# Design and development of an enhanced RSS feed reader

Olivier Jais-Nielsen (s090763) and Andrea Lai (s091088)

Technical University of Denmark, Anker Engelunds Vej 1,
2800 Kogens Lyngby, Denmark

**Abstract.** The rise of Web 2.0 has contributed to the rapid growth of information available, and it can often be a time-consuming process to navigate through it all. While RSS has been able to help make the increase in online self-publishing more manageable, media content is continually increasing in a wide variety of formats. This additional information can provide valuable contextual background to information presented within an RSS entry, offering a richer reading experience. The concept of the semantic web has emerged in recent years, describing a more automated organization of online information through analysis of language. In this paper, we describe the design and development of an application that makes use of textual analysis to identify people, places, and organizations within an RSS entry and automatically provide relevant background information about the items, sparing the user the need to interrupt his or her workflow by having to perform a separate search to obtain relevant context. We further extend our RSS reader by providing an interface for examining the similarities and associations between people, places, and organizations named in a feed. A working prototype written in Python is included with this report.

**Keywords:** web 2.0, semantic web, RSS, mash-ups, Python, XSLT

## 1 Introduction

Web 2.0 is a term coined around 2004 to describe the increasingly social nature of the internet [1]. Users gradually stopped simply passively receiving information and sites were developed that allowed users – the everyday average individual – to start contributing content of their own. The publishing of information has not slowed, and a number of methods have arisen to help manage the steady flow of information.

One important factor in dealing with the growing amount of data available online has been the standardization of publishing formats. “Really Simple Syndication” (RSS) allows for the automatic syndication of content, and grew in popularity in usage alongside Web 2.0’s development. Indeed, one today is unlikely to find a blog or other published online content that does not allow for such aggregation.

Though the usage of RSS may still be growing – in 2005, it is cited that only 9% of Americans said they knew what it was [2] – there is little doubt that content aggregation adds a lot of value to our online experience. Indeed, over time, we have seen the rise of social networking sites such as Facebook, which not only allow the

publishing of a variety of content all within one page, but also the means to do so within a framework of one's friends and acquaintances. The Financial Times notes that "social networking sites are changing the way people navigate the information landscape and share and consume media. [3]"

The rise of Web 2.0 has changed the way we move around the internet. Some rely heavily on content aggregators, be it a feed reader or a social networking platform, to keep up with the flow of information. Mechanisms behind information sharing, no doubt, have changed and applications that facilitate the flow of information must also reflect the dynamism that accompanies Web 2.0.

The Semantic Web has grown out of this accumulation of online content, and has been described as a "vision of information that is understandable by computers, so that they can perform more of the tedious work involved in finding, sharing, and combining information on the web [4]." Ontologies, defined in this context as a schema to set hierarchies and relationships between different resources [5], allow computers to parse natural language into something more machine-friendly. Their aim – at least, in the context of the web – is to provide a common vocabulary for a given domain, integrate data sources, and analyze the domain knowledge. [6]

Furthermore, given the vast amount of information available online today, tools can be developed to try and uncover relationships hidden in data. As long as there are comparable data points, the Euclidean distances or Pearson correlation between objects can be determined, and such data analysis has numerous applications.

In this paper, we examine the syndicated feed reader and how semantic analysis may be applied to facilitate the organization of the acquired content. How can one obtain the most information out of the fewest numbers of click-throughs? We outline the development of an application to not only add additional layers of information to one's RSS feed, but also to allow for a more user-intuitive method of sorting and moving through the data.

It is worth noting that the hope is that this linguistic analysis can be applied to micro-blogging platforms such as Twitter as well. However, due to the 140 character limit, which adds the additional complexity of having to infer context from very little information, we have limited our application development to traditional RSS feed readers for the time being.

## 2 Problem Definition

Though we couldn't find any numbers as to exactly how much time a person spends on average sifting through his or her subscribed RSS feeds, we assume that it takes a fairly reasonable amount of time to get through them all. The number of productivity blogs encouraging one to "unsubscribe to all RSS feeds that aren't unmissable [7]" or some other variation and the 386,000 Google hits for "RSS bloat" [8] suggest a need for some better mechanism to organize and go through one's (often lengthy) list of RSS feeds.

Not only might there be a lot of content to be covered in one's aggregate feed, the feeds themselves often span a large variety of topics with no particularly structured way of keeping track of them. Furthermore, a news feed, for example, may contain a

reference to a person or place with which you are not familiar, and then the workflow may be interrupted if you need to look up information from a new source. While this is, admittedly, a small task, it is one that comes up sufficiently often to be worth looking at.

Take, for example, the following post from *The Dieline*, a design blog:

> *New work by Laura Berglund, student at The Kansas City Art Institute: "This project was focused on branding a self-proposed topic, mine being a hot air ballooning company, Reve (a french term meaning "to dream", since the hot air balloon originated in France) that ends each tour with a picnic and champagne. Most ballooning companies out there are very childish and unrefined, so my goal was to create a ballooning brand that was more than that– something classy and sophisticated, yet still with an easy-going and friendly vibe, to keep it approachable.*
>
> *Many more images of this beautiful line after the jump!*
> *Continue reading "Student Spotlight: Laura Berglund" »*

The post in question (found here: http://www.thedieline.com/blog/2009/11/student-spotlight-laura-berglund.html) includes photos of her work, along with links to look at a larger image. However, after a cursory read-through, I want to know – who is Laura Berglund? Does she have her own design site? What can I find out about the Kansas City Art Institute? There are additional layers of information that an RSS feed alone doesn't provide, and it takes additional time and human effort to obtain them.

