

Browser based Web Accessibility Measurement Component

IKT407

Autumn 2005



HØGSKOLEN I AGDER

Authors: Katja Soukhikh
Dag Tommy Steen

Supervisors:
Morten Goodwin Olsen
Nils Ulltveit-Moe

Version: 1.0

Pages: 18, including this. - 1 -

Status: Final

Modified date: 2005-20-11

Keywords: Python, Linux, wam, xpcocom,
pyxpcocom , open source

Abstract:

In the project we have looked on the possibilities for using Mozilla for assessing accessibility to web pages. As the web today grows to be a bigger part of our daily life, so is also the need to access information found on the web in one way or another. We have looked upon the things that are needed to make about any program take use of the web, and those structures already defined there as standards, by using existing technologies.

License:

This work is licensed under the Creative Commons Attribution-ShareAlike License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/2.5/> or send a letter to Creative Commons, 534 Howard Street, 5th Floor, San Francisco, California, 94105, USA

Table of Contents

1. Introduction.....	3
2. Background.....	4
2.1 General background.....	4
2.2 Harvestman.....	4
2.3.1 Mozilla.....	4
2.3.2 Accessibility and the DOM tree in Mozilla.....	5
2.4.1 XPCOM.....	6
2.4.2 PyXPCOM.....	6
2.5 Python.....	7
2.6.1 Other possible technologies.....	7
2.6.2 Accessibility and the DOM tree in IE.....	7
A note on the difference between COM and .Net.....	8
2.7 Comparision of IE and Mozilla DOM support.....	8
3. Problem Description.....	9
4. Requirements specification for project.....	10
5.1 Discussion.....	11
5.2 Mozilla.....	13
5.3 Internet Explorer.....	13
5.4 And the choice is.....	14
5.5 Project outcomes.....	14
6. Conclusion.....	15
7. Appendix.....	16
7.1 Reference.....	16
7.2 Information gathered from the web.....	16
7.3 Glossary.....	17

1. Introduction

The main task of this project is to look at the possibility of using an existing web browser and combine it with another application with help of some programming language. This will make the web browser to serve as an interface for the program against the internet. By using open source and free software, this can be done in a way that allows the browser to call preinstalled components by itself, and are going to be easy to use.

We will discuss the different technologies that exists today, and why some of them are more suitable for this kind of fusion than others. We will also take a peek at some of the difficulties about this, and will take a brief view on how is it possible to solve some of the problems.

Although this have been already done before in some degree, we will make a more theoretical approach, and look on the benefits that are gained from this kind of integrating, and discuss how it could be done.

One example that goes thru this rapport is the example of implementing a webcrawler named Harvestman into Mozilla, and steps how to do this. It seems to be a good example to show the reader why we have chosen to look at this case. It shows the benefits of being able to use a browser as an interface towards the web; instead of for example using standard libraries or built-in classes straight form any programming language. We will also take a brief view on the technologies around the different platforms and problems with their use.

Since this is a quite large subject, we have decided to set some limits. We will not go so deep into the details that we dig into “code level”, but we are neither going to “swim on the surface”. We have also included only those subtopics here as we consider being the most important and rather made links available to the readers for further study on each subtopic.

The internet today contains a lot of information on these subjects, and sometimes it is hard to know what to trust and what not to. We have been critical to our sources, and we hope we managed to filter the information right.

We also thought it was important to remember that there are many different technologies and solutions available. For this intention, we have kept our minds open, and tried to look at the different solutions with as much unbiased approach as possible.

Enjoy the reading!

Grimstad 2005-11-21

Katja Soukhih & Dag Tommy Steen

2. Background

2.1 General background

As the need for information mining on the web grows, so do also the sources change. Today many pages containing JavaScript's flash elements and/or spawned windows for example. One of the most used web crawler today, Harvestman, does not support harvesting information from these sites. This makes some interesting websites inaccessible for Harvestman, and therefore the user will possibly miss some of the vital information.

2.2 Harvestman

HarvestMan [1] is a multithreaded off-line browser. It has many features for customizing offline browsing through URL filters, word filters, domain filters, URL priorities, and many more. It is useful to download an entire Web site or certain files from a Web site to the hard disk for offline browsing later. It supports HTTP/HTTPS and FTP protocols and can work across proxies.

