Ontofly: Ontology Engineering Using the Web

David Urbansky
University of Technology Dresden
Dresden, Germany
david.urbansky@tu-dresden.de

Robert Willner
University of Technology Dresden
Dresden, Germany
robert.willner@gmail.com

Alexander Schill
University of Technology Dresden
Dresden, Germany
alexander.schill@tu-dresden.de

***Abstract*—Ontology engineering is the task of creating and refining a knowledge model for one or multiple domains. This process is difficult and a great deal of time is required to refine the ontology. In this paper, we present a system that semi-automatically aids the user in creating an ontology using a standard web browser. Our system helps to create ontologies whose target domain can be found on the web and which will be used for extracting knowledge to populate the ontology later on.**

I. Introduction

In recent years, industry and academia have developed tools, systems, and technologies that aim for semantic representation of data. Many tasks, such as knowledge modeling, ontology population, ontology matching, and ontology engineering, play major roles in the movement towards semantic data. The semantic web initiative [1] started the trend almost a decade ago for the unstructured web data, resulting in a continuously growing web of data.

In this paper we focus on browser-based ontology engineering with suggestion functionality to ease the engineering process. Our major research questions are as follows:

1) In which ways can we assist an ontology engineer with creating a domain ontology?
2) How can we suggest potentially useful information (properties, concepts, instances, synonyms) to the ontology engineer in order to minimize his/her cognitive work?
3) What vocabulary extensions are necessary to equip the ontology with enough information to be used in an ontology-based web information extraction system?

Our hypothesis is that a domain expert can create a domain ontology, which should be used for ontology-based extraction, far quicker and more effectively (more relevant concepts, properties, synonyms, etc.) if the ontology can be created close to the real web data instead of using an ontology editor that is not connected to the corpus where the later extraction should happen.

Our major contributions in this paper are as follows:

1) We describe the design of an ontology editor that works on real web data to guide the engineering process.
2) We explain a technique that automatically suggests potential property candidates for the ontology.
3) We show a vocabulary extension to better specify the range of properties that are added to the ontology. This extension is especially important when the ontology is used in an ontology-based web information extraction system such as WebKnox [8] afterwards.

In the next section we will describe a use case that led to the requirements and design of our system, which are explained later.

II. Use Case

In our use case, we want to create a knowledge base with factual information about mobile phones. We know that this information can be found on the web and we want an ontology-based web information extraction system to extract it automatically. In order to guide the extraction system, we need to supply an ontology that is structured and named very similarly to the representations on the web. A domain expert now creates such an ontology using a state-of-the-art ontology editor such as Protege [5]. The expert might now visit mobile phone producer websites to understand which properties and facts are associated with the concept mobile phone. He or she will now find that, among others, mobile phones have an “internal memory”, a “display size”, and a “talk time”. These properties are now added to the concept using the ontology editor. The expert now faces several problems:

1) The domain expert has to switch between the source of information (the website) and the ontology editor. This reoccurring process is time consuming and error prone since the expert has to copy the property names manually.
2) The expert finds a table with facts about a mobile phone. All properties in the table are of interest to the ontology so the expert has to manually transfer them into the ontology editor and check whether some properties are synonyms of existing ones. Furthermore, relevant property names might be buried in the text of the web page and are difficult for a human editor to discover.
3) The expert finds additional information about the property ranges. For example, after researching a couple of mobile phones he or she might conclude that the “internal memory” property has values between 32K and 32GB. This range restriction would greatly help the subsequent extraction process, but the domain expert needs to express this knowledge manually in the ontology.

In the following sections we explain how our web ontology editor tool solves these problems and eases the process of creating an ontology.

III. Ontofly

Our ontology editor can create ontologies as you "fly" through the web and is therefore called *Ontofly*. In this section we will explain the architecture of our system, discuss its capabilities, and describe vocabulary extensions that we made to better suit the final ontology for subsequent extraction processes.

