|epcc|

THE UNIVERSITY OF EDINBURGH

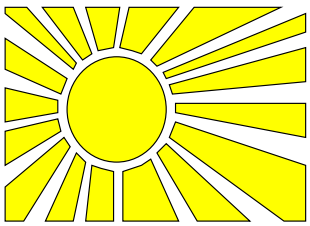
EPCC-SS-2000-12

Student Tracking: Mother, Mentor, Mate

Katarzyna Zając

Abstract

The aim of this project is to explore the area of student tracking to monitor how student use web based courses, from where, at what time, which the browsers are used and the coverage of the course material. The Department of Meteorology at the University of Edinburgh [7] has already established some of the underlying principles through the EuroMet [6] and NLN [8]. projects and has a continuing interest in student tracking. The goal is to build a portable tracking system independent of the number of servers on which the courses are hosted. The tracking system should be independent of the operating systems and software that have been installed on each server and on the client side.



Contents

1 Introduction

The main goal of tracking an individual user is to allow for the creation of student profile records that can be examined and utilised to best cater for students' learning needs and interests. Specifically, it would provide a feedback mechanism for the course organizers to appropriately focus, modify and target the courses to their students needs, enriching the online, interactive learning environment.

To achieve these goals one must know what information would be useful to record (§ 4). There are three main aspects with which we are concerned

- User identification – allows a complete trace of all of the activities of an individual student.
- Pedagogical data – to know the coverage of the course material to improve the contents of the course.
- Technical data – in order to match hardware and software requirements with that available to a user.

One can examine all the information at an individual level or collectively (statistics).

However, there are many problems one has to cope with while designing the tracking system. Using the access log generated by a local web server can only give a partial picture as to how the learning-material is being accessed. As http [2] is a stateless protocol it is impossible to get information about a session that consists of multiple requests directly from the access logs. Still, there are some possible ways of tracking an individual user (see §2). However, one must also consider privacy issue – some people do not like others gaining information about their web usage and they have the ability to prevent people from tracking (by controlling some browser's settings).

The next problem is the action of the caching mechanism [17] – Once someone has accessed a page on a server , subsequent calls to the same page will often just retrieve the page from a proxy server or from the local machine's memory or disk rather then from its original destination. The caching mechanism is very useful in order to minimise traffic and latency, but it makes accurate tracking more difficult to implement.

Recorded data is also difficult to analyse when a course is distributed across multiple servers. A very important requirement for a tracking system is its independency of the number of servers at which the courses are hosted as well as the operating systems and the software base installed on each server.

Possible solution for the two latter problems have been presented in [17]. However, the problem of tracking an individual still needs to be solved according to the implementation described in [17]. Actually there are several ways of tracking an individual (session tracking) that are used on the web (see §2).These were used for designing the tracking system for that purpose (see §3 and §5).

Further to the above, the specification of storing data should also be considered. There are some disadvantages and advantages of using a database management system in order to store tracking data. One of the advantages is using special data structures based on relation algebra in order to minimise the amount of storage space. The proposed entity-relationship model illustrating this is described in §6.

2 Tracking methods and tools

2.1 Common ways of modeling state within HTTP

This subsection outlines the four common ways of implementing session tracking (that is tracking of an individual user) on the web. Each of these has its advantages and disadvantages. However, their downsides are serious enough to have warranted the provision of session tracking API in web server extension libraries described in the next subsection (see §2.2).

User Authorisation is implemented at the server side – a web server requires authentication for the user to be able to access web pages. This can be used to track a session. The main advantage of using this method is that it is universally available on web servers. We are also able to identity users even if they use different locations and browsers. A disadvantage is its scalability over a large user population. What is more, pages protected with HTTP authentication are marked as private by default, so they will not be cached by shared caches (see caching issues §1 and [17]). However, you can mark authenticated pages public with a Cache-Control header. HTTP 1.1 compliant caches will then allow caching. It is best to minimise the use of authentication – for instance, if the images are not required to be private, it is good idea to put them in a separate directory and configure the server so as not to force authentication. That way, those images will be naturally cacheable and there will be no bandwidth problem .