Written by Anand B. Pillai, Harvestman is an open source fully customizable multithreaded web crawler, written in Python using urllib2, httplib and HTMLparser. This means that the program is capable of downloading whole web pages from the net into a local machine, so that they later can be used for offline browsing. It is a re-write of the earlier 'PIDER'(Program for Internet Data Extraction and Retrieval) program.

The name "HarvestMan" is derived from a kind of small spider found in different parts of the world called "Daddy long legs" or Opiliones. Since this program is a web-spider, the analogy was compelling to name it after some species of spiders. The process of downloading data from websites is sometimes called harvesting. Both these similarities gave rise to the name HarvestMan.

2.3.1 Mozilla

In 1998 something happened, Netscape gave away their source code [2] to the open source community to ensure that Microsoft did not achieve monopoly lock on the browser market. This put a lot of pressure on the open source community, because if they proved that they were not up to the task, the open source concept would be discredited in such a way that the commercial world would not touch it again in many

<Group 8>
Browser based Web Accessibility Measurement Component

years. Luckily, the community pulled it off, and Mozilla based browser [3] are now having about 20% of the browser market share.

Mozilla is not a web browser. It is rather a framework for building web applications using web standards that have evolved over the years. The Mozilla project is an open source development project, which means that literally anybody capable and willing to participate in the project are allowed to help further development of the Mozilla application suite (known as Seamonkey). This suite contains a lot of different programs, not only the web browser itself. E-mail client, news client and chat clients are some of the examples of programs included in this suite.

One thing about Mozilla is that the source code can be downloaded by whoever wants it, and change it accordingly to their needs. This can be seen in the number of browsers that are built upon the Mozilla framework, like the new versions of Netscape, Firefox and Camino to mention a few.

2.3.2 Accessibility and the DOM tree in Mozilla.

Every node in the DOM tree could be important to 3rd party assistive technology. The accessibility API's on each operating system have assumptions already built-in about what is the most important information, and how an accessibility service like Mozilla should use the API's programmatic interfaces to view this information to an accessibility client (the 3rd party technology)[4]. Even though platform's accessibility API has different assumptions, there are a number of common characteristics. For example, they all expose an accessible name, or text representation, of each object, and they all use an enumerated integer value from a list, to expose the role and behaviour of an object.

Given that there is a fair amount of commonality between accessibility API toolkits, it makes sense to write the code in a cross platform manner, and then deal with the platform differences on a consistent manner [5]. The shared code makes itself available to the toolkit-specific code via the generic XPCOM interfaces that return information about objects that is exposed.

The accessibility tree in Mozilla is constructed on demand. When some document requests need accessibility, the operating system must return this document accessibility. One benefit of this solution is that the accessibility data is not loaded before it is actually needed.

2.4.1 XPCOM

XPCOM[6] is a shortened for Cross Platform Component Object Model. It is a framework for writing cross platform software. In our case, it acts like an interpreter that takes makes sure Harvestman and Mozilla understand each other. It has multiple language bindings and IDL descriptions so programmers can plug their custom functionality into the framework and connect it with other components.

It is one of the main things that make the Mozilla application environment an actual framework. It is a development environment that provides the following features for the cross-platform software developer:

- Component management
- File abstraction
- Object message passing
- Memory management

This component object model makes virtually all of the functionality of Gecko available as a series of components, or reusable cross-platform libraries. These can be accessed from the web browser or scripted from any Mozilla application. Applications that want to access the various Mozilla XPCOM libraries (networking, security, DOM, etc.) use a special layer of XPCOM called XPCConnect, that reflects the library interfaces into JavaScript (or other languages).

On the developer side, XPCOM allows you to write components in C++, JavaScript, Python, or other languages for which special bindings have been created, and compile and run those components on dozens of different platforms. Including these and others where Mozilla itself is supported. The flexibility to reuse the XPCOM components from the Gecko library and develop new components that run on different platforms, facilitates rapid application development and results in an application that is more productive and easier to maintain. The networking library, for example, is a set of XPCOM components that can be accessed and used by any Mozilla application. File I/O, security, password management, and profiles are also separate XPCOM components that programmers can use in their own application development.

