

**Web Structure Mining****IKT407****Autumn 2006****Group 1****HØGSKOLEN I AGDER****Agder University College***Faculty of Engineering and Science*

<b>Author:</b> <b>Li wenjie, Yang guang</b> <b>Alonso, Ignacio</b>	<b>Supervisor:</b> Olsen, Morten Goodwin
<b>Version:</b> 1.1 <b>Status:</b> Final	<b>Pages:</b> 39 including this page <b>Modified date:</b> 2006 – 11 – 22
<b>Keywords:</b> <b>Mozilla XUL Xulrunner Browser based measurement DOM tree</b>	
<b>Abstract:</b> <p>This report describes attempt to develop a measure component for measuring web page's elements. A series of introduction about theoretic expatiate and practice experiences in this report in order to demonstrate a concept, which is how a web browser based measurement component works.</p> <p>Comparing all the solutions we what we have worked out, the best so lution is a combination of Xulrunner and Python. This solution can gain different kinds of elements and their amount in webpage. What we have done, in wish to give further development in this area some significant effects. Also, we want to use our application in this area to demonstrate the measurement based browser.</p>	

## Table of content:

<b>1. introduction</b> .....	<b>4</b>
1.1 Goal of project.....	4
1.2 Strategy of project.....	4
1.3 What was done and how?.....	4
1.4 Report outline.....	5
<b>2. Background</b> .....	<b>6</b>
Mozilla.....	6
DOM.....	7
XUL.....	8
Xulrunner.....	9
SOAP.....	9
SOAPpy.....	10
Python library.....	10
WAI and WCAG.....	11
MSAA.....	12
<b>3. Problem description</b> .....	<b>14</b>
<b>4. Solution</b> .....	<b>15</b>
4.1 Requirement.....	15
4.1.1 Environment requirement.....	15
4.1.2 Functional requirement.....	16
4.2 Design.....	16
4.2.1 Python(HTML).....	17
4.2.2 Python(XML).....	19
4.2.2-1 XML MODEL(xml.dom).....	19
4.2.3 Python+Xulrunner.....	20
4.3 Implementation.....	21
4.3.1 Python(HTML).....	21
4.3.2 XML Implementation.....	22
4.3.2-1 TIDY(HTML to XML).....	24
4.3.3 Xulrunner.....	24
4.3.3-1 Our browser(XUL Application).....	25
4.4 Test and Compare.....	30
<b>5. Discussion</b> .....	<b>32</b>
5.1 Different in Implementa tion.....	32
5.2 Different in Output.....	33

<b>6. Conclusion.....</b>	<b>35</b>
<b>A. Appendix.....</b>	<b>31</b>
<b>A.1 Glossary and Abbreviations .....</b>	<b>36</b>
<b>A.1.1 EIAO.....</b>	<b>36</b>
<b>A.1.2 W3C.....</b>	<b>36</b>
<b>A.1.3 URL.....</b>	<b>36</b>
<b>A.1.4 MSAA.....</b>	<b>36</b>
<b>A.1.5 web page.....</b>	<b>36</b>
<b>A.1.6 XUL.....</b>	<b>36</b>
<b>A.1.7 XML.....</b>	<b>36</b>
<b>A.2 Reference.....</b>	<b>37</b>
<b>Project progress.....</b>	<b>39</b>

## Table of Figures

<b>Figure 2.1 DOM tree of test.html .....</b>	<b>7</b>
<b>Figure 2.2 SOAP message structure .....</b>	<b>10</b>
<b>Figure 2.3 MSAA nodes .....</b>	<b>13</b>
<b>Figure 4.1 Folder structure.....</b>	<b>25</b>
<b>Figure 4.2 URLs input.....</b>	<b>25</b>
<b>Figure 4.3 Browser interface.....</b>	<b>27</b>
<b>Figure 4.4 Output.....</b>	<b>27</b>
<b>Figure 4.5 DOM tree of our application .....</b>	<b>29</b>
<b>Figure 4.6 Test 1.....</b>	<b>30</b>
<b>Figure 4.7 Test 2.....</b>	<b>31</b>
<b>Figure 6.1 Process of the browser .....</b>	<b>35</b>

## Table of Tables

<b>Table 4.1 DOM objects.....</b>	<b>20</b>
<b>Table 5.2 Comparison two methods.....</b>	<b>33</b>
<b>Table 5.2 Comparison of Mozilla and Traditional parser .....</b>	<b>34</b>

# 1. Introduction

Nowadays, we become more and more depend on Internet to get various information, by this way, we are easy to deal with our study and work, and sometimes, we use it just for entertainment. There are several browsers we can choose. Mozilla is a special one, can provide a flexible framework, in which user can build their own third part plugs. Mozilla Accessibility architecture make s Mozilla as a container for WAM measurement system. We can make use of those features to build measurement components simplify some accessibility measurements, like identifying screen flickering, redirection, JavaScript links or spawned windows, by subscribing to events via the interface or the DOM tree.

## 1.1 Goal of project

The goal of this project is to demonstrate the concept of a web browser based measurement component using Mozilla.

## 1.2 Strategy of project

At very beginning, we try to understand all concepts and build a mind about what kind of architecture we should to build. Then we found that there are several ways we can use, we need to find the most efficient way. One of most important purpose is the way which we use in our project should be a typical way to build a measurement component in Mozilla.

## 1.3 What was done and how?

We build a Xulrunner application consists of a SOPYpy based server that communicates with Mozilla, and a JavaScript that loads documents on demand from the python server, and present the links found in the document.

## 1.4 Report outline

This report has five chapters including the introduction and conclusion.

Firstly, we introduce the background of our project, in chapter two. Then we describe the problem of our project in chapter three. Chapter four is about our solutions, including our main solution and other comparing solutions, validation and testing. We put discussion in chapter five and conclusion in chapter six. In the appendix a glossary and abbreviation, reference list, figures and tables list, plan of our project progress can be found.

## 2. Background

### Mozilla

Mozilla is a computer term which has had many different uses, though all of them have been related to the now-defunct Netscape Communications Corporation and its related application software.

Mozilla is used as free software / open source software project that was founded in order to create the next-generation Internet suite for Netscape. The Mozilla Organization was founded in 1998 to create the new suite. On July 15, 2003, the organization was formally registered as a not-for-profit organization, and became Mozilla Foundation. The foundation now creates and maintains the Mozilla Firefox browser and Mozilla Thunderbird email application, among other products.

Because of Mozilla is built on a flexible framework, it always can be used as the base for application other than a browser. Many enthusiastic developer and programmer prefer to use it, take a fancy to it free, faster, and powerful. It combines stability, security with incredible flexibility. A funny guy who is called Neil Deakin has list out 101 items what Mozill a can do but IE can not do.

