Web Personalization Design and Implementation of a User Assist System for Mail Composition

Vidyuth Dandu and Mangesh Bedekar

***Abstract*—Web usage assistance through client side computing has become a mandatory feature in new extensible browsers with decrease in hardware prices and increase in computation power on client side. Utilizing client resources to aid client is the aspect in which this paper deals with. This paper presents design of User Assist System for Mail Composition (UAMC), a tool which helps users to send everyday emails quickly and efficiently.**

***Index Terms*—JavaScript, PHP, model view controller, Mac Apache MySql PHP (MAMP), grease monkey.**

I. Introduction

WWW (World Wide Web) has become the most important place for all network users, either connected by a personal computer, laptop or a personal device. Web has changed from a static information guide to a collaborative framework where users from novice to professional use it without needing much assistance or training unlike old days. Browsers became a medium to run applications which replaced the static web pages. With client side scripting being rampant, the pages are designed to respond to users' needs and alert based on users preferences. The same principle of recommend-ing the user based on his/her preferences while composing mail to finish it quickly and efficiently is discussed in this paper. Today, one of the main applications of internet is for transferring messages. Several mail applications help people to send messages to others.

This paper presents the design of a user assist system for mail composition (UAMC), which is a recommender system that helps users while typing emails for quick and concise emails. The Mail Clients these days are free and offer ad- ditional flexibilities such as maintaining friends list, mail list, trash, filtering spam, etc. All these capabilities of mail increased its usability and reduced the effort in organizing and sending the mail. But till now, no mail client offers a clean solution for sending a mail quickly without much tedious work of typing, organizing content and checking especially for mails which have almost have the same content but needed to be changed a little and to be send to a person based on the context. Consider a scenario where a mail needs to be sent from person A to person B. A is not proficient in english and hence types his mail in his own language (say A's mother tongue) and assume that B is acquaint with A's mother tongue. So he needs to type the entire matter and then send it. Say A wants to send the same mail to B again but with some changes in it, since its mother tongue, copy pasting the previous mail may doesn'tnecessarily work. What if, while A is typing the message, he can get the entire word or sentence to autocomplete on key press of just his first few letters, and also show other options like recommending the user about his/her most used words/sentences for particular context so that the user can either choose a word or sentence to autocomplete. Observe here that the user need not type the full sentence saving him the manual or laborious work. If this process is applied to a user who is typing a mail, he can finish a normal mandatory mail in less than a minute with the help of the recommender system. This helps the users to not only quickly complete a mail but also send it in an efficient and customized way. There are other simple mail applications[1] like force mail to use secure connection i.e https instead of http, alert while replying, these are applications without any extra components that can be written using Grease Monkey[2]. UAMC also uses Grease Monkey [3] along with other components as discussed in the later part of the paper.

The above example gives the gist of the system. However the conceptual thinking and implementation part of the system is dealt in next few sections. The paper is organized in the following manner, Section II i.e next part deals with the methodological framework of the system, Section III shows the algorithm used, Section IV deals with the design and implementation of the system which briefly describes each component, Section V speaks about the Design Considera- tions. Then the paper is concluded in Section VI with some remarks on the system along with future research needed to develop this system into a more reliable, efficient system.

II. Methodological Framework

This part of the paper presents the background of the application. The basic idea for this application is that it should minimize human effort in composing mails which have repeated words or sentences in them. Typing a mail all over again is a tedious job. For ex: In a company mail, if say a person always starts with "Good Day to you", then for every mail, he needs to type the sentence. With the help of system which processes the user's pattern we can provide the user a direct solution rather than the waste time typing the mail. Another simple example in this case would be, in GMail application if we want to send a mail to someone and type first few letters of their name in the 'To' box, it autocompletes the mail address for us, so this concept is extended for the'Content' area of the mail. UAMC considerably saves users time by capturing user

Manuscript received November 17, 2011; revised December 20, 2011.