Another limitation of feed readers is that they only compile ones items by the blog or other source from which it came, not the content itself. Thus, the topics – let's say an upcoming political election, for example – can be scattered throughout your unread feeds, and the same topic may unnecessarily come up over and over again. Along similar lines, there is no easy way to identify feed items with related topical content within your own feed aggregate.

Indeed, a lot of applications already exist to help one find new feeds to read – but as far as answering the question of *how* we process our existing (and often plentiful) feed lists, there appears to still be room for work. This will be further examined in the following section.

# 3 Analysis

## 3.1 Examination of existing applications

Before we began development of an application that could address our defined problem, the seeming lack of tools to change the way users interact with RSS and the overabundance of feed discovery tools, we had to first take a look at what was already

out there. We identified a few tools that organized RSS feeds by semantic terms, such as “Feedzz” and “Power RSS” (unfortunately, both seemingly defunct) and a few others that pulled in other information based on RSS content, but we could not identify any existing products that included both features.

First, we examined the application of semantic and other linguistic analysis to RSS feeds. Indeed, the idea of using more sophisticated language tools to organize our reading material is not a new one. At least three applications found through Programmable Web apply semantic web tools to RSS feeds: Feedzz, Power RSS, and TagCloud [9, 10, 11]. The first two are now defunct, which might lead one to question whether this is a valuable problem – if applying linguistic analysis to RSS feeds is so useful, why did these two applications fold? However, the last one, TagCloud, is a ranking tool that evidently is under overhaul due to high popularity.

The other category of enhanced RSS feed readers we looked into were those which offered a mixed-media content display based on RSS. One such example was *San Diego Fire Feeds* [13], which used Google news feeds to identify articles relevant to the topic and pull in information from Twitter, Google Maps, and Google search.

Curiously, of the noted applications above, only TagCloud still has a valid webpage associated with it. One might argue that this demonstrates that an application need not be developed to address this problem – it is, after all, a minor one. San Diego Fire Feeds likely folded since it is a pretty niche interest; similar mash-up content is continually produced, and our application aims to be more generalized. As for the semantic organizers of RSS feed, it is possible that the other applications suffered from a bulky or non-intuitive design. A cursory glance at TagCloud’s website shows sleek, lightweight site-design, which keeps very much in line with the Web 2.0 aesthetic. One of our design challenges, then, is to keep a clean interface design in mind while managing a whole host of content sources. Our design process is detailed in Section 4.

### 3.2 Room for improvement

As far as we could tell, there are no published applications that combine both semantic analysis and a more conventional mash-up of online content. We believe that the combination of both elements can contribute to a more pleasant RSS-feed reading experience. In Section 2, we demonstrated some of the questions that may arise while reading a blog post. The figure below aims to illustrate how the RSS feed reading experience might be improved. This time, the example post (again, from *The Dieline*) provides a link to the design studio, giving some context, but I want to know more about the beer and the brewing company as well. As such, I find that I then have to search Google myself for more information about the product or some other detail that may have influenced the design. It is a small task, but it is a tedious one, and I’d much prefer to have the context on screen as I navigate their posts.

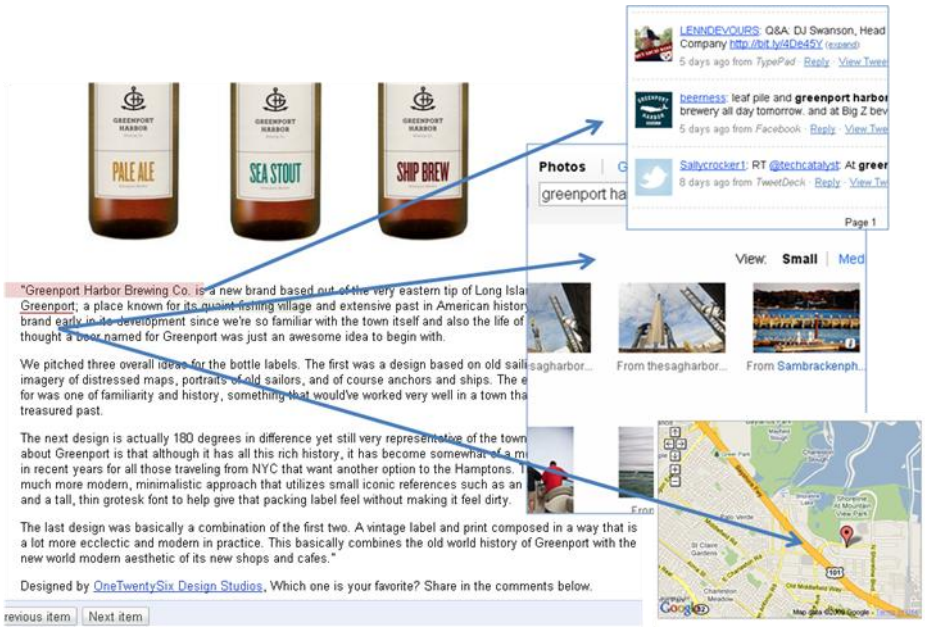


**Figure 1: Illustration of the sort of information lacking within a given RSS entry and what kind of information is also pertinent**

In Section 2, we briefly touched on how efficiency can be lost when one has to look up a number of these separate references. The rise of a "mashable" internet culture can contribute to a solution that can display the relevant content in a single interface, enhancing the original syndicated content and reducing the user's need to switch from page to page to obtain a rich reading experience.

