The Web Object Store: an infrastructure for mining semantics from web resources and their usage

TECHNICAL REPORT

Mirco Nanni[1*], Fabrizio Silvestri[*] Fosca Giannotti[*], Dino Pedreschi[**]

[*]ISTI-CNR
Pisa, Italy
{mirco.nanni, fabrizio.silvestri,
fosca.giannotti}@isti.cnr.it

[**]Dipartimento di Informatica,
Università di Pisa, Italy
pedre@di.unipi.it

Abstract. The development of methods for an effective and efficient access to the information contained in large masses of digital documents is a long-standing objective in computer science research, and its importance is emphasized by the growing availability of large information repositories. With the advent of the web, the methods for content delivery evolved in the services offered by search engines, categorization and topic search services, related pages services, etc.: the main innovation needed was a shift from content-only analysis methods to the combined analysis of contents and *hyperlinked structure* of web documents, as witnessed by the PageRank metric for document relevance. However, as the web explosion continues, the limitations of the current generation of access services to web contents are becoming clearer, in terms of scarce quality and freshness of the results, etc. The overall vision presented in this paper is the development of a new generation of services for enhanced content delivery – web search, document classification, question answering, etc. – tailored for a large-scale community of web users, and based on the use of knowledge extraction methods for enriching raw data with automatically-extracted semantic information. We refer to such category of services as Usage-enhanced Web-Access services (UWA), emphasizing the fact that they are based on a combination of web usage, web content and web structure mining. Usage data are those that the community of web users decides to share, on a privacy-preserving basis, in a participatory style. Usage-enhanced Web-Access services (UWA) applications are complex, for several reasons. They deal with enormous volumes of data. They deal with continuously incoming streams of data. They deal with different abstractions of the data. They apply computationally expensive data mining algorithms on the data. The needed infrastructure for supporting the development of UWA applications is called, in our project, *Web Object Store* – WOS – a web data management system specialized in dealing with web content, structure and usage data. The WOS is designed to provide persistency, compression and efficient access methods for data structures representing basic web objects (Web documents, URIs, Citations, and HTTP requests), and to help the development of sophisticated applications that need complex data structures and advanced analysis methods.

Keywords: Data Mining, Web Usage Mining, Web Semantics, Knowledge Discovery Support Environment, Web Data Management System, Enhanced Content Delivery services.

---

[1] Contact author.

1 Introduction

The development of methods for accessing, retrieving, filtering, and discovering of desired contents within large masses of digital documents is a long-standing objective in computer science research, which received always increasing attention – due to the increasing availability of larger and larger information repositories. To this general purpose, methods developed for information retrieval, intelligent query answering, document classification and categorization, text mining, etc. are based on the analysis of document *content*. With the advent of the web, the methods for content delivery evolved in the services offered by search engines, categorization and topic search services, related pages services, etc.: the main innovation needed was the combined analysis of contents and *hyperlinked structure* of web documents, as witnessed by the PageRank metric for document relevance.

Despite the vast popularity of search engines, as the web explosion continues, the limitations of the current generation of access services to web contents are becoming clearer, in terms of:

- scarce precision and quality, difficulty of matching users' desire to obtain relevant high-quality information in response to queries or questions;
- scarce readiness to novelty: the ranking mechanisms of search engines penalize new important pages that enter the scene, in a inertial phenomenon where "rich get richer" and fresh important information is not delivered to users (see [5]);
- inadequate conceptual tools for managing the complexity of query results.

If these weaknesses are due, in part, to the overwhelming over-abundance of low-quality, irrelevant web material that obfuscates the search, it is widely accepted that the main impediment lies in the lack of *semantics* of the web documents: there is a lot of tacit knowledge hidden in such documents – background information and experience, social conventions – and it is often impossible to discover such knowledge from the syntactic nature of the available web resources. The awareness of these limitations is precisely at the basis of the Semantic Web effort, aimed primarily at finding ways to annotate web documents with semantic tags in order to access knowledge instead of rough unstructured material. While we adhere to this agreeable, illuminist, far-reaching vision of the semantic web, we are also aware that it is essentially a long-standing goal, which will require time and large efforts to be achieved. What to do in the meanwhile? The answer seems obligate: try to extract as much semantics/knowledge as possible from the available web information, and use such semantics/knowledge to improve the services to access, retrieve, filter, discover desired web contents. While we wait for the optimum – the semantic web – let's try to live with an imperfect approximation – the web semantics that we can derive using data mining and knowledge discovery methods over available web-related information. It should be observed that mining web semantics is not in contraposition with semantic web, as the achieved techniques may easily evolve synergistically as the semantic web becomes a reality (see [20]).

To this extent, another crucial source of information entered the scene in the last few years: the *web usage data*, recorded in log files by web servers or proxy servers; potentially, these data contain information on how people use (access) the web resources, and *web usage mining* techniques have been put forward to extract such information. Accordingly, a first generation of web usage mining applications has been proposed, ranging from:

- adaptive sites, which adjust dynamically their structure in response to users' behavior (e.g., see [19]);
- recommendation systems, which exploit user profiles extracted form the usage data to suggest alternative products, related pages, etc. (see [1]);
- proxy-level intelligent caching and prefetching of web pages, which improves performance in accessing contents on the basis of previous experience (see [17]).

So far, web usage mining applications are mostly limited at improving local resources on the basis of how the external world looks at you (your own site, your own proxy) on the basis of the locally gathered usage information. Albeit interesting, these applications just scratch the surface of the potential knowledge that is hidden in the usage data – in principle, there are unprecedented opportunities to learn precious knowledge from the traces of people's access to the web resources. So far, no application is based on the analysis of how a given group of users access the general web, with the aim of exploiting such usage information to the purpose of providing enhanced access to web resources to the users from this group.

The point of view advocated in this paper is that it is possible to deliver better contents with combined web content, structure and usage mining. If a sufficiently large group of web users is willing to share its usage data, on a privacy-preserving basis, then it is in principle possible to learn from these data new models and patterns that, in combination with document content and structure analysis, may yield enhanced, semantics-based content access and delivery – better search services, better categorization and document classification services, better question answering services.

The key idea is that, starting from usage data and network traffic information, is not only possible to derive models of the users' behavior, but also, indirectly, discover semantics of the content of the accessed resources. Two examples are the following.

Example 1. By analyzing traffic data, e.g., at proxy level, it is possible to reconstruct sessions consisting of queries to search engines followed by accesses to some of the pages presented in the query result. From these query sessions it is possible to extract models, which describe how users in the community use some search services, in order to provide to such users sharper focus on content that emerges as important from the community's behavior.

Example 2. From usage data it is possible to reconstruct the web subgraph that is accessed by the community, with links adorned with traffic measures. Information about contents can be induced from this "traffic graph", e.g. about new interesting sites visited by users of the community – even weak signals about new pages with high potential quality can be discovered, fed to the crawler of the community search service for download, and

promoted in their rank adopting a generalized PageRank metric that combines content, structure and usage of web pages.

1.1 Vision

Summarizing, we envisage the development of a new generation of services for enhanced content delivery – web search, document classification, question answering, … – tailored for a large-scale community of web users, and based on semantics/knowledge extracted by means of an intertwined combination of web content, structure and usage mining. Usage data are those that the community of web users decides to share, on a privacy-preserving basis, in a participatory style.

Enhancements in contents access and delivery is rooted, besides in better analysis of documents' contents and structure, in the ability of combining knowledge induced from usage data, describing how users within the community access to web resources outside (and possibly also inside) the community. This general goals is pursued within an Italian national project called ECD – Enhanced Content Delivery – funded by the Ministry of Research.

Some *Usage-enhanced Web-Access services* – UWA services from now on – of this combined style are sketchily reported below, which are currently pursued within our project.

- Characterization, on the basis of usage only or usage + contents + structure, of new important emerging sites, or irrelevant sites (e.g., advertising sites); this is crucial to instruct the crawler of the community web repository towards fresh, relevant documents while avoiding unimportant documents (see [7]).
- Page ranking based not only on static hyperlink structure, but also on usage information, for achieving a more accurate and dynamic measurement of documents' importance.
- Recommendation of similar/related documents and keywords, on the basis of combined usage/content analysis (see [1]).
- Caching and clustering of web search results (see [18]).

Another example is the intelligent query answering proposed by Chakrabarti et al., that learns from training sets of query-answer pairs accumulated by using the system (see [6]).

