. a raul:WidgetContainer if $u$ is the concept node or a raul:Group if $u$ is a single node on a multi-length property path, through a membership property of the RDF list. An example mapping for the branched properties is shown in Fig. 18 for two single-length associations of a single concept. In this case $p_1$, $u_1$ and $p_2$, $u_2$ each are mapped according to RaUL$_{sl}$ and added to the container of $u$. In a Web form the two or more properties for $u$ are displayed as separate textboxes within a fieldset.

**Example:** An example branched property path shown in the figure is part of SA-4 where the (Publication, publisher, Publisher) and (Publication, year, Int) are mapped to the RaUL$_{L_{ai}}$ graph. The association (Publication, publisher, Publisher) is mapped as a RaUL$_{ln}$ and is referenced from a container of the Publication. Since, Publication has another association i.e. (Publication, year, Int), therefore it is mapped according to the property path (for this example a last node on single length property path) and referred from the same widget container of the Publication. On the Web form, both nodes appear as a textbox within the fieldset for the Publication.

**Axiom instances to RaUL Mapping (RaUL$_{ai}$):** The RaUL$_{ai}$ graph shown in Fig. 19 is created for a property $p$ where the range of the property has defined instances in the ontology. The RaUL$_{ai}$ graph is referred through a membership property in the container class of source($p$). The RaUL$_{ai}$ maps the property and the instances that can be value for that property in...
a `raul:Listbox`. For the listbox the `raul:list` property is set to a list of `raul:ListItems`, where `raul:value` of the list items refers to the instances defined in the ontology. As shown in Fig. 19, `target(p)` is mapped to a listbox, where `l_1` and `l_2` are the list items.

**Example:** Example axiom instances as shown in Fig. 19 are the semantic associations in SA-3 where the property `gender` has two instances defined in the ontology, `Male` and `Female`. The instances of the `source(gender)` can have any one of these two instances as a object value for the property `gender`. A RaUL\textsubscript{ai} graph is referenced from the widget container of the `Person` (i.e. source(`gender`)) concept through a membership property. In the RaUL\textsubscript{ai} graph `raul:label` is set to `gender` and the object value for `rdf:predicate` is `URI\textsubscript{gender}`. The `raul:listbox` of the RaUL\textsubscript{ai} contains two `raul:ListItems`. The `raul:value` for the list items are set to `Male` and `Female`. On the generated Web form, the RaUL\textsubscript{ai} appears as a listbox within the fieldset of the `Person`. A user can select the defined value for the `gender` from a listbox.

**Non-functional property RaUL Mapping (RaUL\textsubscript{nfp}):** A non-functional property `p` can have more than one object value for a single instance of `source(p)`. The two type of mappings for such properties which depend upon the type of property path the participates in are shown in Fig. 20.

If the non-functional property is part of a single-length property it is mapped to a textbox and the `raul:multiple` property is set to `true`. This RaUL property causes the textbox to include a `plus button` to its side on the Web form. With this `plus button` a user can create multiple instances of the same relation.

If the non-functional property is part of a multi-length property where the relationship between multiple concepts for a given concept node are defined through a RaUL container class (see Fig. 15), a `raul:DynamicGroup` (a special type of a `raul:WidgetContainer`) instance is created instead of a `raul:Group`. This `raul:DynamicGroup` itself can hold multiple `raul:Groups` as shown in Fig. 20. The `raul:DynamicGroup` causes the whole group of widget elements to be enclosed by a `plus button` on the mapping to a Web form. With this `plus button` another set of instances of the entire group can be created.
on the Web form, i.e. all widget elements within the group are replicated by the RaUL JavaScript library.

**Example:** In our example ontology, some properties are defined as non-functional e.g. publication and supervises. For single-length property paths, e.g. supervises, the RaUL\textsubscript{nfp} mapping adds a raul:multiple property with the object value true to the supervises property which is mapped according to RaUL\textsubscript{ln}. For multi-length property paths, e.g. publication, the RaUL\textsubscript{nfp} mapping creates a raul:DynamicGroup that references the publication property that was created through the RaUL\textsubscript{ml} mapping via a membership property in the dynamic group. The raul:WidgetContainer for the Person class that originally referred to RaUL\textsubscript{ml} now refers to the RaUL\textsubscript{nfp}.