A. Architecture

Figure 1 shows the schematic architecture of Ontofly. The system is a client server application. The server part is realized as a web service that uses the Representational State Transfer (REST) architecture as described in [3]. The application on the client is realized in HTML and JavaScript. The client application consists of two frames, the Ontofly management frame on the top and an i-frame where the ontology engineer can navigate to any arbitrary page. This design allows our application to work in every modern web browser. The client server communication is done by *Asynchronous JavaScript And XML* (AJAX), that is, whenever the user loads or adds an item to the ontology, the data is transfered from/to the server and ontology changes are done on the server side. We use Jena [2] to read and write the ontology and mirror all ontology changes in a relational database which is then used by the extraction system WebKnox [8] for the subsequent extraction process.

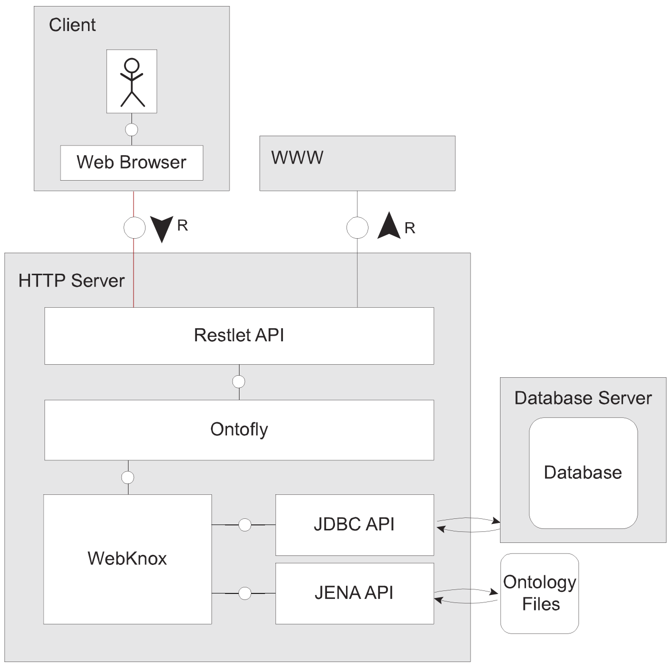
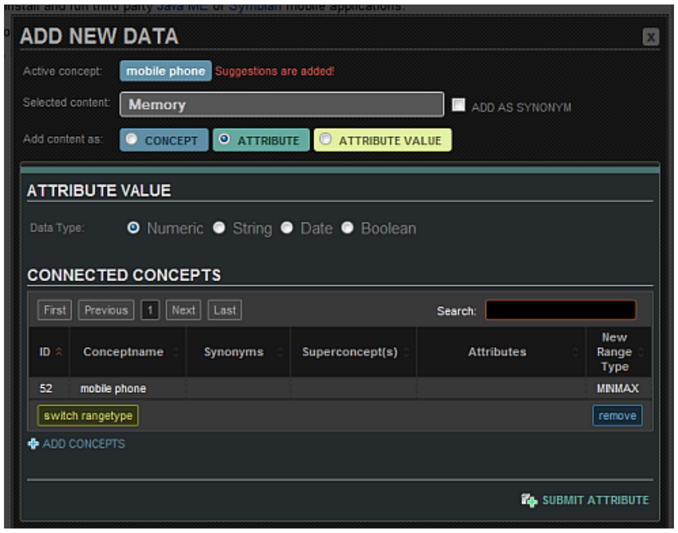


Fig. 1. Architecture of the Ontofly system.

B. Fact Candidate Suggestion

Ontofly allows the engineer to manually add, change, and delete items from the ontology. These items can be concepts (e.g. "mobile phone"), properties (e.g. "memory") and property values (e.g. "160 MB" for the attribute "memory"). Figure 2 shows an example input mask that appears after the user highlighted the property "Memory" in the fact table about a mobile phone. Here we have added the term as a property for the concept mobile phone and have set the data type to "numeric" and the range type to "minmax". We will explain the range type in Section III-C.



ADD NEW DATA

Active concept: mobile phone Suggestions are added!