Are the advocated communities of web users existing, and willing to share their usage data to obtain enhanced access services to contents? Strictly speaking, any group of users which share a proxy server (or a collection thereof) is a candidate community, as web traffic logs of appropriate format can be recorded at proxy level. Examples include (i) students, professors/researchers of a large university campus/research centre, (ii) citizens, administrators and policy makers of a town/province, (iii) customers of a geographically distributed Internet provider. An important caveat to be taken into account is *privacy*. The usage data analysis should be based on privacy-preserving mining methods (see [11]), capable of assuring the individual user that sensitive data (about preferences, accessed sites,

etc.) is never disclosed. In a trustable setting, where sensitive usage data are shared to the purpose of developing public knowledge which helps in achieving better access to contents (and not to the purpose of individual surveillance), the advocated notion of web user community may exist, in a participatory style where everybody contributes to better service to the whole community.

Is this scenario technically feasible? The main objective of this paper is to illustrate the underlying principles and the architecture of an enabling infrastructure for extracting semantics from web resources and their usage data, to the purpose of deploying usage-enhanced web-access services.

1.2 An infrastructure for mining semantics from web resources and their usage

UWA services, such as those described later in this paper, are complex, for several reasons. They have to deal with enormous volumes of data. They have to deal with continuously incoming streams of usage data. They have to deal with different abstractions of the data. They need to apply computationally expensive data mining algorithms on the data. An enabling infrastructure to develop UWA services, which facilitates mining semantics annotation from rough web data, needs therefore to satisfy a number of requirements.

First, it should provide useful abstractions of web-related concepts, and enable their direct deployment at different levels: for instance, a comprehensive ontology of usage concepts is needed, ranging from elementary access log entries to page-views, sessions or specific types of sessions. This is needed because UWA services may refer to usage and content information at different levels of abstraction, or need to create their own specific abstractions from basic ones in a direct way.

Second, it should provide a comprehensive repertoire of pattern and model types, to represent the extracted knowledge/semantics to be used in constructing UWA services. Again, this is needed because UWA services may need a wide variety of mining models (frequent patterns, rules, classifiers, predictors, clusterings), or may need to create their own specific models from basic ones in a direct way.

Third, it should provide extremely efficient data structures for both data and models – both space and time efficient – and should guarantee persistency over time, in order to provide easy and tractable means to deal with previously extracted knowledge. Issues here include data compression and specific access methods.

The needed infrastructure for supporting the development of UWA services is called, in this paper, a *Web Object Store* – WOS – a web data management system specialized in dealing with web content, structure and usage data. The WOS is precisely designed to provide persistency, compression and efficient access methods for data structures representing basic web objects:

- Web documents,
- URLs and URIs,

- Citations and web graphs,
- HTTP requests,
- Page views,
- User sessions.

Besides basic web objects and their access methods, the WOS provides means for accommodating data structures representing the knowledge extracted by information retrieval or data mining algorithms from the basic data:

- Indexes,
- Traffic graphs,
- Classification rules,
- Models, patterns.

The WOS is also designed to support easy extensibility of basic web objects and other concepts to higher-level abstractions.

In this paper we illustrate the design principles and the architecture of the Web Object Store, and show by means of both concrete and visionary examples how the WOS may act as the enabling infrastructure for the construction of UWA services, capable of adding more semantics to web-access on the basis of usage information. The paper is organized as follows: in Section 2 the Web Object Store is presented along with the description of its classes and application categories; in Section 3 we present the methods that can be followed in order to populate the WOS; in Section 4, a sample WOS application is described; finally, Section 5 presents a large scale WOS application and in Section 6 some conlusions are briefly summarized.

We are well aware that this is a very ambitious project, and that very strong competitors exist both on the research and the industry side. We believe however that realizing the WOS has a specific value and a specific orientation (towards web usage) which characterizes the goal and creates novel research challenges. It should be noted that there are at least three possible deployment strategies for the WOS that could be followed, corresponding to incremental maturity stages of the project:

- A research infrastructure, in the spirit of the WebBase project at Stanford University [3], over which new specific algorithms and UWA services may be experimentally created and validated;
- An infrastructure for web analytics services to be offered to third parties, in a spirit close to the WebFountain IBM project [4];
- An industrial Web Data Management Systems aimed at developing and engineering web mining ECD applications.

In any case, a sensible novelty with respect to the mentioned related projects (the WebBase and WebFountain projects, two impressively large university and industrial initiatives) is in the orientation of services towards a community of web users, and the explicit combination of content and usage that the community decides to share.

2 The Web Object Store

The Web Object Store (WOS) is part of the development environment for Enhanced Content Delivery applications.In Figure 1 the components of the WOS architecture are shown. The WOS represents the central Data Warehousing system where data coming from various sources (Web documents, Web logs, etc.) are collected by specialized populating algorithms (Web crawler, Traffic monitor, etc.). Such data are stored in data structures and handled by means of ad hoc algorithms, forming what we call *µWOS*. The µWOS is the kernel that provides the very basic data to be used in ECD applications. It also provides the algorithms to access them. On top of the µWOS several layers of algorithms and (possibly) additional shared data structures can be developed within the WOS. Eventually, such algorithms are exploited to derive languages and applications (including user interfaces) or are used by other WOS algorithms or populating algorithms.

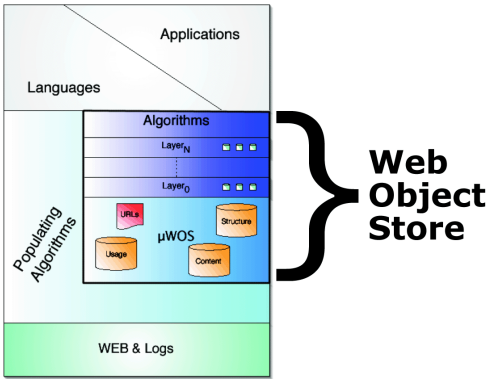

Fig. 1. Architecture of ECD Applications Development Environment

In the following sections the content of the µWOS is described. In the last part an example of extension of the basic data structures by means of a WOS algorithm layer is presented..

2.1 µWOS Classes

The four basic repositories of data provided by the kernel of the WOS are: *HTTPRequests* (Usage data), *Citations* (Structure Data), *Documents* (Content data) and *URIs*. These repositories, modeled as classes in an object-oriented environment, provide a minimum kernel of concepts which allow to extract compound and extended entities that will be exemplified in later sections. A basic set of attributes and methods for each class is provided, together with descriptions and notes on entities modeled where needed.

URI. A Uniform Resource Identifier (URI) is a compact string of characters that identifies a resource. Familiar examples include an electronic document, an image, a

service (e.g., "today's weather report for Pisa"), and a collection of other resources. Please note that not all resources are network "retrievable"; e.g., human beings, corporations, and bound books in a library can also be considered resources. A URI can be further classified as a locator, a name, or both. The term "Uniform Resource Locator" (URL) refers to the subset of URI that identify resources via a representation of their primary access mechanism (e.g., their network "location"), rather than identifying the resource by name or by some other attribute(s) of that resource. The following example illustrates a URI represented by means of a http address scheme: *http://www.math.uio.no/faq/compression-faq/part1.html*.

The kernel of WOS is able to manage URI objects.. Within the WOS, each URI corresponds to exactly one object in the Document class (described later) that stores the content of such resource.

HttpRequest. When a user visits a web page, the browser sends a number of requests to a web server. One request is for the HTML file but individual requests for each of the other elements that make up the web page are also sent. For instance, requests for graphic files, audio files and so on.

We define an HttpRequest as the generic request to a Web server to retrieve a resource, for example an HTML page or its parts. The information stored by a HttpRequest usually complies with the Common Log Format (CLF). The CLF format will be described in some detail in section 3.2.

HttpRequest objects store these values in corresponding attributes, providing methods for accessing them. In particular, the Request field is modelled as a pointer to the URI element that represents the requested resource. URI objects are in *one-to-many* relation with HttpRequest objects because more distinct HttpRequest objects could represent requests to the same resource.

Citation. The link structure of the Web can be represented as a digraph. The Web pages being its nodes and the links between pages being its directed edges. The Citation class exactly describes an edge of such graph, and therefore stores the URI of a *source* web page (the origin of the edge) and the URI of a *target* page (the destination of the edge) which is linked by the first one. Both URIs are modeled as references to the corresponding URI objects stored in the μWOS.

Document. Each object in this class virtually contains a copy of a resource which is in *one-to-one* correspondence with a URI object. . The content of the resource is physically stored as a compressed file in the underlying file system, and can be retrieved and uncompressed by means of the methods provided by the class.

