Crowdsourcing Enabled Ontology Engineering

Gerhard Wohlgennant, Marta Sabou, and X Y

WU Vienna
{alfred.hofmann,ursula.barth,ingrid.haas,frank.holzwarth
http://www.wu.ac.at

Abstract. Recent years have seen an increase in the use of crowdsourcing based methods at various stages of the ontology engineering lifecycle (e.g., verification of subsumption, assigning equivalences between concepts etc) thus laying the foundations of a novel approach to ontology engineering. Take up of this early research by the community at large, especially by practitioners, is however currently hampered by 1) a lack of understanding of which stages of the ontology engineering process can be crowdsourced and 2) tool support in ontology engineering platforms that would allow easy insertion of crowdsourcing into ontology engineering workflows. In this paper we perform an overview of recent works in the area and take a scenario-based approach to identifying those stages of the OE process where crowdsourcing makes sense. Then, we present the uComp Protege plugin, a plugin for the popular Protege ontology engineering platform that facilitates the integration of crowdsourcing stages into the OE process. TBD: clarify novelty (e.g., which new tasks we introduce?), sum up some important evaluation results.

Keywords: human computation, crowdsourcing, ontology engineering, ontology learning, Protege plugin

1 Introduction

RQ1: Which tasks can be crowdsourced? How do they fit in the OE workflow?
RQ2: How to provide tool support for crowdsourcing in OE?

2 Related Work - MS

Noy and colleagues [1] focus on the task of verifying subclass-superclass relations that make up the ontology hierarchy. Their main interest is in understanding whether crowdsourcing via MTurk is a viable alternative for this task and therefore they perform a series of experiments that compare microworkers against students, evaluate performance differences over different types of ontologies (e.g., upper ontologies vs. generic ontologies) and finally assess crowd-performance in specialized domains. The paper concludes that micro-workers are a viable alternative for verifying subclass-superclass relations and also introduces a vision for a tool support that would facilitate the integration of crowdsourcing into OE

workflows. We present such a tool in this paper, that not only allows the ontology engineer to crowdsource subsumption verification, but also a set of other tasks that often appear in ontology engineering scenarios.

[TBD - table]

3 Ontology Engineering Scenarios - Ontology Learning - MS

(1 or both of the following) OL - Gerhards work on automatic OL from text & other sources

OR - using the Watson plugin

TBD: Relate to the "Embedded HC paradigm"

4 Ontology Engineering Tasks for Crowdsourcing - MS

From the scenarios above as well as the related work, we distinguish a set of generic ontology engineering tasks that are suitable for crowdsourcing and which are likely to be of interest in a wide range of ontology engineering scenarios:

T1. Verification of Domain Relevance. Is a concept/instance relevant for a domain?

T2. Verification of Relation Correctness. Does a certain relation between two ontology entities hold? These could be a set of generic relations (sameAs, subClassOf, instanceOf), but also arbitrary named relations to be specified by the ontology engineer. The crowd here would have to vote (yes/no) for a given triple (Subject - Relation - Object). This task is the focus of [1].

T3. Learning of Relation Names. This is also a very difficult task in OL in general - the workers are presented with two terms and can choose between a set of given relations what applies. These relations can be a set of OWL relations that all ontologies have as well as a (restricted) number of domain specific relations specified by the expert (e.g., influences). [BTW, since this is a complex task it could be split up into a sequence of 3 simpler tasks: 1) given two terms workers agree whether these are related or not; 2) those pairs that were judged related are then passed to another task where a correct relation is selected; 3) in the third task, the quality of the relations is checked - practically T2] Available relation labels are some predefined (like subClassOf) and those from the target ontology (all ObjectProperties) – if more than 20 .. spread over multiple windows (depending on window size) Have a limit of 5 relation labels in the Protege interface All pairs with a certain relation (defined by the user in a text field, eg relation) are sent to the uComp API (or directly to CF). If you want to use just a subset of relations: have a window where you select (checkboxes) the actual pair to be sent. In CrowdFlower – also have the choice to add a free text label output: be able to sort by certainty from CF ..

5 The uComp Protege Plugin - GW

(not sure what would be the best name for it :)) describes which of the tasks above are implemented and how + other technical details.