Selected content: Memory ☐ ADD AS SYNONYM

Add content as: CONCEPT [✓ ATTRIBUTE] ATTRIBUTE VALUE

ATTRIBUTE VALUE

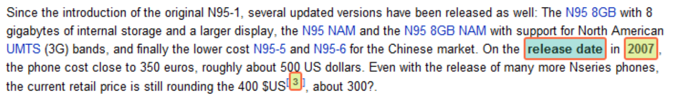
Data Type: [✓ Numeric] String Date Boolean

CONNECTED CONCEPTS

ID	Conceptname	Synonyms	Superconcept(s)	Attributes	New Range Type
52	mobile phone				MINMAX

Fig. 2. Adding a property to the ontology.

After the ontology contains several properties for a given concept the Server will use WebKnox to suggest properties and values that are then highlighted by the client application. This process should make it easier for the ontology engineer to find more relevant terms and values for the ontology. Figure 3 shows how Ontofly suggests the property "release date" for the mobile phone with a possible value of "2007". Terms that are suggestions have a red frame, while property suggestions and possible values are highlighted in cyan and tan respectively.



Since the introduction of the original N95-1, several updated versions have been released as well: The N95 8GB with 8 gigabytes of internal storage and a larger display, the N95 NAM and the N95 8GB NAM with support for North American UMTS (3G) bands, and finally the lower cost N95-5 and N95-6 for the Chinese market. On the release date in 2007, the phone cost close to 350 euros, roughly about 500 US dollars. Even with the release of many more Nseries phones, the current retail price is still rounding the 400 $US[3], about 300?.

Fig. 3. Ontofly suggest facts that might be useful for the ontology.

C. Range Type

Our target ontology is intended to be used as input for a web information extraction process. We therefore want to provide information that helps guide this process. Ontofly introduces a few new vocabulary terms to specify the range of values for a property. These terms use the WebKnox namespace "wx:".

1) The **hasRange** property has a property as rdfs:domain and a blank node of rdfs:range. One property can have several "hasRange" properties.

2) The **rangeType** is the type of range we want to specify. We distinguish between “minmax” and “possible”. The term “minmax” means that we know the numeric range of values for the property, the term “possible” means that we know all possible values that the property can have.
3) The properties **rangeMin**, **rangeMax**, and **rangePossible** specify the values in the range of the property. If the rangeType is “minmax”, we specify the minimum and maximum values with “rangeMin” and “rangeMax” respectively. If the rangeType is “possible”, we enumerate all possible values using “rangePossible”.

Figure 4 shows an example graph for the property “memory”. We see that the property can belong to several domains (“phone” and “camera”) that we declare using “rdfs:domain”. We use “rdfs:range” to specify that the values are integer values and make it an OWL “datatypeProperty” using “rdfs:type”. We now declare one range for the “memory” property using “wx:hasRange” and connect it with a blank node (“rn1”). We use “rdfs:domain” again to declare that the blank node specifies the range type for the memory property that belongs to the phone concept since the memory property also belongs to the camera concept but might have different ranges in that concept. Finally, we use “wx:rangeType”, “wx:rangeMin”, and “rangeMax” to declare that the memory property for mobile phones always has a minimum value of 0.5 MB and a maximum value of 32,000 MB. The extraction process can now use this information and discard all extractions that do not fall into the defined range.

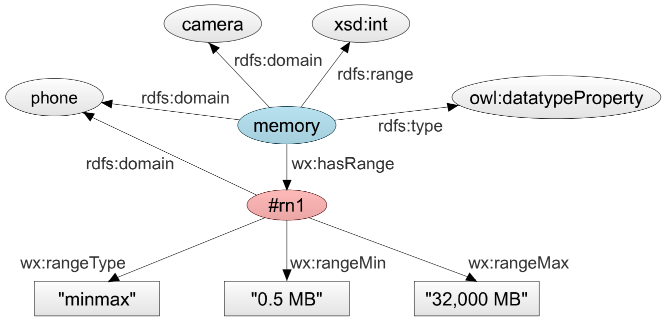


Fig. 4. Modeling range types using blank nodes.

