SIE – Intelligent Web Proxy Framework

Grzegorz Andruszkiewicz, Krzysztof Ciebiera, Marcin Gozdalik, Cezary Kaliszyk, and Mateusz Srebrny

Institute of Informatics
Warsaw University
Banacha 2, 02-097 Warsaw, Poland

Abstract. In this paper we would like to present and describe *SIE*, a transparent, intelligent Web proxy framework. Its aim is to provide efficient and robust platform for implementing various ideas in broad area of Web Mining. It enables the programmer to easily and quickly write modules that improve pages on that site according to personal characteristics of the particular user. *SIE* provides many features including user identification, logging of users' sessions, handling all necessary protocols, etc. *SIE* is implemented in OCaml – a functional programming language – and has been released on GPL.

1 Introduction

We live in the era of information. The rapid development of computer and communication technologies enabled people to quickly exchange data at a low cost. Probably the most popular source of information is the Internet, and the most commonly used service is WWW. HTML pages are a universal way of publicizing knowledge, but it is very difficult to find the exact piece of information we are looking for.

In our paper we would like to focus on one given Web site, containing various pages. The webmaster always tries to optimize the structure of the service in order to help users navigate. But different users have different preferences. When reading a page one user may next want to see page X, another one page Y, etc. When typing a keyword into the search engine, e.g. "chaos", one user wants to read about mythology, another one about fractals, and so on. Sometimes a user does not know which page exactly she is looking for, because she had not visited it yet. One static structure will not fully satisfy all users' needs.

That is why *dynamic page personalization* is so important. It is often possible to predict the interests of the user, analyzing for example the history of her choices. In this case it would probably be helpful for the user if she was provided, on the page she is currently using, with the most important links. It may be even more useful if she had some links to pages similar to the current page to minimize time spent on searching. Of course the better the algorithms to *predict interests* and to *find similar pages* the better the page personalization process.

The main problem with algorithms which try to understand human behavior and predict users' actions is that it is extremely difficult to develop them only theoretically. Humans are often irrational and their language is ambiguous. On many pages considerable amount of information is not only in written words but in pictures, animations or even sounds and these are still nearly impossible to analyze automatically. That is why most algorithms are heuristics based on empirical experiments and developed through testing. To advance the level of research in this field, scientists need to have possibility to test their algorithms and to tune their parameters quickly and cheaply.

Having made this observation we have decided to create a simple, yet powerful framework for constructing intelligent Web proxies. We called it *SIE – Site Improving Engine* and we wanted it not to be limited to personalizing alone – hence “Improving” and not “Personalizing”. *SIE* relieves module writers from re-implementing user identification, network protocols handling, collection of statistics, etc. *SIE* gives users the opportunity to focus on the concept alone and enables them to quickly and comfortably write a new testing module.

In addition we include a few interesting modules showing the variety of *SIE* features.

It is worth mentioning that *SIE* and the modules are written exclusively in *OCaml* functional language. It gives the programmer many interesting possibilities characteristic for functional languages and provides very good performance. *SIE* has been released on GPL. Sources can be found at `http://sie.mimuw.edu.pl`.

2 Solution

We would like to present a framework supporting programmers in creating, changing, testing and fine-tuning intelligent Web proxies – including Adaptive Web systems. Our system – *SIE* – implements basic features, essential for intelligent Web proxy to function properly. In this section we will mention all these features and in the next one we will present related work. Further we will discuss the architecture of the system. Then we will present already implemented modules, their functionality, some theoretical analysis and the way they are integrated with the *SIE* framework. In the last two sections we will discuss possibilities of further development and summarize the paper.

