Pharos, a Collaborative Infrastructure for Web Knowledge Sharing

Vincent Bouthors, Olivier Dedieu

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Pharos, a Collaborative Infrastructure for Web Knowledge Sharing

Vincent Bouthors , Olivier Dedieu

No 3679

Mai 1999

THÈME 1

ISSN 0249-6399

Pharos, a Collaborative Infrastructure for Web Knowledge Sharing

Vincent Bouthors* , Olivier Dedieu†

Thème 1 — Réseaux et systèmes
Projet SOR — http://www-sor.inria.fr/

Rapport de recherche n ° 3679 — Mai 1999 — 20 pages

Abstract: Finding relevant information is one of the biggest problems that Web users experience. This article describes Pharos, a new service that has been developped to help groups of Web users share their knowledge about interesting documents. Pharos relies on a collaborative infrastructure which allows user groups to index and evaluate documents on specific topics. This information, possibly subjective, is synthesized to produce personalized recommendations. Scalability is handled by distributing servers and replicating their databases. Pharos has been implemented in Java and is currently being evaluated.

Key-words: Information retrieval, Community, Knowledge Sharing, Catalogues, Collaborative filtering, Digital library, Scalability.

(Résumé : tsvp)

* Vincent.Bouthors@inria.fr
† Olivier.Dedieu@inria.fr

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
Téléphone : 01 39 63 55 11 - International : +33 1 39 63 55 11
Télécopie : (33) 01 39 63 53 30 - International : +33 1 39 63 53 30

Pharos, une infrastructure collaborative pour le partage de connaissances sur le Web

Résumé : La recherche d'information sur le Web est un problème auquel font face les utilisateurs. Cet article décrit Pharos, un nouveau service développé pour aider les utilisateurs du Web à partager la connaissance qu'ils en ont. Pharos repose sur une infrastructure de collaboration qui permet à des groupes d'utilisateurs de cataloguer et d'évaluer des documents sur un sujet donné. Ces données, qui peuvent être subjectives, sont synthétisées afin de produire des recommandations personnalisées. Le passage à l'échelle du système est assuré par la distribution des serveurs et la réplication de leur bases de données. Pharos a été implémenté en Java et est actuellement en cours d'évaluation.

Mots-clé : Recherche d'information, Communautés d'intérêt, Partage de connaissances, Catalogues, Filtrage collaboratif, Bibliothèques électroniques, Passage à l'echelle.

1 Introduction

The World-Wide Web is the easiest way to disseminate information to the global community. In December 1997, the estimated size of the indexable Web was 320 million pages [14]. Nevertheless, according to a recent surveys [12], about half of the users consider that it is a big problem to find the information they are looking for. The Web infrastructure relies on hypertext and does not provide a mechanism to quickly find a valuable document on a precise subject. As a result, people search information in dedicated sites that index the Web, or consult alternative communication channels such as news groups or mailing lists.

In January 1999, eight out of the ten most visited sites were search engines [17]. Search engines index the content of the Web and provide a query interface on a Web site. Their robots travel through hyperlinks to discover new documents. Each document found is then indexed. Automatic indexing uses full-text algorithms to associate meaningful words in the page to its URL. Therefore, most of the search engines can be queried on words only and not on the concepts of the pages they index[1]. So, query formulation is a difficult exercise if both noise and silence are to be avoided. Search engines using manual cataloging, such as Yahoo![2], return more relevant responses, but the user must trust the expertise of the people who select and organize information. Success of search engines is due to two main reasons: (i) they are the easiest place to find information ; (ii) they have a relatively good coverage of the Web [14, 24]. However, one major drawback of search engines is that they only give links. The user must read and determine himself if the proposed documents are interesting or not. This work is time consuming and must be done by all the users for a given search. Factorizing this repetitive effort would help people to find more quickly the most relevant documents.

Search engines do not provide evaluations about returned documents. People therefore use alternative systems such as news groups and mailing lists. They ask for information, and someone knowing relevant references can send their advice back to the group. Such systems are highly dependent on the people who belong to the groups. The more expert the group members are, the more valuable the information is. Frequently asked questions, off-topic questions or bad responses generate noise that depreciates the group. The lack of structure in the messages' content makes automatic filtering difficult. Furthermore, this requires experience, and new members cannot immediately estimate the expertise of existing members. Because messages are not persistent, the benefit of one's accumulated knowledge cannot be given to new or external members. To address this last point, alternatives are to build FAQs[3] [11], archives, links pages or thematic portals. But these solutions leave unresolved the problem of their localization and are time consuming to maintain. There is clearly a need to provide a system that integrates functionalities similar to search engines

[1] Some of search engines (e.g. `ww.excite.com`, `www.infoseek.com` or `www.altavista.com`) extend requests to morphologic or lexicographic variations.

[2] `http://www.yahoo.com/`

[3] Frequently Asked Questions.

but which references valuable information such as that exchanged within groups of experts.