Hidden Form Fields – here "invisible" form fields are placed in HTML files generated by the web server. The information stored in the hidden fields is particular to the client session and therefore placed in HTML files by the web server with each client request. As soon as the client navigates to a static HTML page, where no hidden fields are generated, session information is lost. Session tracking using hidden form fields only works for a sequence of dynamically generated web forms and causes problems with caching (see caching issues § 1 and [17]). The other disadvantage is that if the user uses the back key too often, information is lost.

URL Rewriting – operates in a very similar manner to hidden form fields. Instead of storing session information in form fields it is encoded within the URL. Web server programs are responsible for rewriting all URLs, local to the web site, so that they contain session information in the form of extra path information and URL parameters. They work for all dynamically created web documents, and not just forms. What is more, they are on the server side, so they cannot be manipulated by the user. URL rewriting handles people who are required to use caching services which cache documents based on the URL. With URL rewriting, every connection will have a unique URL.

Persistent Cookies – in general cookies hold small pieces of information. In the context of the internet, a web server sends a browser a cookie, which the browser saves locally for future reference. Every time the browser accesses the web server, it sends the cookie for the web server to interrogate. Cookies are a defacto standard for session tracking because the web server can use them to uniquely identify a client. The main problems with using this method of tracking is that they can be easily switched off by the user at the client end. They are also difficult to cache. The cookies specification is available from [1].

2.2 Technologies

There are several web technologies that implement the techniques of session managment described in § 2.1. This section is a general overview of each of them. As we already know, the action of tracking can be done at the two ends of an HTTP connection: the server side and the client side (the browser). The former has the advantage of being executed on the machine controlled by the system designers, while the latter can be used to obtain more information about the software and hardware at the user end as well as the local time of access at the remote machine.

2.2.1 Server side

Common gateway interface – scripts are programs that run on a http server. The scripts perform tasks that cannot be accomplished using web page technology alone. Simple web pages are not interactive, they are fixed and will look the same whenever one visit the site. CGI scripts generate the webpages requested by browsers dinamically. The pages are no longer static. The scripts can be written in any programming language. One, that is commonly used for this purpose is Perl as it is ideal for text processing .

A relatively easy way to obtain data about a remote user is to read environmental variables (see appendix A) set up by a server through a CGI script that store information about the users: their location, browsers and the operating systems.

One of the disadvantages of CGI is that it does not scale well. In a CGI-based web server, each CGI request runs as a separate process which is an quite expensive resource. Alternative solutions to this problem are mentioned later in this section.

Mod_perl – this solution will improve the performance of Perl scripts as by using this is possible to write Apache modules entirely in Perl. In addition, the persistent interpreter embedded in the server avoids the overhead of starting an external interpreter and the penalty of Perl start-up time. The disadvantage of using this tool is that it introduces dependency on the http server [13].

Embperl – gives you the power to embed Perl code in your HTML documents – although it can be used offline (as a normal CGI script or as a module from other Perl code) it was designed for running under mod_perl and Apache. It is directly integrated with Apache and mod_perl, it directly uses Apache functions and precompiles the code to avoid a recompile at every request [5].

PHP – can be compiled as a CGI or as an Apache module. It is an HTML-embedded scripting language. Much of its syntax is borrowed from C, Java and Perl with a couple of unique PHP-specific features thrown in. The goal of the language is to allow web developers to write dynamically generated pages quickly. One of the biggest advantages is that PHP can have persistent connections to a database. For more information see [14].

Java servlets/JSP – a servlet engine is a web server extension than runs within a Java Virtual Machine (JVM). Requests made of servlets are processed by threads and not processes, with makes Java servlets good for multithreading. The other advantage is that the server engine has to deal with byte code instead of interpreting the scripts from the beginning. This should make it faster. However, using Java servlets sometimes requires too much effort for initial hit for running a single instance of the JVM and the allocation of the initial Java heap. For more information see [4].

