
John Holst, Fotolia

Semantic web tools of the Simile project

TOMORROW'S TOOLBOX

The Simile project jump starts the semantic web with a collection of tools for extending semantic information to existing websites.

BY OLIVER FROMMEL

A simple Google search shows how dumb the web really is. If you try to search for a solution to a Linux problem, you are very likely to find many other users with a similar problem, but you might not actually find a solution. The problem is that Google simply evaluates the occurrence of whatever keywords the user enters to describe the problem.

A typical Internet search engine does not analyze the document structure or the structure of a conversation, and this means you could turn up articles that have very little to do with Linux when you search for "Linux *keyword*."

Google only performs rudimentary language analysis, correcting typos or incorrectly typed phrases on the basis of statistical methods known as N-grams. However, Google is incapable of interpreting the meaning of the search keys. For example, Google does not distinguish between *bank* as a financial institution or *bank* as a place to sit beside a river.

Some people wonder whether computers will ever be truly capable of sorting through the subtleties of human language; however, the Simile web-developer community is unwilling to accept this dire assessment.

The Simile project develops tools for the semantic web. Semantic web [1] visionaries hope that tools such as the Simile toolkit will one day make machine processing more intelligent, and thus more useful.

Problems

Web developers have an array of sophisticated techniques for serving up dynamic web content that is treated as data by client-side applications. This paradigm, however, is both too labor-intensive and too limited for the needs of the semantic web. One problem is that this kind of functionality must be programmed carefully into the fabric of the website itself, which might be fine if you are building a new site, but if you want to apply the benefits of automation to existing static websites, your only real option is to start over. The other problem is that a conventional server application must be written with very detailed knowledge of what the client will do with the data.

The semantic web addresses these problems by adopting the premise that, if web information has *meaning* in a way that is closer to how real language has meaning, the information becomes readily adaptable for interpretation and automation without extensive coordination between server and client.

For the semantic web to achieve a major breakthrough, new technologies must include the necessary meta-information and also provide a simple means for attaching this semantic information to existing web pages. The Simile project, which is sponsored by the Massachusetts Institute of Technology (MIT),

develops software tools that one day could help to smooth the transition to a semantic web. Simile stands for *Semantic Interoperability of Metadata In unLike Environments*. The tools of the Simile toolkit (Table 1) are designed to expose, extract, associate, and manipulate semantic information.

Thus far, semantic web techniques have failed to make a breakthrough, partly because of the W3 consortium's slow standardization process, but also as a reflection of the huge numbers of different standards and technologies used in practical applications. The most popular format for storing semantic web data is probably the Resource Description Framework (RDF). RDF is already used in some popular RSS news feeds [2], and it is the basis for many of the tools of the Simile project.

RDF Format

RDF format is designed to structure web data in a form that is independent of format. The purpose of RDF is to arrange the data so that it can be interpreted meaningfully rather than just viewed as letters and words.

RDF structures resources into three-part expressions known as *triples*. A triple mirrors the classic form of a sentence, consisting of three parts:

- subject
- predicate
- object

Wikipedia's introduction to RDF [3] offers as an example the sentence, "The sky has the color blue." This sentence could be expressed as an RDF triple with "The sky" as the subject, "blue" as the object, and "has the color" as a predicate defining a relationship between the subject and object.

Figure 1: Without a screen scraper, Piggy Bank only discovers a few snippets of information on HTML pages.

In any language, the possibilities for structuring sentences in RDF format are, of course, infinite. The purpose of RDF technology is not to build a single program that navigates through all natural language alone but to provide a simple

Figure 2: The Firefox Solvent extension helps users write their own screen scrapers for Piggy Bank.

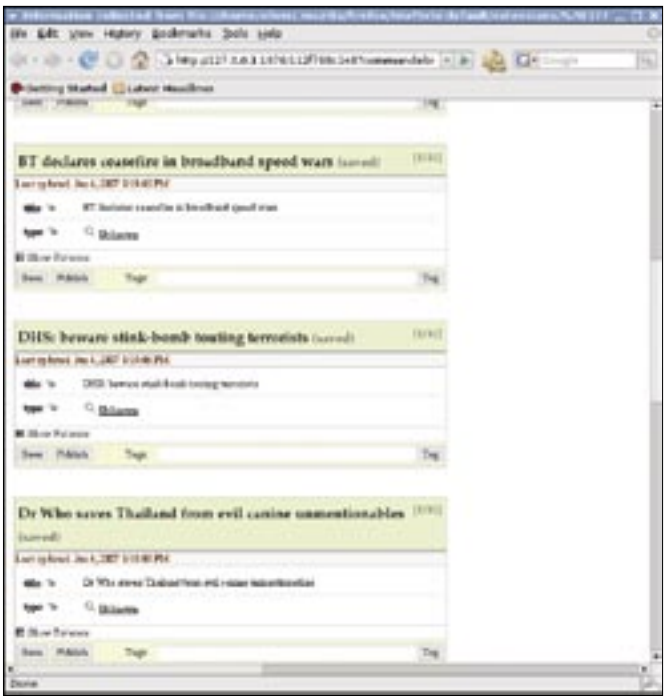


Figure 3: Piggy Bank can display information extracted from a website by Solvent and enhanced with semantic markups.

means for the owner (or viewer) of a web page to express clues about the meaning of the information on the page. Human intervention is still necessary, but RDF reduces that intervention to a simple, concise form that minimizes the disruption to existing web pages.