Pharos is a collaborative infrastructure which allows people to share their knowledge on a specific topic. People finding a valuable document put an *annotation* in the appropriate *channel*. An annotation is a structured datum referencing an URL. It is composed of values such as a title, a rating (a subjective note), a free comment and a list of keywords. A channel is a database of annotations dedicated to a specific topic (e.g. games, music, Java language, Web technologies, and so on). People interact with Pharos by using a *browser assistant*. The browser assistant is a personal proxy with a GUI[4] (see Figure 1). It observes the URLs the user accesses, requests channel servers for annotations associated with each URL and displays them. As a result, the user can quickly evaluate the interest of the document. Pharos also helps to find rated information. Queries can be performed to extract annotated documents matching some criteria (e.g. all the documents rated as "good" by "bob" and categorized with the keyword "documentation:book" and "api:servlet").

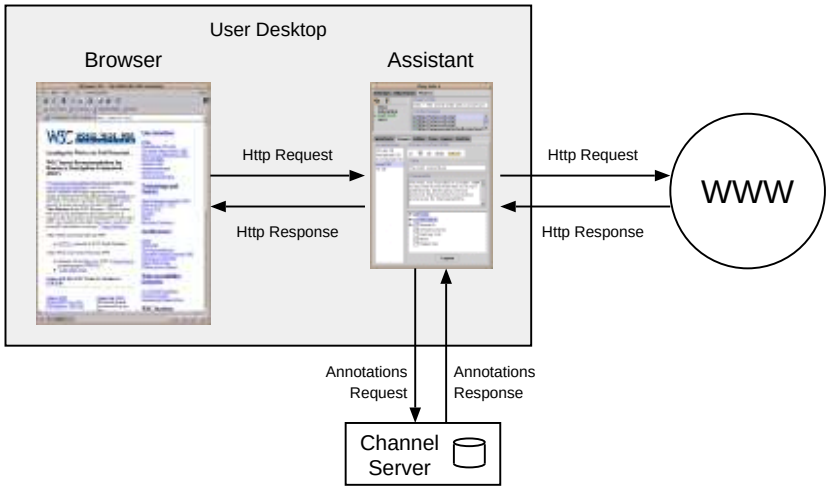

Figure 1: *Browser assistant interaction*

As the amount of available annotations increases it becomes more difficult for users to exploit them. To reduce noise, annotation syntheses are computed. The synthesis algorithm gathers all available annotations about a given document into a *recommendation*. Since all users do not necessarily agree about document ratings, recommendations are personalized according to the users' profiles.

[4]Graphic User Interface.

Pharos is based on a client/server architecture. For each page browsed, the browser assistant requests annotations from the channel server and displays them. If members are numerous this can dramatically increase the load on the server. If users are located world wide the latency of annotation fetching can become unacceptable. To improve performance, channels can be replicated close to users' locations. This creates consistency issues which Pharos addresses with an optimistic protocol, guaranteeing the *eventual consistency* model [4, 25].

In Section 2, we detail the main concepts, the design of Pharos services and the recommendation algorithms. Section 3 presents the implementation of the browser assistant, channel flexibility and the communication model. Section 4 describes the scalability issues and how Pharos addresses them. Section 5 presents a survey of related work. Finally, in Section 6, we present our future work and conclusions.

2 Services Design

The Pharos system is made of two components: the browser assistant and the channel server (see Figure 1). The assistant works with any Web browser and provides a user interface to publish an annotation, to display annotations and recommendations on the documents visited by the browser, to search documents matching some criteria, and to perform administrative tasks for channels. A channel may be replicated at multiple locations. This section describes what an annotation is, how recommendations are computed, and how channels are managed.

2.1 Annotations

2.1.1 Annotation structure

An annotation is a structured datum, published by someone on a channel, to describe a Web document. The user publishes annotations explicitly. There is no "spying" on user behavior nor keeping trace of what has been visited. We believe strongly that it is absolutely necessary, for a collaborative recommendation system, to respect the privacy of users, in order to gain user acceptance.

An important characteristic of an annotation is that it is structured. This data structure depends on the channel class. A basic class, named *BasicChannel*, is provided for general purpose. A BasicChannel annotation contains a *title* which is by default the document's title, a *rating* which is the user's appreciation of the document, some *keywords* which are chosen in an extensible hierarchical list, and a *comment* which is a free textual note.

The rating is a float value ranging from -1 to +1. However, from the user-interface point of view, the rating is a discrete value displayed as an icon or a text defined by the channel

administrator.

An annotation may contain multiple keywords. The keyword hierarchy may be viewed as a simple thesaurus. This thesaurus helps the community to share a common, tree-structured vocabulary when annotating documents. This avoids having keywords that are semantically similar but lexically different (e.g. *browser* versus *navigator*). The thesaurus simplifies searching by keywords and gives a global view of the channel topic organization. Keyword hierarchy may be modified only by authorized users.

BasicChannel can be subclassed to satisfy specific needs. In scientific communities, users would appreciate the ability to handle bibliographical data and to be able to import/export data in the BibTeX [19] format. In the digital library there is a need to support meta-data as defined in the Dublin Core [23, 13]. Another important point is the life-time of annotations. It appears important to leave the user to decide, at creation time, when an annotation will become obsolete. All these examples show the need to be able to extend BasicChannel to add new attributes.

2.1.2 Functionalities

While browsing the Web, annotations associated with visited documents are shown. This information is displayed in the browser assistant. The list of authors that have added annotations is displayed. When selecting a specific author, his annotation is shown. Additionally, recommendations are presented as if they were annotations published by pseudo users. There is one such pseudo user for each recommendation algorithm. The algorithms available in the BasicChannel class are described in Section 2.2.

