CBS COPENHAGEN BUSINESS SCHOOL
HANDELSHØJSKOLEN

Documentation of the First CASMACAT Prototype

Bonk, Ragnar

Document Version
Final published version

Publication date:
2012

License
CC BY-NC-ND

Citation for published version (APA):
Bonk, R. (2012). *Documentation of the First CASMACAT Prototype*. CASMACAT. CASMACAT Project Deliverables No. D5.2

Link to publication in CBS Research Portal

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
If you believe that this document breaches copyright please contact us (research.lib@cbs.dk) providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 04. Jul. 2025

D5.2: Documentation of the first CASMACAT prototype

Ragnar Bonk

Distribution: Public

CASMACAT
Cognitive Analysis and Statistical Methods
for Advanced Computer Aided Translation

ICT Project 287576 Deliverable D5.2

Project funded by the European Community
under the Seventh Framework Programme for
Research and Technological Development.

Project ref no.	ICT-287576
Project acronym	CASMACAT
Project full title	Cognitive Analysis and Statistical Methods for Advanced Computer Aided Translation
Instrument	STREP
Thematic Priority	ICT-2011.4.2 Language Technologies
Start date / duration	01 November 2011 / 36 Months

Distribution	Public
Contractual date of delivery	April 30, 2012
Actual date of dlivery	April 30, 2012
Date of last update	April 30, 2012
Deliverable number	D5.2
Deliverable title	Documentation of the first CASMACAT prototype
Type	Report
Status & version	Draft
Number of pages	14
Contributing WP(s)	WP5
WP / Task responsible	UEDIN, CBS, UPV
Other contributors	
Internal reviewer	Philipp Koehn
Author(s)	Ragnar Bonk
EC project officer	Aleksandra Wesolowska
Keywords	

The partners in CASMACAT are:

University of Edinburgh (UEDIN)
Copenhagen Business School (CBS)
Universitat Politècnica de València (UPVLC)
Celer Soluciones (CS)

For copies of reports, updates on project activities and other CASMACAT related information, contact:

The CASMACAT Project Co-ordinator
Philipp Koehn, University of Edinburgh
10 Crichton Street, Edinburgh, EH8 9AB, United Kingdom
pkoehn@inf.ed.ac.uk
Phone +44 (131) 650-8287 - Fax +44 (131) 650-6626

Copies of reports and other material can also be accessed via the project's homepage: http://www.casmacat.eu/

© 2012, The Individual Authors
No part of this document may be reproduced or transmitted in any form, or by any means, electronic or mechanical, including photocopy, recording, or any information storage and retrieval system, without permission from the copyright owner.

Executive Summary

This document contains details about the implementation of the first prototype of the CASMACAT workbench. It outlines the major components and their usage.

Contents

1 Introduction

The implementation languages for the CASMACAT editor are determined in deliverable D5.1 [5]. Both, JavaScript and PHP, are script languages and have some pros and cons. First, structuring the code and applying common object-orientated programming (OOP) design patterns is more difficult than, for instance, with Java. Even though JavaScript and PHP support OOP, this is very different or seems incomplete compared to Java. For instance, PHP does not support constructor overloading. If this is needed (and it is used quite often in OOP), it must be implemented manually. JavaScript on the other hand does not support protected fields nor abstract classes or interfaces. Both use dynamic typing, with the drawback that data type errors can only be recognized at runtime. Nevertheless, JavaScript and PHP also have advantages: Prototyping can be done very well with both languages. It is fast and easy to develop small and functional programs for testing and investigations without caring so much about structured code. JavaScript (together with the browser extension) looks like the most suitable solution for CASMACAT.

This deliverable describes the most important details of the CASMACAT prototype that has been developed in the past 6 months. As development is still ongoing, the information provided here may be outdated at some points, but the basic concepts will still apply. For a detailed background and conceptual information refer to [5] and [1].

2 Basic Structure

This section describes the basic elements, how they have been wired together and which structuring strategy has been applied for the CASMACAT prototype.

Basically, the prototype is a collection of the following items:

XHTML Templates These are files which contain templates for the concrete web-pages that are shown in the browser. The templates determine the structure and functional components of the web pages. For instance, they specify that a page will have a menubar, a table containing place holders for data and a button. Templates are not shown directly when requested but are first processed by the template engine on the server (PHPTAL in this case [4]) which replaces the placeholders through concrete data or other objects (e.g. different menu entries depending on the user logged in). (X)HTML only specifies that there *is* a button on the page, but not what this button *does*.

