Crowd Sourcing Web Service Annotations
James Scicluna, Christoph Blank, Nathalie Steinmetz
seekda GmbH
Grabenweg 68
Innsbruck, Austria
Elena Simperl
Karlsruhe Institute of Technology
P.O. Box 3640
76021 Karlsruhe, Germany

Abstract
Web service annotation is of crucial importance to enable efficient search and discovery of such services, especially for those that are REST based. However, the process of annotating such services has several drawbacks, whether automated or not. Automated processes are quite inaccurate and do not in general provide quality annotations. Manual processes are very expensive, both in time, effort and consequently financially. This work focuses on easing - and thus reducing the cost of - manually providing annotations. This is done using a user friendly wizard on the seekda Web services portal in combination with crowd sourcing on Amazon mechanical turk. The approach turned out to be very promising in solving one of the crucial problems of the Semantic Web, namely, the acquisition of annotations.

Introduction
There is a vast amount of Web Services available. Such services (e.g. Google mapping APIs, Gravatar, etc) are increasingly being integrated within applications in order to provide a better experience to the end user. Finding such services is notoriously difficult and usually boils down to a Google search with the developer having to manually filter out irrelevant services. Even the process of identifying whether a page links to a service, can be problematic this is especially true for Web APIs. Annotating these services is a first important step towards better search results. However - and very ironically - developers are in general not keen to perform such annotations, making life difficult for themselves and their community. This work describes how the use of crowd sourcing techniques can help obtain such annotations effectively by giving a small monetary reward. This is done through a seamless integration between Amazon’s Mechanical Turk and seekda’s Web Services portal’s easy-to-use annotation wizard. Users on the mechanical turk can access a set of tasks that require the annotation of a number of services selected from the portal. The process starts with the identification of whether a specific document describes or is linked to a Web API. If this is the case, it proceeds to the annotation process which includes category assignment, tag-based annotations and natural language descriptions. The work is evaluated in terms of the effectiveness of this technique, namely, the increase in the number of annotations obtained and the quality thereof. This work has been done in the context of the EU funded integrated project Insemtives ¹, which focuses on insentivising users to annotate resources on the Web. The rest of the paper is structured as follows. First the problem at hand is described. Following that, the approach for obtaining annotations using crowd sourcing techniques is illustrated. Results obtained from our experiments are then described. The paper then concludes with some remarks and future work.

Annotations and Crowd Sourcing
Web services are software components accessible over the internet. There are two basic types: WSDL based and REST based (or Web APIs). WSDL services are “described” by an XML document using the Web Services Description Language². These are easy to detect and if available, their annotations are generally easy to extract. REST (Fielding 2000) services on the other hand, rely entirely on the HTTP protocol and are generally accessible through Web documents. They are not described using a structured document as for WSDL based services, and as such, are harder to detect and even more to annotate. Extensions for microformats - such as hRESTS and MicroWSMO (Kopecky, Vitvar, and Fensel 2009) - allow to provide annotations to REST based services. However, these are rarely provided by the service developer.

Our work concentrates on REST services. They are increasingly becoming much more popular with Web developers, in particular due to their ease of integration with mashups. However, as stated above, such services are generally hidden behind normal Web pages and are therefore hard to find and use, unless they are known apriori. In general, the process of sensibly indexing a Web API for search consists of two main parts:
1. Asserting whether a Web document leads to a Web API
2. Annotating the Web API

¹http://insemtives.org/
²http://www.w3.org/TR/wsdll
In order to automate the detection of Web APIs, machine learning techniques are generally used. A set of known Web API documents is fed to an algorithm which builds a vector model. This model is then used to assert whether other unknown Web documents are in fact Web APIs. Although similar techniques can be used to automate the annotation process, their results are not satisfying. In most cases, only basic annotations are correctly extracted automatically. As our results show, human intervention provides more quality annotations. However, this process requires more effort; this is where crowd sourcing comes into play. This paradigm, in combination with the right tools, allows completing tasks fast and cheap. The focus of this work is therefore to combine crowd sourcing with manual annotation of Web Services in order to obtain such annotations fast, cheap and with quality results.

In the next sections, we provide more insight into annotation procedures, outlining the pros and cons for automated and non. Following that, we describe advantages and disadvantages of crowd sourcing and how we can minimise its drawbacks to use it for effective Web Service annotation.

