COMS W4115

Programming Languages and Translators

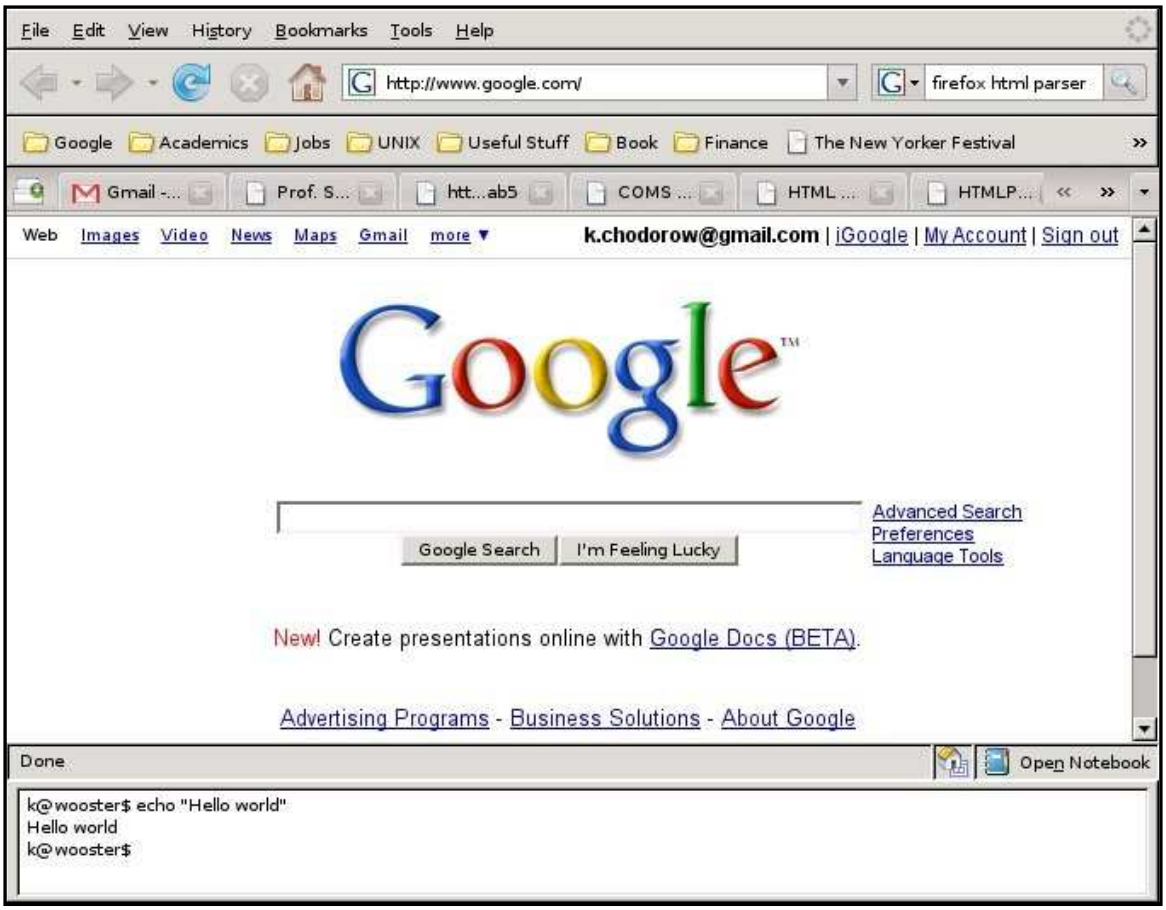
PROJECT PROPOSAL

MASC

Project Team:
Christos Angelopoulos
Kristina Chodorow
Michael Masullo
Vaibhav Saharan

Introduction

There is a cornucopia of information on the internet and one may often find it difficult to tap into this web of possibilities in an efficient manner. MASC comes to the rescue. MASC is a web-oriented processing language that allows users to manipulate the information available on the web in an organized and intuitive manner. MASC introduces a new way of writing scripts that collect the data on the web and perform various operations on this data. It is superior to merely perusing the webpage source code and trying to make sense of the intricate structure of the page. It offers more flexibility than other languages because the user can write his/her own scripts which integrate naturally with the browser, by running directly from a Firefox extension.