7.3. RaUL Mapping procedure

Algorithm 4 describes the process of how the property path mappings defined above are combined to create a complete RaUL Web form graph. The mapping process for each property path encompasses three steps: (1) A RaUL mapping RaUL\textsubscript{pattern}, corresponding to the type of the property path is created; (2) the RaUL mapping created in step 1 is added to the RaUL WidgetContainer RaUL\textsubscript{wC} that is an input of this algorithm and; (3) a triple (URI\textsubscript{RaUL\textsubscript{wC}}, rdf\textsubscript{n}, URI\textsubscript{RaUL\textsubscript{pattern}}) is created where URI\textsubscript{RaUL\textsubscript{wC}} is the URI of a WidgetContainer RaUL\textsubscript{wC} that holds the different RaUL mappings through an RDF membership property and URI\textsubscript{RaUL\textsubscript{pattern}} is the URI of the RaUL\textsubscript{pattern} linking the newly created RaUL\textsubscript{pattern} to the corresponding RaUL\textsubscript{wC}. Since, for every container the first member rdf\textsubscript{1} is always the main concept of the semantic association, n in rdf\textsubscript{n} is the number of semantic association plus 1.

Since, we look for all associations of node ν at (line 1), we are implementing RaUL mappings for all property paths p for which ν is source(p). To implement other patterns, for each association in the semantic set the first property of the association (i.e. p), the first related concept of ν (i.e. r) and all the base property paths ‘bpp’ are extracted (line 3-4).

The bpp is matched to the corresponding case of the switch statement (line 5). If bpp matches to P\textsubscript{ai} then a RaUL\textsubscript{ai} graph is created for (line 7). For a non-functional property p a triple (URI\textsubscript{RaUL\textsubscript{ai}}, raul:multiple, true) is linked using RaUL\textsubscript{ai} URI (i.e. URI\textsubscript{RaUL\textsubscript{ai}}) (line 9) and RaUL\textsubscript{ai} is added to the raul:WidgetContainer RaUL\textsubscript{wC} with a triple defining the container membership (line 11).

If it matches to P\textsubscript{ml} (i.e. a multi-length property path) then a RaUL\textsubscript{ml} is created (line 13). If p is a non-functional property then we create a RaUL\textsubscript{nfp}, and add this to RaUL\textsubscript{ml}. To link the RaUL\textsubscript{nfp} to RaUL\textsubscript{ml} a triple is added to the combined graph (line 14-17). Once the mapping for RaUL\textsubscript{ml} is completed, a triple defining the container membership of RaUL\textsubscript{ml} to RaUL\textsubscript{wC} is created (line 18). To map next properties and concept to RaUL mapping, p and r are removed from the semantic association and the rest of the association comes up to an association set for r (line 19). The SA_RaUL_Mapping algorithm is called...
Algorithm 4. Semantic Association to RaUL Mapping (SA_RaUL_Mapping)

Require: SA set $\mathcal{A}(\nu)$, Node $\nu$, WidgetContainer $RaUL_{wC}$

Ensure: $URIs_{wc} \leftarrow \text{source}(RaUL_{wC}, rdf : \text{list})$ /* URI of the list of a widgetContainer that holds different raul mappings through rdf membership property */

1. Set $p \leftarrow \text{null}$ /* property for which $\nu$ is a domain concept */
2. Set $r \leftarrow \text{null}$ /* target(p) */
3. Set $bpp \leftarrow \text{null}$ /* base property pattern for p */

for each $\pi \in \mathcal{A}(\nu)$ do

1. $p \leftarrow p' = \{ p' \text{ source}(p') = \nu \}$
2. $r \leftarrow \text{target}(p)$
3. $bpp \leftarrow \text{BasePropertyPattern}(p)$

Switch $bpp$ do

1. case $P_{ai}$:
   
   if $p \equiv P_{ai}$
   
   $RaUL_{ai} = RaUL_{ai} \cup \{ \text{URI}_{RaUL_{ai} \cdot \text{raul:} \text{multiple} \cdot \text{true}} \}$
   
   end if

2. case $P_{nl}$:
   
   if $p \equiv P_{nl}$
   
   create a $RaUL_{nlfp}$
   
   $RaUL_{nlfp} = RaUL_{nlfp} \cup \{ \text{URI}_{RaUL_{nlfp} \cdot \text{rdf} \cdot \text{node}} \}$
   
   end if

3. case $P_{mr}$:
   
   if $p \equiv P_{mr}$
   
   $RaUL_{mr} = RaUL_{mr} \cup \{ \text{URI}_{RaUL_{mr} \cdot \text{raul:} \text{multiple} \cdot \text{true}} \}$
   
   end if

4. default:
   
   if $p \equiv P_{r}$
   
   $RaUL_{r} = RaUL_{r} \cup \{ \text{URI}_{RaUL_{r} \cdot \text{raul:} \text{true}} \}$
   
   end if

end for;

in a recursive way to map the association set of $r$ (line 20).