2.4.2 PyXPCOM

PyXPCOM [7] is a component that allows bindings between Python and XPCOM, it allows the developer to access XPCOM objects from Python code and implement XPCOM objects in Python code (instead of JavaScript or C++).

In 2004, ActiveState, perennial home of goodies for scripting languages, announced a Python interface to XPCOM as part of the Komodo project. (Komodo is a cross-platform developer's environment based on Mozilla). PyXPCOM was initially developed by Trent

Mick, and has benefited by the additional efforts of Mark Hammond, who is also the main force behind the Python binding for COM.

2.5 Python

Python [8] is a relative new programming language in the computer world. It is a so called interpreter language, which means that the computer running the program starts at line one, compile that line, and moves on to the next, much as the same way as in programming a microcontroller in assembly. This makes the Python language a ideal language for scripting, and is suitable to built into applications that need to be programmed (like for instance the configuration file in HarvestMan, you program your application to make it act according to your needs). Python is also an object oriented programming language.

It is a programming language that is built upon very logical commandoes, and is thus is easy to both read and write. It also “forces” the programmer to have certain structure in his/her code, since the influx of a code decides whatever the program is run able or not.

Python is also highly portable, and able to run on most operating systems and platforms.

2.6.1 Other possible technologies.

Today, the biggest marked share of browser belongs to Microsoft’s Internet Explorer [9]. It is a integrated component in all current releases of Microsoft Windows, and are also available as a free of charge downloadable application for Mac, Linux and Unix. As this is the biggest browser on the net today, it has been the main target for software that wants to abuse its security weaknesses.

One of the main concerns about IE is that it has introduced an array of proprietary extensions to many of the standards, including HTML, CSS and the DOM. This has resulted in a number of web pages that can only be viewed properly using Internet Explorer. This is viewed often as a use of EEE (Embrace, Extend and Extinguish), a way of using it’s dominat marked share to drive competitors out of marked by forcing them to use propetary technologies that the company owns. IE is licensed as Proprietary software [10], and is thus not Open source.

2.6.2 Accessibility and the DOM tree in IE.

Internet Explorer uses Microsoft’s COM based technology [11] to manage the accessibility tree. Since it is almost a part of the Windows operating system, it uses the accessibility framework provided in Windows.

<Group 8>
Browser based Web Accessibility Measurement Component

Microsoft does not follow W3C DOM model, but have instead created their own, that is almost identical. On noticeable change however, are that the event handler is always reference, not copied like in Mozilla. This means that it always reference to the windows, not the object itself.

A note on the difference between COM and .Net

COM and .Net are complementary technologies. The .Net technology enables bi directional, transparent integration with COM. This enables older COM programs to function with the new .Net.

Both COM and .Net can achieve the same results, but .Net allows developers simpler use, as it handle things like resource pooling, event publications and disconnected applications automatically without that the developer have to think about it at all.

2.7 Comparison of IE and Mozilla DOM support.

Table: A general view of what is supported of W3C recommendations in both IE and Mozilla. 6.0 or 1.0 means that it was fully supported from that version.

	IE	Mozilla
DOM1	6.0	1.0
DOM2	Minor	Major
DOM3	None	Minor

See [12] for reference.

3. Problem Description

Usually web crawlers that are in use today just harvest the code of web pages that is searched through. That causes a problem when the user wishes to extract information from pages containing for example flash elements or JavaScript. By combining a web browser and a crawler, this problem can be solved as the crawler uses the browser for reading the pages, and then return the extracted information that a viewer normally would see on a page.

By finding a possible combination of software and modifying some of the existing programs out there, we would have just that. A crawler that also searches for information and keyword on pages containing JavaScript's/applets and pages built upon flash rather than pure old HTML. Our main task is to get this up and running, and make a few tests to make sure that it is actually working.

The other part of this project is to make some theoretical statements about this project. Like, what can it be used for, and what kind of reasons are there that we wish to have an interpreter between Python and Mozilla?

Another interesting thing today, is that while the net continue to grow, and is getting more and more a part of our daily life it is still hard for those having some kind of handicap to use it. Also the fact of finding relevant and accurate information on the web can be hard.