The tools of the Simile Project work with RDF information derived from

- an explicit definition of RDF relationships created by the owner of a web page and referenced as a link in the header of the HTML page
- a custom *screen scraper* tool specifically designed to render the content of a web page page in RDF format.

The option of expressing the web information through an explicit definition lets the website owner make that information available for integration with custom automation tools. For example, the owner of a hotel chain with a web page that lists names and addresses of hotels could have the chain's webmaster structure the address information in an RDF file that would let travel agencies easily build tools to plot the hotel locations on a Google map.

Importantly, RDF lets you offer this possibility of automation without radically restructuring the existing website. The web page can remain a simple list of hotel names and addresses. The only change necessary is a reference to the RDF definition in the < *head* > section of the HTML file.

Table 1: Simile Projects

Tool	Task
Piggy Bank	Firefox extension that stores the results of Xpath queries locally
Solvent	Auxiliary extension to Piggy Bank for screen scraping
Semantic Bank	Server that stores Piggy Bank information for multiple users centrally
Welkin	RDF graph visualization
Longwell	Faceted browser for RDF data
Gadget	Inspector for large XML data sets
Referee	Metadata extractor and referrer crawler for logfiles
Exhibit	Automatically generates HTML pages from a database
Babel	Format translator; for example, from JSON to Exhibit format
Fresnel	Vocabulary for RDF rendering
HTTPTracer	Sniffer for HTTP traffic
Timeline	DHTML widget for ordering data chronologically
RDFizer	Collection of RDF conversion tools

The second, *screen scraper* option requires even less intervention from the website owner. In fact, the owner doesn't even have to know that the visitor is automating the information. You can write your own screen scraper that processes the text on a web page and renders it into an RDF structure. It is also possible for the web page owner to define a screen scraper to use with the page. Listing 1 shows the example code provided by the Simile project for referencing an RDF or screen scraper file in an HTML header.

At the core of the Simile toolkit is a Firefox extension called Piggy Bank. Piggy Bank lets you view, manage, and combine RDF information from various sources, and you can use Piggy Bank as an interface for managing custom screen scrapers. Another tool in the toolkit, Solvent, helps you write your own screen scrapers for extracting RDF data from websites.

Semantic web tools such as Piggy Bank and Solvent are still largely experimental. This article offers a first glimpse at some of Simile's tools for the semantic web. You'll find more information on the Simile toolkit at the project website [4].

Exploring Piggy Bank

From the Piggy Bank homepage [5], simply click on the link to a Firefox

extension to install it.. The installer does assume that you have the Java plugin.

In our lab, Piggy Bank would not work with plugins from JDK 1.4.2, or with the brand new JDK 1.6.0, but only with JDK 1.5.0. To enable the Java plugin installed with the JDK, you need a create a symbolic link:

```
cd $HOME/.mozilla/plugins
ln -s /usr/java/jdk1.5.0_11/jre➥
/plugin/i386/ns7➥
/libjavaplugin_oji.so
```

If this does not work, Firefox users can't expect much in the way of help. The browser will just not launch the Simile plugin, although you could try launching the browser in developer mode by entering *firefox -P development*.

If all of this works, a new button should appear in the bottom border of the window, or a piggy icon for starting the Piggy Bank browser will appear next to the location bar.

The first time you launch Piggy Bank, the local database will be empty. The new menu item *Tools | Piggy Bank | Collect and Browse* gives you the ability to fill your Piggy Bank.

The Piggy Bank extension will then search the current page for RDF resources and store them locally. If the website does not have anything in the line of semantic information, Piggy Bank will run any available screen scrapers to grab the data off the surface of the screen. Screen scrapers have to be individually written for each website because they depend a great deal on the document structure.

Scraping Information

The Piggy Bank extension includes three ready-to-run scrapers, with several more on the website. This said, it is mainly up to users to write their own scrapers; otherwise, Piggy Bank just displays the few snippets of information that it automatically detects on standard web pages, such as URLs and titles (Figure 1).

Listing 1: Exposing Scrapers and RDF Files

```
01 <html>
02   <head>
03     ...
04     <link rel="alternate"
   type="application/n3"
   title="Screen scrapers'
   information"
05       href="http://people.
   csail.mit.edu/dfhuynh/
   research/downloads/
   screen-scrapers/
   screen-scrapers.n3">
06
07     <link rel="alternate"
   type="application/rdf+xml"
   title="My contact information"
08       href="http://people.
   csail.mit.edu/dfhuynh/foaf.
   rdf">
09     ...
10   </head>
11   ...
12 </html>
```

As an aid to writing your own screen scrapers, Simile includes another Firefox extension, Solvent, which simplifies this process. After completing the installation, you can launch Solvent by clicking the icon that looks like a spray can at the bottom right of your screen. When you do so, Firefox pops up two additional windows in the area occupied by the current window (Figure 2).

