Fifth International
World Wide Web Conference

WWW5

May 6-10 1996
Paris France

Poster Proceedings
Active Files: a New Way for Attaching a Behavior to Web Pages

Vincenzo D'Andrea, Davide Pastorello, Marco Ronchetti and Giancarlo Succi

Laboratorio di Ingegneria Informatica del Dipartimento di Informatica e Studi Aziendali
Università di Trento, Via Inama, 38100 TRENTO ITALY
[Vincenzo.Dandrea, Davide.Pastorello, Marco.Ronchetti, Giancarlo.Succi]@iti.unitn.it

Abstract

In this paper we present a new mechanism for associating an active behavior to URLs. The mechanism we propose, called "Active Files", allows to associate a generic and possibly complex action to the opening of a generic file, and can be deployed in a variety of circumstances. In this paper we discuss both the general concept of Active File and the benefits for WWW applications, where it can successfully surrogated the Common Gateway Interface. We discuss advantages and disadvantages over the traditional CGI approach.

Keywords

Active files, Unix System Calls, Common Gateway Interface

1. Introduction

The overwhelming success of the W3 project comes from the ability to access in a uniform way local and remote information. This capability derives from the clean decoupling of client and server programs by the "information bus" and from a simple yet powerful description language. It is easy to be an actor on the server side (since preparing a HTML page is easy) as well on the client side (since browsers are really user-friendly, and no special setup is needed on different hardware platforms).

A relevant part of this success stems from the ability to go beyond the retrieval of static information by triggering processes on the server side, with the well known Common Gateway Interface (CGI) [McCa98] mechanism. Such processes can be used to create "on the fly" virtual documents which are served back to the client. The most popular applications of CGI technique are:

- to receive information from a WWW client under the hypertext transfer protocol;
- to create active gateways toward databases;
- to create new interfaces for legacy systems [Bou83] [WLD95]

In this paper we present Active Files (AF): an alternative approach which allows to associate an action to a file, so that whenever the file is OPENed the corresponding action is performed. When an URL is requested, the (regular) http daemon tries to open the corresponding file. If the file is an "active" one, before it is actually opened the associated action is performed, and then the file is opened. This allows to give a richer behaviour to "static" URLs, like the possibility to include common informations (e.g. a common background) or to respond in different ways to requests coming from distinct place. Stub files can be used to activate processes. Parameters given by the client can be passed to the action-performing process.
The mechanism is similar to the CGI but we believe that it is easier to use for non-experts, and that some flexibility is added. For instance, Active Files can present the same behaviour when accessed via the HTTP server and by other application programs, such as file viewers or editors. The access can be uniform, regardless of the tool used for accessing the information contained in the file.

Active Files (AF) can therefore be regarded as a way to extend the concept of CGI (i.e. to associate an action - the execution of a script - with a "page") even to other than the HTTP daemon. In the next section we will discuss the general idea of Active Files. Section 3 discusses some general implementation issues of AF. In section 4 we will present the application of the Active Files to HTTP servers. Section 5 contains a few examples of use, and discusses advantages and limitations with respect to CGI scripts. The paper is completed by a few concluding remarks, acknowledgments and references.

2. Active files: the concept

Almost all tasks usually performed with computers involve the treatment of "files". As a consequence, a large portion of the capabilities of a modern Operating System (OS) is devoted to the treatment of files. Files can be read, written, copied and modified by means of standard OS commands.

Some OS-specific actions can be associated to the operation of accessing a file. For instance, permission for reading or writing a file is matched with the user characteristics. It is however not possible to associate complex user-defined operations to files, independently of the process requesting to read or write the file. A file is normally conceived as a static entity, its content is modified only as a consequence of the operations explicitly performed by a user or by a program.

We propose an extension of the functionalities of the OS, allowing the user to specify an activity associated to the access to a file. This association should be definable for any kind of file. For example, when a request for reading is issued, the file contents could be modified or updated, complex security checks could be verified, or a program could be executed. Since the file is modified when it is opened, its contents is not static and can be determined only by means of the actual read operation.

In order to be general and portable, the OS extension must not rely on modification of the OS source code, nor on modification of the source code of the application programs. The extension should be placed between the OS services and the application programs, intercepting the requests being made to the OS. The request can then be associated to the execution of the user-defined action, thus extending the functionalities of the OS. This capability has been used by Kom at al. (Ko93) for "adding a new dimension to the file system" and by Krishnanmurti et al. (Kr93). In this way it is possible to define a strategy for associating an operation to each single file. An AF is a normal file with a set of operations associated to the access to the file contents.