The Annotation Process
As stated earlier in the text, there are two primary ways of annotating Web APIs, namely, automatic and manual. The automatic way generally uses information extraction techniques to identify concepts present in a particular document. The approach is generally very fast and annotations can be obtained easily without any cost of human effort. The problem is that the annotations are in many cases inaccurate. For example, it might be the case that the document from where the annotations are being extracted, is not really related to the service. As a consequence, this leads to inaccuracy in search results.

Our focus therefore turns over to manual annotation processes. By definition, anything that requires human intervention means more cost in time, effort and perhaps money. In our context, we are dealing with Web APIs which would require from the person at least basic Web development skills in order to sensibly annotate the service. The following are a different levels of annotation types, starting from the simplest up to the most complex.

- association with a category
- free tagging
- natural language description
- microformat annotation (e.g. hRESTS or MicroWSMO)
- rich semantic annotation (e.g. RDF, OWL-S, WSMO)

Of course, the lower one goes in the list, the higher the effort and costs to manually produce such types of annotations. On the other hand, the quality of the produced annotations is in general much higher than existing automated techniques. Furthermore, the first three types of annotations can be done in a small amount of time, when compared to the richer two annotation types (the last two in the list). It’s also the case that richer forms of annotation are still expensive to process when it comes to perform queries over them and would therefore not provide adequate response times. Existing Web API portals such as ProgrammableWeb 3, provide means to tag such services. However, they do not provide a way to create such annotations in a cheap, fast and systematic way.

Crowd Sourcing
The term crowd sourcing describes the idea of sourcing out tasks to a group of people or a specific community. As opposed to the traditional way of completing these tasks, where a company or institution would either perform itself or outsource them to a third party provider, the goal with crowd sourcing is to use the collective intelligence of the public at large instead. Due to the open nature of crowd sourcing and since it uses a large amount of people at once, it allows to gather those that are most fit to perform particular tasks in a given domain.

Depending on the task, problems are usually not only just submitted for the crowd to solve, but also to review. The participant sorts through the possible solutions, and filters out the best or most complete ones. The participant that submitted a (or the best) solution is often rewarded. In most of the cases this reward is of monetary nature, but it can also consist of some other different form of prize or just recognition in the respective community. To ensure the quality of results, consensus finding mechanisms can be used on top of the solutions.

A simple and well known problem would be the task of annotating images to describe their content. The annotation process could be crowd sourced by dividing it into simple tasks where each participant has to specify a category for a particular image. To improve the quality of results, majority voting is used (e.g. 2 out of 3). The successful submission is then monetarily rewarded with $1. The benefits of crowd sourcing include:

- Problems can be solved very quickly, as people to solve tasks are available globally 24/7.
- The cost of solving tasks is comparatively small for the "crowd sourcer".
- A wide range of talent and a very large number of individuals is available at once
- Payment determined by the quality of solutions (only the successful participants get rewarded or need to get paid, increased value for the invested money).
- Insight into the habits and desires of a target group.

However, there are also a few drawbacks that have to be kept in mind when crowd sourcing tasks:

- Lack of control (since the problems are made public, or made available for a larger audience, the shared information has to be carefully selected)
- Lack of trust and confidentiality (participants are often anonymous, which is most of the time caused by spam).

Due to these concerns, the specification of the task to be crowd sourced is detrimental to the quality of the output. It

3http://www.programmableweb.com
must be detailed enough for a worker to precisely understand what needs to be done, but concise enough to reduce the amount of time required to complete the task.

Therefore, in light of all, our work focuses on combining manual annotation techniques with crowd sourcing. We choose those types of annotations that are manually relatively easy to obtain without sacrificing their quality. We boiled down the whole process into three single easy steps that in combination with crowd sourcing enable us to obtain annotations fast, cheap and with good quality. These steps are:

1. Assess whether a given Web document is a Web API or not
2. Assign categories
3. Provide semantic tags and a natural language description

Initially, the steps were however more (for example, they included assigning a rating for the service). In the next sections we describe how we came down to these primitive steps.

Marrying Crowd Sourcing with Web Service Annotation Tools

As described earlier, our core contribution relies in integrating a simple to use process for obtaining annotations with an effective crowd sourcing technique. The process has been developed within the seekda Web Service Portal and Amazon Mechanical Turk is the crowd sourcing tool we use. In this section, we first give a brief introduction of both applications, followed by a description on how we combine them together.

seekda Web Services Portal

The goal of the seekda Web Services platform is to provide a convenient interface to this index of services, allowing their users to browse and search Web Services in an efficient as well as effective manner. Web services are indexed using a focused crawler. However, in the current state, only WSDL-based services are detected since they are far easier to detect than Web APIs, as discussed earlier. There are currently more than 28,000 WDSL-based services indexed, along with information about the service provider, service availability and latency, documentation coverage, etc. Users interacting with the platform are able to directly invoke particular operations of a Web Service for immediate testing (unless authentication or registration are required by the provider). Furthermore they can provide comments and describe a service in question. The objective of this work is to further improve the existing portal in such a way to allow easy and quick annotations of the Web services. A lot of other improvements have been performed, which are however outside the scope of this paper.