# 4 Design

## 4.1 Overview

Indeed, a lot of applications already exist to help one find new feeds to read – but as far as answering the question of *how* we process our existing (and often plentiful) feed lists, there appears to still be room for work. In the previous section, we discussed some existing applications that do add another dimension to traditional RSS feeds, identifying those that either apply some linguistic analysis or mash-up a variety of other media content. Semantics analysis is continually improving – why not combine the two? This solution thus utilizes the organizational potential of the linguistic analysis and the enhanced reading experience that arises from the layering of additional media content.

We envision our application to, at minimum, identify people and places within an RSS feed and integrate a map and Google search (based on the extracted content) to

enrich the display. People and places, after all, are often the unfamiliar elements that require further look-up as we read. Our proposed minimum prototype identifies what we consider to be essential to one such application.

Our idea was to reduce the interruptions to ones reading that may arise from unfamiliar content items by making them all available on one display. Ideally, this display would be fairly unobtrusive (either collapsible or available only on hover) and smoothly integrate a number of other available API content.

A further expansion of this design could also store element items (people and places) to a database, allowing the user to sort his feeds with a kind of auto-tagging system built upon the linguistic analysis. As noted in the previous section, a clean, intuitive interface design is essential to a useable product. An initial picture of what we had hoped to achieve can be found below in Figure 2. In general, we have to always consider any costs additional features may have on overall usability.

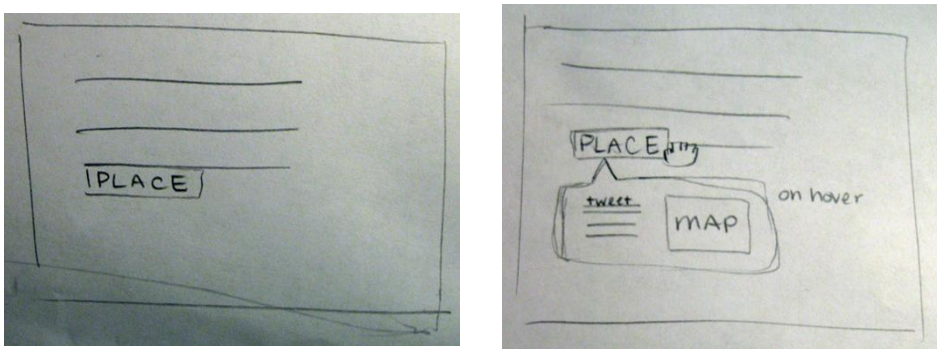


**Figure 2: A quick sketch of how one might want additional content to be displayed with the feed**

The following table shows what kind of information could be valuable when linked to a given entity:

| entity | linked information |
|---|---|
| **LOCATION:** | Google Maps, Flickr |
| **ORGANIZATION:** | Google Search, Twitter |
| **PERSON:** | Google Search, Twitter, Facebook/LinkedIn |
| **SONG:** | Grooveshark, Last.FM, Youtube |

**Table 1: A look at some applications that could offer interesting supplementary data**

### 4.2 Test cases

In order to evaluate the success of our application, we will see how well it performs in context of the problem posed earlier in Section 2. Given a blog post – we could use the very same about Laura Berglund from *The Dieline* – can appropriate entities be identified? Does the content that is automatically generated from the entity answer some of the questions we had posed? Is it relevant? In this particular case, I would expect a Google search to identify Laura Berglund and her online portfolio, as well as search results for the art institute and its location on a map.

We would also have at least a few others not involved in the work on this project test our application and see if they can understand what it is supposed to do and how to use it. If they succeed (and they should), we can demonstrate that the prototype has not only an intuitive purpose but an intuitive interface as well.

## 5 Implementation

This section discusses the coding behind the application. Below, Figure 3 provides an overview of the program structure.

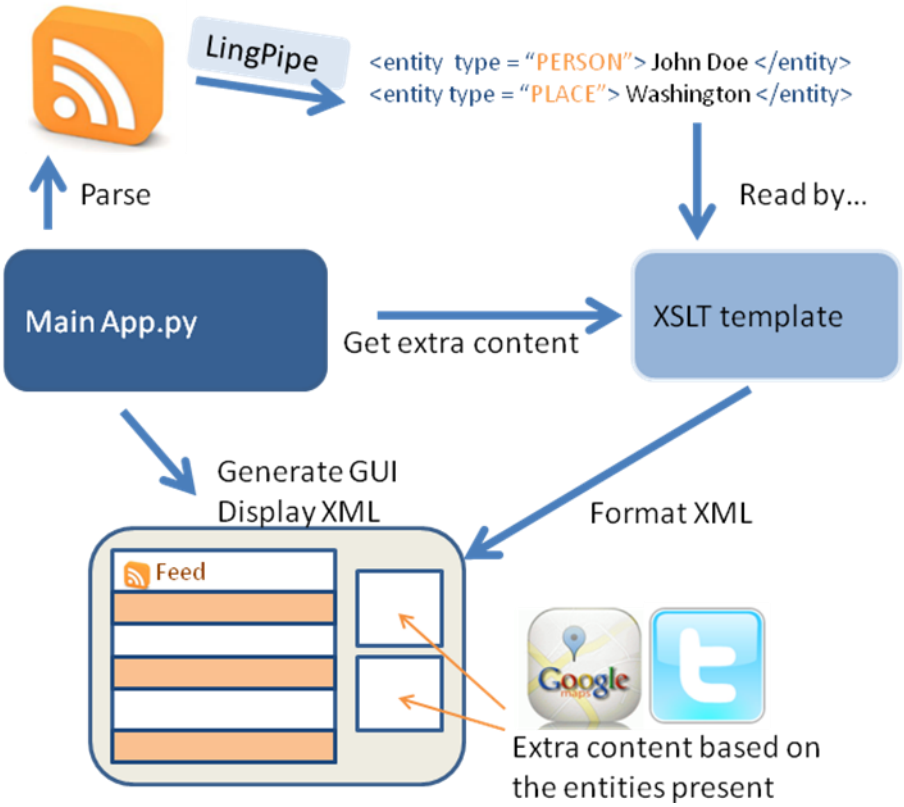


**Figure 3: An overview of how the RSS feed reader obtains additional content for display**

The main program, enhancedfeedreader.py, takes an RSS feed as input, which is then read by LingPipe. The entities of organizations, people, and locations are then marked by XML tags. XSLT (extensible stylesheet language transformations) can be used to not only format XML documents, but also transform XML data into HTML and CSS documents.