The authors are with the Birla Institute of Science and Technology, Pilani, India. (e-mail: mangesh.bedekar@gmail.com).

intent from time to time and shows it when he writes an email. The entire process is captured in a flow chart as shown below in fig 2.

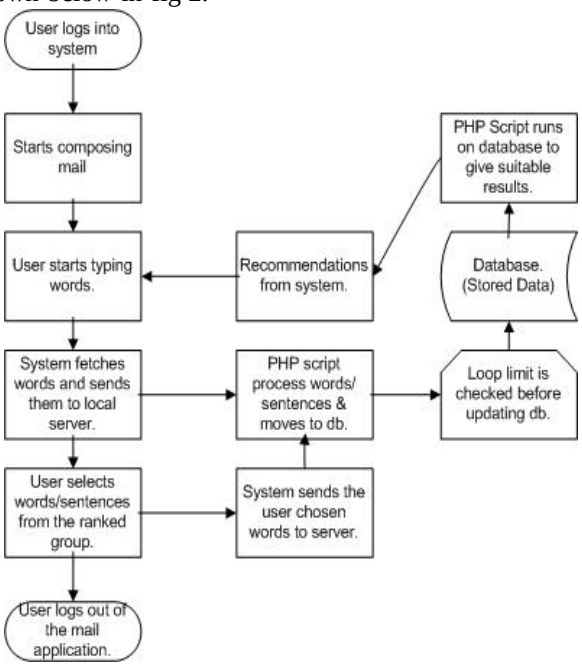

Fig. 1. Flow chart.

The Compose page of the Mail Application has 4 parts, 'To', 'CC', 'Subject' and 'Message' fields. As we mentioned UAMC is a recommender system which operates in Message field. The 'To' field as discussed above already has autocom- plete feature provided by the Mail Application in the form of XMLHTTP requests, as is 'CC' field. The subject field is mainly context based and hence even though recommendations be provided, would be of less use than when it's used in the message field. Hence the UAMC system mainly concentrates on the 'Message' field as its main field and leaves other fields. Message field of the mail is essentially important part of a mail and typing long mails essentially is a time consuming and complicated task.

The usage of the application with respect to a user can be categorized into 3 phases as depicted below.

A. Learning Phase

As the name suggests, this phase learns from the user as he starts using the application. The JavaScript code is responsible for this. It fetches data from the user while he is composing a mail and sends the words to a local server running on the client system. The local server is the MAMP(Mac, Apache, Mysql and PHP)[4] server which runs on the client machine.

B. Processing Phase

This phase, as mentioned earlier and by name is responsible for collecting the data, parsing it. The words are sent by the JavaScript code. The words here are checked for only stop words and also other words of local language, his/her common typing habits. A future inclusion as mentioned later would be to ask user for his options (customize) on the words of inclusion/exclusion. The PHP script also refers to the database for already included words and increment their count. The incrementing is a part of business as it literally gives the idea of most used/unused words. The aspects of the algorithm will be discussed later.

C. Recommendation Phase

Data is sent back after being processed from the script running on the script. The words/sentences sent are generally based on ranking system adopted in the script and the most suitable word for the context. The user can now decide the various options shown for autocompletion. All this is done when the user types first few letters of the word. Depending on the context not only word, but also sentences can be suggested by the system.