The user queries recommendations by filling in a form, indicating a list of criteria. This is a database querying facility, with regular expression search on titles, comments and URLs. Results are not displayed in the browser assistant but in the browser itself. This makes it straightforward for the user to navigate in the recommended documents (see figure 2).

Another way to consult annotations is very similar to bookmarks or favorites provided in Netscape or Internet Explorer. The annotations are visible in a tree having the same structure as the keywords hierarchy. The leaves of this tree are the recommendations themselves. This user interface has been provided to give regular users an interface similar to the one for bookmarks/favorites in the browser.

Web access exists also, providing another way to consult annotations and recommendations through two levels of queries: advanced and simple. The first one is similar to the assistant querying. The second one provides access through a hypertext paradigm.

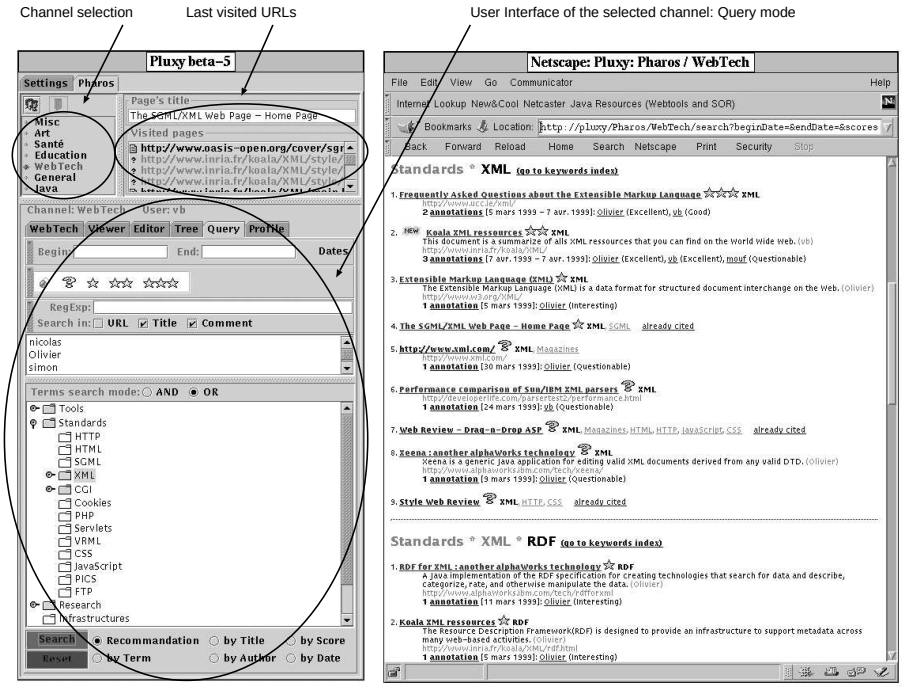

Figure 2: *The assistant is on the left and the browser is displaying results of a query concerning XML.*

2.1.3 Document identification

An annotation is associated with a document. Annotations are stored in a channel server. To identify the associated document, each annotation contains an URL. This raises problems because an URL is a locator to access a document, not a name to identify a document: a document can be updated, moved, mirrored, be temporarily unreachable, or definitively destroyed. Moreover, there is probably a need for annotating not only documents but also objects such as movies or CDs. A simple solution is to annotate a vendor page, but such a page may not exist or there may be more than one such page. A correct solution would be to rely on a naming service (possibly based on bar codes [16]).

2.2 Collaborative Recommendations

A recommendation is the synthesis of all the annotations on a single document. This section describes how recommendations are computed in the BasicChannel. It is important to note that computation of recommendations is made easier because annotations are structured

data.

The important feature is that everybody gets a personalized recommendation. The goal is to avoid, or at least to reduce, two problems: average effect and pollution. If the recommendations were the same for everybody, it would correspond to an average advice. Minority annotations would be diluted in the mass. On the other hand, in the task of taking into account multiple advices, more weight tends to be given to some people, because they are experts or because they share the same taste, depending on the subject. This is what Pharos automates. The second problem is the sensitivity to advertisement pollution. For instance, a vendor could inundate a channel of annotations from fake users praising his product. A solution is to ignore these annotations by giving a null weight to all these fake users. This may be done by other users explicitly. It is done automaticaly by Pharos with the correlation algorithm (see next sections). These Solutions minimize the need for a moderator.

2.2.1 Computing recommendation rating

The rating is a subjective value and plays an important role when computing recommendations. The function estimating the rating of the URL u for the member m, $predictedRating(m, u)$, is defined by the following formula. Let's call $M(u)$ the set of members having added an annotation for the URL u, $weight(m, n)$ the weight or the confidence placed by member m on member n in the range $[-1, +1]$, $rating(m, u)$ the rating of the URL u by the member m in the range $[-1, +1]$:

$$predictedRating(m,u) = \frac{\sum_{n \in M(u)} weight(m,n) \times rating(n,u)}{\sum_{n \in M(u)} \mid weight(m,n) \mid} \quad (1)$$

This function is a weighted average of ratings. If a member m places no confidence in member n, that is if $weight(m, n) = 0$, the rating of member n is ignored when predicting the rating for member m.

