Semantic Annotation Tool For EIS Publications

...using crowdsourcing, ontology and SPARQL

Team : A Q M Saiful Islam
Seyidov Mirsattar

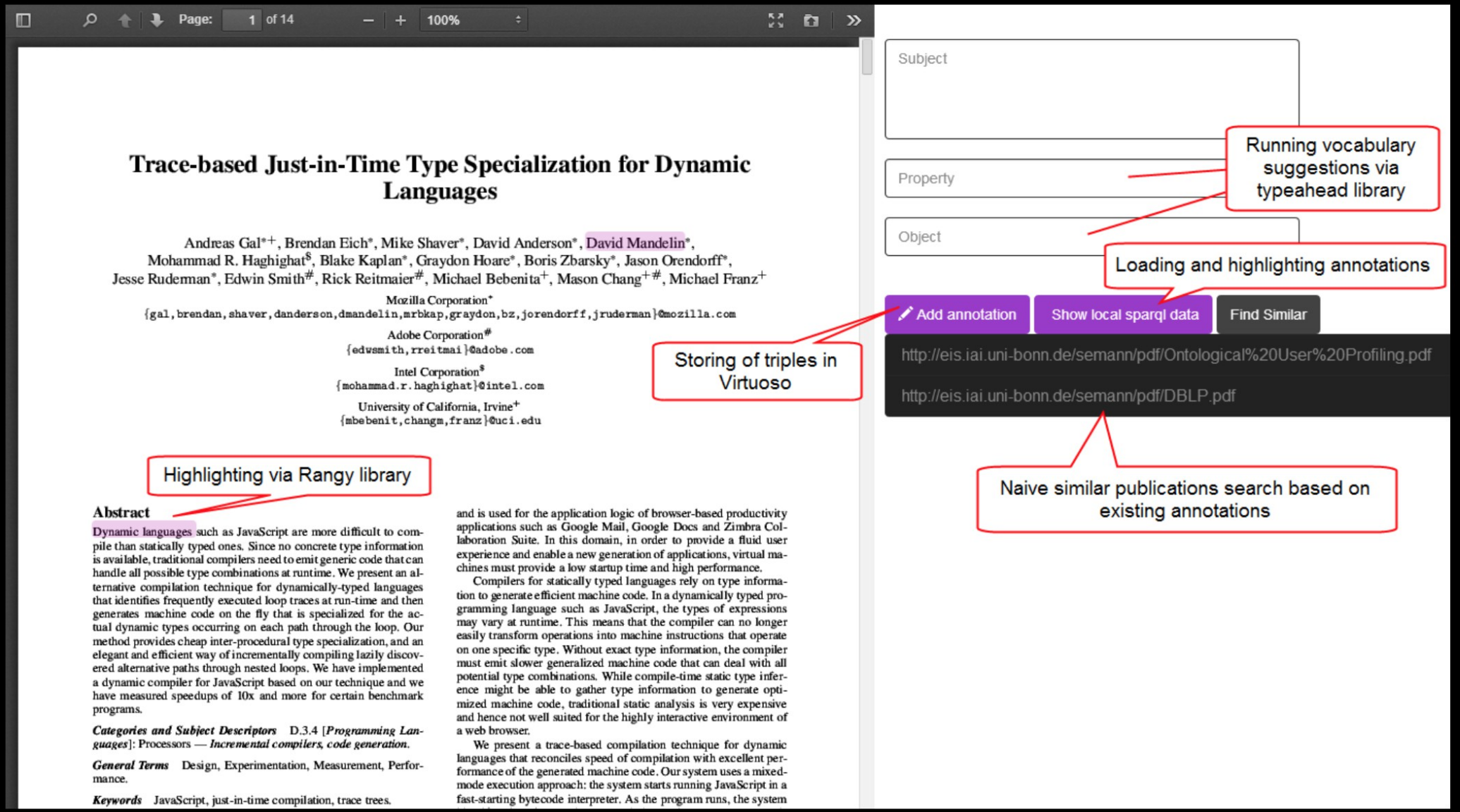
What this tool about?

This tool is about annotating EIS related documents, which normally contain lots of **tabular** information.