Not all the flavors of the Unix OS allow such an operation. We use a feature of Unix System V: the possibility of specifying a library that should be loaded before all the other libraries. If such a library contains replacements for the OS routines, then the replacement will be executed in place of the actual routine (the first name found is the one chosen). We successfully tested such feature on Solaris, HPUX, Digital OSF and Irix.

3. Active Files: the implementation.

The behavior of the Unix OS can be tailored by means of several environment variables. One of these is the LD_PRELOAD variable [LD_PRE] under Solaris or _RDR_LIST under Irix and Digital's OSF, which can be used for specifying a library of routines to be loaded before all the other libraries.

The general idea for supporting the AF paradigm is to develop a set of function with the same names of the System routines associated with file operations (we put our focus on the "open" operation). If the linker finds two functions with the same name, the address of the first one is associated with the name and the second is ignored. Therefore, if the customized function is loaded by means of the LD_PRELOAD mechanism, it will be executed in place of the corresponding system

call. The customized version of the function will first perform the AF operations, and then it will give the control to the original system function.

Concurrent access to a single file must be arbitrated by means of semaphores and shared variables. The performances of the system and of the process accessing an Active File are only slightly affected by the preloaded library. A first set of test has given only a small increase in execution time; such increase depends on course on the number of file accesses and on the action performed, and therefore can vary from case to case.

The operation performed by the preloaded library is to intercept the "Open" System Call and, before giving control to the System function, to verify if a special control file is present. Such control files can be associated to single ordinary files or to whole directories, therefore specifying common actions to be performed on all files in the directory.

The control file contains a description of the action associated with the Active File. The action is described by means of a simple language (which will be described elsewhere [Dau95]) in terms of pairs \texttt{<condition><action>}

Some of the currently supported \texttt{actions} are:
- execute of a user or system program;
- execute of a shell script;
- send a mail message to a user (possibly the file owner);
- increment a counter of the accesses to the file (there can be a set of counters, for instance one for each group of users);
- change the name of the referenced file;
- allow/deny the access to the file to single users or groups of users.

The \texttt{condition} part of the pair can be constructed by testing one or more of the available parameters. Some of the parameters are:
- \texttt{id} and \texttt{name} of the process attempting to access the file;
- \texttt{id} and \texttt{name} (real and effective) of the owner of the process;
- \texttt{name} of the file being accessed (a whole directory or group of files can be specified);
- \texttt{date} and \texttt{time} of the access operation.

These parameters are also available to be passed as arguments to the process or shell executed in the \texttt{action} part.

A powerful action is the execution of a program for modifying the file contents. For instance, part of the information in the file can be updated by consulting a database or user information sources. The preloaded function takes care of avoiding infinite loops by not intercepting subsequent accesses to the file.
the Active File is not performed.

In summary, it is possible for a single user to customize the behavior of the file system to a large extent without modifying the OS itself or the application programs. We are currently working on a graphical tool which should help working with AF, so that even inexperienced users can define and modify the behavior of a file.

More details of the implementation will be given elsewhere [Dan95].

4. Active Files and WWW

Although the AF project is currently being developed for general applications, it was originally started for extending and customizing HTTP servers; in particular to offer an alternative to CGI.

The operation of a CGI script can be summarized with the following steps:
1. the client (browser) issues a request for a specific document;
2. the HTTP server recognizes the requested document as a CGI script;
3. the application program is executed;
4. the program output is associated to the browser input.

The use of AFs can simplify and make more flexible the use of on-line modified documents. The sequence of operations performed when using AF is the following:
1. the client (browser) issues a request for a specific document;
2. the HTTP server access the requested document as a normal file;
3. if the requested document is an AF, and there is an action associated with the HTTP server, the associated action is performed;
4. the document contents is (possibly) modified, or statistics are collected, or any other requested action is performed, and the HTTP server continue with its normal operation, transmitting the file content to the browser.

It must be stressed that the HTTP server does not treat the file differently from other processes, and the file which specifies the associated actions is placed in the same repository as the AF. Each user can therefore customize the access to one or more files without the need to insert a new CGI script in the (system maintained) CGI repository.

One important point is still to be discussed: in order to make AF competitive with CGI, it is necessary to be able to pass parameters from the Web client to the process responding to the request on behalf of the HTTP daemon.