If $bpp$ matches to $P_{mr}$ (i.e. a multi-range property path) then $RaUL_{mr}$ is created (line 22). For a non-functional property $p$ a triple (URI$_{RaUL_{mr} \cdot \text{raul:} \text{multiple} \cdot \text{true}}$) is linked using $RaUL_{mr}$ URI (i.e. URI$_{RaUL_{mr}}$) (line 30) and $RaUL_{mr}$ is added to the raul:WidgetContainer $RaUL_{wC}$ with a triple defining the container membership (line 32). The implementation for the mapping of the branched property path is implicit in this algorithm, since all property paths $p$ for which $\nu$ is source($p$) are already considered.

Example: For the association set $\mathcal{A}(\text{Person}) \mu : \text{CG(Person)} \rightarrow \text{RaUL}$ is defined for each association according to the mappings proposed in Section 7.2 as shown in Fig. 22. Each node in the RaUL graph represents a RaUL mapping for the corresponding property and concept pair, and each directed edge represents the membership property with which different RaUL mappings are connected with each other. All the nodes...
with outgoing edges are RaUL containers (appear with bold lines) that hold other containers or textboxes. Once all the associations of the association set are mapped, the concept graph CG(Person) to RaUL graph mapping is completed. The RaUL graph is rendered in HTML as follows. Each container node (except RaUL{nfp}) is mapped to a fieldset and a textbox, and each RaUL{sl} is mapped to a textbox. RaUL{nfp} is mapped to a plus button along with the fieldset of the member container (i.e. RaUL{ml}). For the RaUL graph extracted above the rendered Web form in HTML is shown in the screenshot in Fig. 21 which we have annotated with red boxes indicating the extracted semantic association set.

8. Evaluation

We carried out an empirical user study to evaluate our hypothesis that ActiveRaUL is: more effective, more efficient and more usable for someone with no prior knowledge of RDF/OWL to create and maintain logical consistent ontology instances, than traditional knowledge engineering methods. We compared ActiveRaUL to the widely used state-of-the-art Web ontology editing tool, WebProtégé [23]. WebProtégé also offers a plugin for form-based editing6. However, similar to other related works such as Callimachus [6], Web form templates have to be manually created requiring in-depth knowledge of RDFs/OWL. In this evaluation we aim to validate our hypothesis that the automatic generation of a Web form template from an arbitrary ontology yields in a User interface that is easier to use than any other state-of-the-art User interfaces that does not need a customization based on the input ontology. Consequently, we can compare ActiveRaUL only with tools that automatically generate a User Interface from an ontology without any need of configuring templates manually. For our user study we used WebProtégé rather than Protégé, the desktop version, for the following reasons: (1) similarly to ActiveRaUL, it is Web-based and runs in any browser; (2) it stores all data centrally, and thus users have distributed access to the same individuals, making the user study more realistic to real-world scenarios in terms of searching for existing individuals; and, (3) it uses a frame-based logic for properties that was also used in a previous version of Protégé that treats rdfs:range as constraints when creating properties (i.e. by default it displays to the user only the individuals of a type that is in the subsumption hierarchy of the rdfs:range of the property relation while still allowing the user to select individuals from any class).

8.1. Participant Demographics

We recruited twelve participants for our user study, including undergraduate and postgraduate students of computer science at the Australian National University and staff in the Information Engineering Laboratory of CSIRO. We asked the participants to rate their: (1) computer literacy, (2) knowledge of the Semantic Web, (3) knowledge of the SSN ontology, and (4) experience with WebProtégé on a five-point Likert Scale with anchors from “Novice” to “Expert”. None of the participants have ever used ActiveRaUL before. All participants rated their computer literacy as “Expert”. Although we had preferred to include participants in the study who did not have a computer science background, we found that for understanding WebProtégé, participants needed to have at least a rudimentary understanding of object-oriented design to understand the difference between classes, properties and instances of each. Based on the participants’ self-assessments, we separated the users into two groups: semantics experienced users and semantics inexperienced users. We would expect that for a more usable system, the usability measures will be high within both user groups. However, if a semantics inexperienced user is able to comprehend our system more easily, then we would expect to see less significant differences between the usability measures in accomplishing the test cases in ActiveRaUL by the semantics experienced user group.