4. Requirements specification for project

Clearly, any HTML parses in a crawler will need communicate with the crawling scheduler. For example, if Mozilla could be integrated with HarvestMan, PyXPCOM is a candidate that will allow this combination. In order to verify that this is done correctly, the test scripts that are included in the PyXPCOM package will be used. It is also fairly important to keep in mind that it is desirable to keep the setup on the application layer and thus allowing porting to other platforms.

The hardware demands for this project are not that demanding. We deem all of our personal computers as more than quick enough to cover our needs. And, internet connection is already in place, and is fast enough for this purpose.

All the software needed is free, and downloadable from the internet, so this is a non concerning matter.

By making this connection between a web browser and the Python programming language on an application layer, new possibilities will reveal. Programs that will make accessibility to the web easier for many people (screen magnifier, Braille drivers) will have a common entry level.

5.1 Discussion

When buying a computer today, you are always in the need of an operating system to make use of the computer itself. This is maybe especially true if you are in some need of special requirements for being able to use it, like for example Braille keyboard and/or a text reader. This often involves a relative big extra expense to the buyer. By allowing on a purely application layer

to combine a web browser with a programming language that is quite widespread some of these boundaries might be, if not broken, but at least weaken.

By combining Python with Mozilla, existing programs already written in Python will be able to harvest and use information directly from a browser, and thus can making it easier for some to experience the web, or automating processes, or allowing searches where otherwise would be hard to do. By allowing this merge, web crawling can be made possible on not just HTML pages, but also pages containing JavaScript, spawned windows, flash elements and so on. It would also make the crawler report more accurate and relevant information, rather than just the code. One example of this can be viewed in the figure above. As seen, the code itself doesn't make any sense to anyone not familiar with programming, and are therefore irrelevant to end users. The result however, makes sense and is the main purpose the script exists at all. Another example can be a tax calculator based in JavaScript. The code is not relevant, but rather the outcome of the calculations.

Working with this environment is relatively easy. It works the same way as writing a python program. But the environment has proved challenging to set up and get to run

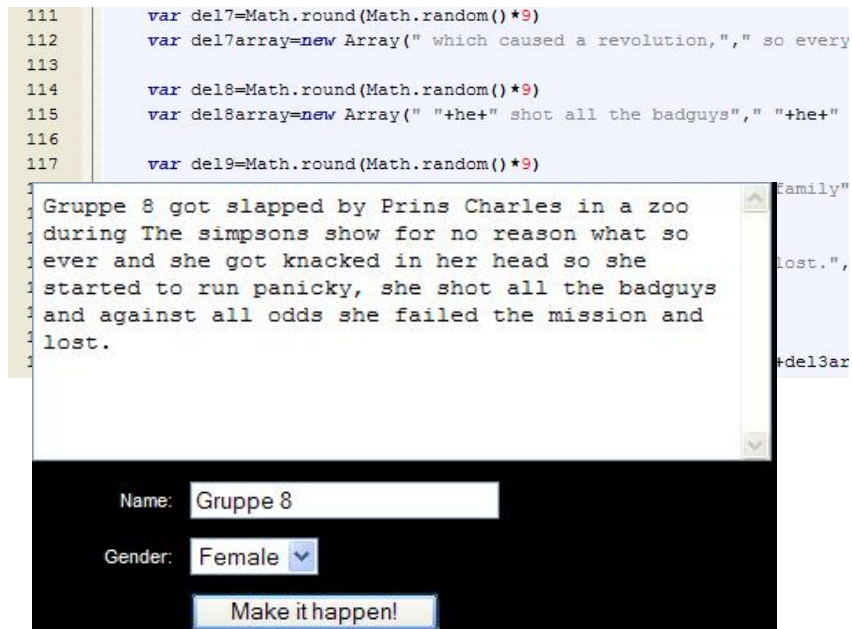


Fig: A JavaScript viewed from a conventional WebCrawler, and the same script viewed in a browser.

<Group 8>
Browser based Web Accessibility Measurement Component

properly. And this is maybe one of the biggest downsides. Getting the whole system up and running can be hard, and may cause the user to lose motivation before even starting. Also, today's distributions of Linux will be trouble free to install on most standard systems, they also can be challenging to get up and running properly on not so standard system.

