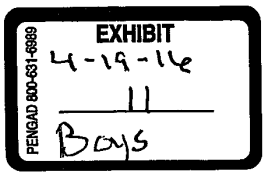
3902



Summarizing the web with AutoLogin

By Inventors P. Venkat Rangan, Sam Inala, Ramakrishna Satyavolu
Date: 12 May 1999

Field of the invention

The field of this invention is in the area of Internet services, mobile communications, and proxies.

Background of the Invention

Currently, the web offers a wealth of personal services. You can store your email on the web, get subscription news articles, manage your bank account, buy and sell stocks, and update a calendar. But with this embarrassment of riches come attendant problems. You become impatient with the time it takes to login to each site and check its information; you grow frustrated with the difficult of navigating web sites through mobile devices; you worry that you can't keep track of all of these varied sites and their user interfaces.

Ideally the web would provide you a way of consolidating all of your personal accounts in one place. You could check all of your personal information with a single page view, you could access your information from any place using any device, and you would always know exactly what accounts you have. The present invention describes how to fulfill these needs.

Summary of the Invention

A user visits the web site offering the web summarization service. For each account the user has, the user gives the username and password. The site records this information in a database. It then periodically issues a request to a gatherer. The gatherer acts like a browser (i.e. it is a HTTP User-Agent) and fetches pages from the web, logging in using information that the user has provided. It parses the page and looks for the summary information that the user is interested in. The gatherer knows how to login to each site and how to fetch the important information by using site-specific scripts. After scanning the page, the gatherer then stores that information in a canonical format in the database. Then when the user requests the information about that account, the information is retrieved from the database and rendered as HTML.

What follows is a detailed description of each of the major features of the invention.

1. Gatherer agents log in to sites on the user's behalf.

A gatherer gets a request to update the information for a given user and site. The gatherer understands the login procedure used by each particular site, namely what the field name is for the password, whether or not the site requires that a cookie be set by first visiting the site's front page, etc. The gatherer then submits an HTTP request (or POP3 request, or other network I/O) and gets the response from the server. The request might be a simple POST with the right information, an HTTP challenge/response conversation, or some more complex protocol.

2. Gatherers scan the page for the information to summarize.

Typically, the user specifies what kind of information they would like to see from each site. For example, they would like to see email messages from Hotmail, headlines from the NY Times, and the status of their order at Amazon. Let's say the user wants to see the status of their Amazon order. Then the gatherer parses the page and looks for just the items that the user is interested in, namely the order number, the shipping

date, and the order status. It doesn't try to keep the whole page, only these three text items. It knows where to find these items because Amazon uses a consistent structure in how information is laid out on their HTML pages, and the gatherer knows (for each site), how to look through that structure. Once it parses the document and finds the information, the gatherer stores that information to the database.

3. Site-specific scripts store summarization logic.

How does the gatherer know about how Amazon and Priceline and Ebay store their information? We store this knowledge in site-specific scripts. Each script contains the logic of how to navigate through the page and by what attributes the text of the page should be stored. Here's an example of a summarization script for Amazon:

```
# Site amazon.orders.x - shows status of orders from Amazon

login( 7 );

get( "/exec/obidos/order-list/" );

my @tables = get_tables_containing_text( "Orders:" );

my $order_list = new Yodlee::ObjectHolder( 'orders' );
$order_list->source( 'amazon' );
$order_list->link_info( get_link_info() );

my @href_list;
my @container_list;

foreach my $table ( @tables ) {
    my @rows = get_table_rows();

    foreach my $i ( 0 .. $#rows ) {
        select_row( $i );
        my $text = get_text( $rows[ $i ] );
        next if $text =~ /Orders:|Status/;

        my @items = get_row_items();
        next unless @items >= 4;

        my( $order_num, $date, $status );
        select_cell( 1 );
        $order_num = get_cell_text();

        my $href = get_url_of_first_href( get_cell() );

        select_cell( 2 );
        $date = get_cell_text();
        select_cell( 3 );
        $status = get_cell_text();

        next unless defined $order_num and defined $date and defined
$status;

        my $order = new Yodlee::Container( 'orders' );
        $order->order_number( $order_num );
        $order->date( $date );
        $order->status( $status );

        $order_list->push_object( $order );

        if( defined $href ) {
            push( @href_list, $href );
            push( @container_list, $order );
        }
```

~~HIGHLY CONFIDENTIAL~~
~~ATTORNEYS' EYES ONLY~~

CCPA 000918

```
    }
}

# Instead of just showing order number, we'd like to show the
# name of what was ordered. We visit the order status page for each
# order to get this information.

foreach my $i ( 0 .. $#href_list ) {
    # Note: we're relying on these being not directory relative links
    get( $href_list[ $i ] );

    @tables = get_tables_containing_text( "Items Ordered:" );

    foreach my $table ( @tables ) {
        my @rows = get_table_rows();

        foreach my $j ( 0 .. $#rows ) {
            select_row( $j );

            my $href = get_url_of_first_href( get_row() );

            next unless defined $href;

            my @child_list = get_children( get_row(), 'a' );
            next unless defined $child_list[ 0 ];

            my $text = get_text( $child_list[ 0 ] );

            $container_list[ $i ]->description( $text );
        }
    }
}

result( $order_list );
```

The script does this: It logs in on the user's behalf. Then it visits the order status page. Then it looks for a table titled 'Orders'. This table contains order status information. The gatherer looks at each row with at least four items. It gets the URL of the page with detailed information about the order and gets the status and order number. It then visits each page in succession and gets the detailed name of the item ordered.