The BasicChannel provides two ways of defining the weight: explicitly, or automatically by correlation. In the first case, the user edits the weight he uses for each of the other members. By default all users have the same initial positive weight. This allows each user to distinguish some people because they agree (positive weight) or disagree (negative weight) with them, or don't care about them (null weight).

2.2.2 Computing correlation between users

If there are too many users in a channel it would be tedious to attribute a weight for all of them so the weight would remain the default positive value for most of them. In this

case the predicted rating would be equivalent to the average. To solve this problem, the BasicChannel offers a second way to automatize the weight assignment. The correlation algorithm determines users having similar profiles by comparing the annotations they have added in the past. The weight $weight(m,n)$ given by member m to member n is equal to $correlation(m,n)$, the correlation between the two users. This function takes values in the range $[-1,+1]$: from -1 for systematic contradiction, up to $+1$ for systematic agreement. Let's call $U(m,n)$ the set of URLs annotated by both members m and n.

$$correlation(m,n) = \frac{\sum_{u \in U(m,n)} rating(m,u) \times rating(n,u)}{\sqrt{\sum_{u \in U(m,n)} rating(m,u)^2} \times \sqrt{\sum_{u \in U(m,n)} rating(n,u)^2}} \quad (2)$$

If the denominator is null the function has a default customizable value. One property of this function is to not take into account rating equal to zero.

Other very similar algorithms have been used in recommendation systems [20, 22]. An original point of Pharos is to empower the user by allowing the algorithm to be explicitly weighted. From a user interface perspective, everything works as if there were pseudo-users named "weighted" and "correlated". The pseudo-user "correlated" computes a prediction defining $weight(m,n)$ as being $correlation(m,n)$. The pseudo-user "weighted" computes a prediction using $weight(m,n)$ explicitly given by the user. The pseudo-user "correlated" may be given a weight indicating the confidence of member m in the "correlated" algorithm. This way, the user has a complete and intuitive control over the way the "weighted" prediction is computed. He rates the algorithm by giving a weight, as for any other user. This mechanism provides a natural way to integrate other algorithms: they would simply be associated with other pseudo-users.

2.2.3 Computing recommendation

We have discussed how rating is used to compute correlation between users and how it is used to compute a predicted rating. We now describe how other attributes of the annotations are treated. The rating still plays an important role.

The title and the comment of the recommendation are synthesized using a "best guess" algorithm. When computing a recommendation for member m on URL u, a title is chosen among members n so as to maximize the function:

$$pertinence(m,u,n) = weight(m,n) \times rating(n,u) \quad (3)$$

The rationale for this algorithm is to prefer information from a user both having been assigned a good confidence and having appreciated the recommended document.

The keywords are synthesized another way. A simple algorithm is to compute the union of all sets of keywords associated by all users on the same URL. The drawback is that too many keywords may be returned. A better algorithm is to compute a pertinence for each keyword and to retain only those keywords exceeding a threshold.

2.3 Annotation Channels

A channel represents a community of users sharing the same interest (e.g. the Java language, Web technologies or teaching). The number of users registered in a channel varies according the community. A channel can be composed of a few users annotating documents on a very in-depth topic or, at the opposite extreme, it can concern a more general topic intended for mass consumption on a world-wide scale. The channel stores members' annotations in a database. It can extract annotations associated with a specific document, annotated documents matching some annotation criteria, and compute personalized recommendations.

Channels are autonomous entities. That is, instead of having one huge database for all annotations of all topics, there are as many channels as there are topics. Pharos does not require the channels to all be located in the same place. They can be distributed across the network. This design choice increases system flexibility and performance for several reasons:

1. Channel access control policy can be customized according to each community. Some channels can be public, not filtering member registration, allowing anyone to add annotations and provide Web access to increase their accessibility. Others, can contain private data and be restricted to authorized members.

2. Channel management is flexible. Creating a channel on a new topic is very simple and only requires a machine to host the channel. Tricky operations such as halting a channel for maintenance, exchanging updates for replicated channels, and merging or splitting channels, do not have any impact on other channels.

3. Network traffic and server load is reduced. Channels can be located close to their members and are only accessed by people interested in their topic.

4. Members can replicate on their machine some of the channels they have subscribed to.

5. Relevancy and performance of synthesis algorithms are improved. These algorithms produce personalized recommendations according to users' profiles. Channel separation increases topic locality for each profile. For instance, Alice and Bob can have related profiles in the Java channel but opposite ones in the Music channel.

6. The annotation structure can be specialized for each channel to better fit with the community's requirements. For instance, a channel about distributed computing would have a BibTeX [19] field to annotate research papers whereas ones about financial news whould have a field specifying the duration of validity of the annotation.

Channels are located with an URL. Member creation and authentication is channel-dependent. Some channels can allow unrestricted subscription with identification based on the user name or email, whereas others require the channel's administrator to register a new member and use more secure authentication mechanisms. Once the user is logged into the channel, he can perform the operations for which he is authorized: accessing annotations, publishing an annotation, editing the keyword hierarchy, and administration.