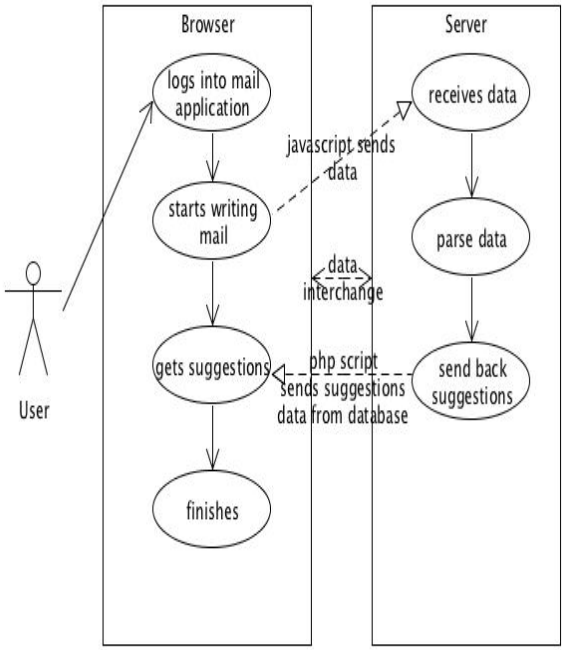

Fig. 2. Use case diagram.

The entire scenario of this application as discussed is depicted in the fig.2 Use Case diagram. The system is essentially a combination of user-side JavaScript code and a php script running on the server (with the database) also at the user side. As do all JavaScript's this acts acts on the webpages used by the user, but as the target of the system is to help user in composing mails. This system as of now isrestricted to run only on mail applications web pages. Briefly the scenario, Once the user logs into mail account and starts composing mail, the JavaScript runs on the web page and it transfers the content of the mail(whatever, user is typing) to the local server which is password protected by the user itself. So there is a decreased chance of stealing important words, also all the words which go into the server are mostly stop words and user habituated words. Important words in mails like passwords or numbers are filtered about by the php script before sending them to the database for storage. So there is no privacy concerns involved here. Once the data reaches the server, it is processed their by the php script, this script is responsible for not only filtering input data and storing in database but also this script offers additional capabilities such as storing user preferred words/sentences as permanent, removing some unused sentences from the database by setting them appropriate decay time, thus it basically monitors the pattern of the inbound sentences to

efficiently manage the huge incoming data. The database attributes as of now are the word field and count field. So as to know the most used words/sentences which helps us to rank them when recommending to the user. So the user has the option of selecting among those top 10 ranked words/sentences. Once the user selects them, the script would update the sentence rank. So it can be viewed as kind of self referential. The future section would define some more capabilities which will make the system more advanced. It should be remembered that there is a clear possibility of saturation of the inbound data i.e this is where the incoming data becomes redundant and hence proper management should be done by the script to reduce inconsistencies in the data. So all the main or core processing part is done by the php script sitting on the server of user. The browser script i.e JavaScript is responsible for sending and receiving the data, it also provides users various options of say, prioritize some sentences/words, remove some sentences/words, it indirectly gives an interface to manage the data on the server side, so the JavaScript acts as a controller between the webpage and the server.

After all this, it should be remembered that database is an open ground here, it is open to play on it, i.e it's continuously gets updated as the user composes mails. Therefore UAMC is a robust system with a robust architecture (MVC) and script. The next part of the paper presents with the design and implementation part of the system (UAMC). Section IV deals with how the system is organized in a component view.

III. Algorithm

This section presents with the algorithm which is currently in use to capture user input from the textarea in the mail webpage. The algorithm makes use of the Grease Monkey API's (Application Programming Interface) [5]. One of the main method/API used is the GM xmlhttpRequest[6] is widely used. It also shows some snapshots of that implementation. Algorithm is given below MailAssistSystem(WebPage):

for TextArea in the webpage: do commited__value = null initial__text = textcontent
while text present commit(initial_text)
do execute commit(value) execute commit(value) on keypress. buffer = null
if textarea loses focus
do buffer = textcontent commit(buffer)
if form is submitted
do commited_value = null commit(commited__value)
commit(value)
if value is significant
call XMLHTTP_POST_REQUEST(value)

This is a primary algorithm on the webpage (i.e mail application). The algorithm starts by accessing the textarea of the webpage and its contents, the text content present in it is committed using the commit function, initial content is committed and the textarea element in the webpage has an 'onkeypress' event handler associated with it, values are committed on keypress, also we save the data in the form of buffer when the page loses its focus just to be on the safe side. The commit function checks whether the text value is significant or not i.e it's a string and not just a whitespace. This particular algorithm focuses only on sending the text content in the textarea to the server where they will be parsed by another algorithm using PHP which not only stores the words into the database and updates their records, but also sends back replies to user as he needs it. The snapshots are shown below.