SIE is implemented as a proxy server, transparent to the end user. It intercepts all HTTP messages that are sent to the server (requests) and its responses to the user. Our system interprets these messages and identifies the user. The identification process consists of two stages: when the user first sends HTTP request, the system finds out if she already has special cookie set, containing a unique ID value. If she does, *SIE* can identify and associate current session with the stored history. If she does not the system provides her with a new ID number. In the second stage, when the user fetches WWW pages, *SIE* does not employ the cookie mechanism. Instead it rewrites all the links pointing to the server, which are on the pages sent to the user, in such a way that they uniquely identify the user and the link followed. Uniqueness is achieved by generating 256-bit numbers and storing generated values in hash tables, which are then periodically purged of stale entries. For example the link pointing to `http://www.icwe2004.org` could be rewritten as: `http://sie.mimuw.edu.pl?SIE_SESSION=AF387GZ2&SIE_LINK=2YUZ19A0&SIE_ORIGINAL=www.icwe2004.org`. The purpose of each parameter is summarized in Table 1. With this method we can identify the user throughout each session, even if her web browser does not support (or has disabled) the cookie mechanism.

Parameter	Purpose
`SIE_SESSION`	Identifies the session (and thus the user).
`SIE_LINK`	Identifies the exact link which has been followed by the user – the page which contained the link and the page the link was leading to.
`SIE_ORIGINAL`	Original link – in case the link is followed after appropriate hash-table is purged (e.g. if it was stored in bookmarks)

Table 1. Parameters added by *SIE*

The core of the whole system is constituted by modules. A module usually consists of two parts: offline and online. Online parts of every module is invoked whenever a request or response is processed by *SIE*. Upon registration in the framework the online part specifies:

– supported content types (e.g. `text/html` or `image/png`) (in case of `text/html` the module may also request a parsed HTML tree instead of pure text)
– callbacks provided (request modifier, response modifier)
– priority (used to determine the order in which modules are called)

When the HTTP message contains a HTML document and there is at least one module which requested to receive parsed HTML documents instead of pure text *SIE* interprets HTML and constructs a parse tree, which is a very convenient form for further analysis and modifications.

When modules are called back by *SIE*, they are provided with all the data they need in an appropriate form:

– HTTP parameters
– parsed HTML contents (when appropriate)
– user identification
– current trail in the traversal tree of current user [1]

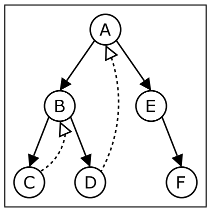

Fig. 1. Example traversal tree representing user session. Solid lines are regular clicks. Dotted lines represent pushing "Back" button in the browser

[1] Formally this would be defined as the shortest path from the root to the current node in the tree (e.g. in Figure 2 when user visited "C" current trail would be given as list: A → B → C)

Modules can modify HTML tree using received information – e.g. choose an appropriate model of user behavior basing on the user's ID, find out which pages may potentially be useful for the user and insert appropriate links into HTML tree. At the end *SIE* deparses HTML back to plain text and passes this new version of the document to the original destination.

Other content may be modified as well – e.g. images may be scaled down to size suitable for small, portable displays found in PDAs[2] or mobile phones. Modules may even generate responses themselves – allowing for custom pages generation.

Before processing the request all important properties of the HTTP request are logged. Apart from the fields found in CLF[3], there are two fields which together make up the strength of *SIE*. These are:

- user ID
- previous node in the traversal tree

Both of these fields are taken from the rewritten link (described above), which allows any module written for *SIE* to easily obtain traversal tree similar to the one presented in Figure 2 from logs. Constructing trees from logs is usually done by module's periodically executed part[4], which can prepare aggregated data for the online part. A traversal tree is a natural representation of behavior of a user visiting a Web site. Many papers dedicate whole chapters to techniques of extracting information about "user sessions" [5] and "episodes" [6] from a CLF-compliant log (e.g. Apache access log). Users of *SIE* are relieved from reimplementing those algorithms and can focus on their modules alone.