Cascading Style Sheets (CSS) This is the state-of-the-art technique to control the layout of web-pages. It is a very flexible technique and allows for much more control than classic HTML does.

JavaScript, Inline Or External JavaScript is used to implement the client-side program logic of the prototype. It may define more or less complex functions and runs inside the browser. JavaScript can be "inline" which means that it is part of a HTML file or it can be placed in a separate external file which is referenced in the HTML file.

PHP Files Those files contain functions which are executed on the server. The execution of such a function is always triggered through a request from the client. This request may either come directly from the browser (e.g. refreshing the page) or through a JavaScript call. The life-cycle of a PHP script is request-response-based, just like the HTTP protocol is. This means that the execution of a PHP script starts with the request and ends when it sends the response back to the client. It *cannot* live longer and the response is also only finished and completely sent when the PHP script ends.

Images Images are very rarely used in the CASMACAT prototype (only two logos). There is currently no need to have images treated in a special way and will not be discussed here.

The prototype consists of one XHTML template per use case, as defined in deliverable D5.1 (except for authentication). Each of those templates is assigned to one PHP file with the same prefix that is responsible for loading and processing this template (replacing the place holders) and delivering the result to the client. For instance, for the "Upload Document" use case there exists a pair of a template file "upload.xhtml" and the PHP script "upload.php". Templates reside in the folder "templates".

If the page generated by the template needs some JavaScript then this is included inline, if the JavaScript is simple and only contains functionality used on this page. If the functions defined in the JavaScript get more complex or they are reused on other pages, then it is realized as an external file. JavaScript files are located in the folder "javascript".

CSS files are placed in the "styles" folder and will become more important in the further development. Currently, only two basic styles exist for the CASMACAT prototype. In a later stage, several different layouts may be designed and tested by simply writing a new CSS file or through a special editor which will enable basic manipulation of CSS values.

One template is always assigned to one PHP file which does the template processing. But depending on the functionality of the generated page, several other PHP scripts may also belong to one template. For instance, the template "translate.xhtml" is processed by the PHP script "translate.php", but when the user interacts with the page, several other scripts are called, such as "update_translation.php" or the "scroll_translation.php". The organisation of the PHP files follows the same pattern as the one for the JavaScript: A script usually only contains the code for its specific task and the reusable code is placed in a separate file. PHP files are placed in the root folder.

There also exists a folder named "include" which contains all external libraries for PHP. Currently, this is only the already mentioned template engine PHPTAL.

3 Default Access

The CASMACAT prototype is a multi-user system that allows two basic access modes: "default user" (user) and "administrative user" (admin). If the prototype is accessed by a default user it offers basic implementations of the "Upload Document" and the "Edit Translation" use cases (see deliverable D5.1). Both implementations will now be described.

3.1 Upload Document

Figure 1 shows the UI for uploading a document. The plotted web-page has been generated by the "upload.php" script using the template engine which populated the "upload.xhtml" (see section 2). In addition to the specification of the minimal prototype this implementation also supports uploading of simple plain-text files. The document or the input text may contain HTML formatting tags. Currently, all HTML tags that may occur inside `<div>` and `<span>` elements can be used. When sumbitting a text or a document, the "upload.php" is called again. In the case of the text this is a simple Ajax call. When submitting a file, this works differently because the default file upload function of HTML always causes a page refresh. To simulate the same behaviour in both instances, a call of a hidden `<iframe>` element is used to which the POST request is redirected. While the PHP script processes the input, a modal dialog informs the user that the upload is running. The "upload.php" script implements some very basic functions of the document management server (see deliverable D5.1): The input text is

segmented by splitting it at the occurrence of dots. The so collected segments are then submitted one by one to an instance of the MT server Moses to generate the initial translations. The calls to Moses are done via XML-RPC as specified in the API in deliverable 5.1. Thereafter the document, the segments and the initial translations are stored in the database and the document is automatically assigned to the user. Finally, the user is notified about the uploading success.