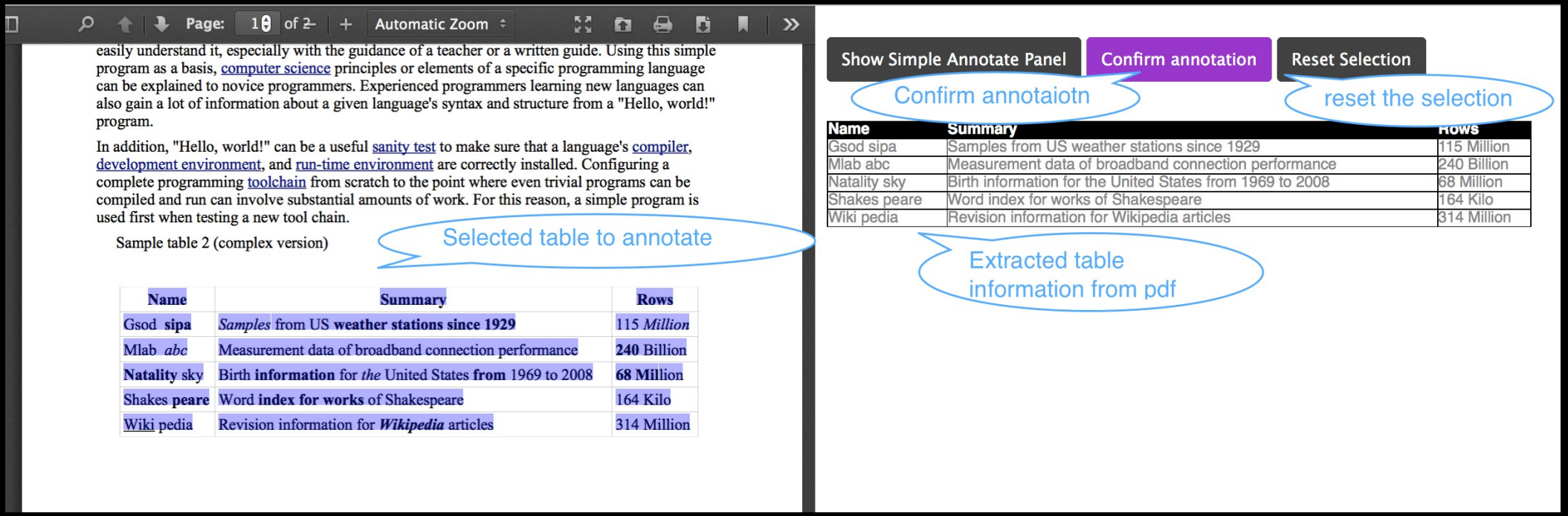
- Annotate Tables
- Transform the table information into **Data cube** vocabulary
- Store data cube into to **virtuoso** server using SPARQL interface.

Poster

Start point


Page: 1 of 14
100%
Trace-based Just-in-Time Type Specialization for Dynamic Languages
Andreas Gal*+, Brendan Eich*, Mike Shaver*, David Anderson*, David Mandelin*,
Mohammad R. Haghighat$, Blake Kaplan*, Graydon Hoare*, Boris Zbarsky*, Jason Orendorff*,
Jesse Ruderman*, Edwin Smith#, Rick Reitmaier#, Michael Bebenita+, Mason Chang+#, Michael Franz+
Mozilla Corporation*
{gal,brendan,shaver,danderson,dmandelin,mrbkap,graydon,bz,jorendorff,jruderman}@mozilla.com
Adobe Corporation#
{edwsmith,rreitmai}@adobe.com
Intel Corporation$
{mohammad.r.haghighat}@intel.com
University of California, Irvine+
{mbebenit,changm,franz}@uci.edu
Highlighting via Rangy library
Abstract
Dynamic languages such as JavaScript are more difficult to compile than statically typed ones. Since no concrete type information is available, traditional compilers need to emit generic code that can handle all possible type combinations at runtime. We present an alternative compilation technique for dynamically-typed languages that identifies frequently executed loop traces at run-time and then generates machine code on the fly that is specialized for the actual dynamic types occurring on each path through the loop. Our method provides cheap inter-procedural type specialization, and an elegant and efficient way of incrementally compiling lazily discovered alternative paths through nested loops. We have implemented a dynamic compiler for JavaScript based on our technique and we have measured speedups of 10x and more for certain benchmark programs.
Categories and Subject Descriptors D.3.4 [Programming Languages]: Processors — Incremental compilers, code generation.
General Terms Design, Experimentation, Measurement, Performance.
Keywords JavaScript, just-in-time compilation, trace trees.
and is used for the application logic of browser-based productivity applications such as Google Mail, Google Docs and Zimbra Collaboration Suite. In this domain, in order to provide a fluid user experience and enable a new generation of applications, virtual machines must provide a low startup time and high performance.
Compilers for statically typed languages rely on type information to generate efficient machine code. In a dynamically typed programming language such as JavaScript, the types of expressions may vary at runtime. This means that the compiler can no longer easily transform operations into machine instructions that operate on one specific type. Without exact type information, the compiler must emit slower generalized machine code that can deal with all potential type combinations. While compile-time static type inference might be able to gather type information to generate optimized machine code, traditional static analysis is very expensive and hence not well suited for the highly interactive environment of a web browser.
We present a trace-based compilation technique for dynamic languages that reconciles speed of compilation with excellent performance of the generated machine code. Our system uses a mixed-mode execution approach: the system starts running JavaScript in a fast-starting bytecode interpreter. As the program runs, the system
Subject
Property
Object
Running vocabulary suggestions via typeahead library
Loading and highlighting annotations
Add annotation
Show local sparql data
Find Similar
Storing of triples in Virtuoso
http://eis.iai.uni-bonn.de/semann/pdf/Ontological%20User%20Profiling.pdf
http://eis.iai.uni-bonn.de/semann/pdf/DBLP.pdf
Naive similar publications search based on existing annotations