With the help of the Python package "lxml" – specifically, the ElementTree API, Python methods in the main program can be called from the XSL document. The template file ultimately formats our final output, and XSL is impressively flexible, and languages can, to an extent, be mixed.

We use the Python application to make calls to additional APIs and other data, and it generates our desired output. However, it needs to be able to receive a variable entity as an input from the XML, and the output has to be passed back to the XML to be displayed. This is where lxml comes into play – its ElementTree component provides a class that allows for the construction of XSLT transformers, so the desired Python methods can be called from the template document as well.

We used PyQt [13], a set of Python bindings for Nokia's Qt UI framework to develop a useable interface for the enhanced feed reader. A more comprehensive discussion of the application in action can be found in Section 6.

### 5.1 Internal XML Flow

The application's internal data structure is based on XML. The graphical output uses HTML. Here is a list of the structures involved.

- The feedparser output. From the feedparser module, the program gets, for each entry, the title, date, web link and the HTML rich text.

Example of HTML rich text:

```
<div>
 John Smith lives in Washington.
 <br/>
 He works for Microsoft Corporation.
</div>
```

- The Lingpipe output. The program feeds Lingpipe with HTML rich text, which outputs HTML rich text with extra HTML tags describing the entities found in the text.

Lingpipe output example:

```
<HTML>
 <BODY>
  <DIV>
   <s i="0">
    <ENAMEX TYPE="PERSON">John Smith</ENAMEX>
    lives in
    <ENAMEX TYPE="LOCATION">Washington</ENAMEX>
    .
   </s>
   <BR/>
```

```
     <s i="0">
      He works for
      <ENAMEX TYPE="ORGANIZATION">Microsoft
Corporation</ENAMEX>
       .
     </s>
    </DIV>
   </BODY>
  </HTML>
```

- The internal database. It is an XML document containing, for each entry, the title, the date, the web link, information about the entities, and the HTML rich text with entities references marked. The entity marking process is discussed in more detail in Section 5.2 For each unique entity, the information stored is the real name and, for locations, the spatial coordinates if they can be found by GeoPy.

Internal database example:

```
  <efrDb>
   <entry title="Life of John Smith"
          link="www.john-smith.com/news"
          date="Tue, 01 Dec 2009 16:27:11 GMT">
    <head>
     <entity type="PERSON">John Smith</entity>
     <entity type="LOCATION"
             coordinates="38.8951118 -77.0363658">
      Washington
     </entity>
     <entity type="ORGANIZATION">Microsoft</entity>
    </head>
    <body>
     <div>
      <span class="SENTENCE">
       <span class="ENTITY PERSON" title="John Smith">
        John Smith
       </span>
       lives in
       <span class="ENTITY LOCATION" title="Washington">
        Washington
       </span>
       .
      </span>
      <br/>
      <span class="SENTENCE">
       He works for
       <span class="ENTITY ORGANIZATION" title="Microsoft">
        Microsoft Corporation
       </span>
       .
      </span>
     </div>
    </body>
   </entry>
  </efrDb>
```

- The HTML output. It defines what the main window displays as rich and dynamic HTML (using CSS and JavaScript). It can be either feeds from a database document, in which case extra information about the entities is added, or tag clouds from a list of name associated to a number that defines the relative size.

All XML transformations are performed by XSLT style sheets.

