**UNIVERSITE LIBRE DE BRUXELLES**
Faculté des Sciences appliquées

Année académique 2001 - 2002

# Extraction par déformatage du contenu des pages web dynamiques semi-structurées

Directeur de mémoire:
Mr **Esteban Zimànyi**

Travail de Fin d'Etude
présenté par **Gérard Materna**
en vue de l'obtention du grade
d'**Ingénieur Civil Informaticien**

# Remerciements

Je tiens à remercier Monsieur Esteban Zimànyi, chargé de cours à la Faculté des Sciences Appliquées de l'Université Libre de Bruxelles (ULB), et directeur du service informatique de la Faculté des Sciences Appliquées de l'Université Libre de Bruxelles, pour avoir accepté d'encadrer et de promouvoir ce Travail de Fin d'Études, et pour m'avoir encouragé dans la voie que j'ai choisi.

Un tout grand merci à Monsieur Jean-Pierre Norguet, mandataire de recherche FNRS, doctorant au Service d'Informatique de l'Ecole Polytechnique et responsable du projet WASA, dans le cadre duquel s'inscrit ce Travail de Fin d'Etude, pour sa patience, son temps, et ses précieux conseils.

Un grand merci aussi à Olivier Samyn, Chercheur au Service d'Informatique de l'Ecole Polytechnique, pour son aide et ses conseils techniques.

Je remercie également les programmeurs avec qui j'ai pu collaborer activement par l'intermédiaire de l'internet et dont les réalisations m'ont souvent été indispensables pour mener à bien mon travail. Je tiens à exprimer ma gratitude envers tous les participants de la mailing list des développeurs de modules Apache.

Merci également à toutes les personnes de mon entourage, qui m'ont supporté, encouragé et relu.

# Résumé

Ce document présente le travail que j'ai réalisé dans le cadre de la thèse de doctorat de Monsieur Jean-Pierre Norguet : « Mise en ligne d'informations relatives à l'analyse sémantique de la consultation de sites web dont le contenu présente une évolution temporelle complexe et une mise en forme à la fois statique et dynamique. »

Le but de ce travail consiste à résoudre le problème de la mise à disposition du contenu textuel et significatif des pages web dont la mise en forme est essentiellement dynamique, en vue de son analyse sémantique et statistique ultérieure.

Les technologies actuelles dans le domaine de la génération dynamique de page web sont très diverses et connaissant une évolution explosive. Les nouvelles potentialités apportées par XML et ses technologies dérivées (XSLT, XML Schema, ebXML, SOAP, UDDI, Xlink…) révolutionnent actuellement ce domaine.

Les solutions proposées dans ce travail sont issues d'une analyse approfondie de ces technologies, de leurs évolutions et de leurs influences sur les techniques de mise en forme dynamique du contenu sémantique de pages web.

Ce document reprend l'ensemble de l'évolution de l'étude de ce problème bien particulier pour aboutir à une solution stable autant dans son implémentation que dans le temps, dans le sens où elle est devenue indépendante des technologies de génération de pages web et de leurs évolutions.

# Avant-lecture

## Conventions

Dans ce document, différentes conventions de langage pouvant être inhabituelles sont utilisées.

Voici donc quelques expressions pouvant porter à confusion et dont il peut être utile de préciser le sens.

**Dynamique** : se réfère aux actions qui prennent place au moment où elles sont nécessaires plutôt qu'à l'avance.

**Journalisation** : action d'enregistrer dans un fichier journal (log).

**Meta** : préfixe commun en informatique, qui signifie « à propos ». Les meta-données sont des données à propos de données.

**Meta-information** : information à propos d'informations.

**Mise en ligne** : on dira qu'une information est mise en ligne lorsqu'elle est consultable à partir d'un navigateur web connecté à l'internet.

**Page web** : un document sur le World Wide Web.

**Page dynamique** : page web dont le contenu est généré dynamiquement au moment où la requête a été soumise au serveur web pour le document.

**Page dynamique brute**: page web dynamique contenant du code exécuté dynamiquement par le serveur web (server-side script). Ces pages sont reconnues par le serveur grâce à leur extension (.asp, .jsp …). Par convention, « brute » signifie que l'on fait référence à la page telle qu'elle est stockée sur le disque dur, avec son contenu statique et son script, et non à la page statique, souvent sous forme de HTML, résultante de la génération par le serveur.

**Page dynamique générée**: page web dont le contenu a été généré dynamiquement au moment où la requête a été soumise au serveur web pour le document. Par convention, « générée» signifie que l'on fait référence à la page statique, souvent sous forme de HTML, résultante de la génération par le serveur.

**Serveur web** : toute confusion doit être évitée entre un serveur physique proposant le service HTTP et le logiciel exécuté par la dite machine réalisant ce service, la distinction doit donc se faire entre le matériel (hardware) et le logiciel (software).