Creating a cross-platform application in Mozilla just need putting together existing components using JavaScript and XUL. New component to extend Mozilla's functionality can be written in C++ or JavaScript. Our project focuses on this point; we design a measurement component in Mozilla which conform to the web services based plug-in interface for EIAO Web Accessibility Observatory.

When it comes to measurement accessibility, Mozilla is interesting as a container for WAM measurement systems due to Mozillas Accessibility architecture[MOZ -A] which is used by 3rd party software like screen readers, magnifiers, and voice dictation software, which need information about document content, UI controls and events like changes of focus. Mo zilla supports two accessibility APIs: MSAA on Windows [MSAA] and ATK on Linux and UNIX [ATK]. Mozilla has also implemented an accessibility model into the browser consisting of transformations from a non-accessible DOM tree to an accessible

## DOM

Document Object Model (DOM) is a description about HTML and XML documents. This description is represented in a tree structure. DOM provides a structure that facilitates access to the elements of an HTML or XML document by scripting languages with object-oriented features (e.g. JavaScript).

Example:

Test.html

```
<html>
<head>
<title>
show dom
</title>
</head>
<body>
<p> this a simple example</p>
<img src="">img</img>
</body>
</html>
```

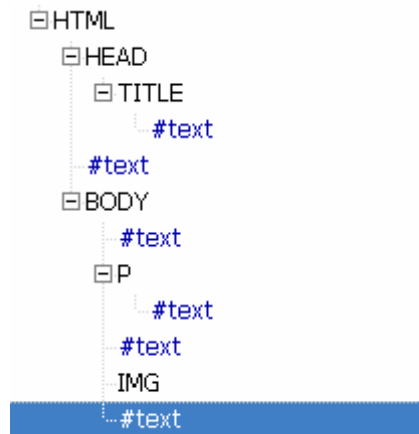


Figure 2.1 Dom tree of test.html

PS: The graph a get from Dom inspector in Mozilla.

User use web browsers convenient because all the web browsers' interfaces are friendly. Various interfaces were initially implemented by web browsers to manipulate elements in an HTML document through JavaScript. This prompted the World Wide Web Consortium (W3C) to come up with a series of standard specifications that defined the W3C Document Object Model (W3C DOM).

The W3C DOM specifications are divided into levels, each of which contains required and optional modules. To claim to support a level, an application must implement all the requirements of the claimed level and the underlying levels. An application may also support vendor-specific extensions if they don't conflict with the W3C standards. As of 2005, Level 1 and level 2, and some modules of Level 3 are W3C Recommendations which means they have reached their final form.

Depends on our project, now we comparing Level 1 and level 2, as follows:

### Level 1

Navigation of DOM (HTML and XML) document (tree structure) and content manipulation includes adding elements. HTML-specific elements are included as well.

**Level 2**

XML namespace support, and filtered views and events.

**Xul**

XUL was created to make development of the Mozilla browser easier and faster. It is an XML language so all features available to XML is also available to XUL. In Mozilla, XUL is handled much in the same way that HTML or other types of content are handled. When you type the URL of an HTML page into the browser's address field, the browser locates the web site and downloads the content. The Mozilla rendering engine takes the content in the form of HTML source and transforms it into a document tree. The tree is then converted into a set of objects that can be displayed on the screen. CSS, images and other technologies are used to control the presentation. XUL functions in much the same way. Actually, HTML and XUL shared many features. Because all document types in Mozilla, whether they are HTML or XUL are handled by the same underlying code. However, there are some features that are special to HTML such as form, and others which are special to XUL such as overlays. This is good for us, if we familiar with one of this we can master another.

For certain security reasons, content from remote sources, regardless whether they are HTML or XUL or another document type, are limited in the type of operations they can perform. For this reason, Mozilla provides a method of installing content locally and registering the installed files as part of its chrome system. This allows a special URL form to be used called a chrome URL. By accessing a file using a chrome URL, the files receive elevated privileges to access local files, access preferences and bookmarks and perform other privileged operations. Obviously, web pages do not get these privileges, unless they are signed with a digital certificate and the user has granted permission to perform these operations.

The packages that are provided with Mozilla are located within the chrome directory, which you will find in the directory where you installed Mozilla. There are three types of chrome packages are: Content, Skin and Locale. Based on our project, we are going to Content and Locale now.

**Content** - Windows and scripts

The declarations of the windows and the user interface elements contained within them. These are stored in XUL files, which have a xul extension. A content package can have multiple XUL files, but the main window should have a filename that is the same as the package name. For example, the

editor package will have a file within it called edit or.xul. Scripts are placed in separate files alongside the XUL files.

### **Locale** - Locale specific files

All the text that is displayed within a window is stored separately. This way, users can have a set for their own language.

## **Xulrunner**

Xulrunner is a command-line app to run XUL applications; it is a Mozilla runtime package that can be used to develop XUL+XPCOM applications. It is a convenient tool in case developers are very familiar with command-line develop environment.

## **SOAP**

A web service is a network accessible interface to application functionality, built using standard Internet technology. If an application can be accessed over a network using a combination of protocols like HTTP, XML, SMTP, or Jabber, then it is a web service. A web service consists of three components:

A listener to receive the message

A proxy to take that message and translate it into an action to be carried out the application code to implement that action

SOAP's place in web services technology stack is as a standardized packaging protocol for the messages shared by applications. The specification defines a simple XML-based envelope for the information being transferred, and a set of rules for translating application and platform-specific data types into XML representations. SOAP's design makes it suitable for a wide variety of application messaging and integration patterns. SOAP is an application of the XML specification. It relies heavily on XML standards. Because of this, XML messaging provides a flexible way for application to communicate. This is the basis of SOAP.

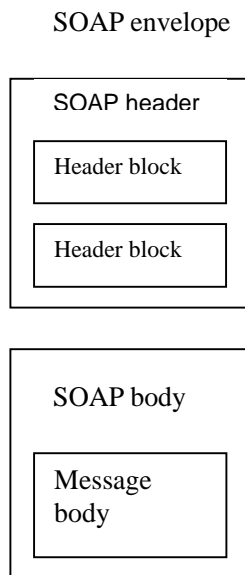


Figure 2.2 SOAP message structure

## SOAPpy

There are many programming languages can be used to develop a SOAP web service, such as Perl, JAVA, and so on. Our choice is Python, considering that should be easy. Actually, it makes our work easier than we have supposed, because Python is not too difficult to master, and our project is also not complicated. Most programmers do not need to understand all details of encodings and envelopes. SOAP provides powerful toolkits which can be installed in different programming languages. There are SOAP toolkits for all the popular programming languages and environments. (Java, C#, C++, C, Perl, PHP, and Python and so on).