A screenshot of the MASC interface:

Motivation and Purpose

MASC is a language for creating fast information extraction tools. The MASC shell is embedded in the web-browser and allows users to write code snippets in MASC while browsing the web. It offers the user a number of interesting features to process the data on the web-page, like crawling the web-page, extracting various types of information, saving different types of media etc. Furthermore, it also allows the user to author general purpose stand-alone programs. It is more stable, concise, and occupies less memory than existing GUI tools.

MASC helps to work with the information available on the web in a smarter way. Consider some examples to this end. Search results often return web-pages riddled with advertisements and irrelevant data. MASC can help derive the relevant information and meaningful links from such results. Alternatively, it can help crawl a web page to find links that go to relevant results and even save such results for future reference. It can help save the various media elements like images, sound snippets, videos etc., embedded in a web page. Such uses of MASC unbound.

Key Features

Basic Data Types:

MASC uses three data types:

string: This data type represents a sequence of any number of characters. It is represented by the keyword *string*.

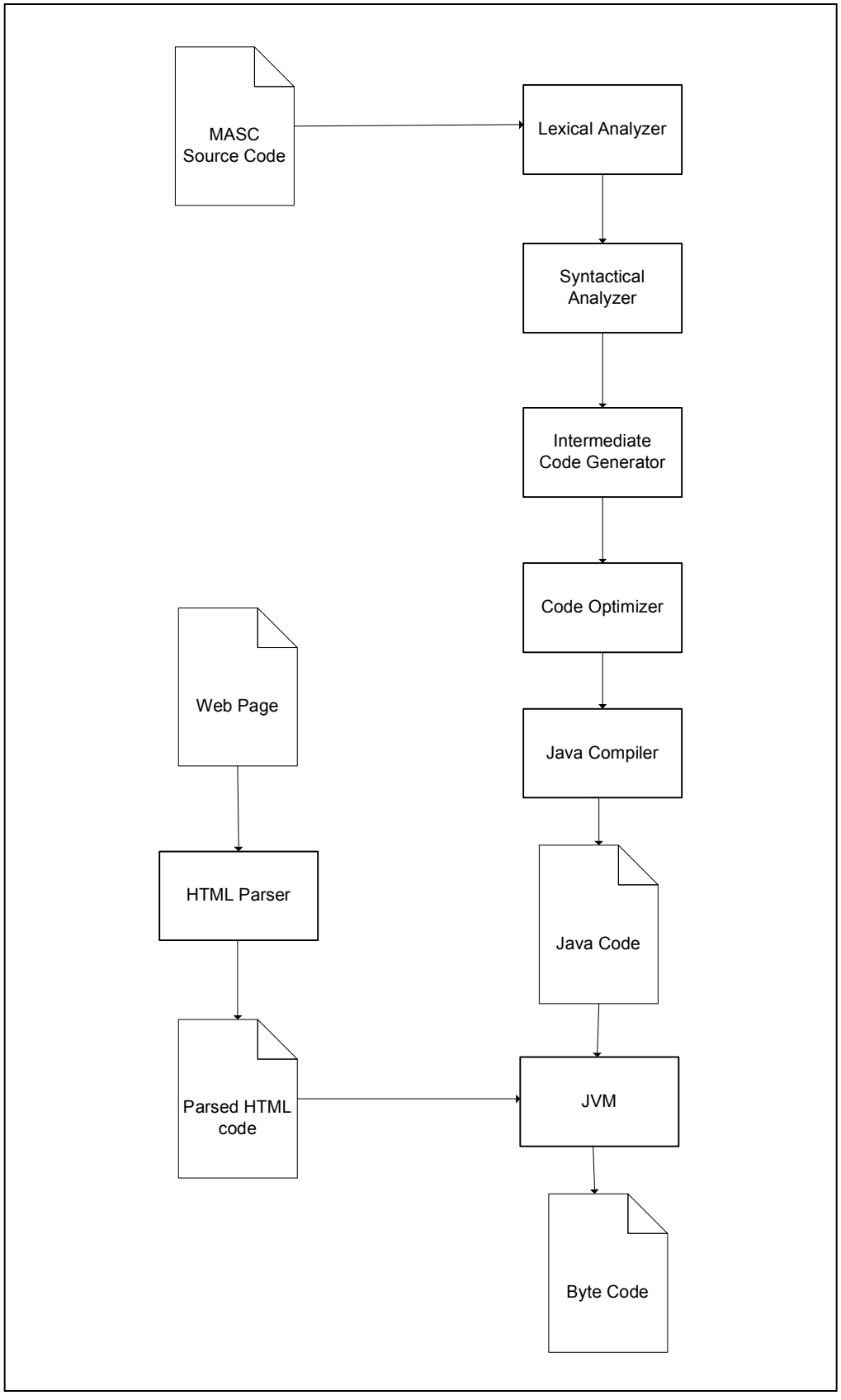
int: This data type represents all 32-bit whole numbers. It is represented by the keyword *int.*

float: This data type represents floating-point numbers. It is represented by the keyword *float.*