2.2 WOS Application Categories

The WOS allows the building of three different application categories. Aim of this section is to introduce a precise terminology that will be used later on. To this purpose we are going to describe the characteristics and the Application Programming Interface (API) of WOS *algorithms*.

2.2.1 WOS Algorithms, User's Applications and WOS Languages

A WOS application can be defined as "*a software module that is designed to exploit the repositories and access methods of the μWOS in order to produce non-trivial information and, eventually, to store it in the WOS"*.

Basically, when referring to WOS applications we should distinguish among three distinct types of software systems:

1. *WOS algorithms;*
2. *User's applications;*
3. *WOS languages.*

We have three different types of modules that can be implemented using the WOS framework since each of them is suitable for different purposes. Referring to Figure 1, we notice that the first kind of modules are part of the WOS, and are thought as extensions of the basic μWOS, while the last two are built on top of the WOS architecture.

WOS Algorithms. As described before, WOS algorithms are essentially software layers that extend μWOS repositories and methods. WOS algorithms are built on top of the μWOS and of a number of other algorithm layers, to the purpose of providing other, more complex services to be used by upper level layers, including user's applications and WOS languages. As an example, we could think about a service that is able to provide, given the URI of a Web page, URIs of semantically related pages or URIs of pages reachable traversing at most a given number of links. These kinds of applications, in particular, are not directly accessible by end-users, and can be accessed only by other WOS applications that can possibly provide also some kind of interface towards end-users.

User's Applications. User's applications may be essentially defined as software interfaces built on top of the μWOS and a certain number of underlying algorithm layers, which is able to interact with end-users. In Section 4, a WOS user's application, called *Suggest,* will be presented in detail. Such user's application is aimed to output suggestions to users of a Proxy Server.

WOS Languages. They are strictly related to User's Applications, and the main difference relies on the exposed user interface. In this case the interfaces represent a kind of higher-level access methods to the capabilities of the WOS architecture. An example of WOS Language may be a query language designed to allow the specification of queries to extract patterns from Web-Usage/Structure data.

2.2.2 WOS Algorithms Interface

While the WOS architecture can be freely extended by users with the algorithms and data structures they need, adopting the application interface they want, two basic principles have been followed so far in the development of such extensions, especially on the algorithm interface level, which are recommended also to third-party developers:

1. Homogeneity: the user interface should provide a standard set of methods for populating and accessing the resource (i.e., repository) they are creating. Such methods should include: constructors for creating and populating the repository (where the application-dependant parameters are specified), destructors for erasing (part of) it, and a set of indexes (at least one) for accessing the objects it contains. For each index the methods for random access and sequential access via cursors on the repository should be provided.
2. Flexibility: as far as possible, general purpose algorithms should be applicable to any data type which provides a minimal set of standard access and manipulation methods, strictly required by the specific algorithm. As an example, some kind of clustering algorithms, such as the hierarchical agglomerative ones, simply require to be able identify the objects and to compute object-to-object distances, regardless of the data type.

A pragmatic way to achieve such goals consists in the adoption of a hierarchy of data type classes, which is built on the base of the access capabilities they provide. Each algorithm, then, will accept as input any data type satisfying its data access requirements, which correspond to the simplest data type class with that characteristic. So far, a simple hierarchy has been adopted, amenable to step-wise refinements and extensions, which (i) distinguishes *data repository types* in sequential access and random access types, possibly combined together, and (ii) classifies the objects stored in such repositories in relation to the operations they are able to perform, such as returning their (unique) ID or the ID of the transaction they belong to (useful in transaction-based algorithms), and computing some kind of distance w.r.t. another object. As an example, Figure 2 depicts the access capabilities (listed on the top row) and object-manipulation methods (on the right column) required by some data mining and data navigation algorithms: clustering, frequent patterns and citations navigation (i.e., navigating the web structure).

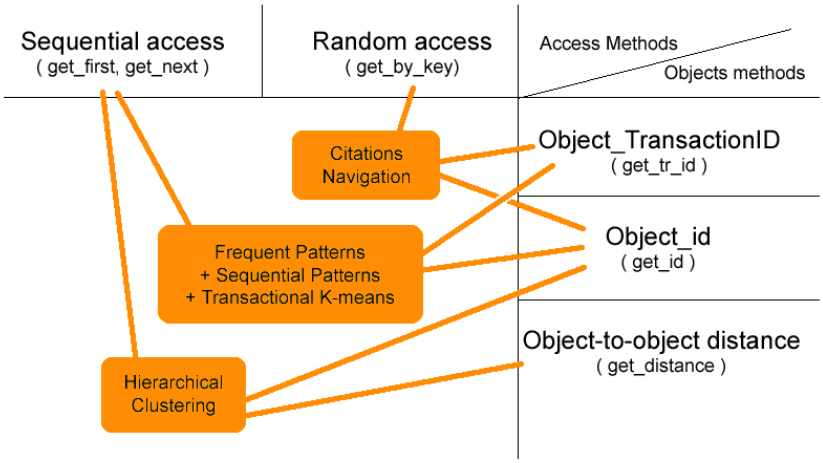

Fig. 2. Sample Data Types for general-purpose WOS algorithms

We notice that in most of these examples, a simple sequential scan of the data is required, since the algorithms usually build an internal, minimal and efficient, representation of the data for their successive computations. In more complex cases, however, more sophisticated access methods can be required, e.g., efficient range queries, which are heavily used by any density-based clustering algorithm.

2.3 Sample WOS Algorithms for Semantic Information Extraction

In this section, we provide two examples of algorithms designed to be integrated in the WOS architecture. Both of them can be considered as tools for enriching the basic usage and content information managed by the μWOS with higher-level concepts and mappings from raw data to such concepts. To give a glimpse of the wide range of possible WOS applications of this kind which can be developed, the first example will be a simple case which provides basic abstractions of usage data by means of elementary heuristics, while the second one will extract complex information on content data, making use of elaborated data analysis techniques.

2.3.1 Extraction of Usage Data Abstractions

The abstractions presented in Section 2.1 represent a minimal core to be extended. Preprocessing routines extract new objects having HttpRequest, Citation, Document and URI as basic data. So, the μWOS can be extended to model new significant entities or to group existent ones into collections in order to handle aggregated information easily. As an example, we describe here three abstractions of the basic HttpRequest entity, also depicted in Figure 3:

- *PageView*: is defined as the visual rendering of a Web Page. In other words, a Pageview consists of several items, such as frames, text, graphics and scripts that construct a single Web page. A PageView is represented in WOS as a set of HttpRequest objects.

- *Session*: is defined as a sequence (i.e., an ordered collection) of PageViews requested by the same user in a browsing session in a chronological order. Session objects can be intra-site, i.e., composed of PageViews belonging to the same host, or inter-site if the session tracks the user through more hosts.

- *User*: is defined as a chronologically ordered sequence of sessions together with the available information about the users.

Depending of the context, several variants of the above mentioned abstractions could be developed. As an example, we describe here a specialization developed in order to evaluate the usefulness of the results returned by a search engine:

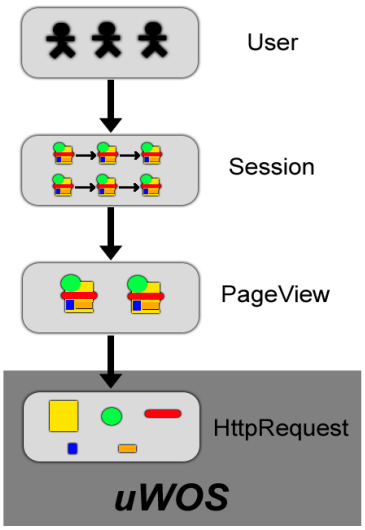

Fig. 3. Usage Data Abstractions

- Q-Session: is defined as a maximal subsequence of a Session, which starts with a query to a search engine (usually explicitly represented as a parameter in the URI requested to the search engine) and does not contain any other query. The Q-Sessions are represented as couples (Q,S), Q being the submitted query string and S being the Q-session originated from that query. Some applications of Q-Sessions will be outlined in Section 5.