SOAPpy is a SOAP toolkit provides for Python to develop a Python web service.

## Python library

Python is a big topic, if we deep into programming thought and every detail of it, of course we can learn a lot, but we can not finish our report and it will beyond all readers' patience.

As many other programming languages, Python has its own library. That means we can use lots of common or special functions directly. The bulk of the library, however, consists of a collection of modules. There are many ways to dissect this collection. Some modules are written in C and built in to the Python interpreter; others are written in Python and imported in source form. Some modules provide interfaces that are highly specific to Python, like printing a stack trace; some provide interfaces that are specific to particular operating systems, such as access to specific hardware; others provide interfaces that are specific to a particular application domain, like the World Wide Web. Some modules are available in all versions and ports of Python; others are only available when the underlying system supports or requires them; yet others are available only when a particular configuration option was chosen at the time when Python was compiled and installed.

Now we turn to discuss two of those modules: `sgmlib` and `urllib`. They are useful in our initial solutions.

The first module defines a class `SGMLParser` which serves as the basis for parsing text files formatted in SGML (Standard Generalized Markup Language). In fact, it does not provide a full SGML parser -- it only parses SGML insofar as it is used by HTML, and the module only exists as a base for the `htmlib` module.

The second module provides a high-level interface for fetching data across the World Wide Web. In particular, the `urlopen()` function is similar to the built-in function `open()`, but accepts Universal Resource Locators (URLs) instead of filenames. Some restrictions apply -- it can only open URLs for reading, and no seek operations are available.

## WAI and WCAG

As one part of W3C, it stands for Web Accessibility Initiative. It focuses on development of uniform protocols to make Web sites more accessible. It is an effort to improve the accessibility of the World Wide Web for people using a wide range of user agent devices. So it not just standard web browsers. It is especially important for people with disabilities which need extra devices to access the Web.

The WAI Guidelines contains mainly four branches: Web Content Accessibility Guidelines (WCAG), Authoring Tool Accessibility Guidelines (ATAG), User Agent Accessibility Guidelines (UAAG) and XML Accessibility Guidelines (XAG).

In designing of our browser, we need to take those in mind, especially the Web Content Accessibility Guidelines (WCAG) which are a set of guidelines on making content accessible, primarily for disabled users, but also for all user agents, including highly limited devices, such as cell phones.

## MSAA

Architecture of Mozilla's accessibility API module provides support for platform accessibility API. Those accessibility APIs are used by 3rd part software like screen readers, screen magnifiers, and voice dictation software, which need information about document content and UI controls, as well as important events like changes of focus. We follow the project's aim find what are the two accessibility APIs in Mozilla: MSAA on Windows and ATK on Linux and UNIX.

Microsoft Active Accessibility (MSAA) API used on Windows operating systems to suppose assistive technologies for users with disabilities.

Every node in the DOM tree could be important to 3rd party assistive technology. Accessibility APIs on each operating system have built-in assumptions about what is the most important information, and how an accessibility server like Mozilla should use the API's programmatic interfaces to expose this information to an accessibility client (the assistive technology). Each platform's accessibility API has made different assumptions, although 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 finite list, to expose the role of an object. Examples of accessible role constants are `ROLE_BUTTON`, `ROLE_CHECKBOX` and `ROLE_LIST`, although they can have slightly different names and values in each API. In general, the accessibility APIs uses similar concepts, but use different method, constant and interfaces names.

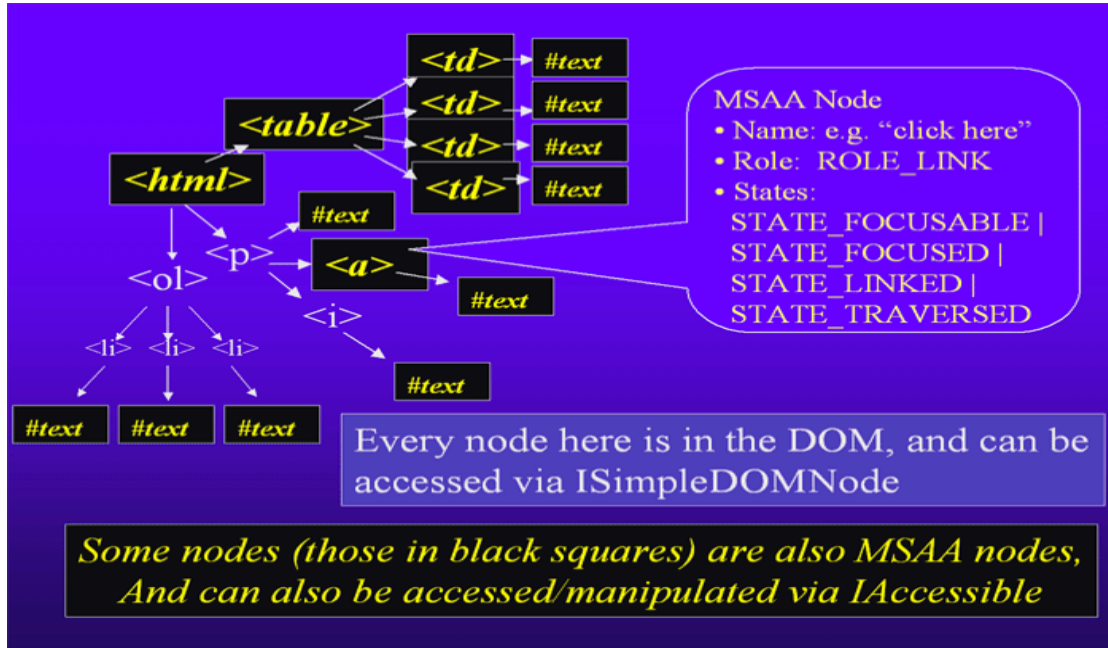


Figure 2.3 MSAA nodes

### 3. Problem description

Our project is to demonstrate the concept of a web browser based measurement component using Mozilla [Moz]. That is to build an API's programmatic interfaces to get what elements users are interesting. Main idea of this project is to find a way. By this way we can achieve the certain elements which we want to get to know about some certain information such as amount.

We want follow the way that a normal man to get knowledge of the browser and the World Wide Web. There are several problems:

What kinds of files are present in browser?

What traditional browsers (such as IE) do?

What is a DOM tree?

Those methods provide to deal with Dom.

What a measure based browser do?

We would like to find the amount of the certain elements to answer those questions. That will help us to demonstrate the measure based browser.

The accessibility API toolkits can be used for cross -platform programming, and because of those assistive technology, we can get the concept of build a measure component in Mozilla. What is more, we can try to find a good way for further development in this area.