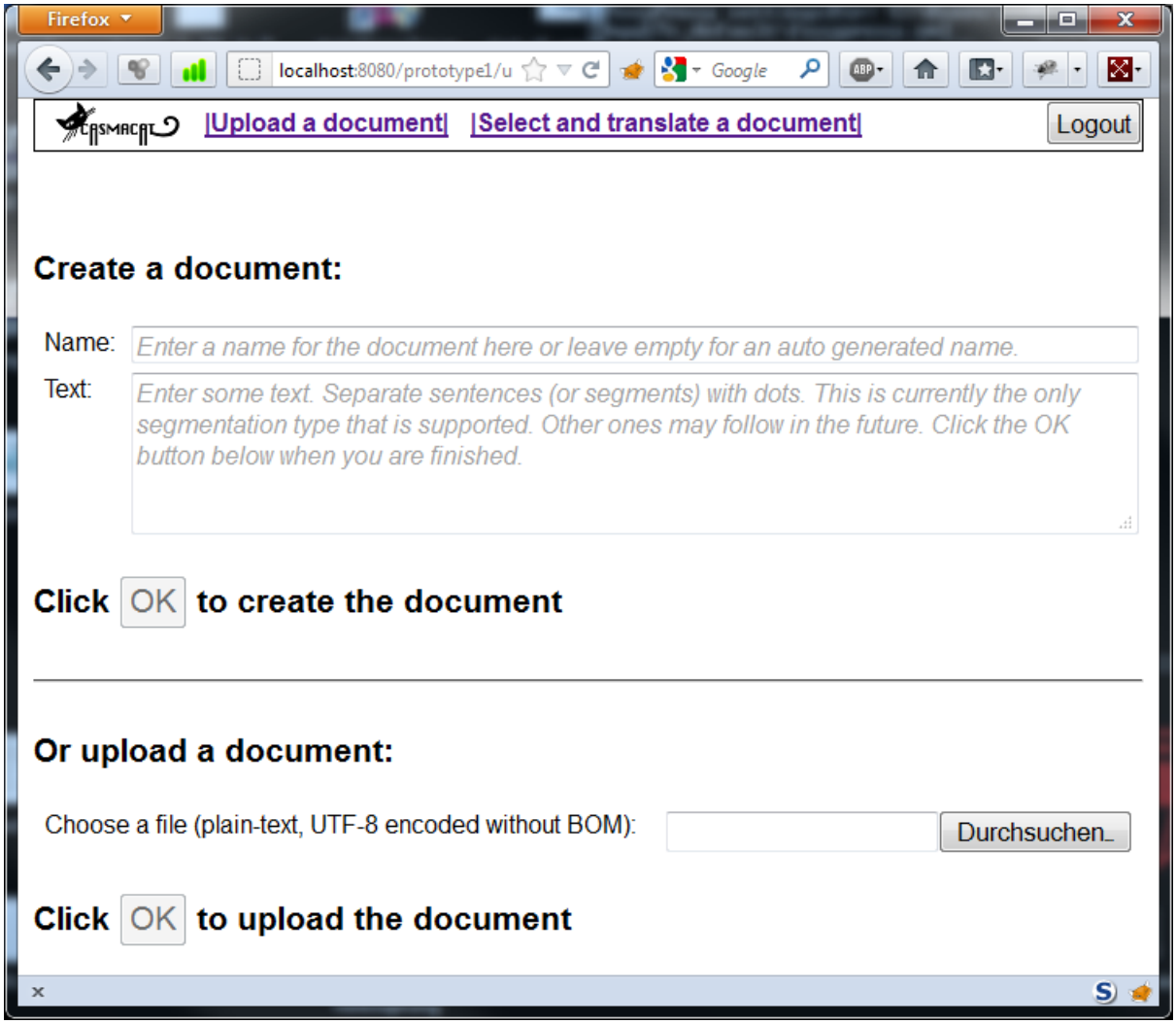

Figure 1: UI for the creation of documents

The current prototype is still missing the possibility to chose the MT system and, consequently, the source and the target language. An appropriate mechanism has already been prepared but at the time of writing there was still an ongoing discussion on how to do this in the best way and which partner of CASMACAT is responsible for hosting the MT instances.It is anticipated that a simple dropdown menu will be added which will be populated with the language models available at CBS.

3.2 Translate Document

This section deals with the implementation of the "Edit Translation" use case. This is the most important part of the CASMACAT prototype and many different approaches have been discussed, developed and tried out. The implementation presented here was used to test the basic concepts for logging with or without Eye Tracker (ET) and it could be tested whether the system architecture fits the specification needs, especially with respect to integrating ET, and also whether it is ready for the first field tests that will be carried out in month eight of the

CASMACAT project by Celer Solutions in Spain. Even though no working logging functionality has been required for those tests, the prototype already allows a basic logging functionality.