Security is also an area that can cause problems. By allowing a browser to run python code, could lead to some unpleasant surprises for the users. This can in some cases be avoided as when the code opens, the user has to change protocol from what he is currently using into the built in protocol in PyXPCOM, called chf, in order to run the program itself. This gives the user a possibility to read the code before executing though. But as humans generally are lazy by nature, surely not all users will read the source before executing, and surely not if the user gets the code from a "trusted" zone.

As it is possible as previously stated to download the source for Mozilla, it is also possible to build a build of Mozilla already implemented with PyXPCOM and a program that has a function of some sort. As everybody in theory can do this, some of the builds released surely will have some functions built in that are not optimal for some users. For instance, when trying to reach Google, the program itself would redirect the user to another search engine. This is a common problem today, known as browser hijacking, caused by some kind of software that has been installed on a user's computer. While it is possible to remove these often "malicious" software from a normal computer relatively easy in most circumstances, getting rid of it when it is built in a build of a browser is another thing.

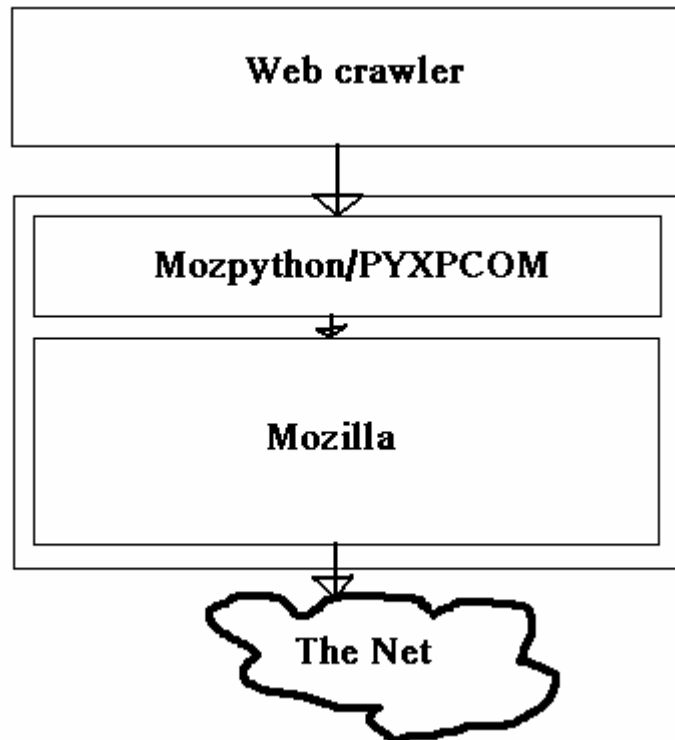


Fig: Simple overview of information flow within a Python/Mozilla builds.

5.2 Mozilla.

Mozilla has an open-source strategy, and officially allows the programming against it browser. They even opened some of the core-code for the developers so they know what to work with. The XPCOM component is to great help to developers, it seems important that it is possible to find the information on the internet about similar projects that have being done before, or general information about how to work against Mozilla and XPCOM/PyXPCOM. Mozilla is also platform independent, and able to run on any platform that can utilize a C compiler. Also, the Mozilla framework has a very good compatibility with the standardized HTML, as defined by W3C.

One of the drawbacks one of the challenges with the programming against Mozilla and Pyxpcom is that it's quite challenging to set up and get to work and run properly. The fact of making a build is quite hard, and requires a lot of background knowledge. All the programming against Mozilla core goes in Linux-style, and it can be difficult to those who are not well known with the system.

5.3 Internet Explorer

Programming towards IE is done in a Rapid application development environment, which is almost like building with Lego. The .Net environment is also a platform independent environment, and supports the Python programming language. It is easy to set up and running.

But, also here there are drawbacks. On of the biggest is the licensing fee that is required in order to use the software. Also the problem with acquire the source code and distribute in further for IE is a case that is rather hard to solve. Also IE is know as previous stated to use the EEE strategy.

