Automatic extraction of local events from web sites

Anton Risberg Alaküla
Lund University, Faculty of Engineering
Computer Science
ada09ari@student.lu.se

Karl Hedin Sånemyr
Lund University, Faculty of Engineering
Computer Science
dt06kh1@student.lth.se

Abstract

In this paper the capabilities of automatically extracting information from various different web sites that lists events are explored. This will specifically be done in the context of formatting the scraped data as RDF, a data format that can be be processed directly and indirectly by computers. Having the data formatted as RDF further allows for interesting applications to be built upon - this is something that also will be explored to some extent in this paper. In conclusion the benefits of having data formatted as RDF are many - this point will be made throughout the paper.

1 Credits

This project is possible thanks to the effort of many different people including Pierre Nugues, the project leader. Håkan Jonsson, stakeholder from Sony. Tobias Arrskog, Peter Exner & Peter Norlander, who together with Pierre and Håkan created the original locevex[1] project[6]. Finally, Karl Hedin & Anton Risberg Alaküla, who wrote the additional code related to this project.

This document was put together by Karl & Anton.

2 Introduction

The basis for this project is a master thesis project (locevex) performed by two students at Lund University, Faculty of Engineering in collaboration with Sony Mobile (Arrskog and Norlander). One of the master thesis project goals was to gather information on the web about events with the use of software. This concept is commonly known as web scraping. The result of the master thesis project was a set of web scrapers that could scrape event data from multiple different websites and store it formatted as JSON in a CouchDB [2] database. There was also a small django-based API in order to access the data and some generic scraper technology built upon Javascript which wasn't used in this project.

In the spring of 2013 a project was started with the goal of converting the data coming from the locevex project into RDF and then make that RDF data searchable via a SPARQL endpoint. This could then be used as a knowledge base to build an applications upon, for instance for smartphones or PC's. Such application would be able answer the questions such as "what is happening near me in the near future".

It was planned that Sindice[5], an organization that hosts SPARQL endpoints, would collect the RDF data produced and present it in their own SPARQL endpoint. Unfortunately due to time constraints this didnt happen in this project. Instead, a small custom SPARQL endpoint was set up. This will be discussed further later in this paper.

3 Converting to RDF

RDF or `Resource Description Framework` is a language which can be used for describing information in "triples". Triples in RDF is information stored in the format:

```
subject - predicate - object
```

For example:

```
Bob - age - 19
Lisa - age - 19
```

These triples can then be searched with the help of a query language called SPARQL. In sparql you can specify what you know and request the holes in your knowledge filled. You could ask a question like this:

[1] The name of the original project this project is based on is "Hyperlocal Event Extraction of Future Events". This will be referred to as "locevex" throughout the document

```
Who is aged 19?
```
Which in the same format as above is:
```
?  - age - 19
```
SPARQL will fill in the question mark and return Bob & Lisa to you.

If you already know that there is a person named Bob but you don‘t know his age, you could ask SPARQL something like this:
```
Bob - age - ?
```
And SPARQL will return 19. Perhaps you know that something related to Bob has the value 19, but you don‘t know what. Then you ask:
```
Bob - ?  - 19
```
And SPARQL returns ”age”. One of the reasons this project wanted the data in RDF was that once a SPARQL endpoint was set up a very simple SPARQL query could perform very complex tasks. For example, you could have a request like this:
```
?  - is - event
?  - starts - today
?  - city - Lund
```
SPARQL would then fill in the question marks with all events that are today in Lund (or another city of your choosing). This is just pseudo code, but the actual query can be written in less than 10 lines of code. This is a very powerful tool! With this you could build an application for searching events and make the search part complex without much effort on your side.

RDF can be written in multiple ways. It has multiple ”syntax notations”, each with their own strengths and weaknesses. Since the data is going to end up at Sindice, they got to choose which particular syntax that was going to be used in this project. They chose ”Turtle” syntax. One of the strengths of Turtle syntax is that it‘s quite easy to read so the choice was welcomed while the system was being developed.

Now, on to the RDF conversion. All the data from the scrapers was available in a CouchDB database. A CouchDB database can, as it turns out, quite easily be dumped to a json file with a small shell script. That takes CouchDB out of the picture. Now, the problem is getting JSON data to RDF.