---

6See http://protegewiki.stanford.edu/wiki/PropertyFormPortlet
and semantics inexperienced user group compared to WebProtégé.

8.2. Test Case Specification

For our study, we had to define test cases based on an existing ontology that fulfills several requirements: (1) to test as many data modelling features as possible the ontology needs to be sufficiently complex, including a subsumption hierarchy, datatype and object properties, OWL property restrictions and the import of ontologies; (2) it should model a domain familiar enough for a non-domain expert to easily understand; and, (3) there should exist some gold standard ontology instances to compare the ontology instances created in the user study to.

We found the Sensor Network Ontology (SSN) [10], developed by the W3C Semantic Sensor Network Incubator group, to best fulfill these requirements. A demonstration deployment of ActiveRaUL set up for the user study already pre-loading the SSN ontology and automatically creating the Web-forms is available at: http://www.activeraul.org/demo/arbitraryOntology.html The SSN describes the capabilities of sensors, their measurement processes and the resultant observations. In particular, in contrast to many other Web ontologies (like FOAF, SIOC, PROV-O), SSN is based on an upper-level ontology, inheriting some of its complex OWL property restrictions. It also describes a domain that is relatively easy to comprehend by non-domain experts. Further, the SSN working group has published a number of use cases with example ontology instances on their project wiki [1] that we could use as our gold standard. In particular, we used the university deployment example from the wiki, because: (1) it includes programming examples in RDF/XML; and (2) it is based on the core SSN ontology without any extension (some of the other examples use extensions). Based on the university deployment example we designed three test cases, each with a number of tasks. The three test cases are increasing in complexity and include the ontology engineering tasks listed in table 8.2. The last column shows
the number of triples that are supposed to be created in each test case if all tasks are successfully completed.

8.3. User study test metrics

To effectively measure the usability of our system for creating and maintaining ontology instances, we consider the usability measures as defined by ISO 9241-11, in particular:

- **Effectiveness**: The ability of the user to complete the task using the system and the quality of the output of those tasks.
- **Efficiency**: The level of resources consumed in performing the tasks.
- **Satisfaction**: A user’s subjective reactions using the system.

8.3.1. Effectiveness

We consider two metrics for measuring the effectiveness of the tools tested for our user study, task success and task accuracy.

**Task Success** measures how effectively users are able to complete a given set of tasks. A task is successfully completed if, and only if, a user enters all the values correctly and within the given time. We combined the success with the time-on-task metric, setting a time-out figure for each test case (i.e. 5 min for Test Case 1, 10 min for Test Case 2, and 15 min for Test Case 3). Consequently, we scored the task success based on the number of correct triples a participant managed to model in the given time. To measure the accuracy/correctness we compared the ontology instances created by the participants in the study with the gold standard instances defined in the SSN working group. We scored each triple that was created by the participant in either of the two systems as “Correct” or “Incorrect/Missing”.

8.3.2. Efficiency

For the efficiency of the system we measured the time-on-task. This metric is related to the efficiency of the system and captures the amount of time spent in completing a test case. As mentioned, we introduced a time limit which may distort our average time numbers (by decreasing our standard deviation). The time limit was introduced to accommodate participants who may get frustrated with the system and would then consequently not follow through with the task.

8.3.3. Satisfaction

After completion of the three test cases in both systems, we asked the participants to rate their subjective reactions on the usability of the systems based on the widely-used System Usability Scale (SUS) [9]. SUS is a highly robust and versatile tool for usability testing and has proven to yield reliable results across different sample sizes [5]. It is particularly suitable for our user study as we are primarily testing the functional differences between the two systems and how they influence the user experience, as opposed to interface design issues that are the focus of some other types of usability scales. We asked the users to rate their experience on a five-point Likert scale with anchors for “Strongly Agree” and “Strongly Disagree” as required by the SUS methodology.

8.4. Procedure

We started each user study with an introduction to RDF, RDFS, Ontologies and the SSN ontology. We varied the length and detail of this introduction based on a participants prior knowledge in semantic Web technologies. However, every participant had strictly the same training in Protégé and ActiveRaUL. Before we started the user study we presented each par-
participant with a sample test case similar to the three test cases we were later testing in the user study. We gave each participant step-by-step instructions on how to complete the sample test case in both systems. Each participant had access to hand-outs with the step-by-step instructions for the sample test case for both tools which they could consult with during the test cases. We asked each participant to perform the same three test cases on both systems. The starting order of the tool was varied between test cases and between participants to balance out any bias from familiarity with the test case. The only difference in the test case description between the two systems was that we included the property hierarchy when asking the participants to create a property in WebProtégé. This was to overcome the lack of a search functionality for properties in WebProtégé which makes it very hard for participants not familiar with the SSN ontology to find a specific property in the tree structure (e.g. the ssn:hasValue property is a sub-property of DUL:hasRegion). After the completion of a test case in both tools we provided the participants with feedback to highlight any errors the participants made in order to avoid repeated mistakes in the subsequent test case.