The data abstractions listed above represent simple examples of higher level concepts built upon raw data (the web usage logs). Such concepts are computed by a set of preprocessing algorithms organized into a library that is accessible by all the components of the WOS. Such algorithms work on the basic classes provided by the µWOS, and implement standard and well-established web log preprocessing heuristics., which can be summarized in the following way:

- Each web page (HTML, PHP, etc.) constitutes a separate PageView, which includes also all its embedded objects (usually multimedia components). Such "embedding" relation can be obtained analysing all the HttpRequests from the same client, either by trivially exploiting the Referrer field, if available, or by selecting all multimedia resources requested after the web page within a time window. In order to treat framesets, PageViews closer to each other than a given time threshold, are collapsed and assumed to be frames of the same set.
- User sessions are extracted by imposing that any couple of PageViews requested by the same client having dates closer than a given time threshold must be part of the same session. The user sessions are then obtained by

computing a simple, linear transitive closure of such relation between PageViews, and keeping them sorted w.r.t. the chronological order of their corresponding HttpRequests.

- Q-sessions are obtained in the same way as user sessions, but selecting only the relevant sub-sessions (those starting with a search query) and extracting the query string from their first PageView.
- Finally, each user is described by trivially collecting all his user sessions and some simple derived information, such as the average length of sessions, the set of browsers and protocols adopted, his IP number, etc.

As for the μWOS components, the new classes provide a simple interface with methods for accessing their content and for navigating the tree structure of concepts they form.

2.3.2 Labelling Content Data with traffic-based semantic information

As the above section demonstrates, reorganizing the usage information contained in the web logs into a significant structure of concepts is a quite easy task (provided that some approximation is allowed), both at the design and at the implementation levels. For content data, including both the documents that populate the Web and their link structure, the situation is completely different. Beside the purely administrative organization of web pages into levels of domains/sub-domains and directories/sub-directories, which is trivially obtained by analyzing URIs, it is difficult to design meaningful, consistent and computable conceptualizations of the raw data. As an example, mapping web pages to concepts like *topics* (sport, economics, etc.) or *user communities* (business professionals, Linux home users, etc.), can easily result into arbitrary choices and presents several implementation issues, due to the intrinsic vagueness of the classes of concepts and to the high inter- and intra-document heterogeneity of web pages.

A large research area which in the last years yielded successful – yet far from definitive – results in such direction is the Search Engines field, at the top of which stand several proposals of algorithms and methods for web pages *ranking,* such as the well-known HITS and PageRank algorithms. Such methods essentially analyze the content of web pages and their link structure in order to induce some kind of mapping between web pages and high level concepts like *interesting page*, *page relevant to query Q*, and so on, usually yielding the strength of such association, e.g., the *importance* rank provided by PageRank.

An important characterization of web pages consists in the distinction of *real content* pages as opposed to *trash content*. The most representative categories of the second type are advertising sites and counter servers: both categories provide some kind of service for other web pages (respectively, ad banners and visitors statistics) but do not contain any meaningful information for the user which is navigating the web. From the viewpoint of a web searching service, they are useless segments of Internet that degrade its performances, by consuming extra time and bandwidth during its web crawling activity and by adding noise during the query answering phase.

In this section we describe a WOS algorithm introduced in [7], which implements an heuristics for recognizing some categories of advertising and counters sites. Actual

methods, either proposed in the search engine literature or applied in commercial systems, are usually based on traditional document classification techniques (i.e., content-only, text analysis heuristics) or on more sophisticated linkage-based approaches, such as the *TrustRank* propagation model proposed in [8]. On the opposite, the idea of our approach is that by analyzing the users' behavior (stored in the usage data) it is possible to induce some semantic information about the content of the visited pages, especially when extreme behaviors occur, as it will be explained later in this section. The algorithm makes use of the usage data collected at the level of a proxy server and stored in the μWOS, adds an intermediate data structure (the Traffic Graph) to the WOS which can be exploited by other WOS algorithms, and finally provides an approximated mapping from web sites to the two categories *content* and *advertising/counters*, also added to the WOS and visible to other algorithms.

a. Usage-induced link structure of the Web

The canonical view of the Web adopted in the search engine field, also implemented in the Citation class of the μWOS, is essentially based on its link structure, which is represented as a simple oriented graph, the web pages being its vertices and the links the (oriented) edges between them. In the context of traffic data, each transition from page *A* to page *B* in a user session corresponds to a link contained in page *A* that points to page *B*. The set of page transitions that can be collected by tracing the web activity of a community of users, then, essentially represents a sub-graph of the whole Web, each vertex corresponding to a page and each edge to a page transition. The same page transition usually occurs more than once, so each transition is associated with a frequency weight. This leads to the following definition:

Definition 1 (Traffic Graph) Let $S = \langle(1,r_1,d_1),\ldots,(N,r_N,d_N)\rangle$ be a sequence of N web requests monitored in a web community over a given time interval, each request being composed of a referrer r_i (i.e. the page where the request is originated) and a destination d_i (i.e. the page requested). Then we define the Traffic Graph of S (or simply Traffic Graph, when S is clear from the context) as a triple TG = (V,E,T), where:

$V = \{r_1,d_1,\ldots,r_N,d_N\}$
$E = \{(r_1,d_1),\ldots,(r_N,d_N)\}$
$T_t(p,q) = |\{(n,p,q) \in S\}| \text{ for } p,q \in V$
$T_p(p) = \sum_{(p,q) \in E} T_t(p,q) \text{ for } p \in V$

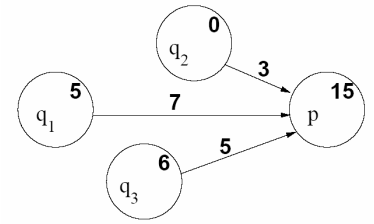

Fig. 4. Example of Traffic Graph

The vertices V are called *pages*, while the edges in E are called *transitions*. Finally, function $T_p : V \rightarrow \mathbf{N}$ is called the *Traffic of pages*, while $T_t : E \rightarrow \mathbf{N}$ is called the *Traffic of transitions*. Hereinafter, the subscripts of the traffic functions are omitted when clear from the context.

Figure 4 depicts (a segment of) a sample Traffic Graph for four pages, where $T(q_1)=5$, $T(q_2)=0$, $T(q_3)=6$, $T(p)=15$, $T(q_1, p)=7$, $T(q_2, p)=3$, $T(q_1, p)=5$, while $T(p, q_1)$, $T(p, q_2)$, etc. are not defined. Notice that, from the mathematical definition, the traffic of a page can be higher than the page traffic.

b. Characterizing web traffic

The *traffic graph* defined above summarizes the traffic load over the web, and some interesting patterns can be defined and retrieved directly on such summary, such as high traffic links and high traffic pages. However, a more sophisticated approach can be obtained evaluating the *success* of web pages, i.e., how often the references to some pages are followed by users (even not voluntarily). We can formalize this notion by defining the following parameter, which computes the *relative* incoming traffic of web pages:

Definition 2 (Relative Traffic) Given a traffic graph TG = (V,E,T), the *relative traffic* R(p) of a page $p \in V$ is defined in the following way:

$$R(p) = \Phi_{(q,p)\in E} R(q,p) \quad \text{for} \quad p \in V$$

where

$$R(q,p) = \begin{cases} 1 & \text{if } T(q,p) > T(q) \\ T(q,p)/T(q) & \text{otherwise} \end{cases}$$

and $\Phi : 2^R \rightarrow R$ is an aggregation operator over sets of real numbers, e.g., average, minimum and maximum.

A similar representation of web traffic can be also found in [9], where estimated probabilities conceptually equivalent to relative traffic are associated to links.

In several cases, reasoning on single pages can be too fine-grained an approach. Therefore, it is useful to extend the definition above to different abstraction levels, such as *web sites* or *web domains.* This can be achieved by simply collapsing the web pages belonging to the same abstraction, summing up their traffic measure and deriving the corresponding relative traffic.

c. Integrating the traffic graph in the WOS

Usage-induced link structure and additional parameters are easily implemented within the WOS. The μWOS and the abstractions presented in the previous section (PageViews, Sessions and Users) can be extended to store and handle the traffic information. In particular, a new usage-oriented version of the Citation class has been provided, which represents the visit of an host from another one, in a referrer-destination format. As mentioned, different abstraction levels can be adopted for traffic analysis, and each level will require managing objects of different granularity. At the lowest level, single pages will be considered, by referring to the corresponding PageView objects. Adopting a higher

abstraction, on the contrary, will require to collapse groups of pages, reducing the size of the traffic graph to be managed. In this situation, the same usage-oriented Citation class is used, where each *higher-level* object represents a set of (corresponding) single pages and contains an aggregation of their traffic function.

d. A Heuristic for high traffic advertising/counters

As already mentioned, we focus on the discovery of two classes of web pages, which represent two classical examples of *uninteresting* pages from the viewpoint of a user navigating the web:

- Advertising: sites containing only pure advertising. In particular, here we will focus on high-traffic advertising sites, which have a strong visibility from other sites, which usually link them by means of pop-up windows.
- Counter services: hosts containing scripts (CGI or other methods) that collect statistics on the access to web pages. Monitored pages usually contain links or scripts that autonomously invoke such services.