The idea for the UI design is to plot a fragment of the source and target document, containing the segment that is currently being edited, as well as past and future segments, just like a window that is moving over the document. Figure 2 shows a document, the window over it and the respective document fragment. The number of segments that a fragment contains is currently limited to seven but this will be made configurable in the future. As shown in Figure 3, a document fragment consists of the following main components:

Source View This area shows the source segments of the current document fragment (left side in Figure 3). The first three segments are already translated source text. The fourth segment is the current source segment. It is marked with a different background colour to improve the readability. It is important, that the highlighting is done by only changing the background colour but not applying a layout that will modify the display size of the segment (e.g. making it bold). This is to avoid flickering to get more reliable ET data. The fifth to the seventh segments contain the next following segments that need to be translated.

Past Target View This area shows the target segments that have already been translated. Of course, when the document is initially loaded, this area is completely empty.

Target Present Edit In this area the currently edited segment is displayed. It has a bigger font to improve the readability. If initially loaded, this field may either be empty or may contain an initial MT translation or the translation the user entered in a preceding session. The content of this area depends on the activated options when uploading the document. Currently, always initial MT translations are generated, but there will later be an option to change this behaviour. Saving of the content of this field is activated 1) if the user scrolls to the next or a previous segment, 2) if the focus leaves the field, 3) if the user clicks a save button in the menu or 4) when the window is closed. The saved content of this edited segment is considered to be the verified translation for this segment.

Future Target View This is a preview of the next target segments that need to be post-edited. Of course, the field will only contain text if either initial MT translations have been generated (see above) or if the user already entered a translation for those segments.

Feature area Here, the other features can be displayed, like alternative MT or TM output and terminology.

In figure 3 shows the interface with the elements just discussed. The red bar in the bottom right corner was for debugging purposes only and will be removed when field tests are started.

This interfaces, like all others, basically consists of a template and an associated PHP script, namely the "translate.xhtml" and "translate.php". As the functionality is more sophisticated than in other interfaces, several other scripts are involved.

On the server side, the two scripts "scroll_translation.php" and "update_translation.php" are also called. The first script is called when the document fragment is scrolled. This means the user moves to the next or the previous segment by pressing ALT+UP or ALT+DOWN. The second script is executed when a translation needs to be saved (see the description of "target present edit" above). On the client side, two additional JavaScript files are used, the "logging.js" and the "logging.keylog.js". The former holds all functionalities and variables that are common for logging like the loglist containing the events logged and functions to start and stop the logging process. Also, all event handler functions that are not related to key logging or capturing ET data are placed here (e.g. mouse moving). The functionality for the key logging is located in the script file "logging.keylog.js". This is due to the reason that key logging is more complicated and separation into a proper file seems to be a good choice. Everything needed for capturing ET is located in the browser extension (see section 5).

4 Administrative Access

This section discusses the control options of the "administrative user" (admin). Those options include more than initially specified (see deliverable D5.1), as new information and ideas evolved. Additional options are also needed for some basic testing without inserting data manually into the database.

4.1 Document Upload

The upload for the admin user works in the same manner as for the user. There are no differences except that the document must be manually assigned to one or more users by the admin before it can be edited. The reason for this being that the admin is not intended to edit documents, therefore, he has no menu entry for this (see section 3.1).

4.2 Assign Documents

To assign a document to a user a very simple UI has been implemented. This UI is shown in figure 4. In the left table there are all documents shown that are currently in the system. The right table shows all available users. If one marks a document and a user and submits this selection through a click on the button, the selected document is assigned to the selected user. The user can now see it in his UI and start translating it.

Assigning documents is mainly intended for the "normal" operation mode of the CASMACAT workbench when not running specific experiments. In this "normal" mode, as opposed to "experiment" mode, the user may self-configure his/her UI layout.

4.3 Configure Experiments

If specific experiments have to be done, then the admin can configure them through the UI shown in figure 5. When experiments are run, the layout of the UI should stay identical for all test participating persons. Also, the document to translate would always be the same. The reason for that is that the data gathered then has a common base and thus is better comparable.

The UI is currently very basic, but already includes two important points mentioned earlier: A particular document and layout can be assigned to an experiment. In a later stage, this UI will also allow many other options, like enabling or disabling ET for the experiment or allowing or disallowing the usage of IMT during experiment execution.