D. Related Work

To the best of our knowledge there is no ontology editor that combines semi-automatic ontology creation from web pages with the goal to use that ontology for information extraction.

Semantic Turkey [6] is the only ontology editor we found that directly works on web content (as a Firefox extension[1]). It focuses on managing personal information and replacing the “Favorites” that are usually used in browsers. While our tool focuses more on the schema or T-Box, the Semantic Turkey is used to store plenty of individual information in the A-Box of the ontology.

OntoGen [4] is a semi-automatic ontology editor. The editor features (un)supervised methods for concept, relation, and name suggestion. Unsupervised learning methods automatically compile sub-concept lists by k-means clustering and latent semantic indexing, the supervised learning techniques identify relevant documents by a given user query and an active learning loop. Another similarity with our system is that OntoGen is data-driven, that is, the suggestion functionality uses the underlying document corpus that the engineer browses to create the ontology. Our system differs in that we use the web as the underlying corpus and our suggestion functionality focuses on properties for a concept rather than concepts themselves.

Protege [5] is a very popular open source Java-based ontology editor from Stanford University. It supports many features such as ontology engineering, exporting, and reasoning. However, there is no suggestion functionality to help the ontology creator nor the possibility to work on real web data. The editor is very generic and therefore offers solutions for many use cases, but does not fulfill our requirements as described in the use case. There is also a web version[2] of Protege which brings part of the functionality to the web without changing the scope of the application.

IV. Evaluation

In this section we will compare Ontofly to the ontology editor Protege. We evaluate the tools in regard to the use case that we described in Section II, which required that the knowledge about concepts and properties be taken from the web. The domain for which the users had to create the ontology was limited to communication devices in order to have a better comparison across the users. The task consists of three subtasks with increasing difficulty:

1) The user has to create an ontology and save it in RDF/XML serialization.
2) The user has to add one concept, one subconcept, and one concept synonym to the ontology.
3) The user has to find and add three properties to one concept.

We divided the group of eight users into two commensurate groups, one that received a short introduction in how to use both tools, and one without any prior knowledge in ontology editing. For each subtask, we measured the NASA Task Load Index (TLX) [7] and the time that was needed to perform the task.

Figure 5 shows the results of the comparison between the two ontology editors. As we expected, we can see that the group that was given a short introduction to the tools performed better on average (lower TLX and shorter time). Only in subtask 3 for the editor Protege did we observe almost no difference between the two groups. The figure clearly shows that using Ontofly led to a lower TLX on average in all three subtasks compared to Protege, which shows users were less frustrated and mentally stressed. These results are