Parsing JSON with java is quite easy using the official JSON classes[4]. Writing to RDF is a bit trickier. In this project Apache Jena[3] was used. Apache Jena is, as stated on it’s official web page:

> “... a Java framework for building Semantic Web applications“

One part of this framework is a RDF parser and writer. Unfortunately it isn‘t widely used, so there is very limited support on the web when the inevitable issues appear. This was somewhat solved due to the fact that the project is open source, so whenever code didn‘t work the way you expected it to you could easily look up why. When it comes to the file format (Turtle/RDF) there is a massive amount of resources online for validating documents, examples files, forum discussions etc. so validating the output of Apache Jena was no issue.

While writing this RDF converter the challenge wasn’t only to convert the data. It was also making sure that the converted data was kept up to date. If a certain event changed time, location, or maybe even was cancelled the converted data should reflect that. This was solved by the RDF converter always keeping track of what the CouchDB dump looked like the previous time the program ran. It could then with these two dumps classify all events into one of the four categories:

1. **New**. This event hasn‘t been seen before.
2. **Changed**. The event exists in the previous dump but with different values.
3. **Removed**. The event has disappeared.
4. **Not changed**. The event exists in the previous dump and is exactly the same.

Then depending on the class, the event was either added to a new file, modified in an existing file or nothing was done to it at all.

The converted files were put in a directory and later served to the web using apache directory listing.

4 Scraper improvements

After the RDF Converter was done it’s output was further inspected. At this time some problems were discovered with the scrapers. One of the problems was that the event descriptions was sometimes short and even missing. The reason for this was that the descriptions were scraped from the search result page where the information was sometimes sparse. When looking further into each event‘s dedicated page there was a lot more information available. The scraping was therefore altered to scrape event information from

each event‘s dedicated page in addition to just the search result page.

Another problem that got discovered was that only <30% of the events got scraped. When looking further into this it turned out that the geocoding, the transformation of an address into geo coordinates, was written in a way that threw away events that would not get successfully geocoded. To deal with this the same approach as with dedicated event pages was used. Some sites hosted dedicated place pages where richer data about the location address was available which then was used instead of the search result page information. This improvement resulted in that roughly 80% of the events got correctly geocoded.

Having geo coordinates for a place is preferable since it allows filtering events in a certain area. When looking into the cases where the geocoding failed it turned out that the authors of the event pages had inputted inconsistent address descriptions and sometimes even complete walking directions. Transforming these into geo coordinates would require sophisticated techniques in order to translate such information into an address. The rule of throwing away events that would not get correctly geocoded was dropped - in those cases the event would be kept but it would lack the coordinates. The reason for this is that it could still be of interest to search for a string within a place name.

The scraper with the most problems of them all was the one for lund.se. The main problem with was that lund.se had moved all of their events to a new page; visitlund.se. This page was built entirely differently compared to lund.se, so the old scraper couldn‘t be used at all anymore. An entirely new scraper was built from scratch for visitlund.se. It used the same method discussed earlier of visiting events dedicated subpage for more information. Once completed, the scraper for visitlund became the most important one. Now, 60% of all events come from that page. And as in almost all kinds of AI, more data is always good.

4.1 SPARQL Endpoint

To make it possible to answer the example question mentioned in the introduction - “What is happening near me in the near future?” - a SPARQL endpoint was needed. By this point, the plan was that Sindice‘s SPARQL endpoint would be up and running so it could be queried for events. Unfortunately they were not ready so another (temporary) solution was needed.

It turns out that the Apache Jena library that was used for reading and writing RDF also contained a SPARQL processor called ARQ. It‘s capabilities was explored and it was determined sufficient as a temporary solution.

In order to be able to input the queries in an easily accessible manner a HTML/PHP form was written that directed the SPARQL query to ARQ which then got presented back in the browser. The browser would pre-fill the query window with a SPARQL query that searches for all events around Lund. This makes it a little easier to adapt it to your own queries, but it‘s still pretty complex. This method had two major drawbacks. First of all it requires you to have some knowledge in the SPARQL querying language in order to modify the queries to one‘s preference. Secondly the resulting output was not very pretty, as it was pure JSON.

4.2 Search Engine

The drawbacks of the SPARQL Endpoint mentioned above led to the development of a “search engine“ for events. This search engine would eliminate the need for knowing SPARQL syntax and it could present the results in a much more user-friendly fashion.