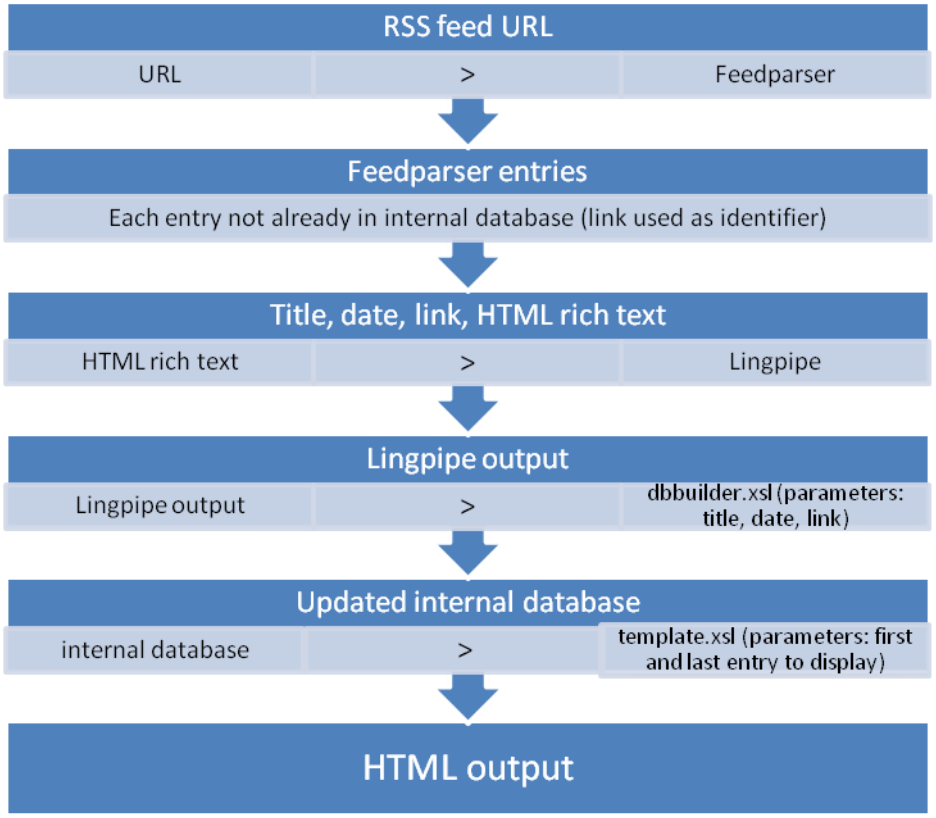


**Figure 4: Feed display pipeline**

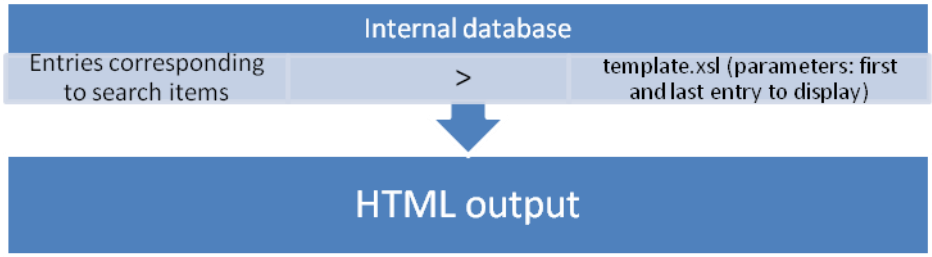


**Figure 5: Search display pipeline**

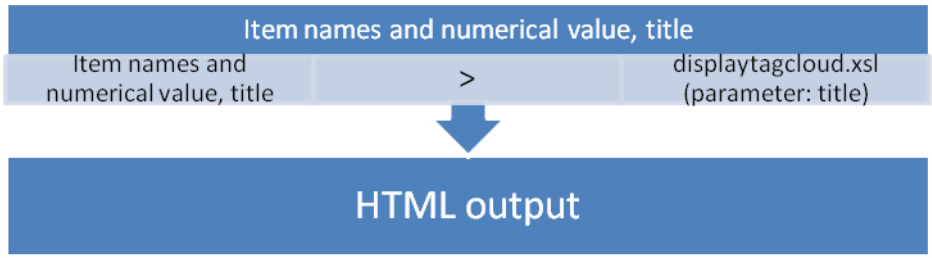


**Figure 6: Tag cloud display pipeline**

### 5.2 LingPipe

One of the first steps in creating the enhanced feed reader was of course, identifying a linguistic analysis program that would meet our needs. The hope was to find one based in Python; however, the best free solution we could find was LingPipe, self-described as "a suite of Java libraries for the linguistic analysis of human language [14]." There are a variety of tools available, ranging from topic classification to parts-of-speech tagging, but we chose to make use only of their "Named Entity Recognition". In text, this module identifies people, locations, and organizations.

For prototyping purposes, these categories are sufficient. We did not think it would have been a valuable application of our time to try optimizing the semantic analysis performance since LingPipe functions sufficiently well for a proof-of-concept. In order to remove some of the false positives (items identified as entities that are in reality, insignificant), each item is queried for a Wikipedia entry and only accepted as an entity if one is present. Admittedly, this method fails for applications that are more personalized than, for example, parsing a news article; however, we will leave that work for a future extension of the project. Even so, the performance isn't perfect – "Kanye West", for example, is identified as a location rather than a person.

### 5.3 getExtraInformation

The method by which additional content is added to the feed reading application is previously discussed in Section 5.1. Before implementation of these methods, we had to identify what content would be appropriate for our application.

Google, of course, is customarily the source we turn to when we are trying to identify new information. As such, it was the first logical choice for a source from which our RSS feed should extract additional information. Google Search API [15] allows for easy web and multimedia search, perfect for gathering information about people and organizations.

In addition to integration with Google, we also thought that it would be valuable to bring in some of the more Web 2.0 elements into play, and we thought Twitter could be a nice complement to news data obtained from RSS feeds. That way, not only is graphical data about a given topic at your fingertips, you can also get near real-time snippets about it. We made use of Twython [16], a Python wrapper for the Twitter API which includes search functionality, to do so.

### 5.3 The database browser: Identifying similarity

For each entity in the database, a vector is associated. It is composed of two parts. The first contains, for every other entity, how often both are simultaneously present in a feed entry. These values are normalized by the total number of feed entries, so the maximal possible value is 1. The second part contains the corresponding earth coordinates for every other entity that is a location.

A distance is defined between two vectors as the combination of two parts. The first is the Euclidian distance between the first part of each vector. The second represents the distance between the second part of both vectors, which are earth coordinate sets. It is calculated as follows:

- For each point of each set, we compute its spatial (spherical) distance to the other set (as the minimum of the distances to all the other set's points)
- We take the mean of all these values.
- We normalize this value so the maximum possible is 1.

This defines a distance between entities. Additionally, we can calculate this distance making use of only specific entity types to calculate the vectors.

The program provides a database analyzer which offers the following functions:

- For a set of user specified entities
    - Display every feed entry containing simultaneously the entities.
- For a specific entity:
    - Display the other entities of the same type with their similarity to the reference entity as a tag cloud. It is possible to choose which types to use to calculate the similarities.
    - Display the entities simultaneously present with the reference entity in feed entries and how often they appear as a tag cloud.
    - Display on a map the localizations involved in entries where the reference entity is present, and how often. Some basic clustering is

performed here: localizations are represented as dots. When several dots overlap, they are merged into a bigger one, centered in the center of mass of the merged dots.