6 Evaluation

6.1 Evaluation Goal

The goal of the evaluation is to assess the improvements that the uComp Plugin could enable in a typical ontology engineering scenario in terms of typical project completion aspects such as time, cost and quality of output. The usability of the plugin is an additional criteria that should be evaluated. Concretely, our evaluation goals can be summarised into the following questions:

Time *How does the use of the plugin affect the time needed to perform ontology engineering tasks?* - We distinguish here the total task time (Ttt) as the time taken from the start of the ontology engineering task until its finalisation; and the time of the ontology engineer spent actively in the task (Toe). In a crowdsourced scenario, Toe ¡ Ttt, because the ontology engineer is only actively working during the outsourcing of the task and the review of the result. In contrast, in a traditional scenario Toe = Ttt. What is of interest to us is the time reduction ratio, that is (Ttt-Toe)/Ttt. (this will be computed as an average over multiple measurements, and over various ontology engineering tasks).

Cost *Are there cost benefits associated with the use of the plugin?* We compute costs related to payments for the involved work-force, that is payments to ontology experts and payments to crowd-workers. Costs of ontology experts are computed by multiplying the time they spend on the task (Toe) with an average salary. In order to allow comparison to other similar cost-focused studies [1], the wage of a research scientist was assumed to be $54,000 per annum.

Quaility *What are the implications on the quality of the resulting output when using the Plugin?* Several earlier studies [e.g., Thalhammer13, Sabou13, Noy?, Sabou13] have shown that the quality of the semantic tasks performed by crowd-workers is in general similar to (or even better than) the quality of tasks performed by ontology engineers. While the quality of the obtained data is not the core focus of our evaluation, we expect to obtain results similar to those already published [TBD: note however, that unlike earlier studies which focus on simple, atomic tasks, we focus on more complex engineering tasks - is this really true?]

Usability *Is the plugin usable?* As any end-user tools, the plugin should be easy to understand and use by the average ontology engineer already familiar with the Protege environment.

6.2 Evaluation Setup

The setup involves working with two groups of ontology engineers, which will perform the same OE tasks.

Evaluators Victims: Adrian, Jrgen (? knows the tool), 2 guys from TU Vienna (Olger, Fajar) Non-Protege victims: Stefan, Matyas, Michael, Philipp, Heinz, Max Gbel

Group 1 This group will make use of traditional (that is manual) methods to perform the three OE tasks. The group consists of X ontology engineers.

Group 2 This group, consisting of Y ontology engineers, will use the Plugin to perform the three OE tasks. Group 2 will be provided a short tutorial about the plugin (30 minutes) and will have to perform a usability questionnaire about the plugin.

Evaluation Data The input to all evaluation tasks are ontologies generated by an ontology learning algorithm from textual sources.

For every ontology snapshot, ie. every result of an ontology learning stage, as well as the resulting ontology the system exports an OWL file (using the Turtle seralization format, which can be converted to RDF/XML easily). The conversion of our internal format for lightweight ontologies is straightforward for concepts (which resemble to OWL classes). WordNet hyper- and hyponym relations are mapped to the OWL subClassOf property. The only challenging aspect is the representation of unlabeled relations in OWL. Our system creates ObjectProperties named "relation_n", where "n" is an auto-incrementing number, as it is not possible to use the same ObjectProperty more than once. The label for those relations is plainly "relation". The periodically created versions of the domain ontology can be uploaded to a triple store and are thereby accessible via the SPARQL.

We evaluate the plugin over two ontologies covering two diverse domains. Some details of the input ontologies are:

	Climate Change Ontology	**Finance Ontology**
Nr. of Classes		
Nr. of Relations		
Nr. of IsA Relations		
Nr. of Un-named Relations		

Table 1. Overview of the ontologies used in the experiments.

Evaluation Tasks We perform the evaluation of the plugin over three different ontology engineering tasks in order to 1) test different functionalities of the plugin; 2) obtain evaluation result over a range of tasks.

Both user groups will perform the following tasks:

Task 1: Check concept relevance for a domain For each concept of the ontology decide whether it is relevant for the domain in question (in our case, climate change). Input: concept and domain name; Output: Y/N ratings – use a relevant_to_domain property – domain experts use Y/N

Task 2: Check the correctness of isA relations For all isA relations verify whether they are correct, that is a generic/specific relation exists between the domain and range of the relation. Manual experts change the value for a certain annotation property (..._subClass_validation_result)

Evaluation Metrics For time: For each participant measure Toe, and Ttt, for each task. Compute averages per participant Compute group averages and differences

For costs: Compute avg costs per participant and per group and differences

For quality: Since we do not have a baseline, we will proceed as follows: Task 1: compute pair-wise inter-expert agreement for both groups; and between groups this will measure how different the output is between the plugin and non-plugin group. If it is small then the results are similar; Compute also Cohens Kappa Task 2: same approach as with Task 1 Task 3: Here I think we should evaluate the quality of labels ourselves (em and Gerhard) for each produced ontology; and then compute a precision value