Amazon Mechanical Turk

Amazon Mechanical Turk is perhaps the most widely known and used tool for crowdsourcing. There are essentially two types of entities involved: requesters and workers. A requester defines a template for the tasks - known as Human Intelligence Tasks, or shortly HITs - that are to be initiated. It describes in detail the procedure and requirements that a worker must fulfill in order for a given task to be accepted. Such a template is initiated by uploading a comma-separated-value file with variable bindings to actual instances. Finally, a financial reward is set to the task. This reward is assigned to the worker either automatically or by manual evaluation of the result of the task. Once all these options are configured, a batch can be published to start running. The system provides a convenient management page where batches can be monitored and tasks approved or rejected. Once finished, the results can also be downloaded for analysis. Figure 2 shows an example of a completed batch from our second iteration of running tasks. These iterations are described in detail later on in the text.

Figure 1: Landing page of the current seekda Web Services Portal.

Crowd Annotating

The combination of the Web services portal tools and Amazon Mechanical Turk are the pillars of this work. Our approach was to create a well described turk task which is ini-
tiated with different Web documents. The task points to a very simple and easy to use Wizard that we developed on the seekda Web services portal. Using this wizard, workers can execute the task at hand by following the steps outlined in the previous section.

Categories are assigned from concepts defined in an RDF classification ontology, which was a result of the Service Finder project ⁶. This allows us to make simple SparQL queries over such an ontology, without sacrificing performance. Also, suggested semantic tags are extracted using a concept bootstrapping service developed as part of the Insemtives project platform (Andrews, Pane, and Zaihrayeu 2011). Given a Web document, this service uses information extraction techniques to provide a list of suggested tags to the user. Note however, that unlike fully automated techniques, it is still the user that decides which of those tags are really relevant. This allows to keep the quality of annotations high, since it is highly the case that tags that are of little relevance to the document are not added. Optionally, other non suggested tags can be added.

Setup and Results

A total of three iterations have been performed, excluding a couple of initial iterations which helped us to figure out how the mechanical turk actually works. In general, each iteration was set up with the following configuration:

- Number of tasks, that is, the number of distinct Web documents that are to be processed by a worker.
- Reward per task in American dollar cents (note that Amazon Mechanical Turk requires an American bank account).
- The maximum duration that the worker should take to complete a task.
- The qualification required for a worker to execute the task.

In this section, we will describe each iteration by illustrating the number of tasks that were processed, the results that were obtained and the adjustments performed. During the process we had to deal with spammers and learned later in the process some techniques to reduce it.

First Iteration

Before even starting the first iteration, we tried a few tasks (around 10) in order to get better acquainted with the Amazon Mechanical Turk environment. Having done so, we set up the experiment, namely by:

- developing the annotation wizard, which included:
  - determining whether a document represents a Web API or not
  - assigning a category
  - assigning tags
  - providing a natural language description
  - determining whether the page is a documentation page, pricing page or service listing
  - provide a rating to the service

- defining a thorough description of the task that the workers must read in order to complete it

An important note is that in the first iteration we did not impose any restrictions in order for a task to be eligible (for example, at least a category must be provided). The setup of the experiment is shown in Table 1.

<table>
<thead>
<tr>
<th>Number of Submissions</th>
<th>70</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reward per task ($)</td>
<td>$0.10</td>
</tr>
<tr>
<td>Duration limit</td>
<td>15 minutes</td>
</tr>
<tr>
<td>Restrictions</td>
<td>none</td>
</tr>
</tbody>
</table>

Table 1: Setup of the first iteration.