The first part of this search engine consisted of building a search form with suitable fields and options. The options that were chosen were *where*, *from date* and *to date*. These would be filled out by the user which when posted would get extracted to form the basis for the SPARQL query. To handle all this a combination of HTML, CSS, PHP and Javascript was used. Since the event sites that were being scraped mostly contained events from the Lund/Malmö area the options for these two cities were chosen. Further, a choice for My Position was added that uses HTML5 to fetch your current location. If you would visit the site from a mobile device the location would be even more accurate since it would utilize the GPS. When searching within a city a suitable center point was located with the help of Google Maps and a large enough area to cover the the city was calculated in order for the query filtering to work as intended.

The second part of the search engine consisted of presenting the search results in a good manner. The result page got formatted with CSS and each

event got a dedicated box containing the event name, description, time and address. If the event had been successfully geocoded the address would be a hyper link taking you to Google Maps with the point marked on the map.

A final improvement that was made to the search result page was to include a Google Map with the area used for the search and each found event marked on the map. This was done with the use of Google Maps JavaScript API v3 [1]. A feature was also added so that each marked event on the map could be clicked which would take you down to the event in the search result list. In the case when many events occurs at the same place they get randomly scattered close around the point to avoid them overlapping.

5 Results

The server used in the project isn't guaranteed to be up forever, and should a server switch occur the url will most likely change. However, if the server is still up then the following urls will show the result of this project:

1. Converted RDF files: `http://ec2-54-234-94-247.compute-1.amazonaws.com/rdf/`
2. SPARQL endpoint: `http://ec2-54-234-94-247.compute-1.amazonaws.com/sparql.html`
3. Search engine: `http://ec2-54-234-94-247.compute-1.amazonaws.com/searchevent.php`

Please note that the data for the search engine isn't updated automatically, so when you read this document the data for the search engine probably only covers a small, outdated time period, possibly showing no events. Use dates around may 2013 and you should get some results.

6 Discussion

This project has a lot of potential for improvements and branches that could serve as new interesting project areas. One of the more obvious is to write more scrapers in order to get access to more data to serve a larger base for the applications that could utilize the RDF data. This is also essential for any of the suggested AI related areas below in order to have a larger pool of data to process.

A more AI specific area of interest previously mentioned in this paper is the issue of performing even more accurate geocoding. By extracting all cases where the geocoding failed one could look into what kind of patterns and classes of cases that these exhibit and then apply appropriate analysis in order to be more successful with the geocoding.

Another area of improvement suggested by Pierre Nugues is performing classification of the events. This would require looking into what categories that the major event web sites has and try to extrapolate what categories that could serve as a common framework for all scraped events. One could then use various AI techniques to automatically classify each event into one of these classes.

Due to the delays with Sindice we have been unable to perform queries against their SPARQL endpoints. However, their endpoints are now up and running so someone should look into using them and/or adapting the search engine to take data from Sindice instead. Switching to Sindice's endpoint could also provide access to events that wasn't scraped by this project but that are still compatible. Perhaps the search engine built in this project could suddenly be usable for events all around the entire world.

It needs to be pointed out that web scraping could run into legal issues so depending on the intended use and scale one must look into the terms of use for each site being scraped.

Acknowledgments

We would like to thank everyone involved for the opportunity to partake in this project. Special thanks to Pierre, for providing us with information and material necessary to start the project. Special thanks also to Håkan, for guiding us and on a more detailed level through everything we did. We have learned a lot throughout the project and we hope that the results will be of use to someone.

References

[1] Google Developers. *Google Maps JavaScript API v3*. June 5, 2013. URL: `https://developers.google.com/maps/documentation/javascript/`.

[2] The Apache Software Foundation. *Apache CouchDB*. June 5, 2013. URL: `http://couchdb.apache.org/`.

[3] The Apache Software Foundation. *Apache Jena*. June 5, 2013. URL: http://jena.apache.org/.

[4] JSON. *JSON in Java*. June 5, 2013. URL: http://json.org/java/.

[5] Sindice. *Sindice - The semantic web index*. June 5, 2013. URL: http://sindice.com/.

[6] Arrskog T. and Norlander P. *Event Extraction for Contextual Point of Interest Databases*. Department of Computer Science, Faculty of Engineering LTH, Lund University. 2012.