We are fully aware that sometimes CLF logs are the only source of information available but we also feel that widespread use of systems like *SIE* can quickly change this situation. Deploying *SIE* in front of a Web site – without any additional modules – can gather information useful for analyzing the site and testing new algorithms on it. The next step would be implementing a module which would use data collected previously.

SIE can be easily scaled thanks to its cluster architecture. It is able to distribute the servicing of different clients to separate computers in the cluster. It enables the administrator to increase performance with the growth of a Web site. In addition, thanks to the *Watchdog* function, the system is safe and easy to maintain. It automatically disables the parts which do not function correctly, or shuts itself down completely when the whole cluster has broken down. In such cases the WWW service functions unhindered, but without any additional features provided by *SIE* modules.

SIE has been tested on the server of the Programming Olympiad (`http://sio.mimuw.edu.pl`). We have collected some statistical data concerning the processing of requests by *SIE*. On average the preparation of the request (HTTP parsing, creating threads, etc.) took about 1% of the total processing time of the request. ESEE module took 2%. Then the request has been sent to the web server and 88% of processing time

[2] Portable Digital Assistants
[3] Common Log Format, *de facto* logging standard in WWW servers
[4] We call these parts *offline analyzers*
[5] Defined in [17] as "a delimited set of user clicks across one or more Web servers"
[6] Defined in [17] as "a subset of related user clicks that occur within a user session"

was spent on waiting for reply. Afterward BM used 5%, and the remaining 4% was used by *SIE* to prepare the reply for BM, and to send it back to the user. These results prove that *SIE* has a small time overhead over the web server and LAN connection.

During the tests (about 3 months) the system was stable and fully functional all the time.

Unfortunately, due to organizational and financial constraints no tests of the "intelligence" of the modules have been performed. Such tests should be done on a large Web site, and should include some kind of opinion poll in order to estimate the effectiveness of our ideas. We hope we will manage to realize large-scale tests shortly.

3 Similar systems

An impressive review of implementations of Web Usage Mining systems has been given by Robert Cooley in his PhD thesis [9]. It includes a systematic classification of reviewed systems into five categories:

1. Personalization
2. System Improvement
3. Site Modification
4. Business Intelligence
5. Usage Characterization

SIE with its current suite of modules (*Adapter* and *SEE*), would probably fit into "Personalization" and "Site Modification". Adding other modules (as described in section 6) could spread *SIE* also to other categories.

In [7] authors present system called WebCANVAS which analyzes Web server's logs and displays visualization of navigation patterns on a Web site. It is accomplished by automatic clustering of users and manual clustering of pages on the Web site into categories. For every cluster of users, navigational patterns between categories are shown. These patterns represent habits of Web site's users and can be used for improving high-level structure of the site.

In [13] the author has presented IndexFinder – a tool which assists webmasters in adding so-called "index pages" to the site. An index page consists of links to similar pages. IndexFinder employs *conceptual cluster mining* to cluster pages not only visited together, but also having similar content. Proposed index pages are presented to the Webmaster which chooses if they should be added to the site.

Corin Anderson in his PhD thesis ([4]) described two systems: PROTEUS and MONTAGE. The former is used to adapt Web pages to the needs of electronic devices with small displays, such as PDAs or modern mobile phones. MONTAGE, on the other hand, does not modify content. Instead, it builds personalized web portals, consisting of content and links from sites the user had visited previously.

IBM has created its own framework for creating Web proxies called WBI – Web Browser Intermediaries[7]. Currently WBI is part of the WebSphere Transcoding Publisher and the Development Kit is no longer available for download. [5] introduces

[7] previously Web Browser Intelligence

concept of intermediaries as “computational elements that lie along the path of a web transaction”. This paper also describes WBI as a framework for building and running intermediaries. WBI supports five type of intermediaries: request editors, generators, document editors, monitors and autonomous. WBI, being a framework for creating intelligent proxies is, in many aspects, similar to *SIE* . The main difference between them is the placement of the system between user and the Web server. As described in [6] WBI is placed between the user and the Internet (all servers), whereas *SIE* has been thought as a proxy between the Web site and the Internet (meaning here users visiting the site). Additionally, *SIE* includes several features (briefly mentioned in Section 2 and described in more details in following sections) which would have to be implemented as modules in WBI.