2.2.2 Client side

Javascript – is an interpreted programming language with object-oriented capabilities. The client side version of Javascript allows dynamic programs to be included in web pages that can interact with the user, control the browser within limits, dynamically create HTML content and, what is most important for tracking purpose, obtain specific information about the user's environment (like screen resolution and local time). One of the biggest disadvantages of using Javascript that it can be turned off by the user. To learn more about Javascript see [18].

3 Designing the system – issues

As mentioned in §1 recorded data is difficult to analyse when a course is distributed across multiple servers. A very important requirement for a tracking system is its independency of the number of servers from which the courses are hosted. One solution is to set up a global database server where data obtained from course servers and a client end can be sent and stored.

3.1 Direct connection

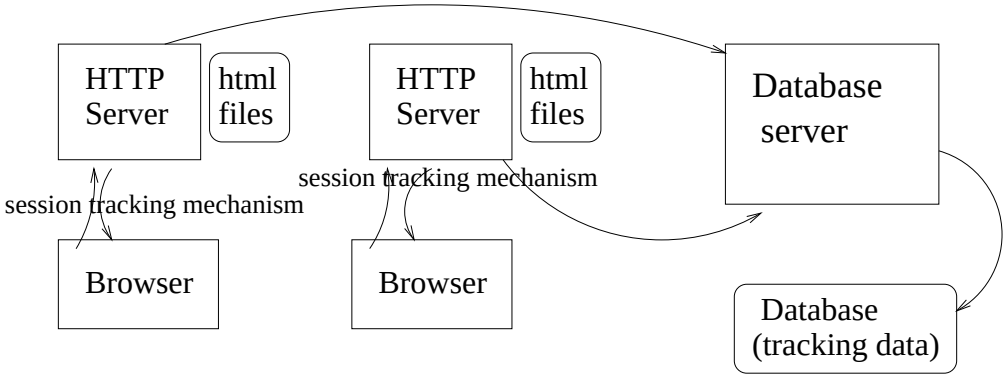


Figure 1: the tracking system – direct connection

This solution is illustrated in figure 1. In the instance where course pages are generated dynamically, this solution can be used . If each page of the course is an output from CGI program, servlet or php script , it is possible to connect the script or servlet to a global tracking database and record data stored in environmental variables (see appendix A). This solution minimises the number of connections needed to send the tracking data to a central database. It also has the advantage for using mechanisms of session tracking already implemented in existing technologies (like for example interface javax.servlet.http.HttpSession [3] using servlet technology). The problem with this method is that we would like our tracking method to be also implementable for existing static courses without putting too much effort in the set up. What is more, we would also like to record data obtained on the client side (for example local time time).

3.2 Asking about authorized image

For reasons mentioned previously, it was changed not to use such sophisticated methods at the course server side. All that was added was a small image tag at the end of each html course page that would requested by the browser from the server. The image is fact a CGI script that passes parameters obtained at the client end to the server. Hence this script must reside on the database server, so it records all the data taken from parameters and the environmental variables This is shown in figure 2.