5 Browser Extension

A browser extension was implemented to connect an eyetracker (ET) to the web-page so as to capture gaze data of the post-editor. In prior versions, some of the ET capturing functions have also been part of the JavaScript code on the page. They resided in another additional external script, "logging.eyetrack.js". While this implementation could shown the basic concepts of integrating ET into the browser to be functional, there has been a delay in the browser reactivity when ET data processing took place during which the application was unresponsive. A decision was made that everything related to ET should be processed inside an extension. Interestingly, after moving the code to the extension, the unresponsive delay disappeared: Firefox seems to run plugins in an own process which enables real time behaviour in the browser. Through simply applying the right software design, the error disappeared.

The browser extension has first been realised for Firefox running on Windows connected to the EyeLink 1000 eye tracker. To ease the development of the plugin included in the extension, the browser plugin framework FireBreath [3] has been used. The plugin itself is implemented in C++ and loads and communicates with the DLL of the EyeLink. The plugin and the DLL are embedded within the extension. The extension additionally contains the code to display a button inside the browsers toolbar and the code that is needed for logging the ET data, implemented in JavaScript. Communication between the web-page and the extension/plugin are event-based.

The currently existing implementation allows the basic functionality for the EyeLink 1000 eye tracker, including calibration and logging. We expect that other eye trackers can easily be integrated into the extension, but will still need some additional code here and there and maybe some other minor adjustments. More details on further ET integration will be reported in forthcoming deliverables.

6 Security

The CASMACAT prototype will be published online and is accessible on the internet for a longer period of time. Even though the knowledge about the URL and IP address of the demo server will not be widely spread, security is still an issue because hackers also use automated tools to randomly scan the internet for hosts that are vulnerable to their attacks. Relying on obscurity as the only mechanism for security is not enough, especially as the demo server for the CASMACAT prototype is not located in a DMZ, unauthorized access to it also implies unauthorized access to the CBS' internal network. And as the other partners of CASMACAT will also connect to the prototype from within their networks they are also endangered.

Therefore, the prototype has been designed and tested to resist common security issues for web applications, like SQL injection, cross-site scripting (XSS) and broken authentication and session management. To prevent those attacks, the prototype has been carefully examined.

6.1 Authentication and Authorisation

The authentication of the prototype is based on a username and password authentication: To access the prototype, a user or admin has to enter a correct username and a password. The password is stored as a sha256 hash on the server which is currently not known to be breakable. The access to resources in the file system on the web server is managed by the Apache server. Only resources that must be readable by the browser are made accessible through the internet.The following access policy applies: the root folder and the folders "javascript" and "styles" must be readable through the browser, otherwise the prototype will not work. The "template" folder is not accessible from outside because this is not needed as the templates are delivered to the browser through calls to PHP files. The contents of the templates may expose internal application logic and should not be readable through a user.

The session management relies on the most recent PHP session mechanism which should be very hard to break and can also be seen as secure. To protect particular PHP files from unauthorized access (e.g. prevent a default user accessing the admin functions), every PHP file includes the "auth.php" (for normal resources) or "auth_admin.php" (for admin resources). Both scripts run always before the called PHP script and check if a valid session exists. In case of the "auth_admin.php" it is also checked if the session belongs to an admin user. If the session is not valid, the execution of the called PHP script is cancelled and a redirect to the login page is performed. In this way access from an unauthorised user is prevented.

Basically, the authentication and session management of the prototype can be seen as secure despite two facts: Firstly, the password is always sent in clear text because only the HTTP

protocol is currently used. Secondly, the session timeout is set to infinite, giving an attacker infinite time to guess a session id, if the user has not logged out correctly but just closed the browser. Event though these are important issues, they are very easy to solve: For the first issue, the protocol for the prototype can simply be switched to HTTPS. For the second problem, only a configuration flag needs to be set that defines a timeout, for instance 10 minutes. Both issues will be resolved shortly.

6.2 SQL Injections

SQL injections are a very common security flaw in web applications. They allow an attacker to send commands directly to the underlying database of the web application. They become possible if parameters that are passed from the browser to the database are not correctly filtered and/or escaped. Fortunately, the PHP interface for the MySQL database used for the prototype already has a mechanism to protect against this: The strategy is to always use prepared statements and the `bindParam()` function to assign parameter values. These mechanisms have to be used consequently. Of course, this has been done during the development of the CASMACAT prototype. To prove that it has not been forgotten somewhere, every part of the prototype has been tested with Sqlmap [2]. But none of the runs with this tool could identify an SQL injection. The prototype can be seen as robust enough against these types of attack.