**Site web** : un site web est un ensemble de pages HTML en ligne, disposées selon une arborescence ; cette arborescence correspond à celle d'un répertoire sur le disque physique du serveur web. On dira qu'un serveur web héberge plusieurs sites lorsque le disque contient plusieurs arborescences mises en ligne, physiquement et logiquement indépendantes.

## Anglicismes

Différentes locutions anglaises seront utilisées dans ce document. Voici quelques indications permettant de mieux les interpréter dans les contextes où elles apparaîtront.

**Browser** : navigateur internet.

**Download** : téléchargement, transfert du serveur vers le client.

**Internet** : réseau informatique mondial.

**Handler** : gestionnaire.

**Log** : fichier journal.

**Logging** : journalisation, enregistrement dans un fichier journal.

**Multi-tier** : se dit d'une architecture client/serveur multiniveau ou multicouche.

**Open-source** : dont les sources sont librement accessibles.

**Process** : processus.

**Server-side scripting** : technologie de scripting où le script est exécuté sur le serveur plutôt que sur le client.

**Servlet** : application Java s'exécutant sur le serveur et respectant une API prédéfinie.

**Stateless** : dont l'état n'est pas conservé, non persistant.

**Tag** : balises utilisées dans les langages balisés, comme HTML.

**Thread** : partie d'un programme qui peut s'exécuter indépendamment du reste du programme.

**Thread-safe** : se dit d'un programme qui peut exécuter tous ses threads sans risque d'interférence.

**Web** : toile, service HTTP du réseau internet.

## Abréviations

**ANSI** American National Standards Institute
**API** Application Programming Interface
**ASP** Active Server Page
**CGI** Common Gateway Interface
**COM** Component Object Model
**CSS** Cascading StyleSheet
**DBMS** DataBase Management System
**DNS** Domain Name Server
**DOM** Document Object Model
**DTD** Document Type Definition
**EbXML** E-business XML
**FNRS** Fonds National de la Recherche Scientifique
**FTP** File Transfert Protocol
**GIF** Graphics Interchange Format
**GNU** GNU is Not Unix
**GUI** Graphical User Interface
**HTML** HyperText Markup Language
**HTTP** HyperText Transfer Protocol
**IP** Internet Protocol
**ISAPI** Internet Server Application Programming Interface
**ISO** International Standard Organization
**JSP** Java Server Page
**NSAPI** Netscape Server Application Programming Interface
**SAX** Simple API for XML
**SOAP** Simple Object Access Protocol
**SQL** Sructured Query Language
**MIME** Multipurpose Internet Mime Extension
**OSI** Open System Interconnection
**PC** Personal Computer
**PHP** PHP: Hypertext Preprocessor
**TCP** Transfer Control Protocol
**TFE** Travail de Fin d'Études
**UDDI** Universal Description, Discovery and Integration
**UDP** User Datagram Protocol
**ULB** Université Libre de Bruxelles
**URI** Universal Ressource Identifier
**URL** Universal Resource Locator
**XHTML** eXtensible HyperText Markup Language
**XML** eXtensible Markup Language
**XSL** eXtensible Style Language
**XSLT** eXtensible Style Language Transformation
**WWW** World Wide Web

# Table des matières

# Tables des figures

# 1 Introduction

Le présent travail s'inscrit dans le cadre de l'application WASA, projet beaucoup plus large, dirigé par Monsieur Jean-Pierre Norguet, mandataire de recherche FNRS, et qui traite du sujet suivant : « Mise en ligne d'informations relatives à l'analyse sémantique de la consultation de sites web dont le contenu présente une évolution temporelle complexe et une mise en forme à la fois statique et dynamique. »

## 1.1 Sujet de ce travail

Ces dernières années, l'idée que l'on se fait d'un ordinateur a beaucoup changé. Avec l'introduction de l'Internet et du World Wide Web, nous pouvons accéder à des informations et même faire du commerce depuis n'importe où.

Un grand nombre de technologies sont apparues pour combler la demande croissante d'applications web rapides, légères, robustes, et surtout interactives. Dans un premier temps, l'interactivité et le contenu dynamique ont été délivrés par des programmes CGI. Bien vite, des solutions de scripting comme JSP, ASP et PHP ont fourni une interface web aux composants utilisés pour gérer la logique des applications et l'accès aux sources de données, accélérant et facilitant ainsi le développement des applications.

La complexité de ces applications web et la diversité des technologies utilisées rendent l'analyse sémantique de la consultation du contenu de tels sites web impossible avec les outils de journalisation (logging) actuels.

Le but de ce travail consiste à résoudre ce problème et à mettre le contenu textuel et les meta-informations de consultation de ces pages web dynamiques à disposition de l'application WASA, en vue de leur analyse sémantique et statistique.

## 1.2 Description de ce document

Ce document reprend l'ensemble de l'étude de ce problème bien particulier pour aboutir à une solution originale et stable.

Il est construit à la manière d'un parcours. Il nous mène au fil des analyses, des solutions et de leurs implémentations, pour aboutir à une solution finale, suivie de conclusions originales et surtout de très intéressantes perspectives.