# 6 The application in action

## 6.1 Initiation

On launch of the program, one finds a menu containing the option to load an RSS feed (url as input). A progress bar will show that the program is, indeed, doing something, and the output will be the "entity database", an xml file that now includes the sentences and entities identified by LingPipe. This xml file can then be saved and reloaded at a later date. After reloading an entity database, one can load another feed and the new entries will be added to the database. The interface displays the "enhanced feed", an html output generated from the database. At the bottom of the interface, there are arrows for paging through the database, as well as the option to set the number of items per page. These items can be seen along with the enhanced feed in following subsection.

## 6.2 The enhanced feed

When the RSS has loaded, the display is not unlike that of any other RSS feed reader. You can click on the title for the full article, and the information one would expect to see is all present. We didn't want to try to change the way the conventional information is displayed; we wanted to add some additional layers to the functionality, not to entirely redesign RSS. As such, we think a conventional approach here helps keep the application useable since new users to the enhanced feed reader don't feel like they have to relearn how they navigate RSS as well. This minimizes overall confusion. The image below shows the display once the RSS feed has been parsed. Here, we display a feed of the top stories from CNN.

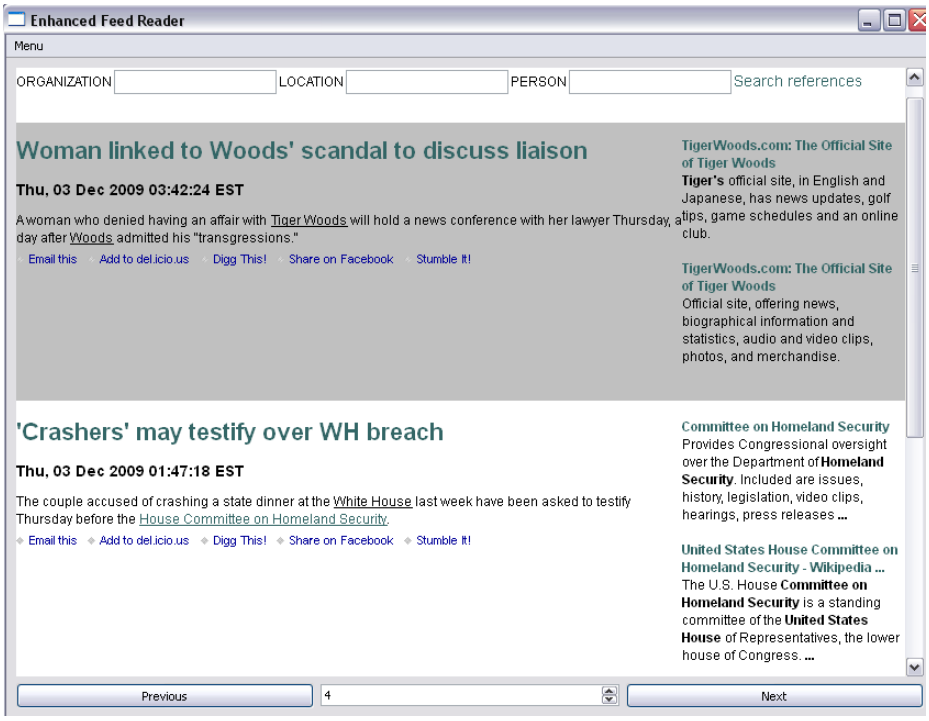


**Figure 7: The enhanced feed reader in action -- the items on the right indicate Google Search results for the people identified by LingPipe**

Note that the entities identified by LingPipe are marked with an underline, and the mouseover is indicated with a change in color, seen in "House Committee on Homeland Security" in the last row. We wanted the additional content to be fairly non-intrusive, so additional information appears in the sidebar on mouseover.

The first two items in our RSS shown above link to Google search entries with their descriptions, while another entry contains a Google Map with the standard zoom and scrolling functions.


**Figure 8: A look at the integration of Google Maps as additional feed content. The embedded map is fully scrollable and zoomable.**

Below, an image of the supplementary Twitter information can also be seen. One can then click on the link to go to the person's Twitter page. Indeed, though the news may be talking about one thing, the micro-blogosphere seems to be commenting on Michelle Obama's fashion sense at this point in time.


**Figure 9: A look at the integration of most recent Tweets regarding a topic as additional feed content. The links lead to the user's Twitter page.**

Table 2 below shows what additional content is provided for a given entity.

| entity | linked information |
|---|---|
| **LOCATION:** | Google Maps |
| **ORGANIZATION:** | Google Search |
| **PERSON:** | Twitter |

**Table 2: The additional content obtained for a given entity in the actual application**

### 6.3 Further extensions – the database browser

The application includes a search bar at the top for references to a specific entity. "Search References" allows the database to be filtered by a given topic. An intersection of topics can be searched using "Add entity". The interface is shown below.