## 4. Solutions

The main solution strategy: We try to use python code combining with Mozilla, thus, we need to find a way to connect them. The main work is to program python code in right way and actualize the connection, make sure all of them can actually run and work.

Here, we have three main ways. They act as marks on our way to solve this project. The first, we try to operate html files with python to get the elements, and count its number. The website not only has html files today but also XML files. Thus the second step operates the XML files in Python; proof the main operation is worked. At last we know how to operate the Dom and combining them together to get the final solution.

### 4.1 requirements

First of all, the requirements of this design problem must be point out. We divide the requirements to two parts.

#### 4.1.1 Environment requirements

In order is to demonstrate the concept of a web browser based measurement components. Some software and support packages are needed.

--**Python**: the main program language we used in this program. It is easily to combine with other program languages and have a high level cooperation with them. It is an interpreted language. It knows as the most famous language used in network designing.

--**Mozilla**: A web browser based measurement components. We work over on it.

--**XUL**: XUL (pronounced "zool") is Mozilla's XML-based user interface language that lets us build feature rich cross-platform applications that can run connected to or disconnected from the Internet. These applications are easily customized with alternative text, graphics, and layout so they can be readily branded or localized for various markets. We learn this language to help us in understanding the process in Mozilla.

--**Xulrunner:** Xulrunner is a Mozilla runtime package that can be used to bootstrap XUL+XPCOM applications that are as rich as Firefox and Thunderbird. It will provide mechanisms for installing, upgrading, and uninstalling these applications. Xulrunner will also provide libxul, a solution which allows the embedding of Mozilla technologies in other projects and products. Thus we use the Mozilla technologies here to build our own application.

--**web pages:** test the application.

--**python packages:** The programming environment in windows is not so good, it not contain the full vision of sources, you should download those pages you self. Those packages will descript later in 3.1.2

## 4.1.2 Functional requirements

After we diagnose the problem, we want to find the number of the specified kind element in the Dom tree to proof the task. Thus some specific functions must be included.

--models: models provide the methods to operate the Dom or the resource text.

--Browser/parser: A Xulrunner application, that functions as a web browser. For us, easily add some functions and test them. It contain main operate to the browser

--web server: a server, it is set up for support our browser to connect with Internet and also provide the information transfers between browser and Internet.

--count function: the main function to count the number of specific element. It is the most important parts of this project.

--output function: this function is used for output the result.

--input function: input the element name, and also check it's a available name or not.

Those are threes basic parts. A few changes would take place when faced the actual situation. As we know more about the project, our method changed and closer to the final solution.

## 4.2 Design

We have three main ways to solve the problem. It will give us some instructions to understand the

different between traditional parser and mozilla.

### 4.2.1 Python (HTML)

When we faced this project at the first time, A website means a html file for us. Thus the first we do is finding those packages in python to handle the html files and later try to know how to use those packages.

We found some models. We will demonstrate it one by one.

The first model is **urllib**. This module provides a high-level interface for fetching data across the World Wide Web. In particular, the `urlopen( )` function is similar to the built-in function `open( )`, but accepts Universal Resource Locators (URLs) instead of filenames. Some restrictions apply -- it can only open URLs for reading, and no seek operations are available.

After we have the html information, we should know the how html processing. It is broken into three steps: breaking down the HTML into its constituent pieces, fiddling with the pieces, and reconstructing the pieces into HTML again. The first step is done by `sgmlib.py`, a part of the standard Python library. We meet another model here `sgmlib.py`. What is it?

**Sgmlib** contains one important class: `SGMLParser`. `SGMLParser` parses HTML into useful pieces, like start tags and end tags. As soon as it succeeds in breaking down some data into a useful piece, it calls a method on itself based on what it found. In order to use the parser, you subclass the `SGMLParser` class and override these methods. `Sgmlparser` parses HTML into 8 kinds of data, and calls a separate method for each of them. We focus on the method start tag and end tag.

----Start tag

An HTML tag that starts from a block, like `<html>`, `<head>`, `<body>`, `<pre>`, or a standalone tag like `<br>`,`<img>`. When it finds a start tag `tagname`, `SGMLParser` will look for a method called `start_tagname` or `do_tagname`. For instance, when it finds a `<pre>` tag, it will look for a `start_pre` or `do_pre` method. If found, `SGMLParser` calls this method with a list of the tag's attributes; otherwise, it calls `unknown_starttag` with the tag name and list of attributes.

-----End tag

An HTML tag that ends a block, like `</html>`, `</head>`, `</body>`, or `</pre>`. When it finds an end tag, SGMLParser will look for a method called `end_tagname`. If found, SGMLParser calls this method, otherwise it calls `unknown_endtag` with the tag name.

Other parts:

#### Character reference

An escaped character referenced by its decimal or hexadecimal equivalent, like `&#160;`.

When found, SGMLParser calls `handle_charref` with the text of the decimal or hexadecimal character equivalent.

#### Entity reference

An HTML entity, like `&copy;`. When found, SGMLParser calls `handle_entityref` with the name of the HTML entity.

#### Comment

An HTML comment, enclosed in `<!-- ... -->`. When found, SGMLParser calls `handle_comment` with the body of the comment.

#### Processing instruction

An HTML processing instruction, enclosed in `<? ... >`. When found, SGMLParser calls `handle_pi` with the body of the processing instruction.

#### Declaration

An HTML declaration, such as a DOCTYPE, enclosed in `<! ... >`. When found, SGMLParser calls `handle_decl` with the body of the declaration.

#### Text data

A block of text. Anything that doesn't fit into the other 7 categories. When found, SGMLParser calls `handle_data` with the text.

The sgmlparser divide html files. It also provides some methods to deal with each piece. It gives us a chance that we can reload the start tag method to add an arithmometer in the function. Thus we can get the number of that specific kind element while the sgmlparser is working.

This method based traditional parser; help us to know more about it. Thus it is find the differences.

This way goes without Dom. Second we seek for Dom applications.

## 4.2.2 Python (XML)

Now we know some basic things of this project, such as how did the parser do when they receive a web-page? But as it mentions in the problem description, Mozilla has also implemented an accessibility model into the browser consisting of transformations from a non-accessible DOM tree to an accessible DOM tree. Thus we also find information about DOM tree.

--DOM. "The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page." this define is found in <http://www.w3.org/DOM/>

After we know the main conception of DOM, and then we try to find the packages in python, which support the operation with DOM tree. The normal DOM tree is based on XML files. Therefore we found the package XML.