6.3 Cross-Site-Scripting

This is probably the most common issue in web applications and also the most difficult to defend. It occurs when users can submit data that is shown in the browser afterwards. For instance, if the user uploads a document to the prototype, this document is displayed later in the edit mode. As it is currently allowed to include nearly any HTML element in the document (see section 3.1), also `<script>` tags may be included. These tags can now contain JavaScript code that is executed when the page is loaded. To avoid this kind of security gap, complex filtering of the input would be needed. In case of the prototype this has not been done, but anyway this does not lead to a security flaw: The current prototype allows users to edit documents that were uploaded only by themselves or by an admin. A user cannot upload a document containing some malicious code that is then executed by another user. The only one who would be able to do so is the admin, but an admin can always damage or misuse a system.

In conclusion, the CASMACAT prototype is also resistant against XSS without special actions taken. Of course, if later the concept changes so that a user could edit the translations of another user, special considerations are needed here.

References

[1] Ragnar Bonk. Development of a web-based translation process research tool. Master's thesis, University of Copenhagen, 2012.

[2] Damele and Stampar. Homepage of sqlmap. `http://sqlmap.sourceforge.net`. Accessed 2. April 2012.

[3] FireBreath. Homepage of FireBreath. `http://www.firebreath.org/display/documentation/FireBreath+Home`. Accessed 14. April 2012.

[4] Motion-Twin, Aardvark Media Ltd. Homepage of PHPTAL. `http://www.phptal.org`. Accessed 12. April 2012.

[5] Ortiz, Alabau, Koehn, Bonk. Specification of CASMACAT workbench. `http://www.casmacat.eu/upload/Deliverables/d5.1.pdf`. Accessed 28. March 2012.