4 Architecture

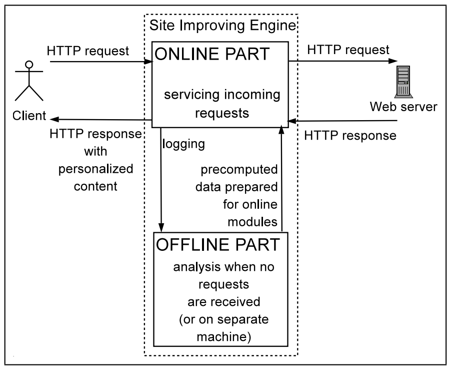
Site Improving Engine
HTTP request
ONLINE PART
servicing incoming requests
HTTP request
Web server
Client
HTTP response with personalized content
logging
HTTP response
precomputed data prepared for online modules
OFFLINE PART
analysis when no requests are received (or on separate machine)

Fig. 2. *SIE* architecture.

As mentioned before, *SIE* is divided into two parts with different functionality:

online - this part serves the client directly, analyzes the information flow between client and server, updates the log file and invokes online parts of modules.

offline - this part is active only periodically – it performs some time-consuming analyses for the online part.

SIE is just a framework to run special modules, which constitute the core of an intelligent Web proxy. Usually every module consists of an *online part*[8], which is executed for every request, and an *offline part* – the *analyzer*.

[8] Sometimes we will use the term *module* – when we do, it will be clear from the context what we are referring to.

4.1 The online part

To make *SIE* as robust as possible, it is crucial to have the lowest possible overhead in request processing and achieve maximum throughput. That is why it is very important to make as many calculations as possible in the offline part, and to use the computed results online.

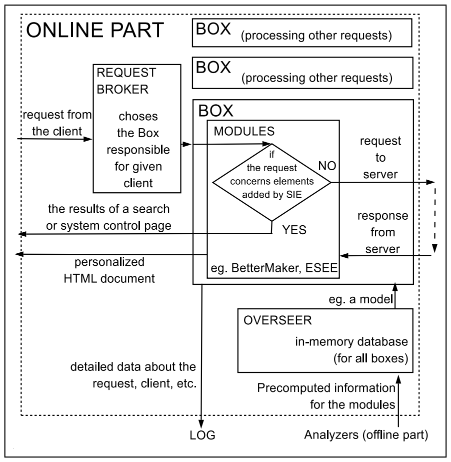
ONLINE PART
BOX (processing other requests)
BOX (processing other requests)
REQUEST BROKER
choses the Box responsible for given client
request from the client
BOX
MODULES
if the request concerns elements added by SIE
NO
YES
request to server
response from server
the results of a search or system control page
personalized HTML document
eg. BetterMaker, ESEE
eg. a model
OVERSEER
in-memory database (for all boxes)
detailed data about the request, client, etc.
Precomputed information for the modules
LOG
Analyzers (offline part)

Fig. 3. Processes in online part. Arrows indicate information flow (data, document, message, etc.)

When a request is received by *SIE* it is processed by *Request Broker*, which chooses a *Box* and forwards the request to it. There may be many concurrently running Boxes, each of them on a different computer in the network. We have implemented a cluster architecture, which means the workload is divided among several computers functioning in parallel, all of them performing the same task. Performancewise, it is crucial to send all requests from a particular user to the same Box during one session. Otherwise computers in the cluster would have to utilize some kind of a shared memory which would hold all information about sessions. This could easily became a bottleneck of the whole system. Therefore we have decided that Boxes should run completely separate, and logs they generate should then be combined into one big log periodically by *Gatherer* (e.g. via NFS).