This script serves as a typical example of the scanning and summarization logic common to most sites.

4. HTML renderer presents summary with auto login links.

We store the scanned information in a canonical format in the database. For example, a mail message is always stored as an array of four elements, representing the sender, recipient, subject, and date. The HTML renderer simply gets the information and formats it so that it appears within an HTML page, a very simple procedure.

One other feature is that most of the summarization data links back to the original site from which the information came. With one click, the user can select their Hotmail message and be automatically logged in to their Hotmail account without having to enter their password information for Hotmail. This is accomplished by including the form used by the site in the page. When the user clicks, Javascript code within the page auto-submits the form, taking the browser to a new page. The new page will be the destination site, not the summarization site itself. Alternatively, the page may submit it to the summarizer site that does some server side processing and then presents a page that auto-logs in the user without user intervention. The Javascript code could also bypass any prompts for HTTP authentication.

5. Any renderer for summarization data may be used.

We expect that most users will access their summarized information through a web page. Still, many users may want to access their information in other ways. The renderer can also present the information in a number of different formats. Examples include XML, plain text, VoxML, HDML, audio, video, and the like.

6. Summarized information may be accessed from any device.

Let's say you want your Palm Pilot to get your summary page. This invention affords the ability to hot-sync your mobile device to your summarization page. The ability to access your page is not limited to the following examples:

- Cellular phones
- Palm-sized devices
- WebTV clients / Video game consoles
- Regular telephones (via VoxML, Motorola's voice markup language)
- Programmatically: Any application may access summarized contents as XML documents through XML-RPC, DCOM, CORBA, RMI, or other forms of RPC

7. Client-side plug-ins may assist auto-login

For summarization, we never need a client side plugin. For the auto-login feature, almost every side may be done using Javascript. Some sites, however, make it impossible to get all the way through the login procedure with one or even two submissions. For these sites, we provide a plugin extension to the browser (or potentially a standalone browser with this feature), that knows you are trying to auto-login to the site and assists the browser in bypassing the forms to reach the final destination page.

8. Client software may summarize without trusting our server.

Some users may not want to give us passwords to sensitive sites, such as their bank accounts and stock portfolios. In these cases, we provide them a program that does summarization completely on the client machine, without having to store the passwords on our server. The only item the client software needs is the summarization scripts from our server (even these could be included with the software.)

The way this works: we write summarization scripts in Java. The client software regularly schedules an update for the page, and uses a browser control to fetch the pages and parse them appropriately. The client software gets the summarization logic from the summarization server. The client software stores the users passwords only on the user's local machine, encrypted and stored in a secure area.

9. Proxies can assist auto-login.

Let's say you're an ISP and you proxy user's connections. Then you can deploy this service and have auto-login to all other sites (including problematic sites), by extending the proxy with a plugin or using a drop-in replacement for the proxy. The extended proxy would automatically do additional form submissions and the like on behalf of the user, without any user intervention, when it knows you're trying to auto-login.

10. Caches may use summarization to present dynamic pages from the cache efficiently.

Network caches work well for static pages, less well for dynamic pages. Without knowing all of the account information for the user, the cache cannot store a wholly rendered page for that user. This means a user must go to the original site where the information is stored and retrieve the information. But using our summarization technology, a dynamic cache can store the user's account information and a template for dynamically generated pages. When a request comes in for this kind of dynamic page, the caching server can immediately satisfy the request, leading to much better response time for the user.

11. Persistent search results made available for every site.

Using the same techniques used for user account summarization, we can build systems that show you the results of a search the user does every day. For example, if a user is searching Apartments.com for places under $1000, then we can issue that search three times a day for them and show them the immediate results when they visit their summarized page.

12. Auto-registration using common profile information.

By simply changing scripts, we can have gatherers also auto-submit registration forms on behalf of users. This uses the same HTTP request logic employed to issue searches. Thus once a user has entered registration information, they never need enter it again. Instead, we store that registration information in the database and re-use it when they say they would like to register for a new account in some place.

13. On-demand update provided by summarization site.

Not only can the user see the summarized information for each site they have, they can also request an immediate update of that site's information. We then indicate to the user that that information is being updated and then issue a request to a set of stand-by gatherers that immediately handle the request for an update. Once the information is retrieved, the page is updated to show the newest information.

14. Changing registration information made simple.

Using the same scripting system employed for auto-summarization and auto-registration, the changing of user information may be automated as well. For example, if a user changes their address they need only change it on our site and then say change it on all sites. Our agents go out and do the appropriate actions on each site.

15. Implementation notes.

In this section, we mention several details of a typical implementation.

- Summarization applies to both the Internet and intranet networks.
- The underlying parsing engine used by gatherers may be any parser or parsing engine. We can using IE's HTML parser, a Perl parsing engine, an XML parser, a regular expression scanning engine, an SGML parser, a hand-built parser, or any combination of the like.
- The system of gatherers and web sites is a scalable implementation of distributed machines. The servers may run on a single, powerful machine, single process or multi-process, or any number of variations on scalable server architectures.
- The system is language independent.

Abstract of the Invention

A user visits the web site offering the web summarization service. For each account the user has, the user gives the username and password. The site records this information in a database. It then periodically issues a request to a gatherer. The gatherer acts like a browser (i.e. it is a HTTP User-Agent) and fetches pages from the web, logging in using information that the user has provided. It parses the page and looks for the summary information that the user is interested in. The gatherer knows how to login to each site and how to fetch the important information by using site-specific scripts. After scanning the page, the gatherer then stores that information in a canonical format in the database. Then when the user requests the information about that account, the information is retrieved from the database and rendered as HTML.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.