In the typical HTTP daemons, a parameter-passing routine is activated whenever the requested URI is recognized as a CGI request. In general this routine is not activated for static URI. Using AF, any file could have an active behavior associated to it: the parameter-passing routine will therefore be activated in most cases.

It is possible to devise solutions which solve the problem by applying a small patch to the HTTP server code; however we were not satisfied with this approach. Therefore we devised a simple alternative: when the AF library is triggered a function of the OPENing process can be fired. Next figure describes the sequence of events.

![Figure 3](image)

When the HTTP daemon tries to open an active file (1), the open() call is trapped and the control is passed to the AF library. Here a check is performed to see if a routine in the calling process must be activated (a reference table associates processes and functions). In particular, in the case of an HTTP server the AF Library calls (2) the HTTP daemon routine for getting the parameters from the client (e.g. through the usual GET and POST methods). When this routine ends (3), control is returned to the AF library, which can perform the necessary action (4). In such way the process activated by the AF mechanism can get parameters coming from the originating client.

5. Examples and discussion

In this section we present some examples of AF use. In these examples we also compare AF and the CGI mechanism so as to show the additional flexibility allowed by the mechanism we implemented.

Example 1: Including common informations in files.

The current definition of HTML lacks an INCLUDE directive, which would facilitate the maintenance of large projects. Future versions of HTML might solve the problem, but at the moment we have to live with it. Some servers offer their own extensions to cope with this problem, but the solution is not general.

For instance, it would be useful to be able to define a common header and footer and to include a common background to a set of Web pages. A similar problem can arise with the use of Netscape Frames, where often one has to provide the same information twice: once for the frame-capable browsers, and once within the <NOFRAMES> brackets for the other browsers. A solution can be devised with CGI: pages can be composed on the fly by putting together the needed elements.
(body, header, footer, background) passed as parameters (some of which can be taken as default by the script activated by the CGI); however such solution is rather cumbersome.

One of the major applications of AF is the possibility of modifying the information contained in the file before the file content is transferred to the requesting process (the HTTP server in our case). In order to simplify this kind of operation we defined an extension of the <action> part defined in section 3, which activates a parser which recognizes some directives. The available commands allow to:

- include current date
- include current time
- include a file
- include the output of a process

and can easily be extended to support other functionalities. As a result, the Active File Library edits the file content before allowing the requesting process to access the file itself. We would like to stress again that such behavior is not limited to accesses done from the HTTP server, but is uniform throughout the system (so that a "cat" command would also present the edited content). Access to the unmodified file content can be achieved (e.g. for maintenance) by disabling its associated control file

Example 2: Responding in different ways at different time, or to different machines.

Sometimes it would be nice to show a different content to different machines (e.g. one could wish to hide some information when responding to requests originating outside the local domain, or use French to respond to requests coming from "fr" and Italian for those coming from "it"). In other cases the information provided could depend (statically) on the time of the day, or on the day of the week: e.g. the page could start with a "Good morning", a "good afternoon", and contain either "Our friday special is fried fish" or "On Saturday we suggest our Gnocchi al Ragu specialy".

Such behaviour can of course be obtained with CGI, but achieving it is not straightforward for a generic user. AF allow to obtain this behavior more easily: a normal file acts as a placeholder and, according to the conditions specified in the associated control file, other files are opened at its place. A demo of such behavior can be seen at:

http://www.inf.univ.it/~dpastore/active/demo.html

where the server (a normal HTTP server) was started with the LD_PRELOAD variable set to the Active File Library.

Example 3: Gathering statistics on the access to Web pages

It is frequent to find a page where a graphic counter shows the number of visits since some starting day. One of the most popular mechanism relies on a process that is activated when the image is loaded. Such process retrieves the counter value, increases it by one, stores the new value, prepares the image and delivers it. Actually such mechanism only counts how many times the image has been downloaded: visitors using textual browsers like Lux (or Netscape with disabled image downloading) are like ghosts who can enter the room without being noticed. Other mechanisms can rely on CGI or on a batch procedure which at fixed times examines the log files.

AF can gather statistics in a simpler way: a counter is incremented every time the file is OPENed (one can restrict the counter to count only the "open" performed by the HTTP server). Moreover, with a single directive the counting can be performed for all pages in a given directory.

Again, a demo of this AF usage can be reached through:

http://www.inf.univ.it/~dpastore/active/demo.html

Example 4: Who is enabled to create active gateways?