Le **Chapitre 2** introduit les technologies de base du web et des pages dynamiques.

Le **Chapitre 3** expose la première solution envisagée à notre problème et ses limites. Elle consiste à extraire le contenu statique des pages brutes se trouvant sur le serveur web.

Le **Chapitre 4** présente les nouvelles techniques et technologies de plus en plus utilisées dans le domaine des pages dynamiques. Elles sont principalement basées sur XML.

Le **Chapitre 5** explore plus en profondeur les mécanismes de la communication entre le serveur web et le navigateur dans le cas des pages dynamiques. Il expose et compare aussi deux nouvelles solutions basées sur les résultats de cette analyse : l'écoute du réseau et le module additionnel du serveur web.

Le **Chapitre 6** développe la structure générale du serveur web Apache, son cycle de vie et son API.

Le **Chapitre 7** présente une implémentation de la solution du module additionnel pour le serveur web Apache. Son but est d'intercepter une copie des données envoyées au navigateur après mise en forme.

Le **Chapitre 8** expose les conclusions et les perspectives de l'ensemble de ce travail.

# 2 Bases des pages web dynamiques

## 2.1 Bases du web

Le but de ce chapitre est de donner une vision globale des technologies à la base du web. Plus d'informations peuvent être trouvées dans [1,2,3,4,5,6,7,8,9].

Au début de l'histoire du World Wide Web, toutes les informations accessibles aux navigateurs étaient purement statiques. Une information de l'utilisateur pouvait être capturée au moyen d'un formulaire HTML et transmise au serveur web.

Le fonctionnement du web est basé sur le modèle requête/réponse qui est implémenté sous la forme du protocole HTTP (HyperText Transfer Protocole).

### 2.1.1 HTML

HTML (HyperText Markup Language) est le langage utilisé pour créer des documents sur le World Wide Web. Il définit la structure et l'affichage d'un document web en utilisant de nombreux tags et attributs.

### 2.1.2 HTTP

HTTP (HyperText Transfer Protocol) définit comment les messages sont formatés et transmis, et quelles actions les serveurs web et les navigateurs doivent entreprendre en réponse aux diverses commandes.

Le web définit deux parties, un client web (navigateur) et un serveur web, et leur relation est du type requête/réponse. Le client fait une requête pour une ressource spécifique et le serveur répond avec la ressource requise, si celle-ci existe.

Chaque document sur le web est adressé de manière univoque par son URL (Universal Ressource Locator).

Une des caractéristiques intéressantes d'HTTP est sa simplicité. La relation entre le navigateur et le serveur est très courte, dès que le document est reçu, la connexion se coupe. On dit que la connexion entre le serveur et le navigateur n'est pas persistante (stateless), le lien qui les unis n'est pas maintenu d'une requête à l'autre.

## 2.2 Applications web

Le contenu statique des pages web est vite devenu limitatif, une fonctionnalité aussi simple qu'une recherche textuelle n'est possible que si les résultats peuvent être affichés à l'utilisateur. Or une armée de concepteurs de pages web ne suffirait pas à créer les milliers de pages de résultats en anticipant toutes les recherches possibles qu'un utilisateur pourrait faire.

Le succès populaire et l'engouement commercial pour Internet ont été les moteurs de l'évolution de ce premier modèle statique du web vers un modèle interactif, où le contenu des documents est généré selon les besoins. C'est l'avènement de l'Internet comme plateforme de déploiement d'applications web.

### 2.2.1 Le modèle client/serveur multiniveau

L'architecture des applications web découle de l'évolution de modèle client/serveur, qui est l'un des paradigmes les plus dominants des technologies de l'information.

L'idée centrale de ce modèle client/serveur est de fournir une architecture d'application qui permette à un processus informatisé d'être partagé en plusieurs tâches moins complexes coopérant via un mécanisme de messagerie. La notion clef du partage du problème est de fournir des couches (niveaux) de fonctionnalités qui peuvent être écrites séparément et déployées sur plusieurs machines d'une manière efficiente.

Un exemple typique de découpage d'une application en couches fonctionnelles est le suivant.

- Logique de présentation (presentation logic) : gère la façon dont l'utilisateur interagit avec l'application via une interface utilisateur (GUI).
- Logique de l'application (business logic) : gère les règles de l'application (business rules).
- Logique d'accès aux données (data access logic) : gère le stockage et la récupération des données.

Le découpage de la logique de l'application en couches permet de minimiser l'impact des modifications d'une couche, de rendre possible la réutilisation du code et d'augmenter sa fiabilité.

Le modèle multiniveau (multicouche ou encore multi-tier en anglais) est un aboutissement logique du modèle client/serveur où:

- Le media utilisé par la logique de présentation est indépendant de la logique d'application,
- la logique d'application est partagée et distribuée sur plusieurs machines, et

- la logique d'accès aux données peut supporter plusieurs bases de données ainsi que d'autres services comme des data warehouses, legacy systems, etc.