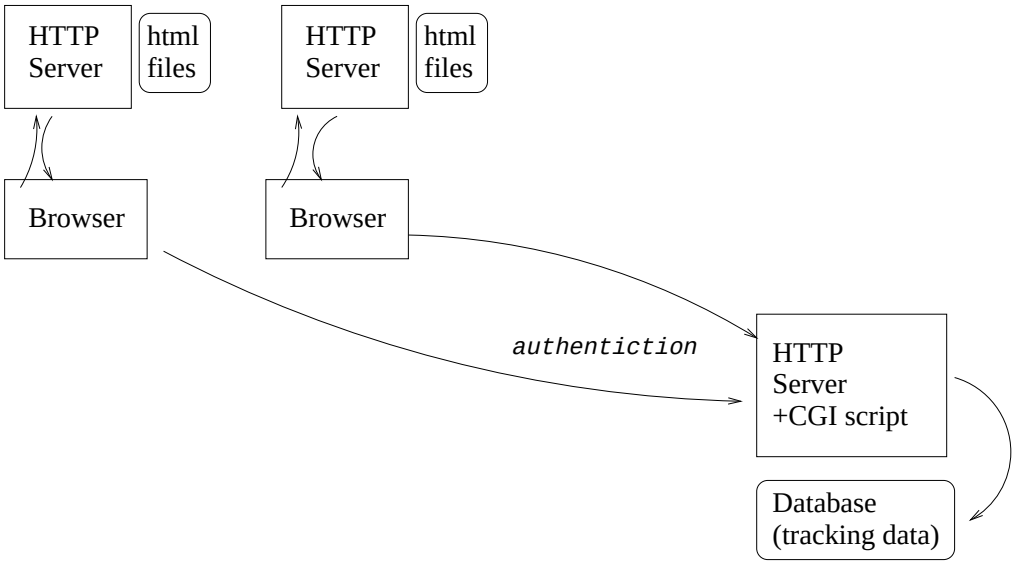


Figure 2: the tracking system-asking about authorized image

This solution has already been implemented in the existing Euromet [6] and NLN [8] projects. However, there was no framework for tracking an individual user within this set up.The most convenient idea is to restrict the CGI script to require authorisation and thus identify the user using the REMOTE_USER server environmental variable (a full list of environmental variables can be found in Appendix A) [1]. The only problem is that this could make the user login twice as the course pages also require authorization.

3.3 The course server responsible for authorization

Due to the problems with double login it was decided to make each server hosting a course be responisible for providing the identification of its own users. The server will then put the user id into each course html file using server side includes [16] or PHP [14] .This solution was used to build the first prototype for the tracking system shown in figure 3.

A problem immediately arises if the administrator of the www server does not want to use server side includes because for security reasons or it takes too much effort to reorganise the existing course to use PHP. We should also think about the solution that uses CGI instead of html embedded scripting. This solution is described in more detail in § 5 and it was also implemented during the SSP project.

[1] It is also possible to use the REMOTE_IDENT environmental variable which would in theory give the romote user login id, but this cannot really be relied upon