### 4.2.3-1 XML MODEL (xml.dom)

In the XML model, there are two basic ways to work with XML. One is called SAX ("Simple API for XML"), and it works by reading the XML a little bit at a time and calling a method for each element it finds. This should sound familiar to that's how the sgmlib module works. The other is called DOM, also the one we want to know, and it works by reading in the entire XML document at once and creating an internal representation of it using native Python classes linked in a tree structure (this is the DOM tree). Python has standard modules for both kinds of parsing, but we will only deal with using the DOM in this document.

The second way DOM is contains in xml.dom. A DOM implementation presents an XML document as a tree structure, or allows client code to build such a structure from scratch. It then gives access to the structure through a set of objects which provided well-known interfaces. It also provides some method to deal those elements. We will introduce the objects in DOM first. Second the methods and attributes.

#### --Objects

The definitive documentation for the DOM is the DOM specification from the W3C.

Note that DOM attributes may also be manipulated as nodes instead of as simple strings

Interface	Purpose
DOMImplementation	Interface to the underlying implementation.
Node	Base interface for most objects in a document.
NodeList	Interface for a sequence of nodes.
DocumentType	Information about the declarations needed to process a document.
Document	Object which represents an entire document.
Element	Element nodes in the document hierarchy.
Attr	Attribute value nodes on element nodes.
Comment	Representation of comments in the source document.
Text	Nodes containing textual content from the document.
ProcessingInstruction	Processing instruction representation.

Table 4.1 DOM objects

After we know the basic objects of DOM, there are so many methods also provide to deal with those objects. We can use those methods to search for elements in the domtree.

Some code will give to explain the methods. That will include in the implementation parts.

### 4.2.3 Python+xulrunner

We introduce two ways to solve this problem. They help a lot, but they both are not the best way. First one its only html based and it works like SAX. It is useless to solve, just help us to learn the DOM more clearly. The second one, the XML package in python, it explain most of the work with DOM. It provide us know the main methods deal with DOM. The package it's not power enough to deal with both html and xml. You should change the html to XML when you parse it. And those two methods are our based the traditional parser. If we want find the differences, we must also make an application of mozilla.

After diagnose the weakness in the given two methods, we start to do something with mozilla and xulrunner. Try to merge those methods together and implement in mozilla or xulrunner.

We choose xulrunner, as we mentions above its more free to development.

In the previous words, we just say what xulrunner it is, and no description about how to develop in it. I think we should talk about here. It is a big topic, it a waste of time to deep in programming thought.

Thus we just provide the conception of how a xulrunner application can be build, and object using in this project. We will provide you know that while we explain the final solution.

## 4.3 Implementation

**This part will descript the implementations of each way.**

### 4.3.1 Python (HTML)

**Code:**

```
>>> import urllib, blinknum
    #import those packages
>>> usock = urllib.urlopen("http://www.google.com/")
    #using the urllib.urlopen for getting information about and actually retrieving data from
    "http://www.google.cm"
>>> parser = blinknum.num()
    # set a parser ,it's defined below, it's a inherit class of sgmlparser
>>> parser.feed(usock.read())
    # html processing
>>> usock.close()
>>> parser.close()
    #free the momery
>>> print blinknum.num
```

**Model blinknum**

```
#set a inherit class of sgmlparser, and reload the function start_blink,
class blinknum(SGMLParser):
    def reset(self):
        SGMLParser.reset(self)
        self.num =0    # initial the num of ' a ' element
```

```

def start_a(self, attrs):
    #when sgmlparser find a start tag blink, the number plus one.
    self.num+=1

```

This is the easiest way to find the number of specific kind element. There are some advantages and disadvantages in this method. It is traditional parser based.

### 4.3.2 XML Implementation

A test XML files and a pieces of codes are include.

**Binary.xml**(test xml file):

```

<?xml version="1.0"?>
<!DOCTYPE grammar PUBLIC "-//diveintopython.org//DTD Kant Generator Pro v1.0//EN"
"kgp.dtd">
<grammar>
<ref id="bit">
    <p>0</p>
    <p>1</p>
</ref>
<ref id="byte">
    <p><xref id="bit"/><xref id="bit"/><xref id="bit"/><xref id="bit"/> \
        <xref id="bit"/><xref id="bit"/><xref id="bit"/><xref id="bit"/>
    </p>
</ref>
</grammar>

```

**Code:**

```

>>> from xml import minidom
>>> xmldoc = minidom.parse('binary.xml')
>>> xmldoc.childNodes

```

1

```
[<DOM Element: grammar at 17538908>]
```

```
>>> reflist = xmldoc.getElementsByTagName('ref')           2
```

```
>>> bitref = reflist[0]                                   3
```

```
>>> print bitref.toxml()
```

```
<ref id="bit">
```

```
  <p>0</p>
```

```
  <p>1</p>
```

```
</ref>
```

```
>>> bitref.attributes                                     4.5
```

```
>>> bitref.attributes.keys()                             6
```

```
[u'id']
```

```
>>> bitref.attributes.values()                           7
```

```
>>> bitref.attributes["id"]                             8
```

1. Every Node has a childNodes attribute, which is a list of the Node objects. A Document always has only one child node, the root element of the XML document (in this case, the grammar element). We can use childnodes[0] to get the first child. Here this method helps us get the view of DOM tree. firstChild attribute, which is synonymous with childNodes[0] (There is also a lastChild attribute, which is synonymous with childNodes[-1].)

2. GetElementsByTagName takes one argument, the name of the element you wish to find. It returns a list of Element objects, corresponding to the XML elements that have that name. In this case, you find two ref elements. As a list, you can easily know how many elements you find. Using len(reflist).

3. Let bitref point at the first node of reflist

4. Each Element object has an attribute called attributes, which is a NamedNodeMap object. This sounds scary, but it's not, because a NamedNodeMap is an object that acts like a dictionary.

5. Treating the NamedNodeMap as a dictionary, you can get a list of the names of the attributes of this element by using attributes.keys(). This element has only one attribute, 'id'.

6. Attribute names, like all other text in an XML document, are stored in unicode.

7. Again treating the NamedNodeMap as a dictionary, you can get a list of the values of the attributes by using attributes.values(). The values are themselves objects, of type Attr. You'll see how to get

useful information out of this object in the next example.

8. Still treating the NamedNodeMap as a dictionary, you can access an individual attribute by name, using normal dictionary syntax.

Now, we already know, how to deal with xml, how to use the XML.DOM to search the element in the DOM tree. As it mentioned in 2, the method `getElementsByTagName` can search the elements in Dom tree, returns a list. So for a list you can get the number of the element. Other methods can let you know more about the operation to the Dom. But this package only provides the method to deal with dom in XML files. If you want deal with html or others, you should first transfer the html to xml.

### 4.3.2-1 TIDY (HTML to XML)

This method we use another package called tidy.