La figure suivante (extraite de [5], page 11) schématise les couches d'une application multiniveau.

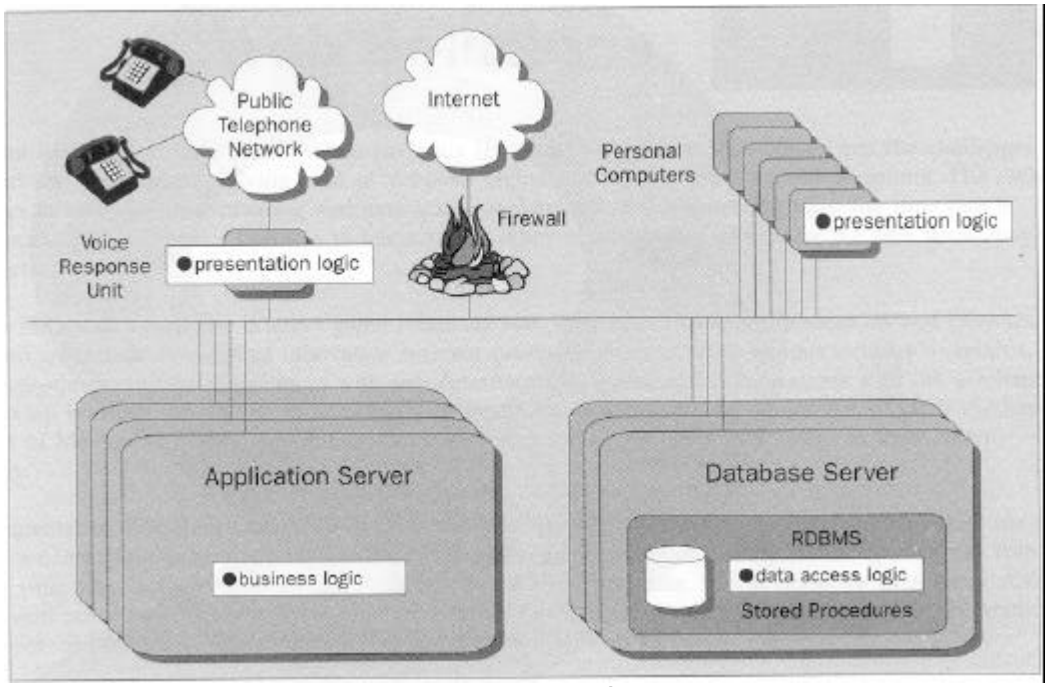


*Fig. 2.1: Application distribuée multiniveau.*

Cette solution est la plus flexible et la plus extensible.

### 2.2.2 Applications web

La demande de contenu dynamique sur le web a transformé la programmation web vers de nouvelles implémentations de l'architecture multiniveau.

La figure suivante (extraite de [6], page 13) montre une application web simplifiée suivant un modèle trois-niveaux (three-tiers).

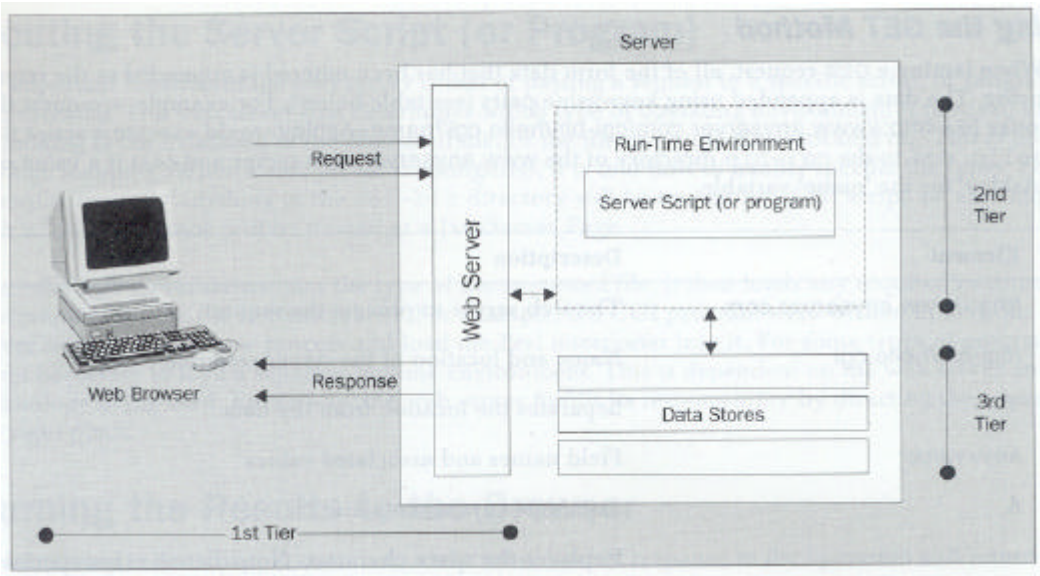


*Fig. 2.2: Application web simplifiée en architecture trois-niveaux.*

La couche de présentation forme le premier niveau qui inclut non seulement le navigateur web mais aussi le serveur web, qui est responsable de l'assemblage des données dans un format présentable. Le deuxième niveau est formé par la couche de l'application (business layer), qui consiste en un ensemble de script, de programmes ou de composants. Finalement, le troisième niveau fourni au deuxième les données dont il a besoin.