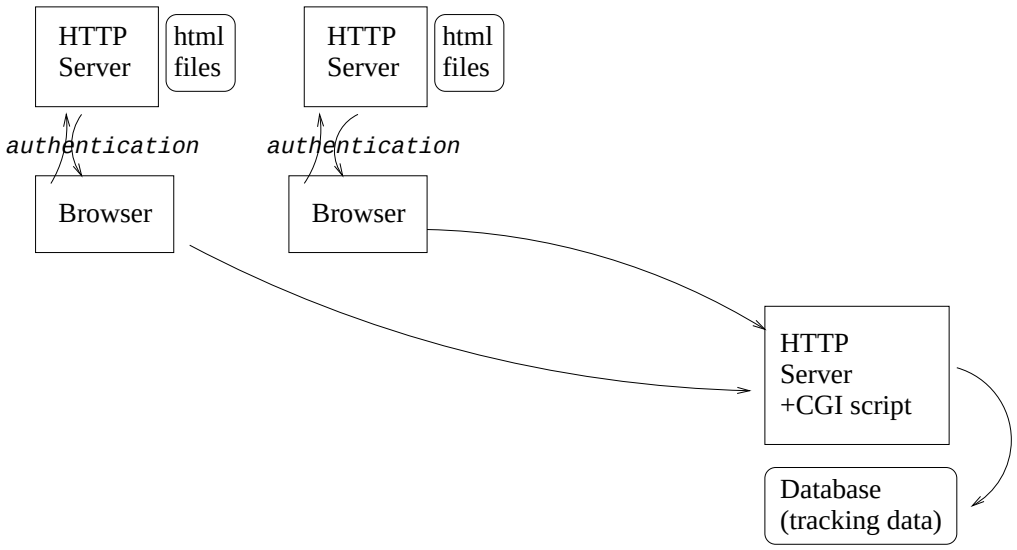


Figure 3: the tracking system-the course server responsible for authorization

4 Recorded data

This section gives an overview of data that can be recorded in a database in order to obtain the tracking goals mentioned at the beginning of this paper. As we want to track individual users we would like to record some unique identifiers. Next, as we need to know their software and hardware requirements, attention to technical issues should be paid as well. Last, but not least, we would like also to record pedagogical information in order to know the coverage of the course material a student is accessing. Once we obtain this information we would like to analyse it at an individual level and collectively. The full list of tracking details is shown below.

1. Users (obtained as described at point 4 of section 5)
   - authorised user (the server specific login) – in order to allow a complete trace of all of the activities of an individual student.
2. HTTP servers (obtained as described at point 4 in section 5)
   - IP and domain name – to know locations of the course.
   - http server type and version – to store information about the course server.
   - server OS type, OS version and CPU – same as above.
3. Client data
   - obtained as described at point 6 in section 5
     - full URL of page requested
     - IP address and domain of client computer – to know a location of a client.
     - client OS type, version and CPU – to know client hardware and software requirements.
     - browser type and version – to know client software requirements.

– time and date page opened (server time) – to track activities of students on specific date.

- obtained as described at point 5 in section 5
  - referrer page – to know more about the navigation of students though course material.
  - time and date page opened and closed (local) – to track activities of students between specific dates.
  - client screen resolution and number of screen colours – to know the client hardware requirements.

5 The tracking system - the final design

As was mentioned before, the goal is to build a portable tracking system independent of the number of servers where the courses are hosted. It should also be independent of the operating systems and software that is installed on each server. A system that satisfies these requirements is shown in figure 4.

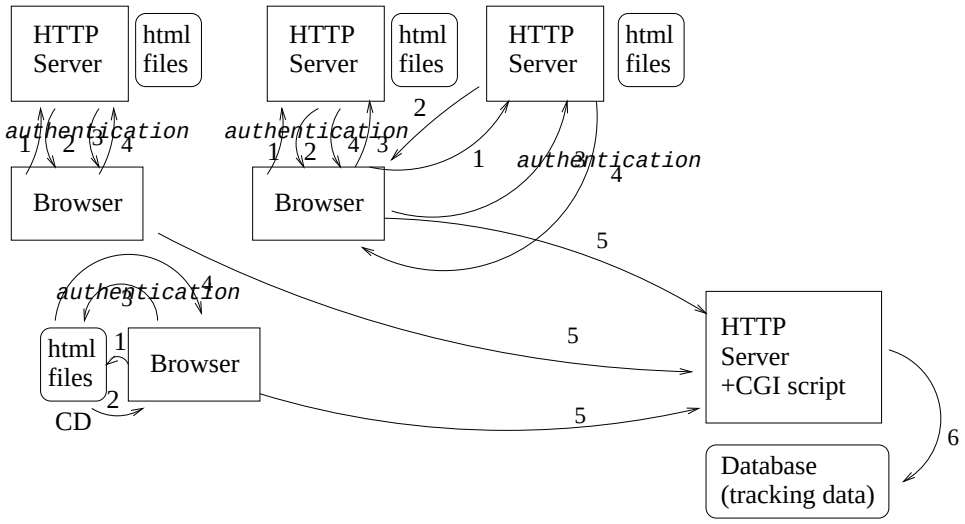


Figure 4: the tracking system.

The data is obtained from three different locations: the http server that stores course pages, a local computer of the user and the http server that executes script recording data described in §4.

The flow of tracking information is as follows (numbers refer to those in figure 4) :

1. A browser makes a request to the http server that stores html course files .
2. The html file obtained has a special SCRIPT tag that tells the browser to request the Javascript source file. (For non-Javascript browsers there is an IMG tag that tells the browser to ask for the image).