We won't deep into this method, in my opinion if you want a get a element in html files, you spent time to change it to XML. It takes so much time.

You can find more information on this website:

<http://www.xml.com/pub/a/2004/09/08/pyxml.html>

If you are interested in this area, you can go search more information.

### 4.3.3 Xulrunner

Xulrunner is a Mozilla runtime package that provide you can develop you Xul applications individually, it allows you to run Xul applications without explore. Xulrunner also provide libxul, a solution which allows the embedding of mozilla technologies in other projects and products.

We will use mozilla technologies to make a browser application.

Firstly, a xulrunner application has a standard folder structure. There are more complicated ways to do it, but this seems to be the bare bones.

```

/applicationName
  /chrome
    /applicationName
      your app files
      chrome.manifest
    /defaults
    /preferences
      prefs.js
      application.ini

```

Figure 4.1 Folder structure

Look deep in this structure.

--**Application.ini**: XULRunner works by reading an application.ini file, and then loading things up by looking at specific directories.

--**prefs.js**: This is a preferences file, where you set initial values of things. Like gecko engine settings or application specific settings.

--**chrome.manifest**: This file performs the mapping between a chrome:// url and your application files:

Content applicationName file:applicationName

Those three are the configure of the applications. Most functional files you application files, for us a Xul file for the interface difine, the other one provide the script source.

### 4.3.3-1 Our browser (XUL Application)

Firstly, we use a python script server.py as the local servers. soappy.py is import in order to make it useful. The URLs are input in this area.



Figure 4.2 URLs input

Secondly, this is the interface of the application. It is defined in jsawam.xul.

**Jsawam.xul** Code:

```
<script src="jsawam.js"/>
```

```
<hbox>
```

```
  <button id="back" label="&jsawam.back;" oncommand="back();" disabled="true"/>
```

```
<button id="forward" label="&jsawam.forward;" oncommand="forward();" disabled="true"/>
<button id="reload" label="&jsawam.reload;" oncommand="reload();"/>
<button id="stop" label="&jsawam.stop;" oncommand="stop();" disabled="true"/>
<textbox id="urlbar" value="" flex="1" onchange="go();"/>
<button id="go" label="&jsawam.go;" oncommand="go();"/>
</hbox>
<hbox>
  <label value="Input the element name, which you want to test:"/>
  <textbox flex="1" id="testinput"/>
  <button label="test" flex="1" id="etest" oncommand="test();"/>
</hbox>
<browser flex="1" id="browser" src="" type="content -primary"/>
<statusbar>
  <statusbarpanel id="status" label="" crop="end" flex="1"/>
  <progressmeter id="progress" mode="determined" value="0%" style="display: none"/>
  <statusbarpanel id="security" label="" style="display: none"/>
</statusbar>
```

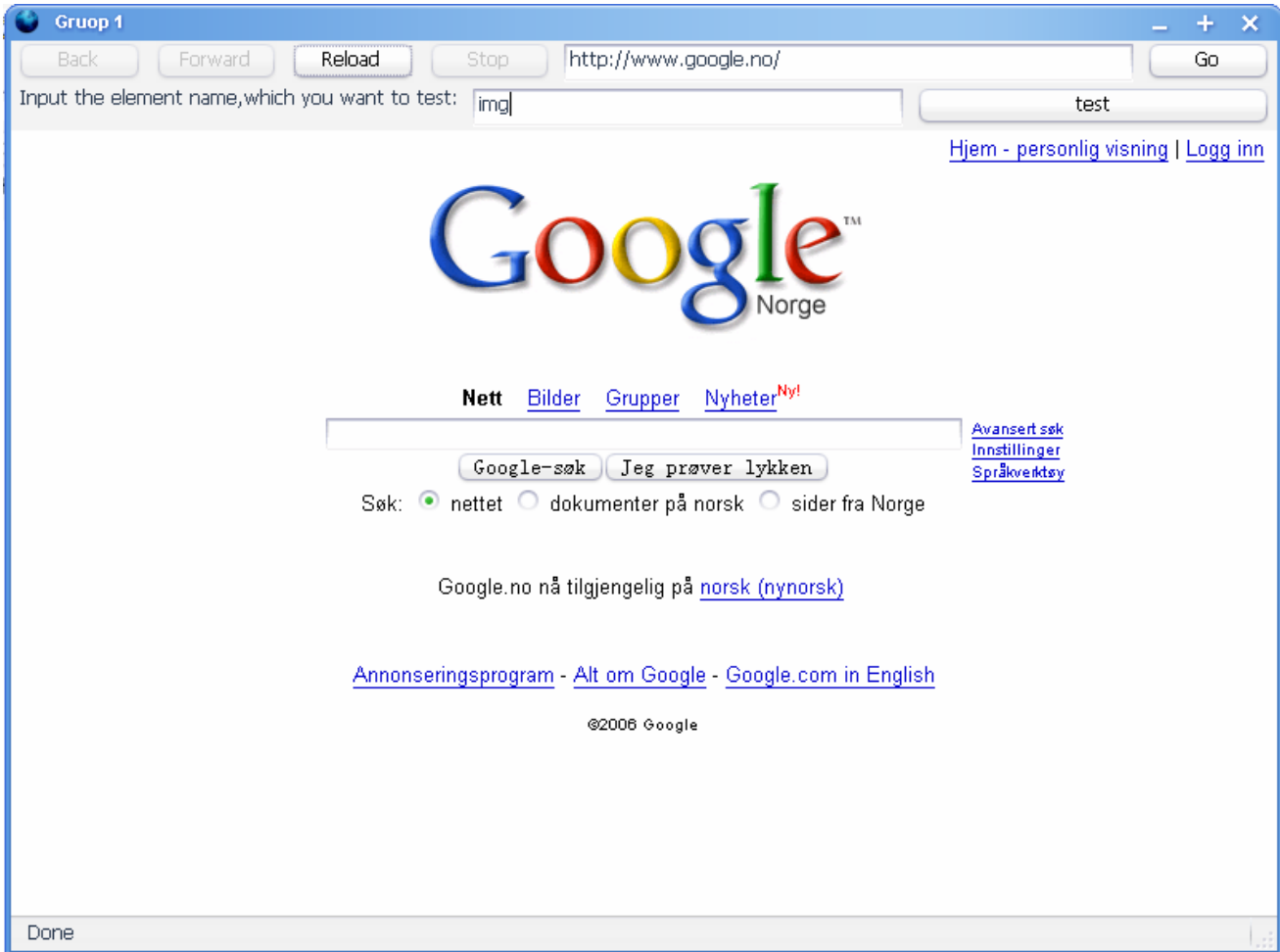


Figure 4.3 Browser interface

As you see in the windows, there is a label” input the element name, which you want to test.” And a textbox element go follow it, you can input the element name here and press the bottom next. The output is here.