Bien qu'HTML permette de créer des documents web qui sont des interfaces utilisateur délivrables par tout serveur web et lisibles par tout navigateur, il reste un langage de description de données statiques par nature. CGI fut la première technologie à apporter un peu d'interactivité et de contenu dynamique au web. Bien vite suivie par des implémentations d'APIs spécifiques aux serveurs web, comme NSAPI et ISAPI. C'est ensuite qu'apparurent des solutions de script du côté serveur (server-side scripting) comme JSP et ASP, qui ont simplifié le développement d'applications web.

### 2.2.3 CGI

CGI (Common Gateway Interface) est la technologie côté serveur la plus commune, tous les serveurs web d'aujourd'hui la supportent.

Un programme CGI peut être écrit dans n'importe quel langage, mais le plus populaire est Perl. Les serveurs web implémentant CGI agissent comme une passerelle (gateway) entre la requête de l'utilisateur et les données demandées. Il réalise cela en commençant par créer un nouveau processus dans lequel le programme sera exécuté (voir la figure ci-dessous, extraite de [6], page 16). Il charge ensuite le moteur d'exécution (run-time environment) requis ainsi que le programme lui-même. Finalement, il invoque le programme en lui passant un objet requête. Quand le programme est fini, le serveur lira la réponse sur la sortie standard (stdout).

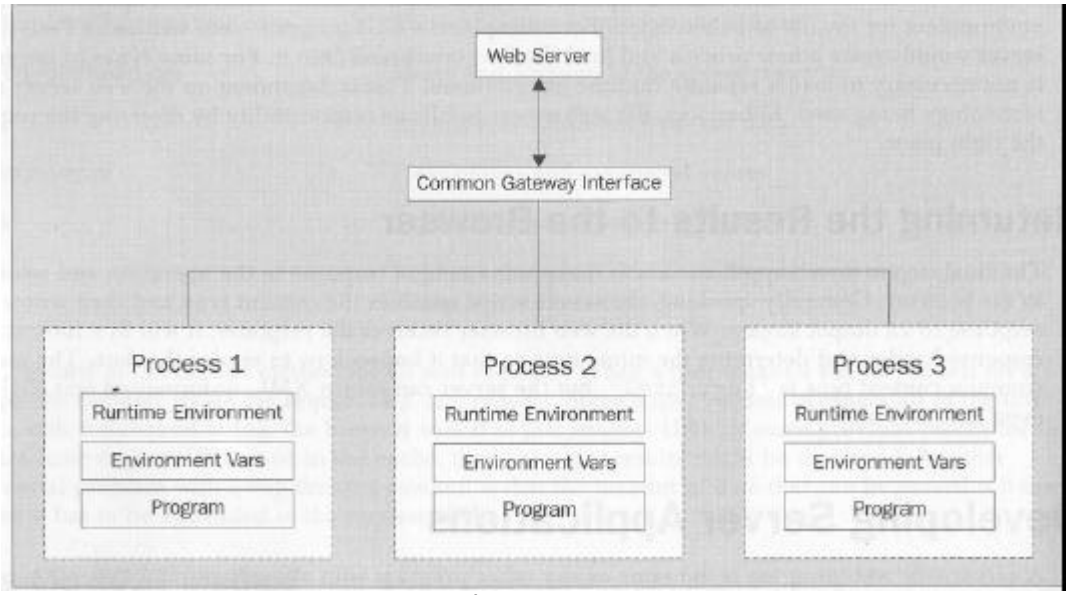


*Fig. 2.3: CGI.*

Le plus grand désavantage de CGI est qu'il n'est pas très extensible. Chaque fois qu'une requête est reçue par le serveur web, un nouveau processus complet est créé, possédant ses propres variables d'environnement, sa propre instance de moteur d'exécution, une copie du programme, et une allocation de mémoire à disposition. Le serveur devient donc extrêmement sollicité dès qu'il y a un certain nombre de requêtes concourantes.

### 2.2.4 Web Server APIs

En réponse aux inefficiences de CGI, Microsoft et Netscape ont chacun développé leurs propres APIs pour permettre aux développeurs d'écrire des applications sous forme de librairies partagées.

Ces librairies sont conçues pour être chargées dans le même processus que le serveur web et pour traiter plusieurs requêtes sans devoir créer de nouveaux processus.

Mais ces librairies ont aussi leurs problèmes :
- elles dépendent du système d'exploitation pour lequel elles ont été développées,
- elles doivent être thread-safe, vu qu'elles sont accédées par plusieurs utilisateurs en même temps, et
- si un programme cause une erreur fatale, cela peut bloquer tout le serveur, vu qu'ils sont dans le même processus.