8.5. Results

In this section we present the detailed results of our user study, where we aim to prove our hypothesis that ActiveRaUL is indeed easier, more effective and more efficient to use than the state-of-the-art ontology editing tools for the creation of RDF data. These results are based on the performance and feedback of twelve participants: five of which, based on their self-assessment, were categorised into the semantics experienced user group, and seven categorised into the semantics inexperienced user group.

Accuracy/Correctness: Table 4 shows the average number of correct triples for each test case for both tools. The table displays the results for all participants and for each of the two user groups - the semantic experienced users and the semantics inexperienced users. Table 5 shows the overall accuracy over the three test cases which shows that the participants clearly performed better in ActiveRaUL, managing to create 91% correct triples compared to 82% in WebProtégé. Only one participant created less correct triples in ActiveRaUL than in WebProtégé, which was only due to him inadvertently pressing the “Close” button instead of the “Fill In” button after successfully entering the values in on of the pop-up windows. The most common mistake in WebProtégé was that users chose the wrong type for an individual while creating a relation; while the most common mistake in ActiveRaUL was that users filled the search box for individuals instead of the label box. In both tools the experienced user group performed better, although the difference was less than 10% for both tools for both systems. For ActiveRaUL, the accuracy of the participants was already very high in the first test case, even though no participant has ever used the system before. This confirms our hypothesis that a Web form-based user interface is familiar enough to computer literate users to create RDF data correctly, even if the participants are inexperienced in semantic Web technologies. Interestingly enough, there was very little difference between the performance of the experienced and inexperienced user group in WebProtégé for the first test case, with both groups only managing to correctly model 2/3 of the triples. The significant increase in accuracy in the second test case is due to the feedback the participants received after completing test case 1, which identified their mistakes.

8.5.1. Efficiency

Table 6 shows the average times participants required to complete a test case. As mentioned previously, we used a cut-off time of five, ten and fifteen minutes for each test case, respectively. This time limit was particularly relevant in the first test case in WebProtégé, where six participants did not complete in time. However, this was due to five of six participants getting stuck, not knowing how to proceed further. For the second test case only one participant could not finish in time in WebProtégé. In the third test case, also in WebProtégé, only three participants did not finish in time. Again, only one of these participants actually ran out of time, whilst the other two were unable to proceed further. All participants finished in time in ActiveRaUL. Both participant groups, inexperienced and experienced, were significantly faster (between 27% and 56% faster) completing the test cases in ActiveRaUL compared to WebProtégé. Particularly pronounced were the differences in the first test case, confirming our hypothesis that even without prior experience with ActiveRaUL, users were able to quickly and accurately create RDF data.

8.5.2. Satisfaction

After completion of the test cases participants were asked to anonymously fill out an online survey cov-
Table 4
Accuracy in completing test cases in WebProtégé and ActiveRaUL

<table>
<thead>
<tr>
<th>Test Case 1</th>
<th>Test Case 2</th>
<th>Test Case 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebProtégé</td>
<td>ActiveRaUL</td>
<td>WebProtégé</td>
</tr>
<tr>
<td>No.</td>
<td>%age</td>
<td>No.</td>
</tr>
<tr>
<td>Exp. Users</td>
<td>3.40</td>
<td>68.00</td>
</tr>
<tr>
<td>Inexp. Users</td>
<td>3.29</td>
<td>65.71</td>
</tr>
<tr>
<td>All Users</td>
<td>3.33</td>
<td>66.67</td>
</tr>
</tbody>
</table>

Table 5
Overall accuracy in completing test cases in WebProtégé and ActiveRaUL

<table>
<thead>
<tr>
<th>Total Accuracy</th>
<th>WebProtégé</th>
<th>ActiveRaUL</th>
</tr>
</thead>
<tbody>
<tr>
<td>%age</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exp. Users</td>
<td>82.05</td>
<td>91.03</td>
</tr>
<tr>
<td>Inexp. Users</td>
<td>76.92</td>
<td>87.91</td>
</tr>
<tr>
<td>All Users</td>
<td>82.05</td>
<td>91.03</td>
</tr>
</tbody>
</table>