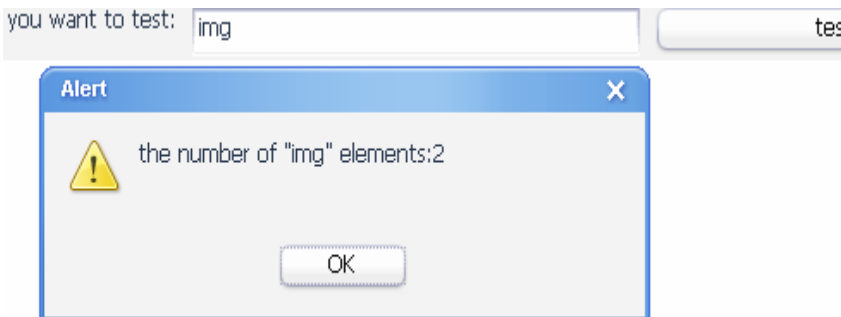


Figure4.4 Output

Most functional functions a defined in jsawam.js.

Jsawam.js is the script source. It provide connect and information transfer with the local servers. Beside these functions, it also define s those functions which used in interface.

After we know the main structure of a application. This is the time to show the advantages of xulrunner.

**The key function:**

```
Function test () {  
    var browser = document.getElementById("browser");           1  
    var input=document.getElementById("testinput");  
    //alert(input.value);  
    links=browser.contentDocument.getElementsByTagName(input.value);2  
    alert('the number of "'+input.value+" elements:'+links.length);    3  
}
```

Here we present the DOM tree of the application windows in order to help us to demonstrate how it works.

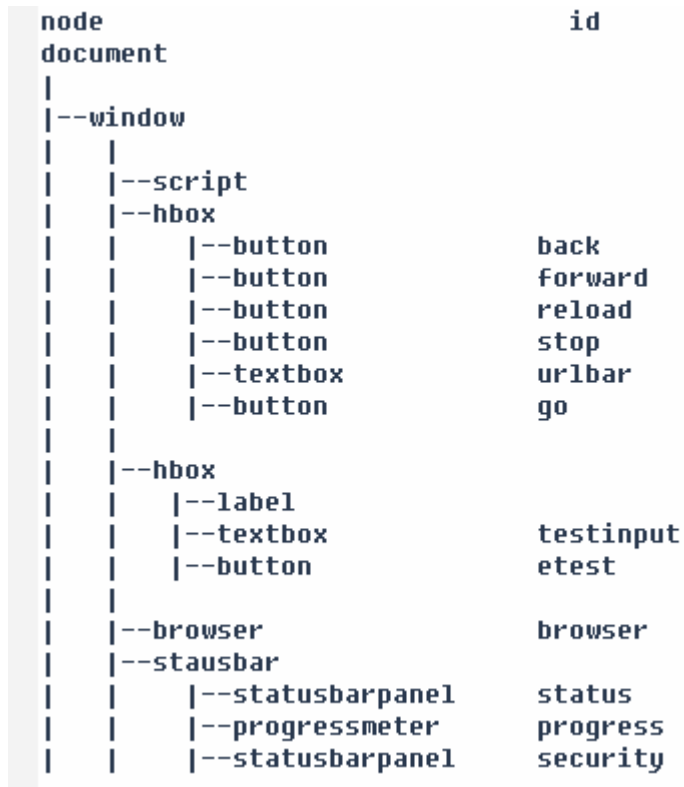


Figure 4.5 Dom tree of our application

We see some methods are used here :

1. `document.getElementById`: the mozilla provide this method for you can search for the element by ID in the document. Every element has its predefined methods.
2. `browser.contentDocument.getElementsByTagName`: `getElementByTagName`, did you remember this where we have talked in chapter 5.3. It's a method to search for elements in DOM tree. It is almost the same here. This one are contain in xulrunner. It's a method in mozilla technologies.

Until now, we know the xulrunner already provide those methods for us. An d the DOM tree already there when the webpages arrive in you browser.

## 4.4 Testing and Compare

We already introduce the main ways and how to implement it. Some differences have found. But besides those differences, no differences happened while deal with those elements. We use some pages to find them out.

Python (XML) and xulrunner almost the same, we just add python (xml) to help us know how to deal with DOM. So we won't test python (xml).

### Test 1

**URL:** <http://www.google.no>

**Testing element:** img

**Output:**

**Traditional parser:** 0

**Mozilla +xulrunner:** 1



Figure 4.6 Test 1

### Test 2

**URL:** <http://www.overseasstudy.com>

**Output:**

**Traditional parser:**



Mouse stop here. Words “my future at my fingertips” moves follow you mouse movement.

**Mozilla+xulrunner:**



Mouse stop here. You see no words follow!

Figure 4.7 Test 2

Testing 3:

Test the element: blink

Traditional parser:

The date in blink element is not blink.

Mozilla:

The date is blink.

## 5. Discussion

Why the output is different? We will discuss the problem and compare the ways.

### 5.1 Differences in Implementation

There are some differences in the ways of implementation. Also we want to discuss why we use those ways, and why we make those changes.

#### Python (html)

##### Advantages:

1. It is an easy way to solve the problem. It is processing before you see the pages.
2. It is easy to understand, that how the browser do with the html files.

##### Disadvantages:

1. You can only compute one element at one time.
2. It is output are too simple and if you want more information, you should change the model. For different elements you have different changes.
3. Is there anything mentions about Dom tree? No, we still don't know what is it and how it works.
4. We use urllib.open function to obtain the information from Internet, but in this method we can just read the files no seek operations are available.
5. This method can only afford HTML files. But, as we know there is not only html in world web sites.

## DOM and Html+XML.SAX

We use a form table to compare with those two. We focus on the differences happened when they are deal with files and elements.

	DOM	HTML AND XML.SAX
Access elements in one time	The DOM is extremely useful for random-access applications	<ol style="list-style-type: none"> <li>1. SAX only allows you a view of one bit of the document at a time.</li> <li>2. If you are looking at one SAX/html element, you have no access to another.</li> <li>3. If you are looking at a text node, you have no access to a containing element</li> </ol>
applications	Dom provide most of the methods for you, you can just use them and create you application.	When you write a SAX application, you need to keep track of your program's position in the document somewhere in your own code. SAX does not do it for you .

Table 5.1 Comparison of two methods

It's obviously to see the different between Dom and SAX. Dom are more powerful when you search more then one element, and easy to deal with the element with given methods.