3. The browser makes the request again to the http course server and asks for the script (for Javasript enabled browsers) or image (for non-Javascript browsers).
4. The http server executes the CGI script that adds the information about the user identifier, the type and the version of the OS and software (CGI environmental variables).

   For Javascript users the output is the Javascript code.

   For non-Javascript browsers the script produces a redirection header to the CGI script that resides on the tracking server . It also passes the user id and additional information about the server as a parameter to this redirected location.
5. The Javascript browser executes JavaScript code in the html file
   - it adds additional information at the client end (i.e. local time).
   - it requests a small image located at the end of the page that is in fact a CGI script at the one central server connected to the tracking database and passes the tracking information as parameters.
6. The central server records data in the database adding information obtained from its own CGI environmental variables

6 Data storage

The other aspect of the tracking system presented here is the data storage. Information can be kept in a flat ascii file as well as be put it into the database system. The former solution's advantage is its simplicity. Is good for small and non-expanding data as there is no need to care about installation and administration. However, for larger sets of data, a database seems to be a reasonable alternative. The main feature of the database management system such as Mysql [11], Postgresql [15] or Oracle [12] is that the time of execution of queries does not grow much with the data size. These systems give good performance in data manipulation while using SQL – an easy language designed for that purpose.

In order to save space required by the data, a logical model of storage based on relation algebra is used. The main concept of this model is to keep the main entities of the data model in separate tables and to define the relations between them. The example model of storing the tracking data presented in §4 is shown in the figure 5. There are four different entities : users, client locations, servers and accesses. Each entity is represented by a separate table and consists of its attributes (i.e. one of locations attributes is its OS type). The attribute that identifies the single entity is called a key (i.e. unique id).
There are three relations:

servers–users – as we can not determine if the user logged at one server is the same person as the user logged at another one, we may assume that each user (represented by login name specified on the server) is related to only one server. That results in adding the identifier of the server to the set of users attributes (this is illustrated by means of the arrow in figure 5).

accesses–users – as one access involves only one user and one user makes many accesses, a user key (that consists of a user identifier provided by a server and the server IP) is added to the accesses table.

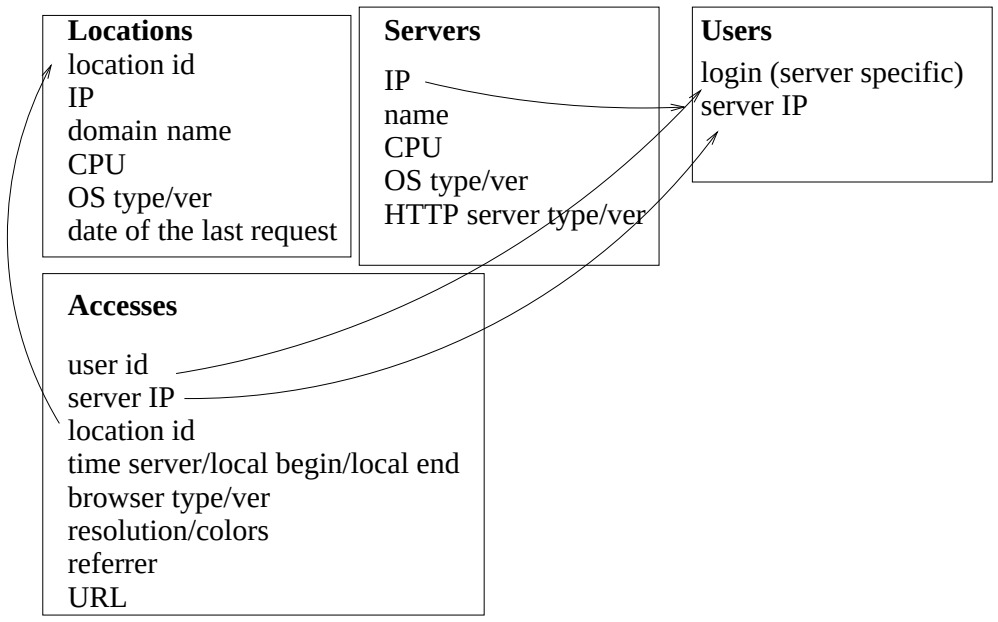


Figure 5: the data model

accesses–locations – the key for the locations table was added to the accesses table for the same reasons as mentioned before.