We start with summarising the identification results:

- 21 real APIs identified correctly as APIs
- 28 Web documents (non Web APIs) identified correctly as not being a Web API
- In total, 49 documents out of 70 were correctly identified (70% accuracy)
- On average, a task was completed in 02:20 minutes

For annotations - category, tags and natural language descriptions - the submissions were manually analysed. Of course, the figures here apply to those services that were correctly classified as Web APIs. The following were the observations regarding the tasks:

- 4 were very good and complete
- 8 were acceptable (the reason being that not all steps in the wizard were completed)

There were in total 23 distinct workers that were involved in executing some of the tasks, with some of them completing more than one. Amongst the contributors, there were unfortunately some spammers. This is a well known problem in crowdsourcing, and there exists no solution that can completely remove spamming. However, one could tweak the restrictions and process to reduce it (described further below). Nevertheless, we chose the top 3 workers to give the reader a high level view of the individual contributions:

- A particular worker completed 15 tasks (2 very good, 2 acceptable)

  ⁶http://www.service-finder.eu/ontologies/ServiceOntology
Another one completed 10 tasks (1 very good, 1 acceptable)
Another one completed 5 tasks (2 acceptable)

Following these observations, a set of improvements were set up to be used for the next iteration. First off, the maximum time allowed to complete a task was reduced from 15 minutes to 10. However, and considering the relatively low overall quality of the annotations, the price was to be increased to $0.20. Restrictions were also to be imposed in order for a task to be accepted. These included selecting a category and providing at least 2 tags and a meaningful description. The amount of annotation types were also to be reduced, specifically, determining the type of the page and assigning a rating were removed from the wizard. The idea is that these adjustments would lead to better quality annotations.

Second Iteration

Given the adjustments identified after the analysis of the first iteration, the second iteration was set up as shown in Table 2.

<table>
<thead>
<tr>
<th>Number of Submissions</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reward per task ($)</td>
<td>$0.20</td>
</tr>
<tr>
<td>Duration limit</td>
<td>10 minutes</td>
</tr>
<tr>
<td>Restrictions</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 2: Setup of the second iteration.

The results of the second iteration were already much better. Although 10 documents were erroneously identified as Web APIs, 80% of the correctly classified Web APIs were accurate and had very satisfying annotations. This time round, there were 27 unique contributors and the average amount of time to annotate took 02:36 minutes. It is interesting that despite more strict rules for a task to be valid, the increase in average time taken was not that high. This is very positive since it is clear that the tweaks we provided improved the quality of the annotations without sacrificing too much the time taken to obtain those quality annotations.

Considering these results, we decided to keep the same setup and rules for a next final iteration. However, we also introduced a survey at the end of the process in order to get more information about the worker. This is described next.

Third Iteration

The setup for the third iteration was the same as for the second (Table 3). Rather than improving the already good results obtained in the second iteration, the focus of this iteration was to get information about the workers through a small survey that we established for this purpose.

<table>
<thead>
<tr>
<th>Number of Submissions</th>
<th>150</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reward per task ($)</td>
<td>$0.20</td>
</tr>
<tr>
<td>Duration limit</td>
<td>10 minutes</td>
</tr>
<tr>
<td>Restrictions</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 3: Setup of the third iteration.

The survey results are summarised in Table 4 below. Note that the number of survey submissions is equivalent to the number of distinct workers. No particular conclusion can be drawn from the age group distribution. The bulk of the workers are aged between 23 and 30 years old. The most interesting observation is related to the profession. As expected, most of the contributors work in the IT field and we observed that they were also the ones providing the best quality annotations.

<table>
<thead>
<tr>
<th>Survey submissions</th>
<th>48</th>
</tr>
</thead>
<tbody>
<tr>
<td>Male workers</td>
<td>30</td>
</tr>
<tr>
<td>Female workers</td>
<td>18</td>
</tr>
<tr>
<td>Popular origins</td>
<td>India (27), USA (9)</td>
</tr>
<tr>
<td>Popular age groups</td>
<td>15-22 (12), 23-30 (18), 30-50 (16)</td>
</tr>
<tr>
<td>Profession</td>
<td>mostly IT</td>
</tr>
</tbody>
</table>

Table 4: Survey results.

Conclusion

This paper has shown how the combination of crowd sourcing and adequate annotation tools are an effective means to obtain good quality annotations for Web Services effectively. This is a very positive achievement in general for the Semantic Web community, and in particular for Semantic Web Services. As described earlier, one of the main bottlenecks of the Semantic Web is in fact the acquisition of the annotations and any means to eliminate - or reduce the burden - of that bottleneck is an achievement. Nevertheless, part of our future work involves submitting larger batches and tweak further restrictions in order to maximise the quality of the annotations as much as possible. In the bigger picture, the annotations using crowd sourcing will serve to bootstrap the amount of annotated Web Services on the portal and thus yielding to better search results and experience to the user. Another very useful case for these bootstrapped annotated Web Services is to use it as a positive set for the Web API detection algorithm, mentioned earlier in the text. The bigger this positive set, the more accurate the detection will be.

References