The CGI approach requires, under some HTTP servers, that the executable files be gathered in a specially designated directory (the HTTP server in our case). The only possibility to permit to all users to build their own active gateways is to make the CGI-directory writable for everybody. In such case, the only solution is inconvenient. With the AF approach, the active files can be anywhere, and therefore the problem is solved.

AF can give an advantage even in cases (like the NCSA server) where the webmaster has more flexibility. With the NCSA server it is for instance possible to define a set of directories where the executables should be gathered (via the ScriptAlias directive) or to declare that all files with a particular extension (defined through the AddType directive) can be activated through the CGI mechanism.

Let's imagine that the webmaster wants to grant the power of building gateways to a (subset of) users. The definition of a special extension would enable any user to create his own active gateways, and is therefore not the desired solution. The alternative options are:

- to gather the authorized users in a group, and make the cgi-directory writable for everybody in that group;
- to enable a list of directories (one for each of the authorized users).

The first solution is not practical, since users could interfere with each other. The second is cumbersome since it requires a certain amount of administrative bookkeeping.

AF allow a more elegant solution: it is possible to limit the active behavior for a specific process (e.g. httpd) to any file belonging to a given set of user (or to a given group), thereby reducing the needed administrative load.

6. Conclusions

We presented the concept and applications of Active Files for Unix systems, both in general and for WWW use. The advantages of such approach are mainly in the possibility of extending and customizing the behavior of any application program without modifying the program itself or the Operating System. The main drawback is that Active File can only be used on Unix System V systems.

Among the subsystems that can benefit from the AB extension, HTTP servers are one of the most suitable. Improved security check, updating information on the fly and simple replacement of CGI scripts are possible applications. The advantage with respect to CGI scripts is mainly in the possibility of allowing the transformation of any file into a file containing self-updated information. Besides that, the access to an AF is completely uniform, its content will be updated if it is accessed via a remote connection to the HTTP server or via a simple Unix "cat" command. Finally, we believe that the AF mechanism is easier to use for non-expert users than the Common Gateway Interface.

We therefore believe that this approach can improve the functionality of HTTP servers, without being tied to a specific implementation of one or another server.

7. Acknowledgements

We are grateful to Balachander Krishnamurthy who introduced us to the trapping of the system calls.

8. References

[Kor90] D.G.Korn, E.Krell A new dimension for the Unix file system, Software Practice and Experience 20,51/19-51/34, June 1990
Why and How Not to Use Plain HTML

D. Muller
Centre de Génie Electrique de Lyon - URA CNRS 829
Ecole Centrale de Lyon
email: muller@trotek-ec-lyon.fr

Abstract
The developers and maintainers of Web servers are faced to specific problems like hiding the pages under development, generating multi-lingual pages (e.g. in English, French, German, ...), mirroring often accessed pages to other servers, getting paper copies of the server contents, etc. We propose a method solving all these problems and offering additional features like the possibility to define macros resulting in complicated sequences composed of many html tags.

Keywords
World-Wide Web, html, Authoring environments.

1. Introduction
At first, the development of a Web server had to be made using one's favorite editor to create the documents by writing plain html. After a reasonable delay, Lynx (What You See Is What You Get) html editors appeared, either written from scratch, or resulting from a browser added to an existing text editor or from some interaction added to an existing browser. On Jan 11th 1996, a list of roughly 80 such editors was accessible on the Web [5]. However, we think that these tools are generally not adapted to professional development of big servers. This needs some more explanations.
In fact, surfing over the net rapidly shows that there are two kinds of servers - this distinction not at all being related to the quality of the documents or of their contents - the professional servers being proposed by companies, universities or other institutions, and the hand-made servers often known as "personal home pages". What we call the "professional servers" - due to their often important number of documents and robustness requirements - need optimized development and maintenance methods relying as much as possible on automatic procedures. In addition, the need for distributing the information world-wide with often poor net performances, imposes to mirror the information using for example a country-based strategy. Being French, we could not avoid mentioning the problem of maintaining various versions (at least two) of the documents written in different languages, being in our case French and English.
Finally, let us point out that a simple observation of the world of computer users will show that despite the large number of available mouse-driven drag-and-drop smart text editors, there is still a large community of people preferring old-fashioned but powerful tools like vi or emacs. The tools and methods we propose in this paper try to fix the problems above and are more particularly intended for this second family of users, more probably issued from the world of programming than from the world of text editing.