Users can quickly create a new channel by instantiating and customizing a BasicChannel (see section 2.1). BasicChannel is a generic channel which has an all-purpose annotation structure (title, rating, comments, and a list of keywords). It permits searching, Web access publishing, and provides a thesaurus manager. At channel creation time the thesaurus is empty. Authorized members can add, delete, move or rename terms when needed. Finally, if the basic channel does not fit exactly with the community requirement, Pharos provides extensible framework which is able to receive any kind of channel. In this case, the community must develop a new channel class or extend an existing one to match their specific needs.

3 Implementation

Pharos has been developped in Java [1] which provides useful features for building dynamic and portable architectures. This section details the implementation of the channels and of the browser assistant.

3.1 Channel Components

Channels are mainly composed of two parts: the backend and the frontend. The backend is the server part of a channel. It manages member subscription, the annotation database and additional data such as the thesaurus of keywords. The backend processing queries from frontends. It uses batch process to pre-compute data used later to produce recommendations. If the channel is replicated, each backend exchanges updates with its peers to maintain a consistent replica at each backend. If the channel has Web access, the backend provides HTML interface to query or publish annotations.

The frontend is the client part of a channel. It contains the channel GUI hosted in the browser assistant. It allows the user to add, display and query annotations, to edit the thesaurus and to customize his profile.

Frontend / backend communication relies on RMI [26]. RMI (Remote Method Invocation) is a Java API based on the RPC paradigm extended for the Java object model. Each backend exports a remote object which is invoked by frontends. However, some updates are propagated from the backend to frontends thanks to a lazy notification mechanism detailed

in Section 4.1.

To factorize certains resource consumption (e.g. network port, Web access, JVM[5], code), backends hosted on the same machine can be aggregated in a *PharosServer*. It has its own RMI access which is used by the browser assistant to list available channels on a machine. This helps users to discover and choose the channels they want to subscribe to. Finally, the PharosServer has a Web access which gives links to backends that have registered a Web access.

3.2 The Browser Assistant

Pharos has been designed to annotate Web documents. Frontends must be aware of the URL the user is browsing. To this end several techniques exist, such as HTTP proxy [15], browser parasite [16], customized browser, applets or browser plug-ins. The proxy solution has been used in Pharos because it works with any browser and provides the assistant with a high degree of control to observe the traffic and to enrich the content of returned pages. The browser assistant relies on *Pluxy* [7], an extensible HTTP proxy. Pluxy can dynamically aggregate a set of proxy components, the *pluxins*. Each pluxin can observe requests and responses, modify them, or provide a response.

The browser assistant is made of a pluxin which (i) observes request URLs in order to fetch associated annotations and (ii) extracts the HTML title of the page (if any) from the responses to fill the title field in the annotation editor. In an upcoming version, it will also be used to enrich HTML hyperlinks with icons pointing out the relevancy of targeted documents.

When the user requests the annotation database, the frontend builds an HTML page containing the results and displays it in the browser (see Figure 2). However, the proxy model does not allow the proxy to send requests to the client. The frontend therefore uses a mechanism to tell the browser to fetch a special URL. When the proxy receives this special URL, it calls the targeted frontend which then returns the HTML result page. This notification mechanism is browser- and OS-dependent. Under Unix/X11, Netscape provides a special option to open an URL[6]. Under Windows, Netscape and InternetExplorer provide a DDE[7] communication handler.

3.3 Channel Flexibility

Pharos handles a variety of channels thanks to a composition architecture. This architecture relies on the JPlug framework [6] which is intended for building modular and extensible applications in Java. JPlug allows an application to confine some functionnality within

[5] Java Virtual Machine.

[6] `netscape -remote openURL(url)`

[7] Dynamic Data Exchange.

components and gather these components into a container. A JPlug component is a set of resources such as code (Java classes), configuration files and icons for the GUI. JPlug allows any component to be dynamically installed, loaded, started, stopped and unloaded. Installing a component means fetching it and installing it on a local disk to be loaded. Loading a component means creating an instance of this component and initializing it. Start and stop events control its activity. Finally, unload removes the component from its container. Combined with the load functionality, this helps to debug and to tune a component without restarting the whole application. Furthermore, JPlug also provides facilities to update and reinstall new version of a component automatically.

When loaded, each component receives a sandbox, i.e. a node in the file system, where it can create new files. A SecurityManager [9] ensures that a component does not access files outside of its sandbox. A component can rely on other components. Such a component expresses its dependencies in its description file. JPlug ensurse that all the required components will be loaded in the right order[8]. JPlug supports component inheritance. That is, a component receives the resources (including the code) of its parent component but can override some of them.

Channels' frontends and backends are JPlug components which are respectively plugged into the browser assistant and the PharosServer. When a user subscribes to a new channel, the corresponding frontend is downloaded, installed and loaded. The BasicChannel can be specialized with the component inheritance mechanism. The specialization consists only of properties and file settings. So, combined with the component loading mechanism, users can quickly and dynamically create new channels by inheriting the BasicChannel component. The starting, stopping and unloading facilities allow PharosServer's administrators to perform tricky operations on a channel without stopping all the other ones.