Gatherer is an external program, which reads logs generated by different Boxes and outputs a combined log. All concurrency issues emerging from accessing one shared log file in Boxes (lock contention, network issues originating from the use of a distributed file system, etc.) are thus avoided.

On the other hand, there has to be centralized configuration so that every module that runs inside Box uses the same model and parameters. This task is fulfilled by *Overseer* – a simple in-memory database. Analyzers insert models, needed by online parts, into the Overseer. Some modules may query Overseer about specific information they need on demand and some read the current version of the model contained in Overseer when Boxes are starting up. Later, when new models are calculated by offline analyzers, a special signal is generated, which informs all Boxes that objects in Overseer have been changed. Upon receival of that signal modules can update local copy of the model. As a result all Boxes have up-to-date versions of model.

At the beginning of the processing of every message, the Box logs appropriate properties of request (as described in Section 2). Then the request is passed to registered modules, which may modify it or even generate response without involving the WWW server. SEE[9] (a personalized search engine and *SIE* control center) takes advantage of this functionality. SEE checks whether the request refers to Web site's search engine. When it does – a response is produced and returned to user. Otherwise request is forwarded to the WWW server. Similarly, the check if the request is for the *SIE* control page is conducted. Control page is a place where the user can modify individual parameters for different modules. For example, she can set how many links will be added by BM to every page or set the favored search criterion described in section 5.2.

When response arrives from the WWW server, the same Box which processed the request modifies all the links found in the HTML document sent in the response, so they uniquely identify user and her session. Additionally a parse tree of the HTML page is constructed if any of the modules uses this form. The tree is then passed to all active modules. In current implementation, only module called *BetterMaker*[10] uses this functionality and adds personalized links to every page. BetterMaker uses data prepared by *eXPerimenter*, an offline analyzer described below.

4.2 The offline part

Offline parts are run periodically (e.g. using standard UNIX `cron` daemon) during low system load or on a computer dedicated to this task. The main idea is to perform some calculations using a log produced by Gatherer, which would be impossible or too expensive to do online. In addition *SIE* provides the modules with the current, analyzed content of the web server. A special program called *Robot* is responsible for preparing this data.

Offline parts should analyze the log and information about all the pages on the site in order to produce data, which would be useful for their respective online parts. For example, running an analyzer which would generate a model of Web site users' behavior every night would make the site truly adaptive, as the model would be updated daily.

Currently there are two analyzers implemented:

eXPerimenter - analyzes traversal trees generated by Web site users. Then it generates a model for BetterMaker (the online counterpart), which uses this model online to personalize pages (by adding potentially interesting links to them).

A-SEE - analyzes features of pages which have been visited by all users. E-SEE (the online counterpart) uses generated model to personalize search results for users by appropriately reordering links returned by search engine.

[9] Details concerning the example modules can be found in the next section.
[10] See next section for more information.

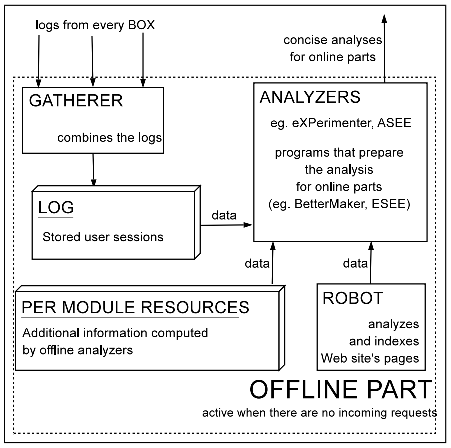
logs from every BOX
concise analyses for online parts
GATHERER
combines the logs
ANALYZERS
eg. eXPerimenter, ASEE
programs that prepare the analysis for online parts (eg. BetterMaker, ESEE)
LOG
Stored user sessions
data
data
data
PER MODULE RESOURCES
Additional information computed by offline analyzers
ROBOT
analyzes and indexes Web site's pages
OFFLINE PART
active when there are no incoming requests

Fig. 4. Offline part as a support for online part. Arrows indicate information flow.