### 2.2.5 Server-side scripting

La programmation d'application web est rendue encore plus facile grâce à l'introduction d'un framework complet disponible depuis un script intégré directement à la page web.

Les pages dynamiques sont donc constituées d'un mélange d'informations statiques, souvent en HTML, et de scripts destinés au serveur. Lorsque le server reçoit une requête HTTP pour un tel document, une réponse est générée en mémoire en utilisant une combinaison de l'information statique et de celle générée par le script.

Le framework fournit au script toute une série d'objets intrinsèques au serveur qui gèrent automatiquement un grand nombre de tâches, et donc minimise ainsi la quantité de développement requis. Ces objets fournissent généralement aux scripts le concept de session d'utilisateur et permettent aux variables de persister d'une page à l'autre jusqu'à ce que la session soit abandonnée ou que le browser soit fermé. Ils fournissent aussi une grande quantité d'informations sur le contexte et un accès direct aux composants de la couche de logique de l'application.

## ASP et COM

Cette technologie Microsoft combine du HTML, du code script, et des composants serveurs en un fichier appelé Active Serveur Page (ASP).

Lorsque le serveur reçoit une requête pour un fichier ASP (reconnu par son extension.asp), il va d'abord chercher la page compilée et va l'exécuter ensuite. Si la page n'a pas encore été compilée, le serveur la compile et l'exécute. Le résultat est un document web retourné au navigateur.

Une page ASP peut être écrite en utilisant du HTML, du JScript et du VBScript. Le script est délimité par des balises : "<%" indique le début et "%>" la fin.

Au travers du script, l'ASP peut accéder aux composants du serveur. Ces composants peuvent être écrits dans n'importe quel langage tant que celui-ci fournit une interface COM (Component Object Model, la spécification des composants Microsoft).

Un des seuls désavantages des ASP est qu'elles ne peuvent être utilisées qu'avec un serveur web Microsoft sur un système d'exploitation Windows.

## JSP et Servlets

Une page JSP (Java Server Page) est très similaire à une ASP. Elle contient du HTML, du code Java et des composants JavaBeans.

Le script est lui aussi délimité par des balises et comme pour les ASP : "<%" indique le début et "%>" la fin.

Une servlet est un programme Java exécuté du côté serveur, qui traite une requête HTTP, et retourne une réponse HTTP.

Lorsque le serveur reçoit une requête pour une JSP, il va d'abord générer la servlet correspondante, sauf si celle-ci existe déjà. Il va ensuite

invoquer cette servlet et retourner le document web résultant au navigateur.

### PHP, ColdFusion et les autres

Il existe de nombreuses autres technologies de génération de pages dynamiques : PHP, Zope, ColdFusion…

Bien qu'elles diffèrent à peu près toutes au niveau des langages de script utilisés, leurs architectures sont très similaires à celles des JSP et des ASP.

## 2.3 Les techniques de programmation

Les techniques de programmation de pages web dynamiques ont connu une période de stabilisation qui prend fin avec l'introduction massive du XML comme support de stockage et de transfert d'informations.

L'idée générale qui ressort de [1,2,3,4,5,6,7,8,9] est que le contenu statique est stocké sous forme de HTML dans la page dynamique brute, et le contenu dynamique provient d'une base de données et est obtenu via le script. Il en résulte qu'une bonne partie du contenu sémantique d'une page dynamique générée est déjà stocké dans la page dynamique brute sur le disque.

Nous verrons plus loin comment XML a révolutionné ces techniques en diminuant le contenu sémantique stocké dans la page brute, celui-ci pouvant, entre autre, provenir de fichiers XML appelés via le script.

# 3 Extraction du contenu par déformatage

## 3.1 Introduction

Après étude des techniques de programmation basées sur le server-side scripting, il ressort qu'une bonne partie du contenu sémantique des pages dynamiques est déjà stocké dans la page sous forme d'HTML. Seul le contenu vraiment dynamique est obtenu d'une base de données via le script.

Une première approche de la solution au problème posé, à savoir mettre le contenu sémantique des pages web dynamiques à disposition de l'application WASA, consiste à extraire ce contenu des pages brutes par déformatage, c'est-à-dire en utilisant un compilateur (ou parseur) qui traduirait un langage source, constitué de l'HTML et du script, en un langage cible, constitué du seul HTML. L'application WASA n'aurait plus alors qu'à récupérer ce contenu et combiner cette information avec l'information recueillie dans les fichiers journal du serveur web.

L'idée générale est d'utiliser JavaCC (Java Compiler Compiler) pour générer un compilateur qui va prendre en entrée la page dynamique brute et va donner en sortie le contenu de cette page amputé du script. On obtient de cette façon le contenu HTML statique de la page.

Un affinement de la solution consiste à corriger l'HTML résultant et le traduire vers XHTML ou XML en utilisant les API JTidy.

Nous verrons ensuite quelles sont les limitations de cette solution et pourquoi elle n'est pas totalement satisfaisante.