4 Scalability

Pharos distributes communities of users into channels and then pushes scalability issues onto the channels. A channel is said to be scalable if it can handle the addition of users and annotations without suffering a noticeable loss of performance [18]. This section describes two techniques used by the BasicChannel to reduce server load and to increase their availability: (i) frequently used data are cached in the frontend side, (ii) backends can be replicated close to their members.

4.1 Data Caching

To reduce network exchanges, frontends cache some frequently used data such as the users list and the thesaurus of keywords. These data are infrequently updated but when a backend

[8] JPlug loads components according to a topologic sort; the dependency graph must therefore be acyclic.

receives an update it must propagate it to all of its frontends. However, since RMI is a point-to-point communication protocol, forwarding an update to all the clients requires enumerating a list of all the frontends, to contact them and to send them the updates. Such an approach would be very inefficient and would reduce the scalability of a backend. We use therefore a *lazy notification* mechanism. Each operation is assigned a timestamp ts. The frontend piggybacks the last timestamp ts it received from the backend on each request. When a backend is invoked, in addition to the request processing, it extracts updates with a greater timestamp than ts and piggybacks them on the returned values of the request.

4.2 Channel Replication

Replication is another solution to scale in distributed systems [18]. It increases performance by reducing network distance and improves reliability by multiplying places where data are available. However, the main difficulty is to keep all the replicas mutually consistent. That is, when a write operation is done on a replica, the system must ensure that it does not corrupt the global consistency model of the system.

4.2.1 Consistency models and protocols

Consistency models fall into one of two categories: strong or weak [10]. The former ensures there are no inconsistencies in the logical view of the group. The latter allows replicas to contain invalid data. Weak consistency requires fewer agreement message exchanges and fits better with the network constraints we address. It can be achieved with two kinds of protocol: pessimistic or optimistic. The first prevents any write operations from producing conflicts when replicas exchange their updates. Indeed, some data are dependent on the order in which the operations are processed[9]. The second allows arbitrary write operations on any replicas and delays potential conflict resolution until later.

4.2.2 The replication protocol

The complete details of the replication protocol are beyond of the scope of this article. We give in this section the main principles which have been chosen.

The replication protocol of BasicChannel has been inspired by Bayou [25] which guarantees *Eventual consistency* [4]. It is a weak consistency model based on an optimistic protocol. It allows write operation without any agreement with the rest of the group. As a result, replicas can contain out-of-date data. Replicas synchronize their databases by exchanging their updates from peer to peer. Updates are propagated epidemically [8], which guarantees

[9] In the literature [3] executions of such operations are said to be *non-serializable*. For instance, non-commutative operations such as addition and multiplication are non-serializable (e.g. $x \leftarrow x + 1$; $x \leftarrow x \times 2$).

that all the replicas will finally reach a consistent state[10]. In BasicChannel, the partner selection pattern for synchronization is left to the channel's administrator. The protocol does not prevent conflicts due to parallel write operations. We propose a new model to detect and handle conflicts. Conflict detection is semi-automatic. We distinguish conflicts concerning data structure integrity and those concerning the semantic integrity of the data. The former is handled by the collections themselves which ensure their data structure is not corrupted (e.g. an operation on a tree must not transform it into a graph). The latter is handled by the application which uses these collections (e.g.: a thesaurus mapped on a tree ensures there is no double term in a same node level). When a conflict is detected, we keep the conflicting operations which are considered as *propositions*. The administrator is responsible for accepting one of those operations or rejecting all.

5 Related works

5.1 Collaborative Indexing

Several works address the indexing problem through collaborative systems. Marais and Bharat [16] uses both content and collaborative indexing. Their desktop assistant[11], Vistabar, continuously indexes[12] the full text of all viewed pages. In addition, they allow people to add *comments* about Web pages. A comment has a unique structure composed of the page's URL, the page's title, a set of categories choosen in a hierarchy, a short subject, a free text area and an icon indicating the nature of the comment. They do not address comments with specifics for some community. They do not have a rating model nor synthesizing mechanisms to reduce noise. The comments feed a centralized database which represents the *CommonKnowledge* of a community. As a result, they do not address scalability issues as describe in Section 4.

The Open Directory Project[13]'s goal is to produce a comprehensive directory of the Web, by relying on volunteer editors. The indexing is mapped onto a tree structure. Each node of the tree represents a category (e.g.: Sports:Tennis, Computers:Hardware, and so on). The user interface is very similar to Yahoo!; people browse in the categories and have a full-text search from each category. Volunteers are responsible for editing entries in categories. This project uses delegation rather than collaboration. There is only one editor per category. Anyone can submit a site for a category but it will be validated by the editor of that category. Nevertheless, adding a site matching several categories requires submitting it to all editors of these categories. In fact, the tree does not help to categorize entries but rather to refine the search. Each leaf of the tree is composed of a link on the site and of a short description.

[10] In fact, this state would be reached if and only if all replicas stopped exchanging updates. That is, if no more write operations were done.

[11] They have developed a parasite browser which works with InternetExplorer.

[12] They uses the Altavista NI2 library to build full-text indexer.

[13] `http://www.dmoz.org/`

A very few of them may have an icon indicating that it is a good site. Furthermore, users must trust the editors and they cannot give their opinion of the indexed sites. So, it helps to find information but not to evaluate it.