Result


Page: 1 of 2
Automatic Zoom
easily understand it, especially with the guidance of a teacher or a written guide. Using this simple program as a basis, computer science principles or elements of a specific programming language can be explained to novice programmers. Experienced programmers learning new languages can also gain a lot of information about a given language's syntax and structure from a "Hello, world!" program.
In addition, "Hello, world!" can be a useful sanity test to make sure that a language's compiler, development environment, and run-time environment are correctly installed. Configuring a complete programming toolchain from scratch to the point where even trivial programs can be compiled and run can involve substantial amounts of work. For this reason, a simple program is used first when testing a new tool chain.
Sample table 2 (complex version)
Selected table to annotate
Name
Summary
Rows
Gsod sipa
Samples from US weather stations since 1929
115 Million
Mlab abc
Measurement data of broadband connection performance
240 Billion
Natality sky
Birth information for the United States from 1969 to 2008
68 Million
Shakes peare
Word index for works of Shakespeare
164 Kilo
Wiki pedia
Revision information for Wikipedia articles
314 Million
Show Simple Annotate Panel
Confirm annotation
Reset Selection
Confirm annotaiotn
reset the selection
Name
Summary
Rows
Gsod sipa
Samples from US weather stations since 1929
115 Million
Mlab abc
Measurement data of broadband connection performance
240 Billion
Natality sky
Birth information for the United States from 1969 to 2008
68 Million
Shakes peare
Word index for works of Shakespeare
164 Kilo
Wiki pedia
Revision information for Wikipedia articles
314 Million
Extracted table information from pdf

Unit Test Result

EIS Table Annotation :: Unit Test

☐ Hide passed tests ☐ Check for Globals ☐ No try-catch

Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:32.0) Gecko/20100101 Firefox/32.0

Tests completed in 3722 milliseconds.
16 assertions of 16 passed, 0 failed.

1. **Sparql Unit Test for our EIS Lab:** **Sparql Server Url Validity** (0, 1, 1) Rerun
2. **Sparql Unit Test for our EIS Lab:** **Camel Case test** (0, 2, 2) Rerun
3. **User table selection testing:** **Testing user selected items validation** (0, 5, 5) Rerun
    1. Table range successfully calculated
    2. No table suggestion applied here
    3. Same number of element is found after the validation
    4. First item match of the selected elements after validation
    5. Last item match of the selected elements after validation
4. **User table selection testing:** **Testing user selected parse data** (0, 3, 3) Rerun
5. **DataCube testing:** **Testing Dimension and property insertion** (0, 1, 1) Rerun
6. **DataCube testing:** **Testing Data set insertion** (0, 1, 1) Rerun
7. **DataCube testing:** **Testing observation insertion** (0, 1, 1) Rerun
8. **DataCube testing:** **Testing column header insertion** (0, 1, 1) Rerun
    1. Column header successfully inserted
9. **DataCube testing:** **Cleaning up the test database** (0, 1, 1) Rerun

Demo

Issues & limitations (Demo)

Implemented Features

- Extend the implementation from **text to table** annotation
- Introduces **suggest** selection mechanism
- **Accommodate** the GUI for new implementation
- Add a application **set up script** when the application run first time
- Extend our **wiki** doc to support **Mac** and **Ubuntu** installation.

Lessons learn

- PDF.js is not developer friendly, it has **limited** access over PDF documents.
- Get support from Virtuoso to enable Cross-origin resource sharing (**CORS**).
- JavaScript test unit (**QUnit**) is different then normal programming language unit test.
- Making the **Poster** was an interesting experience.
- Handling a **team** in this lab is very different/difficult then in real world professional environment.

Future work

- I will continue working on it as part of my thesis.
- We addressed some issues here, will continue working on them.
- Put it on the web for public use.
- Will collect the user feedbacks and consider constructive suggestions.

Thanks for your time