

Mini project

Topics in Operating Systems

Title: External information snippets provider

Igal Shilman shilman@cs.bgu.ac.il



Introduction:

Saya is a humanoid departmental secretary robot. Assembled to provide human-like communication, and provide different kind of information, like faculty members schedule, office numbers etc'.

Saya achieves that by means of speech recognition and speech synthesis.

This project is about extending Saya's response bank by connecting to external content providers.

The goal was to create an efficient, simple dynamic and extendable Java package that enriches Saya with external content sources.

Motivation:

When using speech synthesis based on a set of pre defined rules and a finite answer bank, it is possible to the conversation to get "stuck".

Hence it would have been helpful to be able to extend Saya's ability to retrieve information in real time and dynamically prepare it for speech synthesis. Then finally deliver that information to the querying user.

There were two major methods of choice to provide external content:

- a. Data mining – extracting information out of HTML.
- b. RSS [1] – Rich Site Syndication – Is a name of a family of standards which enables the content providers (i.e., yahoo, AP, ynet, etc'), to expose "snippets" of information to their users.

Today almost all major content provider sites provide an RSS feed to the user, so it seems redundant to prefer data mining over RSS for that purpose.

Project goals:

As promising as RSS seems it's variety of standards bears a problem.

Different content providers are using different standards. Hence one of the goals of this project was to create a unified API and interface which hides all the differences.

Another goal was to provide a level of cache / pre-fetching, since Saya must answer in real time due to user's request we can not fire an RSS request to a distant location, due to network latencies, so a pre-fetching mechanism is required.

Another goal is to provide simple yet reliable method of mapping sentences to a specific content provider. For instance: "local news" would map to "www.ynet.co.il", and "what's new in the world?" would be mapped to "A.P"

Architecture:

There are two main components in this package: **FeedPool** , **KeywordStore**. The **FeedPool** is responsible to fetch feeds from remote hosts and store them in the pool. Periodically this layer renews its feed pools. The layer is backed by Java's Executors Framework and Rome [2] – Rome is an set of open source Java tools for parsing, generating and publishing RSS and Atom feeds. The core ROME library depends only on the JDOM XML parser and supports parsing, generating and converting all of the popular RSS and Atom formats including *RSS 0.90, RSS 0.91 Netscape, RSS 0.91 Userland, RSS 0.92, RSS 0.93, RSS 0.94, RSS 1.0, RSS 2.0, Atom 0.3, and Atom 1.0*. You can parse to an RSS object model, an Atom object model or an abstract SyndFeed model that can model either family of formats (taken from the project description page)

KeywordStore this component is responsible to match user's input (sentences) into the most relevant data provider.

Each web feed is annotated with keywords which describes it. For instance "AP" would be described with the following keyword set {associated, press, what, new, world, international, flash, news, ap, global}.

The incoming user's query will be tokenized, for instance: "global news" will be transformed to { global , news } .

Then for each feed in the **KeywordStore** a proximity rank will be calculated

by the following formula: $p(U, F_i) = \frac{|U \cap F_i|}{|U \cup F_i|}$

When U is the user's tokenized sentence and F_i is the feed under consideration. The maximal p_i value indicates that F_i is the feed requested.

Here is pseudo code of the algorithm:

BestMatch(sentence)

$U \leftarrow \text{split_by_whitespace}(\text{sentence})$

$m \leftarrow 0$

$F^* \leftarrow \text{Nil}$

for $F \in \text{Feeds}$

$p \leftarrow \frac{|U \cap F|}{|U \cup F|}$

If $p > m$

$m \leftarrow p$

$F^* \leftarrow F$

return F^*

User guide:

The most important part of the project is a file called Feed.xml, which describes the supported feeds and the keywords.

A single feed entry is described by:

- Name – the name of the feed.
- URI – the address to that feed.
- Refresh interval in minutes.
- Keyword set – which describes the feed. The users input will be matched against this set.

Here is an example for a simple "Feeds.xml":

```
<feeds>
  <feed>
    <name>ynet</name>
    <uri><![CDATA[http://www.ynet.co.il/Integration/StoryRss3082.xml]]></uri>
    <refreshEvery>1</refreshEvery>
    <keywords>
      <keyword value="ynet" />
      <keyword value="new" />
      <keyword value="home" />
      <keyword value="politics" />
      <keyword value="sports" />
      <keyword value="flash" />
      <keyword value="news" />
      <keyword value="Israel" />
      <keyword value="local" />
    </keywords>
  </feed>
</feeds>
```

After establishing the Feeds.xml, all you have to do now is simply:

```
SayaNews m = new SayaNews();
m.loadFeeds( "Feeds.xml" );
// wait for a few seconds until all the feed will be fetched for the
// first time.
System.out.println( m.getSnippetsBySentence("what do you know about
local politics ?"));
```

Future work:

This project covers all the basic goals, which is to provide a uniform, efficient way to aggregate information from web feeds.

But tagging a sentence is the most difficult task and more effort could be made at that area –For instance currently words with the same root are considered different, so it would've been helpful to reduce all words to their root

For example: buy, buying, bought → buy

Another useful transformation is stripping common words like “this”, “the”, “is” thus reducing the keyword space.

Another approach is to modify the algorithm that is currently used in the KeywordStore, to use weights for different keywords – such that more relevant keywords would possess a higher weight than others, for instance Under **www.ynet.co.il** we would use { (ynet, 10), (news, 10), (sport, 5), (local, 4), .. }

Then if $U \cap F_{ynet} = |\{news, sport\}| = 15$

And $U \cup F_{ynet} = |\{news, sport, ynet\}| = 25$

Then the total rank should be $\frac{15}{25}$

Totally different approach can be using **Wikipedia** articles as a training set for Artificial Neural Networks by partitioning the article, for instance "sport", to sentences and to train the ANN to output "sport" for these sentences.

Bibliography:

- [1] [http://en.wikipedia.org/wiki/RSS_\(file_format\)](http://en.wikipedia.org/wiki/RSS_(file_format)) – about RSS
- [2] <https://rome.dev.java.net/> - Package that read & Parse RSS
- [3] www.jdom.org – XML Parser.
- [4] <http://www.cs.bgu.ac.il/~orlovm/teaching/saya/> - Saya resources