Screenshot 1 is taken for a textarea on a site pastebin.com, this code works on any textarea and is restricted to mail appilcation as of now. So once the text is typed in the textarea, it is continuously saved while typing, along with the frequency count of it i.e the number of times its being used in the textarea. The JavaScript is responsible for taking the words splitting them and also putting the sentence into the database and once it reaches the database we use PHP to retrieve it using simple queries when needed. Screenshot 2 shows these words and frequency attributes stored in database. Screenshot 3 shows the system implemented. When the user types 'hel' he gets suggestions when he uses right click(i.e context menu) and choosing any option automatically fills the field. Two types of suggestions are given as seen in the above menu, one would be context related and other is words/sentences. Context related filling is by taking the current page into consideration. The above mail had the 'To' field as 'vidyuthd@gmail.com', so the suggestions in the first group would be considering the To field into account. For example, if the user starts with'Dea' then the suggestions would be in the form of 'Dear

Vidyuth', 'Dear Vidyut', etc. Screenshot 4 shows the options available so far for the word 'you' and so shows the word your and by selecting it the user can fill it. And also note that till now no sentence has been completed with the word.

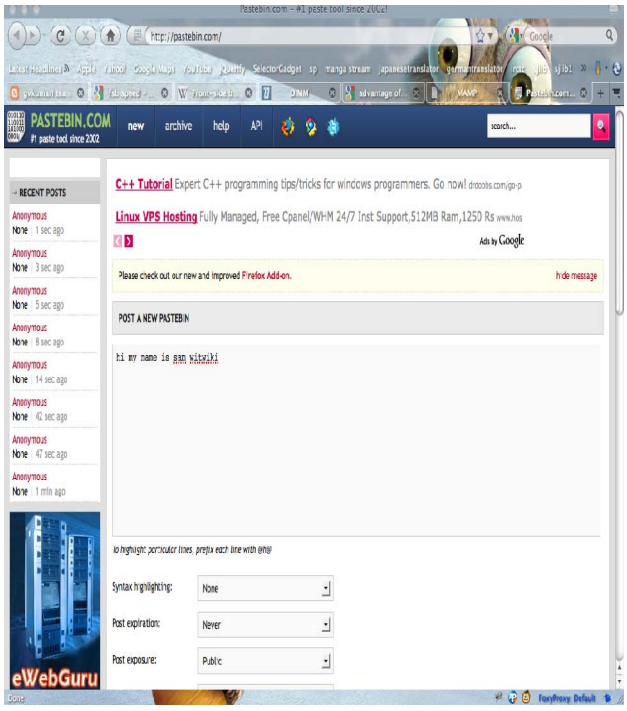

Fig. 3. Screenshot 1.