We selected the *host* as a suitable abstraction level for the computation of the traffic graph. Advertising services, for example, are usually implemented on the level of hosts (real and virtual), while the internal organization of their single pages can be confusing and dispersing (in terms of web traffic).

By an experimental analysis of real usage data, we noticed that the presence of advertising hosts and counters is particularly dense on high values of the $R()$ function. From such observation, we drew the following hypothesis: the higher is $R(h)$ for a host h, the higher is the probability that h contains advertising or counters. Therefore, we have used the $R()$ values to rank hosts, filtering out the top N values, with N being a parameter of the method. An additional peculiarity of the hosts we want to spot is their link popularity: in fact, advertising and counter services are usually delivered on *impressions* on several different sites. As a consequence, from a web link structure perspective, several sites link the same advertising host. Therefore, any reasonable candidate advertising/counter host should be linked by at least n other hosts, n being another parameter of our heuristic. The above observations can be summarized into the following algorithm, which extracts a set of potential advertising/counter hosts from a dataset of web requests:

Algorithm AdvertisingHosts(S,n,N,Φ)
Input: a sequence S of web requests, two integers n and N, an aggregation operator $\Phi \in \{\min, \max, \text{avg}\}$.
Output: A list of hosts.

1. Build the Traffic Graph TG from S;
2. Compute the Abstract Traffic Graph TG' = (V',E',T') from TG using abstraction A = (ψ, α), where ψ = {hosts} and α (p) = "host of p";
3. Let H = {h ∈ V'| h linked by at least n other hosts };
4. For each h ∈ H : Compute R(h);
5. Let O = list of all h ∈ H, sorted by R(h) in descending order;

6. Return O[1 : N];

Preliminary experiments performed on small datasets show that the heuristic produces a high-precision classification only over the sites with very high relative traffic – around the top 1% of the monitored sites – while the precision quickly degrades with smaller values. Such results, however, are expected to significantly improve with larger datasets, letting the heuristic to yield reliable results also for smaller traffic sites.

3 Populating the WOS

One of the most important features of the Web Object Store is represented by its capability of managing different kinds of data (i.e. content, structure and usage). It is obvious, anyway, that the WOS (in particular the µWOS) has to be populated with content, structure and usage data before it can be used. There are several ways to feed the µWOS repositories. We identified four main sources of information:

1. Pages and linkage information coming from a *Web Crawler* (possibly focused on a particular topic or group of topics).
2. Information about accesses to Web sites (or portals) contained within the *Access Log Files* of the servers they are hosted on.
3. If we are allowed to access the data coming from the usage of a Web Search Engine (i.e. *Query Logs*) we can integrate them within the WOS too. Note that we may possibly collect information about issued queries to various Search Engines even if we host a Proxy server and we filter out the queries from all the users' requests.
4. Speaking about Proxy servers, another source of information is represented by the *Proxy Logs*.

3.1 Web Crawlers

A Web Crawler is a sort of mobile agent scouring around Web sites getting all the kind of information that is able to discover and, more important, that is allowed to download. Usually, a Web Crawler is designed according to a number of specifications that should be followed in order to guarantee a *fair* use of Internet resources, such as avoiding the massive downloads from the same server, adhering to the constraints specified in *robots.txt* files, etc.

During these years, a large number of different approaches for building Web Crawlers have been proposed. The aim of such systems, in all the cases, is to download the largest number possible of pages from the Web. This task is a very difficult one, since it is strictly related with the possibility of discovering them. To this purpose, they have to face with the so-called *Hidden Web* that is composed by those pages that are *dynamically* built upon a user request. The Hidden Web is the opposite of the Surface Web that, instead, is composed by those pages that are statically *referenced*.

Commento: Qua ci metto un riferimento oppure spiego il signficato di corret and fair? MIRCO: Spiegare, spiegare!

Web Crawlers are responsible for building the underlying database of Web Search Engines. They are committed to download both the content of the Web pages, and the linkage structure given by the Web graph (i.e. the graph obtained by considering web pages as nodes and the links as its edges).

In general, WOS content and linkage structures should be fed by using a Web Crawler. As we will see in the next section since the WOS targets a community of Internet users, we are planning to build a sort of *Focused Crawler* that would automatically download pages interesting for the community itself. In addition, it will postpone pages that are likely to be not relevant for that community.

3.2 Web Servers Access Log Files

Usually Web Servers write log entries in a format known as the *Common Log Format* (*CLF*). This standard format can be produced by many different web servers and read by many log analysis programs. The log file entries produced in CLF will look something like this:

```
146.48.83.47 - frank [10/Oct/2000:13:55:36 -0700] "GET
/apache_pb.gif HTTP/1.0" 200 2326
```

Each part of the access log is described below:

`143.48.83.47` This is the IP address of the client (remote host) which made the request to the server.

`-` This field indicate the value returned by calling the identd service on the client machine. Usually clients do not accept identd requests so, to indicate that the requested piece of information is missing, we put the "*hyphen*" symbol.

`frank` This is the user ID of the person requesting the document as determined by HTTP authentication.

`[10/Oct/2000:13:55:36 -0700]` The time that the server finished processing the request, wriiten following the standard format [day/month/year:hour:minute: second zone].

`"GET /apache_pb.gif HTTP/1.0"` The request line from the client.

`200` This is the status code that the server sends back to the client.

`2326` The last entry indicates the size of the object returned to the client, not including the response headers. If no content was returned to the client, this value will be "-".

It is clear that each line of the access log contains a lot of information. It is also clear that organizing this information in a more structured way will result in an improvement of the quality of the services offered by the WOS. As shown in Section 2.3.1, we provide an internal tool which is able to extract information about *PageViews* (i.e. complete information about a Web page), *Sessions* and so on.

There are many examples of software, often implemented as research prototypes that show possible analysis made on Web Server Logs. Among them, we can find systems for

producing statistics over accesses (e.g., Analog [15]) and, more recently, Web Recommender Systems. In the latter, Web logs are analyzed in order to extract users classification models. These models are then applied to requests made by future users in order to advertise, in a focused way, commercial products, links to Web pages, and so on. In Section 4 Suggest, an example of recommender system, is shown. Suggest has been built by using the facilities offered by the WOS.

3.3 Web Search Engines Query Logs

Web Search Engines keep track of the queries submitted by the users by storing information about them in a log file (i.e. the *Query Log*). Each company uses its own format to store information in the logs. Anyway, there is information that is mandatory:

- The *query topic*, that is the keyword (or phrase) submitted by the user.
- The index of the first result wanted.
- The maximum number of results a user would like to obtain from the query.

Obviously there may be other information such as, for example, an identifier of the user that submitted the query, the result chosen by the user (if the Web Search Engine adopts a sort of "*click tracking*").

The WOS may use this information to infer the usage pattern of a Web Search Engine or, in the case the WOS is used within an Internet Community, to devise Crawling strategies focused on the most important topics on which the community itself is oriented.

3.4 Proxies Usage Logs

As in the case of the Web Server logs, the proxy log contains information about requests made from users of a large community to Web Sites. In contrast to Web Server logs, at the proxy level it is possible to detect the requests addressed to several different web sites, thus allowing to follow the full, usually inter-site, users' navigation paths. Proxy information include also requests made to Web Search Engines and may be used to keep track of the so-called *Q-Sessions* introduced in Section 2.3.1, i.e., navigational sessions that start with a query issued to a Web Search Engine and go on by following a link from the list of results returned.

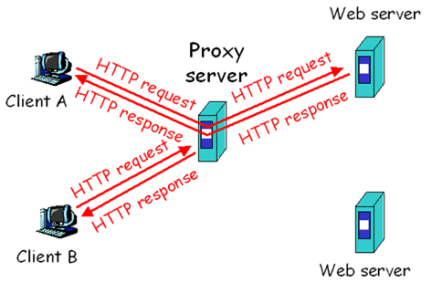

Figure 5: Schema of a proxy server traffic

Usually the format used to represent proxy entries is the same CLF (Common Log Format) used by Web Servers. Thus, the same analysis tools used for Web Servers can be used to analyze proxy usage data as well.