## 5.2 Differences in output

### In testing 1:

Mozilla find an img element in the web site. Python (html) find nothing. We can check the original source code. We find the image are contain in the JavaScript.

### In testing 2:

Mozilla can present the JavaScript, no words follow the mouse. Does it mean this is a weakness of Mozilla. No, the executions in mozilla and traditional parser (IE) are different. Check the source

codes you would find it.

### In testing 3:

Mozilla has method to present the blink elements. Traditional parser does nothing with blink element; just treat it as a unknown elements.

	Mozilla	Traditional parser
Identify JavaScript	Can identify imaged with JavaScript	Cannot identify JavaScript
speed	fast	slow
Element domain	More element can found, provide the methods to deal with them. E.g.: blink	Some element are treat as unknown tag Blink can't
javascript	The way treat the javascript, somewhere are different, you can see the test 2.	

The xulrunner contain the mozilla technologies, you can use those method to deal with DOM tree.

Compare with the previous solutions, It deals with all kinds of situation, either html or xml. It's the best way to solve the problem. Furthermore, Mozilla has many other methods to deal with DOM tree , making your application powerful.

## 6. Conclusion

Now we finish the application of finding the number of specific element. We already know the concept of a web browser based measurement component using Mozilla [Moz].

Present the process what the browser have done.

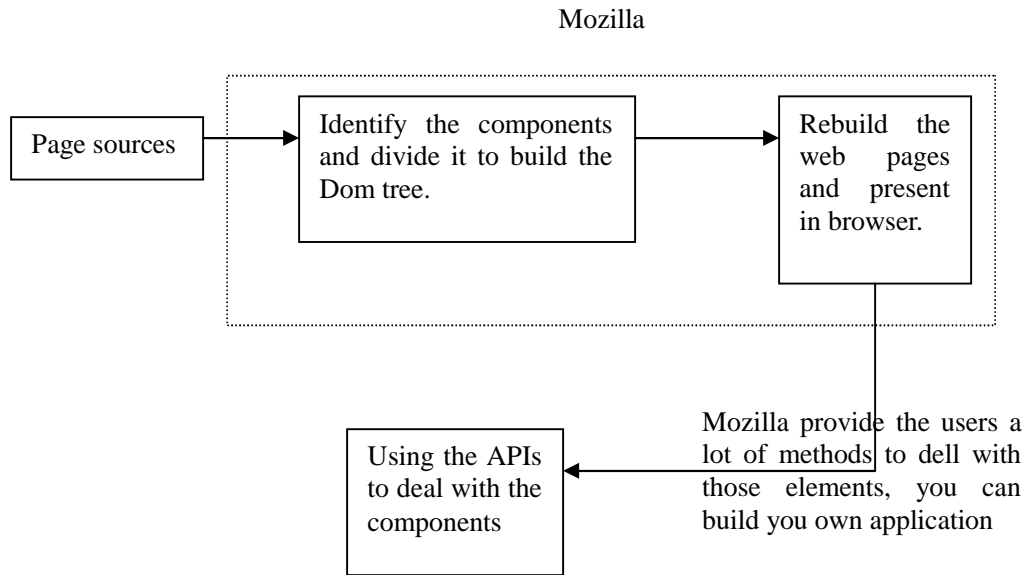


Figure 6.1 Process of the browser

The mozilla has collect those measurable components and build the Dom tree. The users can use the API interface which mozilla has provided you, to get what elements they are interesting.

## **A. Appendix**

### **A.1 Glossary and Abbreviations**

#### **A.1.1 EIAO**

European internet accessibility observatory

#### **A.1.2 W3C**

World Wide Web consortium

#### **A.1.3 URL**

A Uniform Resource Locator, URL, or Web address is a standardized address name layout for resources (such as documents or images) on the Internet (or elsewhere).

#### **A.1.4 MSAA**

Microsoft Active Accessibility (MSAA) API

#### **A.1.5 web page**

A Web page or webpage is a resource on the World Wide Web, usually in HTML/XHTML format and with links to enable navigation from one page or section to another. Web pages often use associated graphics files to provide illustration, and these too can be clickable links. A web page is displayed using a web browser. [8]

#### **A.1.6 XUL**

XML-based user interface language

#### **A.1.7 XML**

Extensible Markup Language

## A.2 Reference:

- [1] W3C- World Wide Web Consortium:  
<http://w3.org/>
- [2] The Joy of XUL  
[http://developer.mozilla.org/en/docs/The\\_Joy\\_of\\_XUL](http://developer.mozilla.org/en/docs/The_Joy_of_XUL)
- [3] xul  
<http://developer.mozilla.org/en/docs/XULRunner>
- [4] dom document object model  
<http://www.w3.org/DOM/>
- [5] xml  
[file:///D:/diveintopython-html-5.4/diveintopython-5.4/html/xml\\_processing/index.html#kgp.divein](file:///D:/diveintopython-html-5.4/diveintopython-5.4/html/xml_processing/index.html#kgp.divein)
- [6] global models  
<http://docs.python.org/lib/module-xml.dom.html>
- [7] xulrunner tutorial  
May 06, 2005 Ryan'work blog
- [8] <http://www.mozilla.org/access/architecture>
- [9] Programming web services with SOAP.  
James Snell, Doug Tidwell, pavel Kulchenko.
- [10] Python cookbook .  
Alex Martelli, Anna Martelli Ravenscroft and David Ascher.
- [11] XUL Tutorial  
<http://www.xulplanet.com/tutorials/>
- [12] Dive into python      Mark Pilgrim    20 May 2004

<http://diveintopython.org/>

[13] A Byte of Python 2003-2005 Swaroop C H

[14] limodou's note of xul

<http://blog.donews.com/limodou/category/35193.aspx>

[15] reading notes of program python 26.5.2004

<http://man.chinaunix.net>

[16] XUL Element Reference

<http://www.xulplanet.com/references/elemref/>

[17] WAI and WCAG Reference

<http://en.wikipedia.org/wiki/WAI>

[18] Mozilla Reference

<http://en.wikipedia.org/Mozilla>

## Project progress

Task	Start	Finish	Deadline	Status
Select project	24th Aug			ok
Search and Learn	8th Sep	15th Sep	15th Sep	ok
Agenda	16th Sep	17th Sep	18th Sep	ok
Problem description	18th Sep	18th Sep	18th Sep	ok
Get familiar with programming environment	19th Sep			ok
First report	21th Oct	23th Oct	24th Oct	ok
First presentation	26th Oct	26th Oct	26th Oct	ok
Application running	28th Oct	8th Nov	10th Nov	ok
Final report	15th Nov	22nd Nov	26th Nov	Started
Final presentation	30th Nov	30th Nov	30th Nov	Preparing