Keywords:

The following words will be reserved by the language as keywords to represent specific functionalities:

for if int float string else return void



A schematic diagram of the MASC compiler

Arithmetic and Comparison Operators:

Arithmetic:
‘+’ Addition operator, overloads as string concatenation operator
‘-’ Subtraction operator
‘*’ Multiplication Operator
‘/’ Division Operator
‘&’ Logical AND operator
‘|’ Logical OR operator
‘!’ Logical NOT operator

Comparison:
‘<’ Less than operator
‘>’ Greater than operator
‘==’ Equality comparison operator
‘<=’ Less than or Equal to operator
‘>=’ Greater than or Equal to operator
‘!=’ Not equal to operator

Misc.:
‘=’ Assignment Operator
‘(:’ ‘:)’ Comment operator
‘;’ End-of-statement delimiter

Control Structures:

MASC has the following built-in control structures:

If statement:

```
if(expression) {
        block
}
else if (expression) {
        block
}
else {
        block
}
```

While statement:

```
while(expression) {
        block
}
```

Functions:

MASC allows programmers to write programs in a more modular manner by writing snippets of code as separate functions. Functions in MASC can be described as follows:

return-type function-name (*argument1*, *argument2*.,.) {
    block
}

Sample Programs in MASC:

The following program looks for the word “Addresses” in the website's header. If it is found, the program finds email addresses using a regular expression and prints them to standard output. If it does not find “Addresses”, it finds all links with inner text containing the word “addresses”.

This type of program could be very useful when dealing with a large HTML document, one with invisible elements, or one where the search terms are more complicated than could be dealt with by Firefox's Find.

```
if (header.contains(“Addresses”)) {
     (: match(regex) is a built-in library function
: returning one match at a time :)
     while(x = match(“[0-9A-Za-z]+@[A-Za-z]+.com”)) {
          print(x);
     }
} else {
     (: matchtag(tag_type[, regex]) is another library
: function returning all matches as an array :)
     @list = matchtag(“a href”, “addresses”));
     print(list);
}
```

This small program will save images from the current page named arrowz.gif (where *z* starts at 1 and increases until there is an error attempting to download an image) to the user's current directory. This will probably be a major use for this language, as many web galleries number images in the order they were uploaded (img1.gif, img2.gif, etc.).

```
x = 0;
error_code = 0;
while(error_code == 0) {
     x = x+1;
     error_code = save(“arrow”+x+”.gif”, “./”);
}
```

HTML Parser

We will be using the a Java HTML parser called (creatively) HTML Parser. It is extremely powerful and versatile. The following program demonstrates it taking an HTML page (the Columbia CS directory site) and extracting all email addresses:

```
import org.htmlparser.Parser;
import org.htmlparser.Node;
import org.htmlparser.NodeFilter;
import org.htmlparser.util.*;
import org.htmlparser.util.ParserException;
import org.htmlparser.parserapplications.*;
import org.htmlparser.tags.*;
import org.htmlparser.filters.*;

public class ParserExample {

    public static void main(String[] args) {

     String URL = “dir.html”

     NodeList list = new NodeList();

     try {
         Parser parser = new Parser(URL);

         //Extracts all links which point to columbia.edu or
cs.columbia.edu email addresses
         NodeFilter filter = new LinkRegexFilter("[0-9A-Za-
z]+@[cs.]?columbia.edu");

         for (NodeIterator i = parser.elements();
i.hasMoreNodes(); ) {
          i.nextNode().collectInto(list, filter);
         }
     }
     catch (ParserException e) {
         System.out.println("Parser exception error.");
         System.out.println(e.getStackTrace());
     }

     // Prints inner html of link
     for(int i=0; i<list.size(); i++) {
         System.out.println(i+":
"+(list.elementAt(i)).toPlainTextString());
     }

    }
}
```

The NodeFilter class can be used to create incredibly complex requests, such as finding a

type of tag if it has a child tag of some type but not if it has a parent of another type and exclusive or that with it having a certain inner HTML.

More information about the parser can be found at its website: http://htmlparser.sourceforge.net/.