## 3.2 Générer le compilateur avec JavaCC

Nous allons dans un premier temps introduire ce qu'est un compilateur. Une information plus complète peut être trouvée dans [10]. Ensuite, nous décrirons sommairement le format d'une grammaire JavaCC en prenant le cas des pages JSP comme exemple.

### 3.2.1 Qu'est ce qu'un compilateur?

En informatique, un compilateur est un programme qui traduit un code source en un code objet. Dans un sens plus général, c'est un programme qui traduit un langage source en un langage cible.

Le rôle du compilateur est illustré à la figure suivante (extraite de [10], page 15).

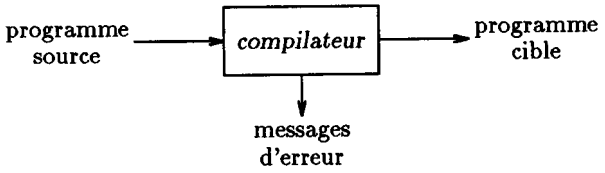


*Fig. 3.1: Rôle du compilateur.*

Il y a deux parties à la compilation :
- L'analyse, où le texte source est partitioné en ses constituants et où une représentation intermédiaire est créée.
- La synthèse, où le texte cible est construit à partir de cette représentation intermédiaire.

En général, c'est la partie synthèse qui requiert les techniques les plus spécialisées.

L'analyse comprend trois phases :
- l'analyse linéaire, au cours de laquelle le flot de caractères formant le programme source est lu de gauche à droite et groupé en unités lexicales, qui sont des suites de caractères ayant une signification collective ;
- l'analyse syntaxique, au cours de laquelle les caractères ou les unités lexicales sont regroupés hiérarchiquement dans des collections imbriquées ayant une signification collective ;
- l'analyse sémantique, au cours de laquelle on opère certains contrôles pour s'assurer que l'assemblage des constituants a un sens.

C'est la traduction dirigée par la syntaxe, une technique de compilation s'appuyant sur la grammaire, qui est utilisé par l'intermédiaire de JavaCC. Pour spécifier la syntaxe, nous utiliserons une notation appelée grammaire non-contextuelle ou BNF (Backus Naur Form).

La traduction dirigée par la syntaxe est illustrée à la figure suivante (extraite de [10], page 42).

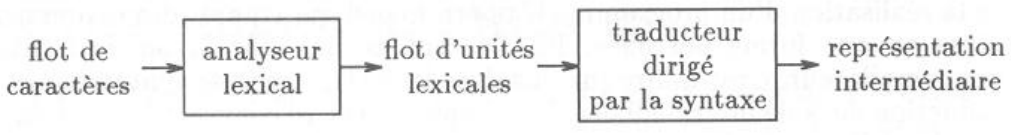


*Fig. 3.2: Traduction dirigée par la syntaxe.*

Dans le cas qui nous concerne, notre compilateur sera assez simple. Notre modélisation du langage source est très basique, il y a le script d'une part et un texte d'autre part. De même pour la modélisation du langage cible, c'est un texte sans trace de script. Nous n'aurons besoin que de deux unités lexicales, les analyses hiérarchique et sémantique seront quasi

inexistantes, quant à la synthèse, elle se borne à écrire en résultat les unités lexicales correspondant à du texte et à ignorer les unités lexicales correspondant à du script.

### 3.2.2 JavaCC

JavaCC est un outil 100% Java, qui permet de générer un compilateur à partir d'une « grammaire ». Le terme grammaire est un peu abusif, puisque cette « grammaire » de JavaCC spécifie non seulement la syntaxe du langage source, mais aussi les unités lexicales, les actions de construction de la représentation intermédiaire de l'analyse syntaxique et la synthèse du langage cible.

Le compilateur généré est lui aussi 100% Java et il fournit des méthodes pouvant servir de points d'accès à la compilation depuis un autre programme.

Le format de la grammaire JavaCC est assez complexe, et bien qu'il soit préférable de le maîtriser pour comprendre ce qui va suivre, nous allons renvoyer le lecteur vers [11] pour la définition complète et nous limiter à expliquer ce dont nous avons besoin.

### 3.2.3 Grammaire JavaCC

Nous allons maintenant utiliser le format de la grammaire JavaCC pour décrire les unités lexicales, la grammaire non-contextuelle et la synthèse, dans le cas qui nous intéresse.

#### Les unités lexicales

Les unités lexicales sont décrites à partir d'expressions régulières. Elles peuvent appartenir à 4 groupes (`TOKEN`, `SPECIAL_TOKEN`, `SKIP` et `MORE`) dont deux nous intéressent plus particulièrement :

- `SKIP` qui définit les unités lexicales qui sont ignorées, et
- `TOKEN` qui définit les unités lexicales ayant une signification dans la syntaxe du langage source.