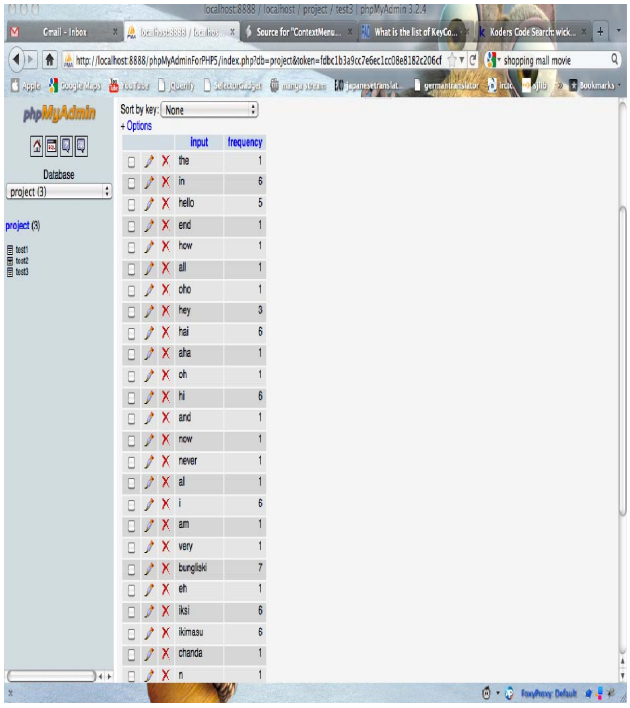

Fig. 4. Screenshot 2.

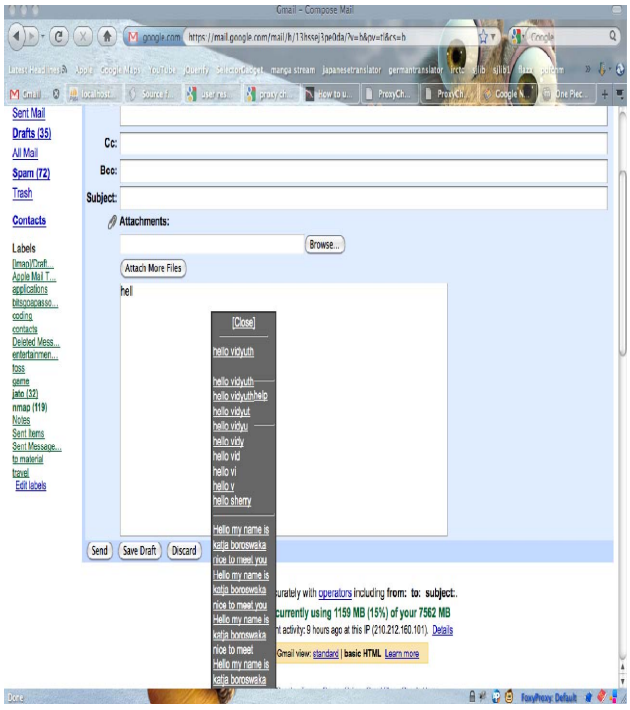

Fig. 5. Screenshot 3.

This is supported in Screenshot 5 shown below Screenshot 4. The other two individual Grease Monkey Scripts, Text area Backup [7] which can be used to save the text typed in any text area on a webpage and Context Menu [8] which can highlight a word—phrase and right click to get a context menu full of search links for that phrase were taken as the starting points for this particular application.

IV. Design And Implementation

The system implemented so far is a proof of concept with JavaScript is responsible for sending the data to the server. The server being used is MAMP (Mac, Apache, MySql and PHP) [9].

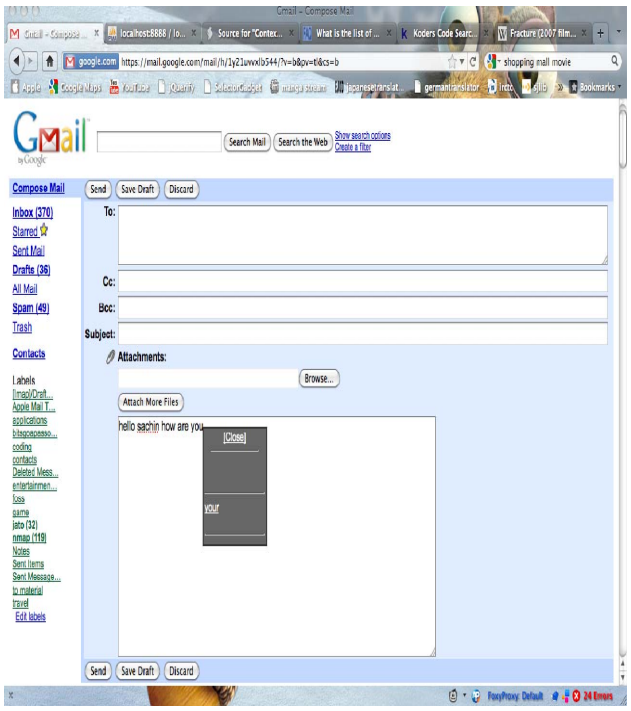

Fig. 6. Screenshot 4.