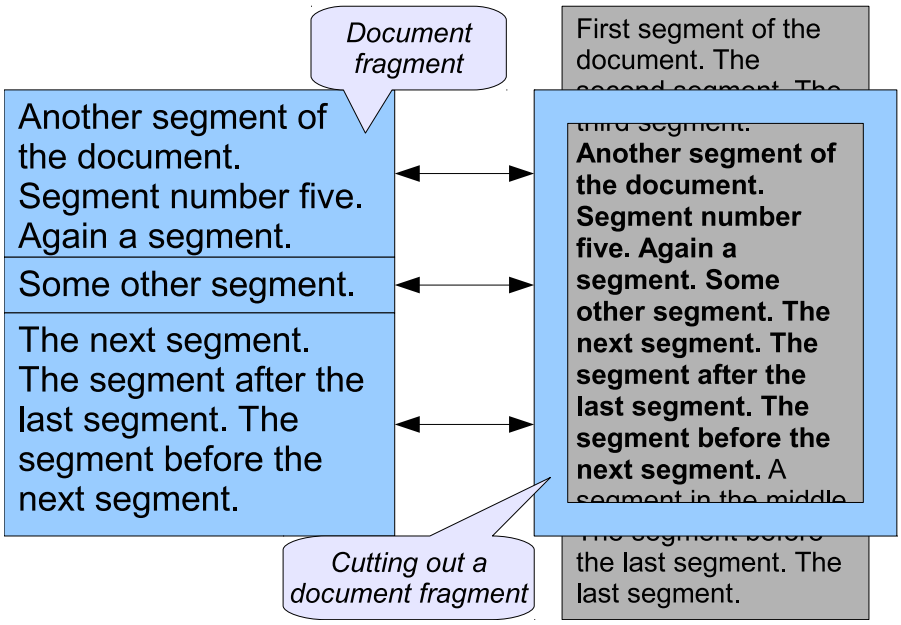

Figure 2: Sketch of a document fragment

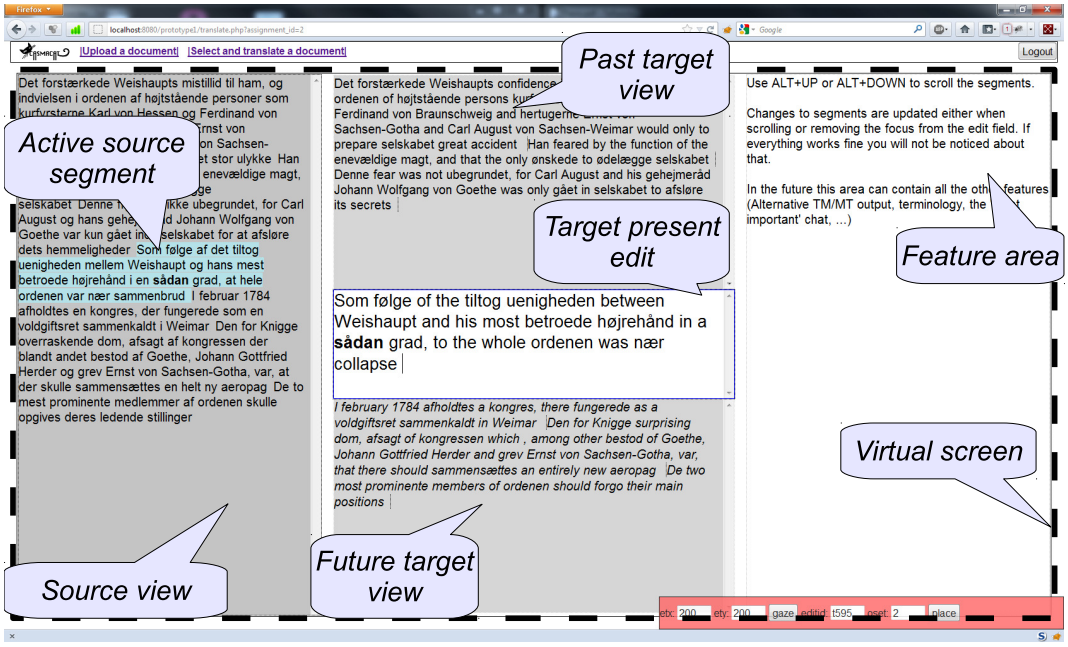

Figure 3: The translation environment of the prototype