To scrape information from a website, users first click the *Capture* button; the cursor changes to a hand shape. Users can then click the HTML elements they are interested in; Solvent will highlight the elements in yellow on mouse over. When you click, the extension automatically selects other elements on the page that match the selected Xpath expression, which you can see in the top line of the right-hand window. At the same time, Solvent displays the selected elements in the bottom half of the window. The blue arrow to the right of the Xpath expression selects elements from the layer immediately superordinate.

After identifying desired page elements, the user has to add semantic information, which the HTML code will obviously not have. To do so, first expand one of the selected items, and then press the *Name* button. The button takes you to a list with a couple of predefined semantic markups, but you can define your own markups, too. In this case, you might select the *URI* element for the URL and *Title* for the title.

Some tags can be assigned via the menu; their names should include a URL to comply with existing standards. Simile draws most of the predefined tags from the Dublin Core [6], which implements a taxonomy that, for example, the OpenOffice.org ODF format uses for its metadata. The Dublin Core has a URL prefix of *http://purl.org/dc/elements/1.1/*.

Typically, you can't expect automatically generated scrapers to return results without some manual attention. At the very least, you will need to modify the element names in the code. To extract information from pages with a complex structure you might need more than the Xpath expressions. In this case, you might need to process HTML with some JavaScript of your own.

Generating Scraper Code

After setting up the page elements and metadata, you can press the *Generate* button to create your scraper code, which Solvent will display in the left-hand window. Another click on *Run* runs the scraper against the current page and displays the results in RDF format on the right. The *Show Results in Piggy Bank* button displays the results in the Piggy Bank browser (Figure 3), and you can then add tags to the information and store your data locally.

As an alternative to a local repository, users can store semantic information on the Semantic Bank server, which is also part of the Simile project. Semantic Bank gives a group of users the ability to collaborate. Other Simile tools support command-line-based screen scraping, converting the resulting RDF data to other formats and displaying the data in chronological or geographic order.

Some Coding Required

Not even the tools the of the Simile project are capable of automatically conjuring a semantic web up from the offerings of the legacy web. The existing website structures are just too different for this to happen. Even screen scrapers require some knowledge of Xpath and JavaScript on the part of the user; just clicking and pointing on complex sites is not going to get you very far. Thanks to the variety of tools, users can experiment with semantic technologies without having to manually code RDF. Soon you can expect screen scrapers and other tools of the semantic web to start appearing, at least for extremely popular sites. ■

Microformats

The confines of the HTML page markup language currently restrict the web to representing a textual layer. Whether this is a list of people or a list of cars, the tags the HTML markup uses are the same: *li*, *div*, or *td*.

Recently, in the context of Web 2.0, microformats have been much hyped. Microformats attempt to add some semantics to the HTML web by adding meta-information to the class attributes of HTML elements, for example:

```
<div class="Street">
Suskind Street</div>
```

As you can imagine, this works better with minimalist markup than with pages that overburden their content with tags. Because of this, the main applications for microformats today are calendar entries and electronic business cards.

With respect to the underlying ontology or taxonomy, so-called *folksonomies* help to standardize individual sites. Users can attach multiple keywords (which are also referred to as tags) to their online data, such as photos on flickr.com. Server-based storage and Ajax techniques mean that users can add tags and use existing tags.

Microformats and folksonomies are referred to by some observers as the bottom-up semantic web because they provide an ad hoc approach to solving semantic web problems. Even some purists do not insist on ontologies being constructed by standardization committees but see the potential of folksonomies. Thus, Web 2.0 can be viewed as a kind of transitional phase, in which more and more semantic technologies will be successively integrated.

INFO

[1] The semantic web: "A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities" by Tim Berners-Lee, James Hendler, and Ora Lassila, *Scientific American.com*, May 2001, *http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&catID=2*

[2] Resource Description Format: *http://www.w3.org/RDF*

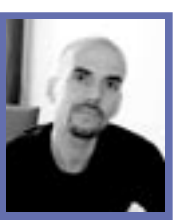
[3] Wikipedia on RDF: *http://en.wikipedia.org/wiki/Resource_Description_Framework*

[4] Simile: *http://simile.mit.edu*

[5] Piggy Bank: *http://simile.mit.edu/wiki/Piggy_Bank*

[6] Dublin Core: *http://dublincore.org/*

THE AUTHOR



For several years Oliver was a sysop and programmer at Ars Electronica Center in Linz/Austria. After finishing his studies in Philosophy, Linguistics and Computer Science he became an editor for the Bavarian Broadcasting Corporation. Today he is head of the Editorial Competence Center for Software and Programming at Linux New Media AG.