A more detailed description of the example modules can be found in the next section.

5 Modules

5.1 Adapter

Adapter consists of two parts

- eXPerimenter (XP) – offline analyzer
- Better Maker (BM) – online module

The most visible to the end user is a little table with links predicted to be most useful to her. BM as an online module modifies responses sent back to the user and adds the table to the page. To generate this table, BM uses a model prepared periodically (e.g. daily) by XP. Model is stored as an object in Overseer and therefore when a new version is generated by XP it is automatically updated in all computers in the cluster. Number of links generated by system can be controlled by the user through a special control panel. In current implementation, the number of links chosen by the user is not stored in any persistent storage, so it is lost upon restart of *SIE*.

To effectively render such table two problems had to be solved. First, basing on previous traversal patterns of users of the Web site, given current user's trail, the algorithm has to predict which pages the user might visit next and with what probability. We chose for that task error-pruned Selective Markov Models described in [10]. The model contains k distinct Markov models, where k is the maximum episode length taken into account. k-th Markov model contains probabilities of visiting page j, having previously visited pages $i_1, i_2, \ldots, i_k$. For $k = 0$ the model reduces to unconditional probability[11]

[11] To be exact instead of probability we use frequency – the maximum likelihood estimator.

of visiting given page in the Web site. With the growth of k, the model would grow enormously, so *pruning* technique had to be applied. Currently, it is done using a subset of logs, which is not used for calculating Markov models. For details on the exact method of *overall error pruning* please refer to [10].

Attributing web pages with probabilities is not enough. Some pages may be buried down in the site's structure (i.e. to reach them user has to click on many links) whereas some others may be easily accessible. To compensate this, BM ranks links using expected number of saved clicks, i.e. the product of probability of visiting the page the link points to and number of clicks that would be saved had the link been put on the current page. To estimate this value, BM uses MINPATH – a simple recursive algorithm given in [2]. The algorithm takes as an input current user's trail and the model generated by XP and returns list of links ranked by the expected amount of saved clicks to the user. Carefully selected maximal recursion depth and great OCaml run-time performance allows for executing MINPATH for every response which is sent to the user.

Currently, Adapter does not distinguish between users – i.e. the same model is used for every user visiting Web site. Of course this approach may cause poor personalization on large sites with many different types of users. In such situation, a more sophisticated model needs to be used. We discuss possible improvements in the next section.

5.2 SEE

Another module is SEE – a search engine which aims at personalizing search results. More precisely, even though all searching users receive the same list of links, they get them in a different order. The order is set by SEE's knowledge about a specific user. To illustrate this, let us refer back to the example mentioned in the introduction. Let us assume that the user is concerned about "chaos" meaning a mythological phenomenon. Therefore she should find pages on ancient gods before those concerning fractals.

First the *SEE analyzer* (A-SEE) indexes the Web site's resources rating each page according to a number of criteria (e.g. amount of text and pictures, number of links, etc.). The value of each criterion is represented by an integer between zero and ten. The criteria vector which is thereby computed describes the characteristics of each page.

SIE is then employed to provide the history of the user's searches. Not only does SEE focus on the keywords the user is searching for, but, more importantly, it takes into account which pages she chooses from the results suggested. This analysis shows which criteria are important for this particular user when she is looking for this particular keyword. It works like this: when the user looks for a word some results are provided by SEE or by any other search engine. The user clicks on one of the links provided. SEE assumes the user has chosen this particular page because she prefers it for some reason. After a period of time, the analyzer computes an arithmetical mean for the criteria values of such chosen pages. This results in SEE obtaining a set of weights for each user-keyword combination. These weights indicate which criteria are important (and to what extent) to this particular user when she is searching for this particular keyword.