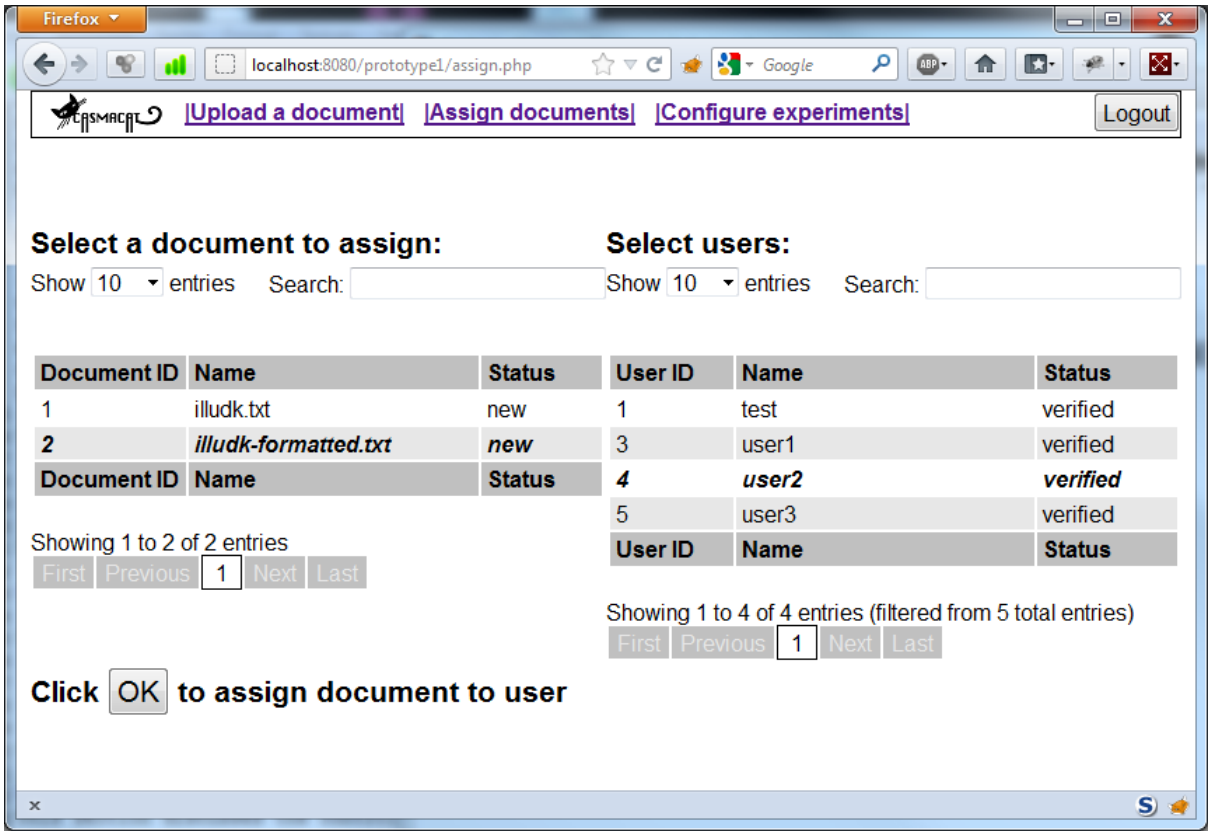

Figure 4: The UI for assigning documents to users

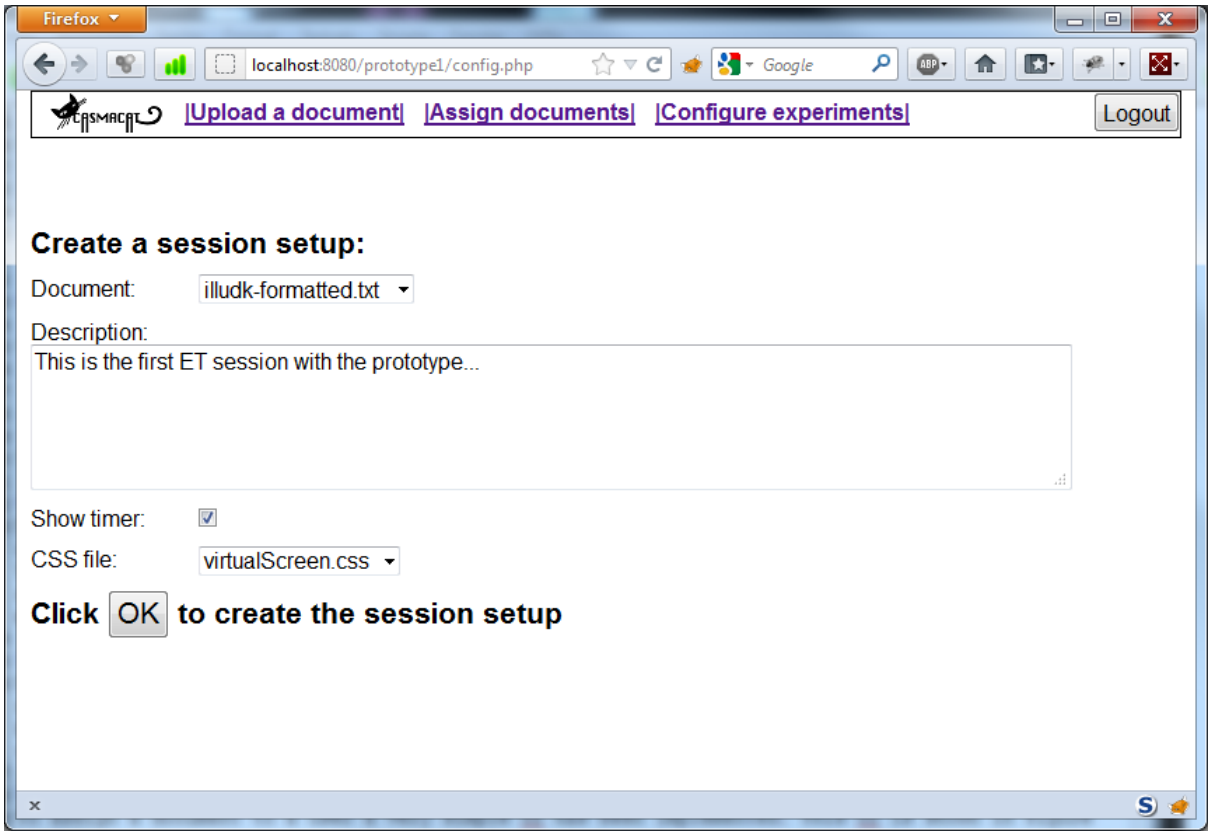

Figure 5: The UI for configuring experiments