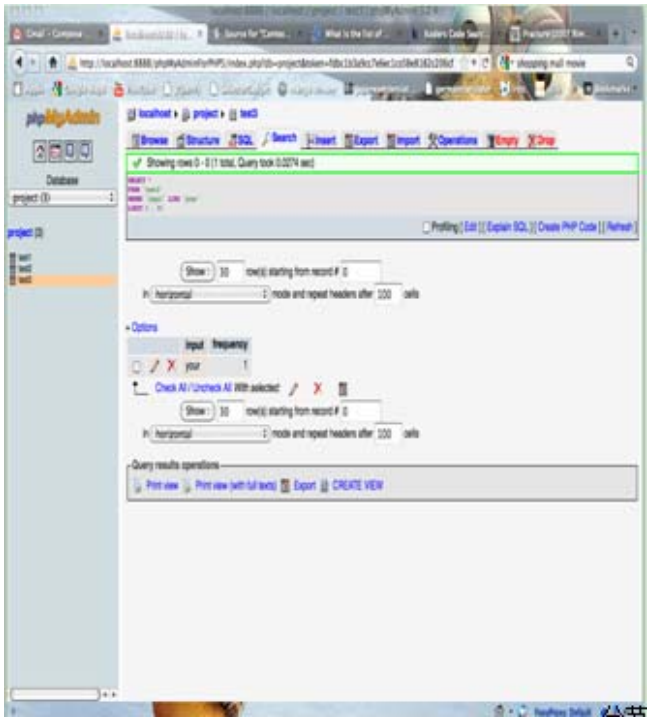

Fig. 7. Screenshot 5.

This server enables us to run the PHP running script on MySql server, which processes the data and is the core component of the system. The main components of the system are the Browser and the Server. The architecture of the system is similar to MVC (Model View Controller) architecture pattern. Here the php script plus database acts as the model, the browser window as the view and the JavaScript as the intermediate controller. The components are briefly described below.

1) MAMP Server:
The server acts as a repository which stores all data regarding user. The php script on the server parses the data and stores it in the database. MAMP (Mac, Apache, MySql and PHP) server which sits on the client is responsible for data storage, move- ment and its management. This server is chosen because it's free and source can be modified to make it a compact server which can sit on user's machines. Most impor- tantly, the server is reliable, compatible with any system through other forms such as WAMP (for Windows) and LAMP (for Linux). Most important thing to notice here is that the server here acts as a medium for the script to be run. The algorithm in the script is essentially the crux or the core of the system.
2) Browser Web Page
This can be seen as the presenta- tion/view part of the system on which the user accesses his mail application. The Browser runs JavaScript with its JavaScript engine.
3) Client-Side JavaScript
This component is responsible for collecting data from the users and sending it to the local server for processing the data, this is done by using JavaScript Event Handlers [10]. It also receives data from the server to show the user some recommendations. The function of this component is to communicate with the server bidirectionally i.e sends and receives data and also show user his/her additional functionalities such as options of deleting certain words/sentences from database, set some of words/sentences as permanent in database,etc. The implemented version is a proof of concept version where JavaScript is used to send user typed data to the server and server stores it. This was implemented using Grease Monkey Addon of Firefox for faster testing purposes ignoring the nuances when writing an addon. Given below is the code

V. Design Considerations

The section adds some more insight into the making of the system, it elaborates on the decisions chosen for the design