As one can see this model reduces the storage of redundant information (as for example the repetition of recording CPU at a location each time a page was accessed).

7 Implementation

The implementation of the system described in section 5 consists of two main parts: the installation on a node with a tracking database and the installation on a course node.

7.1 Install for a tracking node

This part of the software should be installed in a cgi-bin directory of the database server. It should be done only once for all the course nodes that are supposed to be tracked in the same database.

It should be changed in a future according to the way the data will be stored . If it is stored in RDBM system the script should connect to database via Perl::DBI [10] and Perl::DBD [9] modules.

Currently this part of the tracking system consists of the following files:

track.cgi - the CGI script takes data from its parameters and from environmental variables and records them in a file.It also produces a small image as an output.

track.dat - the ascii file with tracking data.

blank.gif - the small image sent to a browser.

7.2 Install for a course node

This part of the software should be installed on each course node to be tracked. It consists of two scripts that are called by Javascript browsers via SCRIPT tag and one script for non-Javascript browsers called via IMG tag. There is also a perl module that contains code common for all the scripts.

Script description:

tracking_functions.cgi The script that produces JavaScript code called by the SRC script tag that allows the tracking to be done at the client end. Also incorporates the REMOTE_USER info to be sent to the client. This script is called at the beginning of each html page and it defines all needed javascript tracking functions as well as recording the time, the page was loaded.

CGIcalling.cgi The script that produces JavaScript code called by the SRC script tag that allows the tracking to be done at the client end. It is called at the end of each html page and actually calls a javascript function defined in tracking_functions.cgi that produces IMG tag.

nph-red.cgi The script that is called by non-javascript browsers. (It is used by these browsers instead tracking_functions.cgi and CGIcalling.cgi). It takes the REMOTE_USER environmental variable as well as variables containing server specific information and passes it as a parameter to the tracking script that resides on the server with the tracking database. It redirects the browser to this script.

userid.pm This is a perl module used by the scripts described above. It was created in order to put in one file the common code that supports Javascript as well as non-Javascript users.

The software consists also of the script converting already existing html course files as they can call IMG and SCRIPT tags to request CGI scripts described above. The script is used only during the installation process, not the tracking. In order to put the whole course into an invisible frame we also needed an html file . The details are as follows:

conver.pl the script that converts an already existing course to use the tracking mechanism. Puts SCRIPT tags into html file (for calling tracking_functions.cgiCGI and calling.cgi) as well as NOSCRIPT IMG tags (for calling nph-red.cgi).

begin.htm this is the html file that puts the whole course in a invisible frame in order to record specific tracking information (like time when page was unloaded).

For course pages that are created dynamically there is a need to change their generator in order to insert html tags described above.

8 Conclusion

The main part of the project was to build a system for tracking an individual user according to what has already been done (see Euromet [6] and NLN [8]) as well as according to existing tracking methods in new, developing technologies. We should also take into the consideration the fact that usually we will not control the course node we want to track, but it can be administered by others who that should agree to install the software. Accordingly, it was desirable to

design this part of the system as easy as possible. The results of the investigation are described in section 5 (the prototype of the system was also implemented).

Next, the problem of storing and retrieving data needs to be discussed in more detail. If a RDBM system is used for this purpose it will also be easier to place a cgi-script on a course node (the information obtained on the client end will be send by a browser on each request as the parameters of this script) and after this script includes the user identifier and server data it will connect to the database server directly via Perl modules. However, the solution described in this paper can be developed as well.