The analyzer's task ends here. SEE comes back into action whenever the user searches the Web again. The resulting list of pages is sorted according to the criteria earlier identified as the ones preferred by the *SIE* user. More precisely, each resource containing the keyword has its criteria vector. For each page SEE multiplies this vector by adequate (for this particular user and keyword) weights and, thus, the ranking is computed.

The more the user searches, the wider SEE's knowledge about her and, thus, the more accurate the search results the user receives.

To provide the user with more control over her searches, SEE allows her to choose one criterion to be used individually. Should the user employ this feature, her lists of links will always begin with pages favored by the criterion.

6 Possible improvements

SIE is in an early development stage and there are many features still to be added. For us, it is most important to develop *SIE* itself as a platform for building intelligent Web proxies. However, we have also a few ideas for the improvement of the already implemented modules and adding of new, equally interesting ones.

6.1 SIE itself

SIE is a framework created to aid the programmers. This is why it is crucial to develop additional technical documentation, tutorials, easy and well-commented example modules, etc. to make learning *SIE* as easy as possible. In the future, we are planning to create a graphical system to automatize basic tasks or to enable them to be performed by mouse drag-and-drop operations.

On the other hand every computer system should be easy to install and maintain. That is why we would like to add an automatic installer as well as create ready-to-use compiled packages for MS Windows and popular Linux distributions. In addition, a graphical user interface is needed for administrative purposes. It would be also very useful to enable the administrator to load/unload the modules without restarting the whole system. To accomplish this the usage of Overseer has to be enhanced. It can be used to provide communication between central administration console and Boxes. The infrastructure is present and working (i.e. the Overseer itself) but there is no code in Box that would allow for remote administration and feedback (e.g. sending of warnings and system logs describing error conditions).

In order to make *SIE* used in practice, we must improve the graphical aspect of our system. Elements added by our modules are readable, but they are behind the aesthetic standards imposed by modern HTML documents.

6.2 Modules

New modules We hope to extend *SIE* by writing new modules ourselves and to encourage others to contribute their ideas as new modules as well. Currently, we see immediate need to add two modules which would show:

1. Top k most popular pages
2. k most recently added pages

We are also developing a module to record and save user session (as in a sequence of user clicks) as a program in *WTL*. WTL is a new script language, developed by us specially for describing user behavior on a web page. Such a program can be executed later, simulating user actions. This simulation could be used as a test, resembling real scenarios of Web site usage allowing to measure Web server's performance or find broken links. It can be also used to automatize some routine tasks done using a HTML interface.

Adapter As mentioned in section 5.1 is a fairly simple module, which was implemented rather as proof of a theoretical concept than a module intended to be used in reality. Many features can be, however, improved or added.

First of all, BM constructs – and XP uses – only one model. For large Web sites it is obvious that no single model could be appropriate for all users. Therefore, basing on clustering of users, Adapter has to use many models, one for every user cluster. Possible approaches to user clustering are described e.g. in [7] and [12].

Another technique, which could prove useful for Adapter, is *page clustering*. Basing on words (terms) contained in the documents from the Web site, the module could group those documents into clusters of pages with similar content. Alternatively such classification could be done manually or semi-automatically (with the help of someone, who would provide keywords for every page). Especially appealing in this context seems to be the algorithm called *Concept Indexing* (described in [11]). For every page, it devises a list of terms (called *concepts*), which best describe the page's content. Having concepts attributed to every page, it is possible to create, for each user, a list of concepts she (or cluster of users) is interested in. Such information can be valuable from the marketing point of view (directing advertisements or communication to the user) and can also help resolve the problem of new pages – when a new page is added to the Web site it is not added to as a suggested link by BM because it is not yet seen in logs. With the help of concepts, BM can find all users potentially interested in reading the new page, and include link to the page on pages viewed by them.