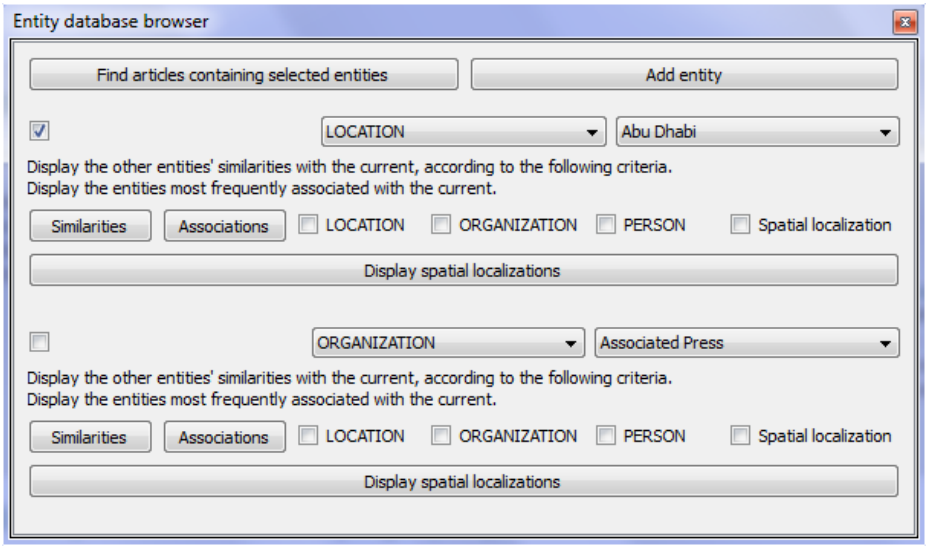


**Figure 10: A look at the entity database browser. It offers not only the ability to search entries, but also to calculate similarities and associations to a given entity and display the results as a tag cloud as well as display on a map localization statistics.**

We can also generate a tag cloud showing similarities between a given item and other defined entities in the feed. Since the relationship between the items is stored in the database (whether or not they are present in the same entry), we are able to make measures of similarity between items based on what items they are often linked to in the news. This can be seen in the figure below.

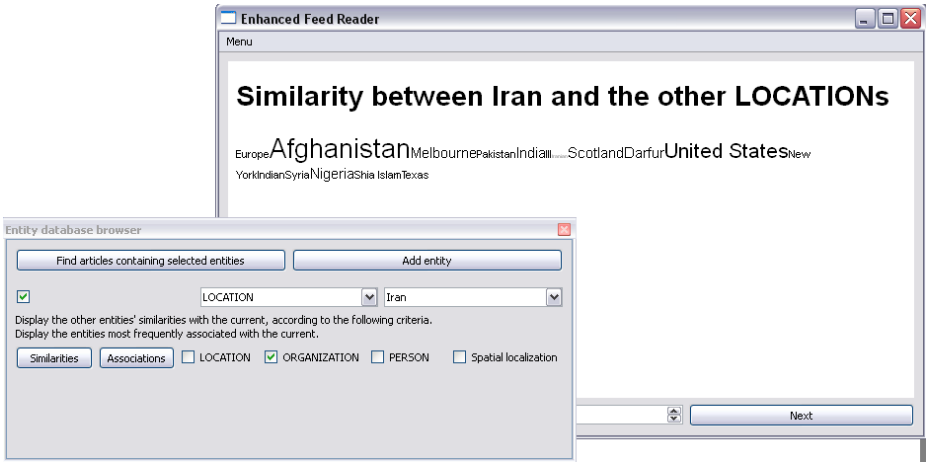


**Figure 11: Similiarity cloud -- location to other location**

Figure 11 shows us that Iran is linked most frequently to Afghanistan and the U.S. if we base our similarity calculations (Euclidean distance) off of organizations. Additionally, we can also see what other entities are associated with another. The figure below shows that Barack Obama is the person most frequently associated with Afghanistan in the recent news.

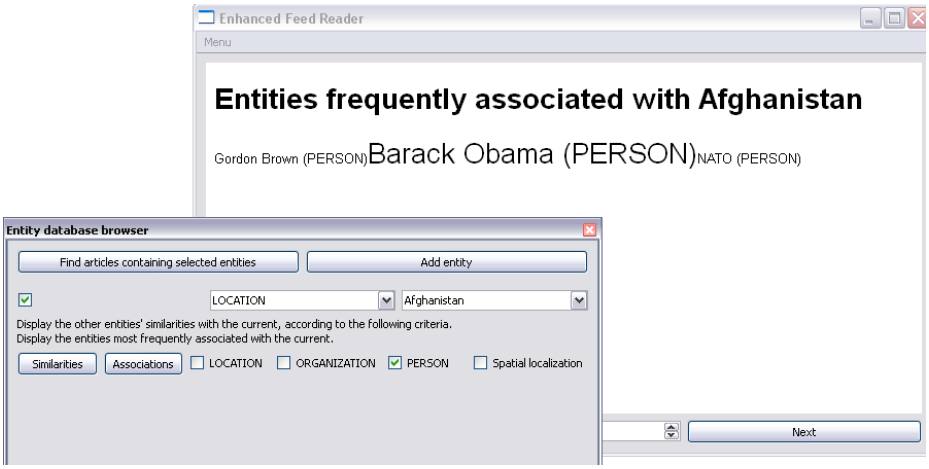


**Figure 12: Association cloud -- people with a location**

Finally, as mentioned before we can display on a map the localizations involved in entries where the selected entity is present, and how often. The landmarks are clustered to emphasize the tendances over wider areas.