3.4.1 Traffic monitoring

As an alternative source of usage data coming from a proxy server, we can consider the following, which is supported by the μWOS and has been used in preliminary experiments. In order to populate the usage-based segment of the WOS (i.e., the HTTPRequest repository), usage information can be extracted from the packet level data obtained from the traffic of a proxy server, by means of packet sniffing techniques. E.g., in some experiments we performed, the network tool Ngrep was used to filter the interesting traffic information by specifying regular expressions to match against data payloads of HTTP requests contained in TCP packets addressed to port 80 of Web servers.

The following example shows a fragment of matched information:

```
T 2003/07/21 09:18:09.877083 216.228.97.23:80 ->
131.114.3.xxx:1247 [A] HTTP/1.1 200 OK..Date: Mon, 21
Jul 2003 08:09:47 GMT..Server: Apache/2.0.47 (Unix)
mod_ssl/2.0.47 OpenSSL/0.9.7b DAV/2 PHP/4.3.2..Last-
Modified: Sun, 20 Jul 2003 22:15:13 GMT..ETag: "81c8-
22b7-f 4a93e40"..Accept-Ranges: bytes..Content-Length:
8887..Keep-Alive: timeout=15, max=92..Connection: Keep-
Alive.. Content-Type: text/html; charset=ISO-8859-
1....<!DOCTYPE HTML PUBLIC "-//W3C//DTD HT ML 4.0
Transitional//EN">.<html>..<head>
```

This continuous flow of raw information has been filtered by means of a data cleaning module, which extracts only the relevant data for our objectives, and re-organize them in a server-like form. As an example, the result obtained by applying the data cleaning task to the fragment shown above, is the record composed of the following fields:

2003/07/21 09:18:09 131.114.3.xxx
http://www.xfce.org/index.html http://www.xfce.org/en/

The meaning of the fields is the following:

- A time-stamp of the request: 2003/07/21 09:18:09
- Client IP: 131.114.3.xxx
- URI of the requested resource: http://www.xfce.org/index.html
- URI of the resource from which the request was originated: http://www.xfce.org/en/

In our experiments, we collected long periods of proxy-level IP traffic originated from "Centro Servizi SERRA" network (covering the *unipi.it* domain). This network segment allows extracting many-to-many web interactions, from thousands of clients belonging to the academic network of Pisa to millions of hosts (the whole Internet). This massive stream of data (~1GB/day raw data) contains sufficient information in order to extract precious usage abstractions of user's behavior, such as:

- Sessions originated from queries to search engines.
- Sessions covering many different hosts.
- Traffic graph and abstractions of the explored Web

4 A Sample WOS Application: SUGGEST

As explained above, the major goal of the WOS programming interface is to simplify the development process of Enhanced Content Deliver applications.

From this point of view the most important component of the WOS is represented by the μWOS layer which represents the minimum core set of functions that allow the development of those high level algorithms manipulating the core set of Web data (i.e. Usage – Structure – Content).

4.1 Web Recommender Systems

The huge quantity of information available on the Web comes from different sources. There are firms and institutions that exploit the Web to conduct their business, customers that daily use the Web to perform every kind of transactions and people that simply browse through pages of their interest. The presence of all these categories has led to the need to make techniques and tools able to accurately extract, filter and select Web information. Web mining methods [2], and Web Usage Mining (WUM) in particular, represent a promising way to tackle such web information extraction problem.

WUM applications typically extract knowledge by analyzing historical data such as server access log files, browser caches or proxy logs. WUM techniques are important for several reasons. It is possible to model users behavior and, therefore, to be able to forecast their future movements. It can be useful to personalize the content of Web pages, to improve the Web server performance, to structure a Web site according to the preferences expressed by users, to help the business to carry out a specific users' target.

A typical application of WUM is represented by the so-called *Personalization* or *Recommender* systems[3]. These kinds of systems allow adapting a Web site to the users' needed. In practice, the WUM personalization process is structured according to three phases: *Preprocessing*, *Pattern Discovery* and *Pattern Analysis* (see Figure 6). Such systems provide mechanisms to collect information describing the user activity and to elaborate this information (Preprocessing). Using these systems it is possible, for example, to determine the number of the server accesses, the pages requested, the interval time between different user sessions, and the IP address of the Web server users. This information are then elaborated to synthesize user profiles (Pattern Discovery) that can be

used to provide user personalized navigational information (Pattern Analysis). As an example we can think about the Amazon recommender system: once a user starts navigating through the books she/he starts to receive recommendations on potentially interesting books selected on the basis of the previously seen ones.

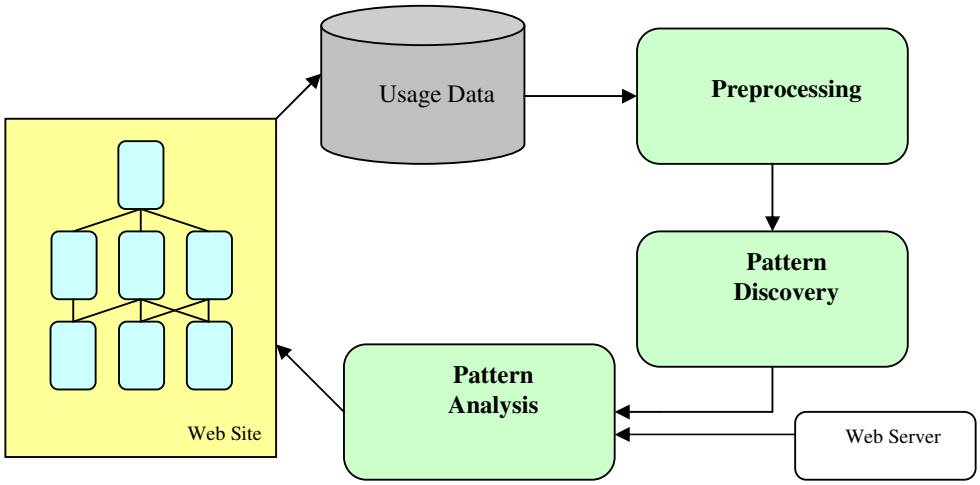

Figure 6: Tipical operations carried out by a Web Recommender System

4.2 Suggest

The Web recommender system Suggest generates recommendations to users by exploiting a clustering of the user's sessions obtained from the proxy logs [1]. In Figure 7 the stages carried out by Suggest are represented.

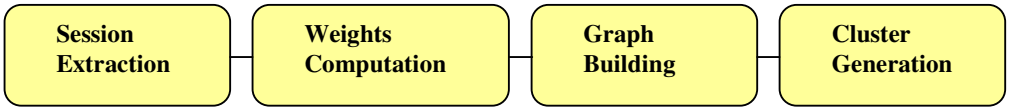

Figure 7: Suggest workflow

As said before our WOS framework heavily simplifies the development of Web applications. In this case, for instance, we can get rid of the first three phases since the data structures needed are already offered by the WOS. In particular, the first phase (Session Extraction) is directly supported by the μWOS while both the Weights Computation and the Graph Building phases basically consist of constructing a simple Traffic Graph, which is already provided by the WOS (see Section 2.3.2). Thus, we just have to generate the clusters from the graph detecting its connected components using a simple BFS visit of the graph itself.

Using the WOS facilities it has been sufficient to implement two simple methods aimed to produce a new persistent class called *sClusters* defined by the followingdeclaration:

```
PERSISTENT_CLASS (sCluster,
                      (KEY(PageView_ID, Field::primary),
                       FIELD(Cluster_ID),
                       FIELD(Session_Count),
                       VARFIELD (_sCluster_Elements, 100)
                      )
);
```

Such declaration syntax essentially corresponds to the META and the CLASS_DESCRIPTOR predicates, adopted respectively by the IXE[4] and Gigabase Object-Relational Database Management System[16]. At the present, the WOS implementation can work with both systems.

The meanings of the fields are quite intuitive. In particular, the key, `PageView_ID`, represent the identifier of the PageView to which this entry is referring, and `Cluster_ID` is the identifier of the cluster containing this PageView. The other fields may be used by future applications to enhance the clustering algorithm or to implement different applications, like for instance, an Intelligent Caching algorithm[17].

The abovementioned methods are used to fill-up the structure and are, respectively: `makelist()` which builds a list of PageViews, each associated with a list of sessions which contain the referred pages, and `makeclusters()` that actually builds the persistent sCluster structure starting from the previously built list. The `makeclusters()` method first materializes in-memory the graph of sessions starting from the previously created list and assigns the appropriate weights. Then it prunes the graph using the two heuristics described in. The method, then, proceeds by executing a BFS visit on that graph to identify the clusters (i.e. the connected components).