[1]https://addons.mozilla.org/en-US/firefox/addon/8880

[2]http://webprotege.stanford.edu

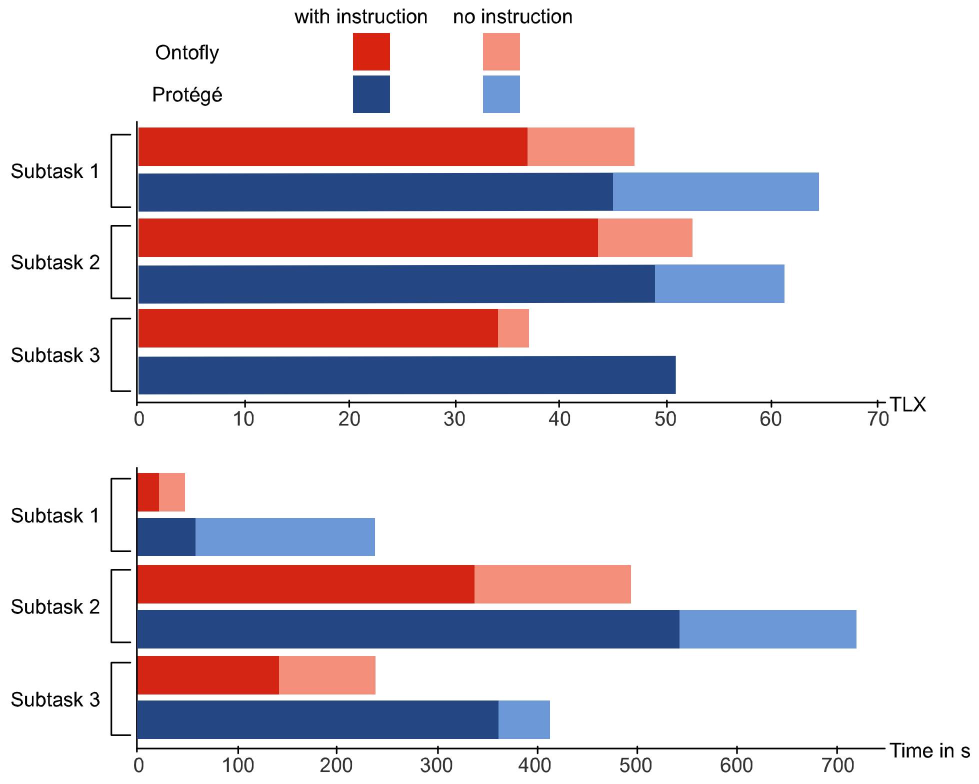


Fig. 5. TLX and time comparison for three subtasks in Ontofly and Protege. Shorter is better for time and TLX.

very likely also an outcome of the reduced time users had to spend for each subtask. The figure shows that users could solve each subtask faster using Ontofly. On average users spent approximately 47% less time with Ontofly and had a 20% lower TLX than when using Protege.

The comparison only shows that Ontofly is better suited for the use case we described. Protege is, however, a much more feature rich tool, which makes it more difficult to use than Ontofly. We evaluated only the editing process that was possible using both tools. Ontofly introduces, however, the feature of specifying property ranges, which would have to be done manually using other tools. This feature makes Ontofly a good choice for creating ontologies that can be used by ontology-driven extraction systems.

V. Conclusion and Future Work

In this paper we have presented a new kind of ontology editor that focuses on creating ontologies that can be used as input for ontology-based information extraction systems. The editor is data-driven and uses the web as an underlying corpus to suggest properties and instance data, which yields a richer ontology than a non-data driven approach. Our last contribution was the vocabulary extension to better express ranges of values for properties using “rangeMin”, “rangeMax”, and “rangePossible” properties. In our ongoing research we want to look into more properties that help the extraction system to find valuable instance data. One of those properties might be “dynamic”, which expresses whether a property might have changing values over time such as the “population” property for the concept “country”. Furthermore, we need to evaluate the system in a user study and compare it to state-of-the-art ontology editors that can also be used for our scenario.

References

[1] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):28–37, 2001.

[2] Jeremy J. Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. Jena: implementing the semantic web recommendations. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 74–83. ACM, 2004.

[3] Roy T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[4] Blaz Fortuna, Marko Grobelnik, and Dunja Mladenic. Ontogen: Semi-automatic ontology editor. *Human Interface and the Management of Information. Interacting in Information Environments*, pages 309–318, 2007.

[5] John H. Gennari, Mark A. Musen, Ray W. Fergerson, William E. Grosso, Monica Crubézy, Henrik Eriksson, Natalya F. Noy, and Samson W. Tu. The evolution of Protégé: an environment for knowledge-based systems development. *International Journal of Human-Computer Studies*, 58(1):89–123, 2003.

[6] Donato Griesi, Maria Pazienza, and Armando Stellato. Semantic turkey: A semantic bookmarking tool (system description). *The Semantic Web: Research and Applications*, pages 779–788, 2007.

[7] Sandra G. Hart and Lowell E. Staveland. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. *Human mental workload*, 1:139–183, 1988.

[8] David Urbansky, Marius Feldmann, James A. Thom, and Alexander Schill. Entity Extraction from the Web withWebKnox. In *Proceedings of the Sixth Atlantic Web Intelligence Conference*, 2009.