Last, but not least, a user-friendly interface to the tracking data needs to be designed and implemented as well. It can be done using any existing technology described in section 2.2 via a connection to the database.

References

[1] http://home.netscape.com/newsref/std/cookie_spec.html .

[2] http://info.broker.isi.edu/in-notes/rfc/files/rfc2068.txt .

[3] http://java.sun.com/docs/books/tutorial/servlets/client-state/session-tracking.html .

[4] http://java.sun.com/docs/books/tutorial/servlets/index.html .

[5] http://perl.apache.org/embperl/Intro.pod.cont.html .

[6] http://www.euromet.met.ed.ac.uk/ .

[7] http://www.met.ed.ac.uk/ .

[8] http://www.nln.met.ed.ac.uk/ .

[9] http://www.cpan.org/modules/by-module/DBD/.

[10] http://www.cpan.org/modules/by-module/DBI/.

[11] http://www.mysql.com/ .

[12] http://www.oracle.com/ .

[13] http://www.perlreference.com/mod_perl/man/mod_perl_tuning.html.

[14] http://www.php.net .

[15] http://www.postgres.com/ .

[16] http://www.useforesite.com/tut_ssi.shtml.

[17] Robert Black, Charles Duncan, Peter Douglas, Martin Morrey, Daniel Gondouin . Accurately tracking the use of distributed Web-based learning courses . British Journal of Educational Technology , 1999.

[18] David Flanagan . Javascript-The Definite Guide . O'Relly & Associates, Inc., 1997.

A CGI Environmental variables

DOCUMENT_NAME The complete local directory path of the current document.

DOCUMENT_URI The local path of the current document referenced to the base directory of the webspace where the ENV script is located.

DATE_LOCAL The current local date and time.

DATE_GMT The current Greenwich Mean date and time.

LAST_MODIFIED The date and time of the last modification of the current document.

REMOTE_ADDR The IP address of the remote client browser. If your are using an anonymous proxy, it's IP will show here.

QUERY_STRING The raw query string sent from the remote browser.

SERVER_SOFTWARE The name of the HTTP server software.

SERVER_NAME The local computer name of the HTTP server.

GATEWAY_INTERFACE The name/version of the Common Gateway Interface served on this HTTP server.

SERVER_PROTOCOL The name/version of HTTP served on this HTTP server.

SERVER_PORT The IP port the HTTP server is answering on.

REQUEST_METHOD The method by which the current document was requested.

PATH_INFO The extra path info that is sent. This information is regarded as virtual (the path is relative to the base directory of the HTTP server).

PATH_TRANSLATED The "PATH_INFO" variable translated from virtual to local (physical) disk location.

SCRIPT_NAME The virtual path of the script being executed.

REMOTE_HOST The host name of the remote client. If your are using an anonymous proxy, it's IP will show here.

AUTH_TYPE The authentication method used to validate the remote client.

REMOTE_USER The user name used to validate authentication from the remote client. Great for use in password protected sites.

REMOTE_IDENT The remote user name if supporting RFC931 identification.

CONTENT_TYPE The content type of the attached information in the case of a POST or PUT.

CONTENT_LENGTH The length of the attached information in the case of a POST or PUT.

HTTP_ACCEPT A comma separated list of mime types that are accepted by the remote browser.

HTTP_USER_AGENT The name of the remote client browser software.

REFERER The URL of the HTML document which referred the remote client to this document.

FROM The name (most likely the-mail address) of the remote client user. Unlikely to be set.

FORWARDED The name of the proxy server through which this document is being processed.

ACCEPT_LANGUAGE The human languages that are acceptable to the remote client.

HTTP_COOKIE The cookie sent by the remote client.

I am currently studying for a Masters degree in University of Mining and Metallurgy in Krakow, Poland. My major is Computer Science.

I would like to thank you to my supervisors Mario Antonioletti from EPCC, Gordon Darling from EPCC, Charles Duncan from Department of Meteorology and Martin Morrey, Department of Meteorology for their help and advices.