An online procedure embedded into the Web Server exploits the resulting clustering structure in order to classify the currently active user sessions and to produce a list of recommendation on the basis of the assigned user category.

Due to the intuitive and powerful API offered by the WOS, the entire code to implement the clustering (offline) phase of Suggest took no more than 500 lines of source C++ code.

5 A large scale Application: Search Services for User Communities

In this section we will outline a complex application that can be realized using the WOS structures and API.

The application considered is a Web Search Engine (WSE) for large communities of users[2]. This means that the WSE should focus all the operations on the topics of interest of the community, trying to improve the accessibility to relevant sources of information and to exclude the irrelevant ones. The WOS can potentially provide a great help to reach such objective, since its integration of usage and content data makes it possible to combine information on the navigation habits of the community, the web contents they show interest in, their reactions to the way such content is organized, and direct feedbacks on the searching service provided. Furthermore, it may help in defining new structures and novel algorithms supporting the engine activities.

5.1 Community-oriented Web Search Engine

Excluding Web Search Engines focused on a particular country or state, to the best of our knowledge a search facility focused on a particular community of Web users has not been proposed, yet.

We could define a *Community-oriented WSE* as a WSE that manages (in principle) only documents related to the topics of interest of a Web Community (as for example the users of a certain department). What are the main differences between a traditional WSE and a community-oriented one? In a department one could identify many different categories of users, but almost all of them are expected to be mainly interested in the same topics, related to department itself (e.g. biology, computer science, natural sciences, engineering, and so on).

On the contrary, in a traditional, general-purpose WSE we cannot make any assumption on the topics about which the users will submit their queries, so the WSE has to collect everything on the Web potentially interesting for somebody on any topic, thus enlarging its indexes as much as it can. Furthermore, since usually a Community-oriented WSE is managed within the community itself, it can make assumptions on several aspects of its working. More important, since a community usually accesses the Web through a single access point (e.g., a Proxy, or a router) a community-oriented WSE could use Web usage information to infer usage patterns and also, for example, to discover previously unknown URLs.

Obviously, since we would like to make assumptions about usage patterns, we are considering "large" users' communities. In this case it is difficult to define the word "large", anyway we assume that a community composed of a thousand of users can be considered "large" enough.

Since we are referring to communities and not to the entire Web, we have to restructure some traditional WSE components. In particular, we have in mind two kinds of extensions/refinements:

1. A Focused Crawler, which tries to spider only potentially interesting portions of the Web (w.r.t. the community's interests);
2. A Focused ranking schema, which gives emphasis to Web pages and sites apparently closer to the community's interests.

---

[2] For an introductory description of Web Search Engines, please see [5].

In the following sections these two aspects will be described in more details.

5.2 Focused Crawler

In recent years there have been many attempts to design and implement focused crawlers. However, so far such proposals obtained only a limited success. The reasons seems to lie mainly in the difficulty of recognizing the adherence of Web pages to a set of predefined topics by just analyzing links and contexts from the already downloaded pages **[6]**.

The seamless integration of content and usage information provided by the WOS enables the renforcement of the traditional, content-only approaches with the available information on the users' activity.

The main assumption we make in our approach is the following: the Web pages visited often by the users and the navigation paths they follow provide a reliable definition of the interests of the community. That means, as an example, that if many of the users visit some Web page, it will be (probably) assumed to belong to the interesting topics of the community whatever is its real subject; in the same way, if they almost systematically avoid the links which point to some Web page, such page will be (probably) considered out-of-topics.

In this work, we suggest three mechanisms which can be combined together in a collaborative way, for enabling a more reliable crawler which is focused on the interests of a users' community:

1. dynamic, usage-based topics updating;
2. web sites/pages selection via labels propagation;
3. trash sites detection and filtering.

In the rest of this section we briefly describe them.

5.2.1 Topics updating

A constant requirement of any focused crawling system is the necessity of compiling a list of topics we are interested in. Sometimes, the user provides such list only implicitly, by manually selecting a set of relevant documents. Such list of topics, then, will be the comparison base for the successive selection of relevant Web pages. It is important to notice, however, that listing and describing the topics of interest of a community in an exhaustive way is in general a very hard task. Missing a relevant topic and providing poor topics specifications, then, are high-probability risks which should be taken into account in the design of any should-be reliable focusing system. Moreover, users are human being, and as such their interests change/grow along time. As a consequence, any list of topics is apt to get outdated at some time, and therefore a re-alignment mechanism between listed and actual topics should be designed to update such list, possibly in an automatic way.

The problems listed above can be tackled by keeping track of the users' behavior: following the assumption mentioned at the beginning of this section, we can deduce that frequently visited documents should belong to the interesting topics list and, if that is not

true, the list should be expanded. Such idea can be further developed leading to the method sketched below:

Algorithm UpdateTopics(L, `t`, `f`)
Input: A list of topics L, a time window **`t`** and an integer threshold `f`.
Output: An updated list L' of topics

1. Select all PageViews requested within the time window **`t`**
2. Compute the access frequency of each PageView and Filter out those below a predefined threshold
3. Associate each PageView with the set of possible topics in the list it can be related to; each topic will be associate with the number of PageViews related to it, while PageViews not related to any known topic are collected in the set `W`
4. [Deletion of outdated topics:] Select all topics associated with less than `f` PageViews
5. [Extension of topics list:] For each PageView in `W`, extract a set of related topics; each new topic related to at least `f` PageViews in `W` is added to the topics list

The (approximated) extraction of the relevant topics for a document can be performed adopting any of the methods described in literature – e.g., based on Latent Semantics Indexing [10] – and will not be discussed here. In literature, an alternative usage-based approach to automatic topics detection can be found in [12], where a clustering method for user behaviors is implemented, by modeling such behaviors as hidden Markov models which corresponding hidden states essentially represent the topics to discover.

5.2.2 Labels propagation

Several categories of sites, the most popular being adults-only sites, can be easily characterized by means of a limited set of keywords, which are also the same that are usually adopted by users when making searches with search engines. That is the usual approach adopted by child-protecting browsing systems, which evaluate the suitability of web pages by checking the presence of dangerous keywords within the document. One of the results of this phenomenon is that a relevant percentage of user queries submitted to search engines are expected to strongly characterize the general topics the users are looking for. If the search is successful, then, it is highly probable that the navigation session originated from the search query – or at least an early portion of it – will be devoted to that same topic. Moreover, the (chronologically) closer is a page request to the search query, the higher is the probability it belongs to the query topics. These ideas can be easily summarized into the following algorithm, which makes use of the Q-session abstraction introduced in Section 2.3.1:

Algorithm PropagateLabels(`t`, k)
Input: a time window **`t`** and an integer threshold k
Output: a partial labelling of PageViews

1. Select all stored Q-sessions which began within the time window **t**
2. For each Q-session (Q,S), with S=(PV_1, ..., PV_n):
3. if all significant keywords of Q belong to domain D
4. then label PV_1,..., PV_m as member of domain D, where m=min(k,n)

Each domain D represents a topic by means of the set of keywords that strongly characterize it. In this simple approach, the k closer pages (i.e., PageViews) requested after the search query are simply accepted as members of the selected domain (if any). More sophisticated variants may assign a weighted membership to each PageView PV_i, which is computed as a function of the number of matching keywords of Q w.r.t. domain D, and the distance "i" from the search query. Such function is expected to monotonically increase w.r.t. the first value, and decrease w.r.t. the second one. This approach has many similarities with Aggarwal's *Collaborative Crawling* [11], where *seeding* pages are manually selected instead of using queries to search engines, and a more complex propagation strategy is implemented.

5.2.3 Trash detection

In Section 2.3.2, a WOS algorithm was introduced, aimed at detecting some classes of useless web sites, namely advertising and counter sites, in order to avoid crawling and/or indexing them. In addition to such approach, which can directly be exploited by the crawler of a community search engine to prune the web search space, a complementary one can be adopted, exploiting again the information stored in the Traffic Graph.

As an extremely high relative traffic is usually the consequence of automated mechanisms, such as advertising and counter banners, an extremely low relative traffic is usually the result of uninteresting contents: the users almost systematically avoid the links which point to the web page under consideration, and so it is highly probable that the page does not belong to the topics of interest of the users. In order to properly quantify such phenomenon, the relative traffic of every link should be compared to its *expected* relative traffic, i.e., the value it should have if users' clicks over the links of a web page followed some given probability distribution. In the field of search engines, the usual reference distribution is the uniform one (an important example is the PageRanking algorithm), so that each of the *n* links in a web page has an apriori probability of being followed equal to *1/n* and, then, a relative traffic also equal to *1/n*. A simple method can be outlined for assigning web pages with a priority value, to be used by the crawler at the moment of choosing the next page to visit: for each candidate page *p* (i.e., not-visited pages which are linked by one or more visited pages), the value $P(p) = R(p)*n$ is computed, $R(p)$ being the relative traffic of *p*; pages with a too small $P(p)$ are removed from the candidate list; the remaining pages, finally, are sorted in decreasing order w.r.t. $P(p)$.