5.2 Recommendation Systems

Only some representative works are presented here. For an overview of systems see [21]. For a discussion on the application of recommendation in digital libraries see [5].

GroupLens [20] is a representative example of a collaboration filtering tool. It was primarily dedicated to net-news. Each user puts a rating on messages. The system computes correlations between users according to their ratings. The main drawback of the system lies in the network architecture. It does not address the scalability issues and the centralized server is often overloaded.

Firefly[14] and NetPerceptions[15], the commercial version of GroupLens, illustrate the main target of today recommendation products. They provide personalization for visitors of commercial Web sites. For instance Amazon[16] recommends books to their customers. This is based both on explicit recommendation and on previous purchases. These kinds of services are limited to visitors and customers of a Web site.

Fab [2] is an example of a hybrid system. It integrates both collaborative and content-based recommendations. The latter tries to recommend documents similar to those a given user has liked in the past. It tries to avoid the limitations of both categories (number of *similar* users for the former, noise for the latter). The annotation contains only a rating, as for GroupLens, and the data are centralized.

6 Future Work

6.0.1 Experimentation

The major functionalities of Pharos have been implemented. The current work concerns the URL issues and the scalability issues. A first prototype has been used internally for some months and the lastest version is released: the browser assistant is freely available.

Currently, experimentation is being lead by CNET[17] with the help of ergonoms. Pharos will be used by teachers to recommend documents that may be pedagogic supports. We are

[14] `http://www.firefly.net/studio/applications/`
[15] `http://www.netperceptions.com/`
[16] `http://www.amazon.com/`
[17] CNET - Centre de Recherche et Développement de France Télécom.

supported by CNDP[18] which provides us the database Educasource to help this experimentation.

Inside INRIA and Bull, some technical channels are used to closely follow technical and scientific evolution in areas such as Java, Web technologies, Distributed computing, or Programming. Some other channels such as *Art* are intended for a larger audience. We foresee the use of a *meta channel* to recommend channels themselves.

6.0.2 Recommendation

This paper has presented two recommendation algorithms. Other algorithms are envisaged. For instance, instead of using correlation between users on their annotations, it is also possible to use profile correlation: if member A places a high confidence in member B, who in turn places a high confidence in member C, then it could be deduced that member A places a reasonable confidence in member C.

We intend to validate the rating prediction algorithms by checking their output with the real ratings published by the users. We are aware that this approach has two limitations. Firstly the *best* function for computing prediction is dependent on the function choosen to evaluate accurary. Secondly members won't annotate most documents and worse, users might annotate documents only when they disagree with the prediction. If they agree with the prediction, they are happy with it and don't fell necessary to give their advice.

6.0.3 Integration with other applications

Pharos does not intend to obsolete existing indexing services. Pharos is mostly a complementary approach. Moreover, Pharos may be integrated with an indexing service to provide advanced functionalities. Firstly, indexing recommended documents would allow the retrieval of documents according to both kinds of criteria: the subjective meta-data of Pharos and the content of the document. Secondly, some indexing tools are able to propose classifications. Pharos could submit these proposed classifications to the user for validation.

It would be interesting to integrate Pharos with a Web cache server. Pharos could ask the cache to tell the user if a document is already present in the cache. In the other direction, Pharos could give the cache a hint of the probability for a document to be retrieved in the future. Basically, the higher it is rated the more it is recommended, and the more probable it will be retrieved.

[18]CNDP - Centre National de Documentation Pédagogique.

Finally, we expect to support the data format RDF[19] based on XML[20] to facilitate export and import to and from other applications. RDF has been proposed as a generalization of the format designed for PICS[21] and is a good candidate to become a widely used standard.

7 Conclusion

Pharos brings a new kind of service to Web users. Some channels are publicly available at `http://webtools.dyade.fr/pharos/`. Compared to indexing tools, Pharos provides a collaborative forum for communities to exchange subjective information. Compared to newsgroups, it copes with heterogeneity of a large number of members by synthesizing automatically a personalized recommendation for each member. Exchanging recommendations is a widely used practice in professional and daily life. We believe that services such as Pharos will be widely used in many areas. In the digital library domain, these recommendation tools will be a new way to manage knowledege and to involve both final users and librarians. In other words, we believe that human beings are the intelligent agents of the Internet.

References

[1] Arnold, K., and Gosling, J. *The Java Programming Language.* Addison-Wesley, 1996.

[2] Balabanović, M., and Shoham, Y. Fab: Content-based, collaborative recommendation. *Communications of the ACM 40*, 3 (Mar. 1997), 66–72. `http://www.acm.org/pubs/citations/journals/cacm/1997-40-3/p66-balabanov%ic/`.

[3] Bernstein, P. A., and Goodman, N. Concurrency control and recovery for replicated databases. Tech. Rep. TR-2, Harvard University,, 1983. serializability for Replicated Databases. Shows problems with Eswaran et al, Rosenkrantz et al approaches. A new theory and analysis of quorum consensus (voting), 'missing write' (by Eager and Sevcik), and 'available copies' (SDD-1) algorithms (which are found to be best).

[4] Birrell, A. D., Levin, R., Needham, R. M., and Schroeder, M. D. Grapevine: an exercise in distributed computing. *Communications of the ACM 25*, 4 (Apr. 1982), 260–274.