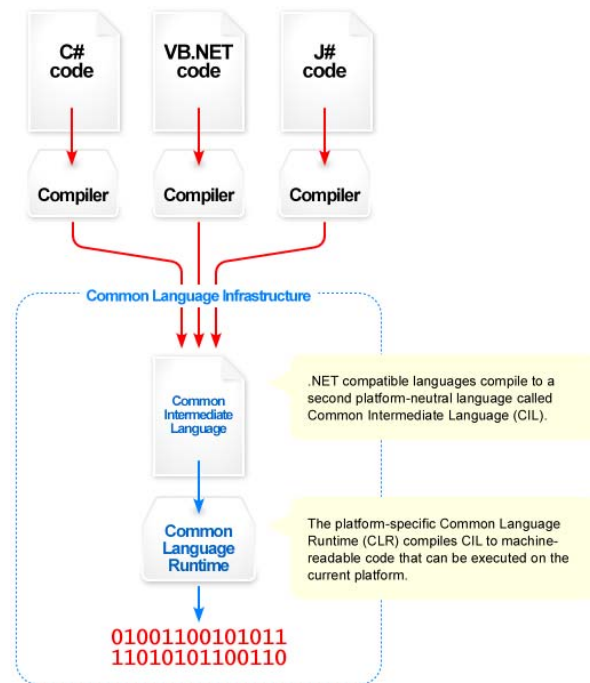


Fig: Visual overview of the Common Language Infrastructure (CLI). Source: www.Wiki.com

5.4 And the choice is...

While IE have some advantages over Mozilla, the choice stills lands on Mozilla due to some simple reasons. All the licenses for programs that would allow working with Explorer just cost too much (we probably would need the Visual Studio C++), while using Mozilla allows us to download the required software from the net. Using pre made work that has been developed in .Net also usually requires a fee for using.

Also the fact that using Mozilla allows the user to redistribute the code with built in enchantments are a strong point. Mozilla also supports the standardized code as it is defined by W3C.

5.5 Project outcomes

After some back and forth, the environment was finally ready to be tested. One way to test if it works is to run the python test scripts bundled with PyXPCOM. This is a normal python program that accepts a string input, and prints it

to the web page. These things is first and foremost all that is needed, the possibility to get text and print it to a page, and get text from another program and use it in a browser (like in the crawler instance, setting url's, complete forms etc).

By confirming this two way communication between the browser and the Python works, it is also shown that it is possible to run python programs on the web with the use of a browser.

MozPython Test Script

```
mozpython.version = 0.1.2  
SERVER_SOFTWARE = MozPython
```

```
get_var is not defined.  
post_var is not defined.
```

[GET Test](#)

ng MozPython in Mozilla

POST Test

MozPython Test Script

```
mozpython.version = 0.1.2  
SERVER_SOFTWARE = MozPython
```

```
get_var is not defined.  
post_var = Using MozPython in Mozilla
```

[GET Test](#)

POST Test

6. Conclusion

As shown, there are possibilities to make set up and use actively an environment combining a web browser and a programming language. By using a language widely spread throughout the Open source community, the availability of not only the programs themselves but also the source code, gives to user the possibilities to alter these programs making them use a browsers capabilities. This allows not only making the program use the parser for the web, but also gives the user directly to the browsers DOM tree, and other built in abilities. By combining existing technologies together, allows a more integrated environment in the programs themselves, and eliminates the need of development of running parameters that use syntaxes defined in other languages. It also helps developers that write new plug-inn's or programs to use the language of their desire, resulting in a community that have it easier to develop something new. The issues concerning these matters are as children's disease, and will be solved almost by themselves in the near future when the technology is widely spread enough.

Since computers are becoming more and more a part of people's everyday life, and therefore the need for more programs that suits special need will also increase. Having a common foundation across platform and language is in need for having effort put into development instead of defining what others already have done, just in another language.

7. Appendix

7.1 Reference

- [1][Harvestman homepage](#)
- [2][Netscape embrace the Bazaar](#)
- [3][Mozilla.org](#)
- [4][Mozilla Accessibility on Unix/Linux](#)
- [5][Mozilla Accessibility Architecture](#)
- [6][Introduction to XPCOM](#)
- [7][PyXPCOM](#)
- [8][Python.org](#)
- [9][Internet Explorer on Wikipedia.net](#)
- [10][Preparatory software definition](#)
- [11][Microsoft COM and .Net documentation](#)
- [12][Comparison of the different layout engines](#)

7.2 Useful links and sources

About Mozilla framework:

- <http://www.mozilla.org/why/framework.html>

About xpcom

- <http://www-128.ibm.com/developerworks/webservices/library/co-xpcom.html>

About pyxpcom

- <http://www.mozilla.org/catalog/architecture/xpcom/pyxpcom/>

Harvestman homepage:

- <http://harvestman.freezope.org/configoptions.html>

Python webpage:

- <http://www.python.org/psf/>

Mozilla Accessibility on Unix/Linux

- http://portal.acm.org/ft_gateway.cfm?id=1061829&type=pdf

Mozilla Accessibility Architecture

- <http://www.mozilla.org/access/architecture>

7.3 Glossary

API - an **application programming interface** is a set of definitions that allows one computer software to communicate with another. It is a method of achieving connection, usually (but not necessarily) between lower-level and higher-level software.

CASE - Computer-aided software engineering - is the use of software tools to assist in the development and maintenance of software. It is what we called “Lego” for programming.

COM - Component Object Model, is a Microsoft platform for software components. It is used to enable cross-application communication and dynamic object creation in any programming language that supports the technology. Although it has been implemented on several platforms, it is primarily used with Microsoft Windows. COM is expected to be replaced to at least some extent by the **Microsoft .NET** framework.

DOM - Document Object Model is an **API** to access HTML and XML documents. It is programming language and platform-independent. Behind the interface the document is represented with an object-oriented model.

Different variants of DOMs were initially implemented by web browsers to manipulate elements in an HTML document. This prompted the World Wide Web Consortium (W3C) to come up with a series of standard specifications for DOM (hence called W3CDOM). A well-structured document can take the tree form using DOM.

DCE - Distributed Computing Environment is a software system that supplies a framework and toolkit for developing client/server applications.

Gecko (layout engine) is the open source web browser layout engine used in all Mozilla-branded software. Written in C++, Gecko is designed to support open Internet standards. Gecko offers a rich programming **API** that makes it suitable for a wide variety of roles in Internet enabled applications, such as web browsers, content presentation and client/server.

IDL - an interface description language or interface definition language is a computer language or simple syntax for describing the interface of a software component. It is essentially a common language for writing the "manual" on how to use a piece of software from another piece of software, almost as a user manual. IDLs are used in situations such as for example: C and Pascal have different ways of calling routines, and in general cannot call code written in the other language. IDLs are a subset of both, a general language to which both can conform to enable language-independent code.

JavaScript is an object-based scripting programming language based on the concept of prototypes. The language is best known for its use in websites, but is also used to enable scripting access to objects embedded in other applications. It was originally developed by

<Group 8>
Browser based Web Accessibility Measurement Component

Brendan Eich of Netscape Communications Corporation under the name *Mocha*, then *LiveScript*, and finally renamed to JavaScript. Like Java, JavaScript has a C-like syntax, but it has far more in common with the programming language “Self” than with Java.

Microsoft .NET is a software development platform focused on rapid application development (RAD), platform independence and network transparency. According to Microsoft, .NET includes many technologies that are designed to facilitate rapid development of Internet and intranet applications.

Self is an object-oriented programming language based on the concept of *prototypes*. It was used primarily as an experimental test system for language design in the 1990s; however, as of September 2004, Self is still being actively developed.

RAD - Rapid application development- includes iterative development system, the construction of prototypes, and the use of **CASE** tools.

A **tree structure** is a way of representing the hierarchical nature of a structure in a graphical form. The starting node is often called the "root." The lines connecting elements are called "branches," the elements themselves are called "nodes." Nodes without children are called "end-nodes" or "leaves."

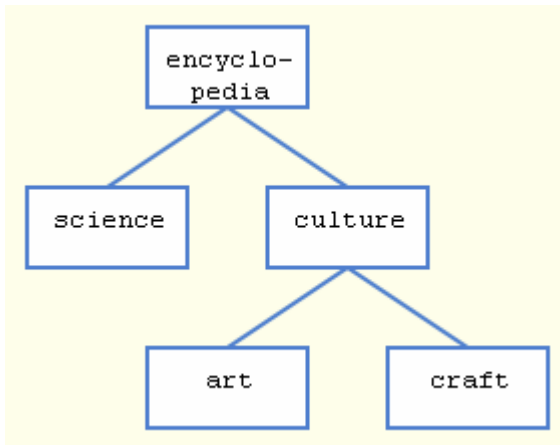


Illustration: en example of a tree structure.