5.3 Community-Based Ranking Functions

One of the most important components of a WSE is the Ranking Module **[5]**. The Ranking function, which is at the basis of the working of the Ranking Module, tries to evaluate the importance of a given document (possibly considering user queries).

There are two general categories of ranking functions: *content-based* and *link-based*. The former category measures the importance of a document on the basis of its content. The

well-known *TFxIDF* proposed by Salton *et al.* in [13] is one of such measures. The latter take care of only the linkage structure of the Web. More precisely, if we make the assumption that a page *P* links to a page *Q* if and only if the author of *P* retains important the content of the page *Q* then we could devise a ranking schema based on the number and the quality of the links connecting to a page.

The two most important examples of such ranking systems are: *HITS* and *PageRank*.

The *HITS* algorithm is proposed in [14]. In *HITS*, each Web page w_i has both a *hub* score and an *authority* score. Roughly speaking, the first measures the usefulness of a page on giving important suggestions (links) to users. The latter, instead, measures the quality of the information contained within w_i.

The PageRank algorithm [5] statically evaluates the importance of a Web page by considering the number and the relevance of links connecting to it.

In a more formal way, the PageRank values for each Web Page can be obtained using an iterative algorithm that searches for the principal eigenvector of the adjacency matrix of the available portion of the Web. The equation underlying the whole process is quite simple. Given a page w_i, the $PR(w_i)$ is given by:

$$PR(w_i) = \varepsilon + (1-\varepsilon) \times \sum_{l_{j,i} \in E} \frac{PR(w_j)}{out-degree(w_j)}$$

where ε is a *dampening factor* usually set between *0.1* and *0.2*; *n* is the number of nodes of the web digraph and out-degree(w_i) is the number of edges originating from a web page w_i.

We could say that PageRank tries to mimic the users' behavior by simulating a typical navigator surfing the Internet. From this point of view, a user goes through a link with probability *(1 - ε)*, while gives up and visits a random page with probability equal to ε.
Usage information could be useful for reaching a better approximation than PageRank. We have, in fact, actual information about users movement through the Web. Nevertheless, if this information is collected from a Web Proxy, for instance, it is related to the pages visited by a particular group of users.

A very similar approach to compute PageRank for small Web sites is described in [9]. Since PageRank is an iterative computation over a probability transition matrix, how the initial distribution is chosen will influence the outcoming of the algorithm. Personalizing the PageRank computation with the knowledge extracted from the Proxy log is quite straightforward. Instead of considering all the pages having the same probability of being accessed, we can assign different probabilities on the basis of the information contained within the Proxy log. Roughly speaking, we are pushing usage information down into the computation of the PageRank scores.

We could also exploit the information coming from the Proxy logs by combining the PageRank value with a measure of importance derived from usage information. Practically speaking, the usage-based measure should be a sort of estimation of the probability of accessing a page obtained by dividing the number of accesses of a page by the total number of accesses recorded in the log. If we call *PR* the PageRank value and *UR* the rank obtained by the Proxy log we can devise a novel ranking function called *MixedRank* (*MR*) by a linear combination of *PR* and *UR*: $MR = \alpha \cdot PR + (1-\alpha) \cdot UR \qquad 0 \le \alpha \le 1 \cdot$

Furthermore, exploiting Q-Sessions (see Section 2.3.1) we can think to a sort of automatic relevance feedback aimed at incrementing the ranking of accessed pages, in the style of [9].

Obviously, the above observation are just hypothesis that need to be accurately verified and opportunely extended. We are quite confident that the information contained within the WOS along with well-tuned ranking functions will improve the precision of the search mechanisms when applied to Community-based contents.

6 Conclusion

In this paper we presented the *Web Object Store* – WOS – a web data management system specialized in dealing with web content, structure and usage data. The WOS is aimed at effectively supporting the development of User-enhanced Web-Access services, and therefore strictly adheres to three main requirements summarized in the beginning of this work. Firstly, it provides useful abstractions of web-related concepts, and enables their direct deployment at different levels; secondly, it offers a comprehensive repertoire of pattern and model types, to represent the extracted knowledge/semantics to be used in constructing UWA services; finally, it provides efficient data structures for both data and models, and guarantees persistency over time.

The sample algorithms and applications described within this paper clearly show the effectiveness of the WOS approach for developing UWA services in a clean and concise way, and strongly motivates its future employment in the development of real UWA services. In particular, in the near future we wil pursue the research lines shown in the large-scale, community-oriented application outlined in Section 5.

References

[1] Ranieri Baraglia and Fabrizio Silvestri. *An Online Recommender System for Large Web Sites*. In Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2004). Bejing, China. September 20-24, 2004. pp 199-206.

[2] Soumen Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan-Kauffman, 2002.

[3] M. Eirinaki and M. Vazirgiannis. *Web mining for web personalization*. ACM Trans. on Internet Technology, 3(1):1–27, February 2003.

[4] Giuseppe Attardi, Antonio Cisternino. *Template Metaprogramming an Object Interface to Relational Tables*. Reflection 2001: 266-267.

[5] Sergey Brin and Lawrence Page. *The anatomy of a large-scale hypertextual Web search engine.* Proceedings of WWW7 (1998), 107-117.

[6] S. Chakrabarti, M. van den Berg and B. Dom. *Focused crawling: A new approach to topic-specific Web resource discovery*. Proceedings of WWW8, Toronto, May 1999.

[7] V. Bacarella, F. Giannotti, M. Nanni, and D. Pedreschi. *Discovery of ads web hosts through traffic data analysis*. Proceedings of the 9th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD 04) (2004).

[8] Z. Gyongyi, H. Garcia-Molina, J. Pedersen. *Combating Web Spam with TrustRank*. Technical Report, Stanford University, 2004.

[9] G.-R. Xue, H.-J. Zeng, Z. Chen, W.-Y. Ma, H.-J. Zhang, C.-J. Lu. *User Access Pattern Enhanced Small Web Search*. In Proceedings of SIGIR2003. July 28 - August 1, 2003, Toronto, Canada.

[10] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, R. A. Harshman. *Indexing by Latent Semantic Analysis*. Journal of the American Society of Information Science vol. 41 n.6, 1990.

[11] Charu C. Aggarwal. *Collaborative Crawling: Mining User Experiences for Topical Resource Discovery*. SIGKDD 2004.

[12] A. Ypma, T. Heskes. *Automatic Categorization of Web Pages and User Clustering with Mixtures of Hidden Markov Models*. In WEBKDD 2002, LNAI 2703, 2003.

[13] Salton, G; Wong, A.; Yang, C. S. *A vector space model for information retrieval*. Communications of the ACM, 18(11):613-620, November 1975.

[14] Jon M. Kleinberg. *Authoritative sources in a hyperlinked environment*. Journal of the ACM (JACM), Volume 46, Issue 5 (September 1999).

[15] *Analog: a WWW logfile analysis*. http://www.analog.cx/.

[16] *Gigabase: an Object-Relational Database Management System*. http://www.garret.ru/~knizhnik/gigabase.html.

[17] F. Bonchi, F. Giannotti, C. Gozzi, G. Manco, M. Nanni, D. Pedreschi, C. Renso, and S. Ruggieri. *Web log data warehousing and mining for intelligent web caching*. Data and Knowledge Engineering (DKE), 32(2):165-189, October 2001.

[18] Giuseppe Manco, Riccardo Ortale, Domenico Saccà. *Similarity-Based Clustering of Web Transactions*. SAC 2003: 1212-1216.

[19] M. Perkowiz and O. Etzioni. *Adaptive web sites: Automatically synthetizing web pages*. In AAAI-98 - Fifteenth National Conference on Artificial Intelligence, Madison, USA, July 1998.

[20] Berendt, B., Stumme, G., & Hotho, A. (in press). *Usage mining for and on the Semantic Web*. In H. Kargupta, A. Joshi, K. Sivakumar, & Y. Yesha (Eds.), Data Mining: Next Generation Challenges and Future Directions (pp. 467-486). Menlo Park, CA: AAAI/MIT Press.