[5] David M. Nichols, M. B. T., and Paice, C. D. Recommendation and usage in the digital library. Tech. Rep. CSEG/2/97, Computing Department, Lancaster University, 2 1997.

[19] Resource Description Framework.
[20] Extensible Markup Language.
[21] Platform for Internet Content Selection.

[6] DEDIEU, O. JPlug, a framework to build modular applications. `http://webtools.dyade.fr/jplug/`.

[7] DEDIEU, O. Pluxy : un proxy Web dynamiquement extensible. In *Proceedings of the 1998 NoTeRe colloquium* (Oct. 1998). `http://www-sor.inria.fr/publi/PPWDE\_notere98.html`.

[8] DEMERS, A., GREENE, D., HAUSER, C., IRISH, W., LARSON, J., SHENKER, S., STURGIS, H., SWINEHART, D., AND TERRY, D. Epidemic algorithms for replicated database maintenance. In *Sixth Symposium on Principles of Distributed Computing* (Vancouver, Canada, Aug. 1987), pp. 1–12.

[9] GONG, L. Java security: Present and near future. *IEEE Micro 17*, 3 (May/June 1997), 14–19. `http://www.computer.org/micro/mi1997/m3014abs.htm` .

[10] HELAL, A., HEDDAYA, A., AND BHAR, B. *Replication Techniques in Distributed Systems*. Kluwer Academic Publishers, august 1996. `http://www.cise.ufl.edu/~helal/books/repl.html`.

[11] HOWEL, J. Faq-O-Matic. `http://www.dartmouth.edu/~jonh/ff-serve/cache/1.html`.

[12] KEHOE, C., PITKOW, D. J., LAWRENCE, D. J. R., AND GILES, C. L. GVU's tenth WWW user survey report. Tech. rep., Georgia Tech Research Corporation, Apr. 1998. `http://www.gvu.gatech.edu/user\_surveys/survey-1998-10/`.

[13] LAGOZE, C. The warwick framework - a container architecture for diverse sets of metadata. *D-Lib Magazine*, SSN 1082-9873 (July/August 1996). `http://www.dlib.org/dlib/july96/lagoze/07lagoze.html`.

[14] LAWRENCE, S., AND GILES, C. L. Searching the World Wide Web. *Science 280*, 5360 (1998), 98.

[15] LUOTONEN, A., AND ALTIS, K. World-Wide Web Proxies. In *Proceedings of the 1rst International Conference on the World-Wide Web* (Geneva, mai 1994).

[16] MARAIS, H., AND BHARAT, K. Supporting cooperative and personal surfing with a desktop assistant. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST-97)* (New York, Oct. 1997), ACM Press, pp. 129–138. `ftp://ftp.digital.com/pub/DEC/SRC/publications/marais/uist97paper.pdf`.

[17] MEDIAMETRIX. Top 25 Web sites, Jan. 1999. `http://www.mediametrix.com/PressRoom/Press\_Releases/02\_22\_99.html`.

[18] NEUMAN, B. C. *Scale in distributed systems*. IEEE Computer Society, Los Alamitos, CA, 1994, pp. 463–489. `ftp://ftp.isi.edu/isi-pubs/rs-94-411.ps.Z`.

[19] PATASHNIK, O. BibTeX 1.0. *TUGboat 15*, 3 (Sept. 1994), 269–273.

[20] RESNICK, P., IACOVOU, N., SUCHAK, M., BERGSTROM, P., AND RIEDL, J. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of ACM CSCW'94 Conference on Computer-Supported Cooperative Work* (1994), Sharing Information and Creating Meaning, pp. 175–186.

[21] RESNICK, P., AND R.VARIAN, H. Recommender systems. *Communications of the ACM 40*, 3 (1997), 56–88.

[22] SHARDANAND, U., AND MAES, P. Social information filtering: Algorithms for automating "word of mouth". In *CHI'95 Proceedings Papers*. `http://info.sigchi.acm.org/sigchi/chi95/Electronic/documnts/papers/us\_%bdy.htm`.

[23] STUART WEIBEL, JEAN GODBY, E. M. Report - dublin core. Tech. rep., OCLC/NCSA Metadata Workshop, march 1995. `http://www.oclc.org:5046/oclc/research/conferences/metadata/dublin\_cor%e\_report.html`.

[24] SULLIVAN, D. Search engine sizes, Feb. 1999. `http://searchenginewatch.internet.com/reports/sizes.html`.

[25] TERRY, D., THEIMER, M., PETERSEN, K., DEMERS, A., SPREITZER, M., AND HAUSER, C. H. Managing update conflict in bayou, a weakly connected replicated storage system. In *15th ACM Symposium on Operating Systems Principles* (Copper Mountain Resort, Colorado, US, Dec. 1995). `http://www.parc.xerox.com/csl/projects/bayou/`,.

[26] WOLLRATH, A., RIGGS, R., AND WALDO, J. A distributed object model for the Java system. In *Proceeding of the USENIX 1996 Conference on Object-Oriented Technologies (COOTS)* (Toronto, June 1996), USENIX. `http://www.usenix.org/publications/library/proceedings/coots96/wollrath%.html`.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399