Additionally, concepts could allow for creating models on a higher level of abstraction than URLs – namely clusters of pages. Such models could be used for visualization of user access patterns (as in [7]) or, as noted in [3], to predict Web page entries on a different Web site but with similar structure.

SEE The way we developed SEE imposed on us the assumption that, before everything else, the general mechanism was needed. Now, when the module sorts links individually for each user, the lack of strong criteria has proved to be its main flaw. The criteria we have implemented only indicate how powerful SEE could be. They mainly test the percentage, on each rated page, of certain HTML tags, inside which are the keywords. The concept of semantics-driven criteria has accompanied the whole process of developing SEE. In other words, SEE could immensely benefit from clustering pages which cover the same topics.

Another issue which SEE should deal with is the size of the model. SEE attempts to store information in pairs: the user and a given keyword. Hence the need for grouping users sharing common interests (in terms of criteria). SEE could also do with a way of clustering keywords that the users perceive as similar.

7 Conclusion

Adaptive web and personalizing Web servers are relatively young fields of computer science. In spite *SIE* is still an immature system, we hope it will help the scientists to test their ideas and develop new modules. Such a framework could prove very useful in social sciences, or in fields that include interaction with humans, as it is impossible to model their behavior in absolutely abstract way.

We were not able to find any similar framework freely available on the Internet. We hope *SIE* will fill this gap and make future research easier.

8 Acknowledgments

First of all we would like to thank Krzysztof Ciebiera, who had the main idea of the project. Without his irreplaceable help as our tutor the project would probably not succeed. *SIE* itself was developed by many authors. Detailed list can be found at `http://sie.mimuw.edu.pl`.

References

1. R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of SIGMOD-93*, pages 207–216, 1993.
2. C. Anderson, P. Domingos, and D. Weld. Adaptive web navigation for wireless devices, 2001.
3. C. Anderson, P. Domingos, and D. Weld. Relational markov models and their application to adaptive web navigation, 2002.
4. C. R. Anderson. *A Machine Learning Approach to Web Personalization*. PhD thesis, University of Washington, 2002.
5. R. Barrett and P. P. Maglio. Intermediaries: New places for producing and manipulating web content. In *World Wide Web*, 1999.
6. R. Barrett, P. P. Maglio, and D. C. Kellem. How to personalize the web. In *Proceedings of the Conference on Human Factors in Computing Systems CHI'97*, 1997.
7. I. V. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White. Visualization of navigation patterns on a web site using model-based clustering. In *Knowledge Discovery and Data Mining*, pages 280–284, 2000.
8. S. Chakrabarti. *Mining the Web*. Morgan Kaufmann Publishers, San Francisco, 2003.
9. R. Cooley. *Web Usage Mining: Discovery and Application of Interesting Patterns from Web Data*. PhD thesis, University of Minnesota, 2000.
10. M. Deshpande and G. Karypis. Selective Markov Models for Predicting Web-Page Accesses, 2001.
11. G. Karypis and E.-H. Han. Concept indexing: A fast dimensionality reduction algorithm with applications to document retrieval and categorization. Technical report tr-00-0016, University of Minnesota, 2000.

12. B. Mobasher, H. Dai, and M. Tao. Discovery and evaluation of aggregate usage profiles for web personalization, 2002.
13. M. Perkowitz. *Adaptive Web Sites: Cluster Mining and Conceptual Clustering for Index Page Synthesis*. PhD thesis, University of Washington, 2001.
14. M. Perkowitz and O. Etzioni. Adaptive Web Sites: an AI Challenge. In *IJCAI (1)*, pages 16–23, 1997.
15. M. Perkowitz and O. Etzioni. Towards adaptive Web sites: conceptual framework and case study. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1245–1258, 1999.
16. P. Pirolli, J. Pitkow, and R. Rao. Silk from a sow's ear: Extracting usable structures from the web. In *CHI-96*, Vancouver, 1996.
17. W3C. Web characterization activity. http://www.w3.org/WCA.