1) Criteria for Choosing Environment:
The MAMP server acts as a perfect choice, as its variant LAMP(for Linux) and WAMP(for Windows) are already available. Also languages such as ASP was not used because it is properiatary, and since we needed to processes the text in a script, PHP would perfectly fit the bill, as HTML is generally for rendering static pages, although with HTML 5 this has changed. We persist with PHP because it is open source, flexible nature. In future when we open the code of this to masses they can change it because of the PHP's license which cannot be done in the same way with ASP.
Efforts are being made to develop the next version of this extension leveraging the power of HTML 5 included database API's (Application Programming Interface) to store the data in local/web storage database which may help us in decreasing the number of calls to the db's which will be a major concern when it is deployed in common for mass users. Such design would indeed help us in increasing the performance and thereby usability of this extension.
2) Security Considerations:
Security and Privacy are the two major concerns that still need to be addressed properly by formatting some rules, as of now they are no rules regarding this, but there are ideas, as this extension goes in for production it would we be implementing these ideas. Some of them are
a)Listing out some sensitive(generic) words and fil- tering them, so that they are not included in the database. For ex: 'Ferrari' is one of the words which are commonly used for passwords, so we check for such occurrences prefixed or suffixed with and omit them while collecting data. This is one of the ideas, this is tedious but it ensures to cut some volume of words.
b) Giving the user the option to turn off to include a particular word into the data collection, this can be implemented easily and hence makes the user responsible in choosing the words he does not want to be stored.
c) Assigning a stale date for each particular word and asking the user to choose the time, after which that particular word doesn't appear in the database if he doesn't use it again. When a word is given this option, it will auto deleted as a part of maintanence check scripts.

The most important thing to remember is that this system is on the client side and hence no issues of sensitive user data being sent across to other domains or to some central server. This are some of the ideas, once after formulating a solid number of rules with people agreeing upon, surely this extension can go into production environment. The aim of this extension is to give the user as much freedom as he wants and hence the rules are designed taking care of that point.

VI. Conclusion And Future Research

This paper mainly focuses on the implementation part of the user assist system for mail composition and its design, this system has a lot of scope for improving as said in the paper. The implemented part so far is a small proof of concept, we plan to include various user related attributes and implement a good algorithm for the php script to manage the data effectively. The other aspects for future consideration are user chosen filter system, ranking system, decaying of usage of words with time. All this aspects have to be taken care of for writing the script (PHP) for producing an advance system which is capable of adapting to user.

References

[1] M. Pilgrim, Grease Monkey Hacks, 1st ed, O'Reilly, Nov 2005, ch 7
[2] M. Pilgrim, *Dive into Grease Monkey*. [Online]. pp.2-3. Available: http://igor.chudov.com/manuals/diveintogreasemonkey-2005-05-09/diveintogreasemonkey.pdf

[3] M. Pilgrim, *Dive into Grease Monkey*, [Online]. pp.9-16. Available: http://igor.chudov.com/manuals/diveintogreasemonkey-2005-05-09/diveintogreasemonkey.pdf
[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern Oriented Software Architecture*, John Wiley and Sons, 1996, pp. 125-144
[5] M. Pilgrim, *Dive into Grease Monkey*. [Online]. pp.75-80. Available: http://igor.chudov.com/manuals/diveintogreasemonkey-2005-05-09/diveintogreasemonkey.pdf
[6] M. Pilgrim, *Dive into Grease Monkey*. [Online]. pp.77-78. Available: http://igor.chudov.com/manuals/diveintogreasemonkey-2005-05-09/diveintogreasemonkey.pdf
[7] Textarea Backup Grease, "Monkey user script for saving text in text area," [Online]. Available: http://userscripts.org/scripts/show/7671
[8] Context Menu Grease Monkey user script, Highlight a word—phrase and right click to get a context menu full of search links for that phrase. [Online]. Available: http://userscripts.org/scripts/show/4912
[9] Setting up Mac Apache Mysql PHP (MAMP) on Mac. [Online]. Available: http://documentation.mamp.info/en/mamp
[10] J. E Resig, Pro JavaScript Techniques, Apress Pro, 2006, pp. 325-340