Table 6
Average times (mm:ss) to complete test cases in WebProtégé and ActiveRaUL

<table>
<thead>
<tr>
<th>Test Case 1</th>
<th>Test Case 2</th>
<th>Test Case 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebProtégé</td>
<td>ActiveRaUL</td>
<td>WebProtégé</td>
</tr>
<tr>
<td>No.</td>
<td>%age</td>
<td>No.</td>
</tr>
<tr>
<td>Exp. Users</td>
<td>3:46</td>
<td>1:50</td>
</tr>
<tr>
<td>Inexp. Users</td>
<td>4:05</td>
<td>2:17</td>
</tr>
<tr>
<td>All Users</td>
<td>3:57</td>
<td>2:05</td>
</tr>
</tbody>
</table>

Fig. 23. System Usability Scale score for ActiveRaUL and WebProtégé
ering the questions specified by the System Usability Scale. SUS yields a single number representing a composite measure of the usability of a system, whereby SUS scores have a range from 0 to 100, 100 being the best score. Fig. 23 shows the individual aggregated scores for each question for both systems. Overall ActiveRaUL scored 72.1 out of 100 points in the System Usability Scale compared with 32.5 for WebProtégé, clearly indicating that the participants found ActiveRaUL easier to use.

9. Conclusion

In this article we presented ActiveRaUL, a system to generate Web forms from arbitrary ontologies. These Web forms can then be used to create and maintain RDF data that is typed according to the input ontology (ontology instances). The process of automatically generating Web forms from an ontology involved a number of novel techniques in mapping the graph-based input ontology to a tree-based Web form. For the mapping we proposed multiple types of widget elements in the Web form that are used for different graph patterns in the input ontology, including object properties, OWL property restrictions and complex relations among the same concept (graph cycles). The resulting widget elements are themselves expressed according to an ontology, the RDF User Interface Language (RaUL) and can be rendered in any browser by using ActiveRaUL.

We evaluated our approach of automatically generating Web forms in a user study based on use cases developed by the W3C Semantic Sensor Network (SSN) Incubator group. In the study we compared the efficiency, the effectiveness and the user satisfaction of the participants in creating RDF data using two interfaces: (1) Web forms automatically generated by ActiveRaUL; (2) Web-based user interfaces automatically generated in the ontology editing tool WebProtégé. The participants in the study created in average 91% correct triples for all test cases in ActiveRaUL compared to 82% in WebProtégé. Further, the participants were significantly faster (between 27% and 56%) completing the test cases in ActiveRaUL compared to WebProtégé. After completing the user study we asked the participants to rate the two systems for RDF data creation based on the System Usability Score. In overall ActiveRaUL scored 72.1 out of 100 points compared with 32.5 for WebProtégé, clearly indicating that the participants found ActiveRaUL easier to use to create RDF data than WebProtégé, a tool that allows both, scheme modelling and RDF data creation.

Future Work Although our approach of automatically generating a Web form from an input ontology results in sufficiently usable user interfaces as demonstrated in our user study, it is arguably a first step in a refinement process to create the best possible Web form-based user interface for a given ontology. Since the resulting user interface templates are themselves expressed according to an ontology and identified by a URI, they can iteratively improved by a developer and assigned a new URI every time the Web form is changed.

One such area that currently still may require manual refinement is the ordering of widget elements on a Web form. Our algorithm does not rank the semantic associations for a concept node, therefore the widget elements appear unordered on Web forms. This becomes problematic when there is a large number of associations for a concept and its related concepts. To overcome this limitation we are considering multiple strategies of ordering the semantic associations on an automatically generated Web form as future work.

Further, our current implementation only deals with rdfs:labels, the other annotation properties (i.e. owl:versionInfo, rdfs:comment, rdfs:seeAlso, and rdfs:-isDefinedBy) are ignored and for datatype properties we do not perform a type check based on its xsd:datatype after submission of the data. In future work we plan to take advantage of the automatic type checks offered by HTML5 and also offer widget templates for common input types such as a calendar for xsd:date.

Another area of future work is to consider a mapping from RaUL templates to templating languages such as Mustache\(^7\). Offering Mustache templates as one of the output types of the ActiveRaUL service would allow a developer to more easily inject customised code in the resulting Web forms. Currently, with RaUL Web forms the structural styling is limited to the features supported in CSS3.

References


\(^7\)http://mustache.github.io/