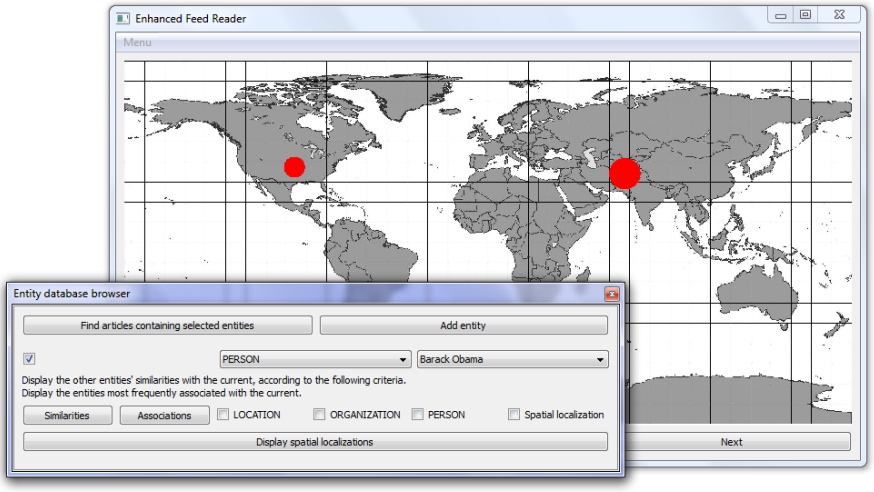


**Figure 13: localizations representations**

Figure 13 shows that currently, Barack Obama is more associated to the Iran/Afghanistan region than to Washington.

When the database browser window is closed, the main displayed is switched back to the enhanced feed.

The similarity and association clouds are, admittedly, rather sparse at this point, but we do demonstrate that such an application is possible. With more entries in the database, we have little doubt that some interesting relationships can be uncovered! Furthermore, this application is not limited solely to news items, but any source which makes reference to people, places, and organizations.

# 7 Conclusion

In this study, we explore some of the potential that has arisen with the Semantic Web. Linguistic analysis can be used to append new labels to items in XML, and we demonstrate that XML metadata can be filtered for greater personalization of content. Furthermore, we propose an application for linguistic analysis in online content to automatically obtain related information from more conventional social media sites (or any other desired location).

## 7.1 Evaluating the application

While the actual functionality of our application may be simple at this stage, it does obtain interesting information without requiring the user to open a new window and perform his own searches.

No doubt, the application could be further tweaked to make the interface a little cleaner, and ultimately, give the user some selection as to which APIs to use to search content. However, the main limiting factor preventing this from this application or something similar from gaining more widespread use is probably the linguistic analysis itself. While LingPipe does an excellent job, it still manages a handful of false positives (identification of non-significant entities), and it sometimes fails to identify the full name of an entity. We work around this by searching for an item in Wikipedia and assuming it is not a significant entity if it fails to appear there; however, for future applications at a more personal level, such as identification of a local restaurant, this will not work. However, as has been mentioned again and again, the Semantic Web has shown tremendous growth in a very short period of time, and we have no doubt that analysis tools such as LingPipe will rapidly improve to a point where such workarounds are not needed.

Ultimately, we do demonstrate that linguistic analysis does have potential use in general online browsing, particularly while we are reading online content. With more time, it'd be interesting to see how well the latent semantic analysis can perform with less input text (such as Twitter and Facebook status updates), since the ability to provide links out to Google Maps and relevant Google searches could also be of value in those contexts.

### 7.2 Further work

One thing that we could do to make the application more accessible is to port it to Nokia mobile devices – QtCore is, in fact, a UI framework developed by Nokia and is cross-platform compatible. Because our GUI is already compatible with Nokia's Maemo OS, it should not be significantly more difficult to make the enhanced feed reader available as a mobile app as well. A mobile specific feature that could be implemented would be to access the GPS coordinates and to add this to the database browser criteria.

Additionally, since LingPipe effectively tags an entity, marking it for easy identification later, there are a number of extensions that could be made to add dimensions to a feed reading experience.

The rise of Web 2.0 has allowed for the combination of data in a number of novel ways and has helped uncover unexpected or interesting relationships hidden in data. In this study, we've worked to tie in linguistics analysis in conjunction with XML metadata to demonstrate how this additional component can identify relevant terms and use them with other APIS to be combined with the more traditional mash-up application.

In the future, we'd hope that the enhanced feed reader concept can be extended to uses beyond very known, global entities such as those found in news articles. With improvements in latent semantic analysis, it may be possible to obtain better entity identification in even short snippets of text, so next time a friend updates his Facebook status or tweets, "Now drinking coffee and getting work done at Taste," the application can map the location and display any other similar recent updates.

Both semantic analysis and general API mash-ups are powerful tools in navigating the vast resources available online today, and we expect that the ability to tie in context-aware (even in the language domain) applications with the user-generated content of Web 2.0 to be able to go a long way.

## References

1. O'Reilly Media, "What is Web 2.0", http://oreilly.com/web2/archive/what-is-web-20.html

2. http://rssdiary.marketingstudies.net/content/rss_still_not_recognized_by_endusers.php
3. Gelles, D, “Friends, not editors, shape internet habits.” Financial Times, Sept. 1, 2009.
4. http://en.wikipedia.org/wiki/Semantic_Web
5. http://www.altova.com/semantic_web.html
6. Wilkowski, B. “Representing technologies, describing the world.” [lecture] Technical University of Denmark, October 21, 2009.
7. http://www.toprankblog.com/2009/09/optimizing-time-spent-in-social-media/
8. http://www.google.com/search?q=RSS+bloat
9. Feedzz, http://www.programmableweb.com/mashup/feedzz
10. Power RSS, http://powerrss.sasthas.com/
11. Tag Cloud, http://tagcloud.com/
12. San Diego Fire Feeds, http://www.programmableweb.com/mashup/san-diego-fire-feeds
13. PyQt, http://wiki.python.org/moin/PyQt
14. Lingpipe, http://alias-i.com/lingpipe/index.html
15. Google Search API, http://code.google.com/apis/ajaxsearch/documentation/
16. Twython, http://github.com/ryanmcgrath/twython