Nous avons vu que dans le cas de pages dynamiques contenant du script, ce script était toujours délimité par des balises. Dans le cas de notre exemple (les pages JSP), ces balises sont `"<%"` et `"%>"`. Ce qui nous donne l'expression régulière suivante pour l'unité lexicale correspondant à du script.

```
"<%" (~["%"])* "%" (~[">"] (~["%"])* "%")* ">"
```

Elle exprime que le script JSP commence par la balise d'ouverture `"<%"`, se poursuit par un nombre quelconque de caractères différents de `"%"`, ensuite, lorsqu'un caractère `"%"` est rencontré, celui-ci peut être suivi de `">"` ou d'un nombre quelconque de caractères différents de `"%"`, et ainsi

de suite jusqu'à rencontrer un caractère "%" suivit de ">", formant ainsi la balise de fermeture "%>".

Les unités lexicales du script seront ignorées, il ne reste alors qu'un langage source constitué d'une suite de n'importe quels caractères. L'unité lexicale correspondant à un caractère du langage source, baptisée ELEMENT, sera défini par l'expression régulière suivante.

```
["\u0000"-"\uffff"]
```

Et la syntaxe sera spécifiée par une grammaire dont l'unique règle de production sera de la forme suivante : `InputText := (Element)*`.

Nous allons voir comment ceci se traduit dans le format de la grammaire JavaCC.

L'unité lexicale de script, de type SKIP, se définit de la manière suivante.

```
SKIP :
{
 <"<%" (~["%"])* "%" (~[">"] (~["%"])* "%")* ">">
}
```

Pour l'unité lexicale ELEMENT, de type TOKEN, nous avons la définition suivante.

```
TOKEN :
{
  <ELEMENT : ["\u0000"-"\uffff"]>
}
```

## Les règles de production et les actions

Les règles de production sont définies de plusieurs manières dans JavaCC (cf. [11]), mais pour ce qui nous intéresse, nous allons utiliser la production BNF, dite non-contextuelle, dont le format est le suivant.

```
Java_return_type java_identifier "(" java_parameter_list ")" ":"
Java_block
"{" expansion_BNF "}"
```

Chacune de ces productions a un membre de gauche (par convention : avant ":") qui est un non-terminal, et un membre de droite (après ":") qui définit ce non-terminal en termes d'expansions BNF.

Ce non-terminal est écrit exactement comme une déclaration de méthode Java (elle sera traduite en fonction Java par JavaCC). Les non-terminaux des expansions BNF seront aussi écrits à la manière d'un appel de méthode Java. Passer des valeurs vers le haut ou vers le bas de l'arbre syntaxique se fait donc suivant le même paradigme que les appels et les retours de fonctions en Java.

Il y a deux parties au membre de droite, java_block et "{" expansion_BNF "}". La première est un ensemble arbitraire de déclarations et de code Java. Ce code va être placé au début de la méthode générée par JavaCC pour le non-terminal.

Les expansions BNF forment la deuxième partie. A nouveau, leur définition est assez complexe et le lecteur la trouvera dans [11].

Elles peuvent, entre autres, être constituées de block de code Java, appelés actions, et d'expressions régulières. Celles-ci étant elles-mêmes constituées à partir de non-terminaux BNF. Lorsqu'une de ces expressions régulières est vérifiée, un objet java de type Token est créé et il peut dès lors être accédé en l'assignant à une variable.

Nous allons donc écrire un non-terminal `Input` correspondant au texte de départ, et le définir à l'aide d'une expression régulière.

```
void Input(PrintWriter pw) :
{}
{
    (Element(pw))* <EOF>
}
```

La partie de gauche de la règle de production est écrite comme une fonction java qui prend un `PrintWriter` en entrée. La partie déclaration de l'expansion est vide et vient ensuite une expression régulière qui définit le non-terminal `Input` comme un nombre quelconque de non-terminaux `Element` suivi de `<EOF>` (unité lexicale prédéfinie dans JavaCC correspondant à la fin du texte source). Il est à noter que le `PrintWriter` est passé en paramètre au non-terminal `Element`.

```
void Element(PrintWriter pw) :
{
  Token t;
}
{
    t = <ELETEMENT>
      {
        pw.print(t.image);
      }
}
```

La première partie de l'expansion du non-terminal `Element` contient cette fois la déclaration de la variable `t` de type `Token`. La deuxième partie est constituée d'une expression régulière vérifiée par une unité lexicale `ELEMENT`, suivie d'une action de synthèse sous forme d'un bloc de code Java. Lorsque l'expression régulière est vérifiée, le `Token` correspondant est assigné à la variable `t` déclarée plus haut. Nous nous servons ensuite de cette variable pour écrire l'unité lexicale dans le `PrintWriter` passé en paramètre. Nous verrons plus tard que nous écrirons dans le fichier résultat via ce `PrintWriter`.

La grammaire JavaCC complète se trouve en annexe.

JavaCC génère les fichiers Java TokenMgrError.java, ParseException.java, JspParserTokenManager.java, Token.java, JavaCharStream.java, JspParserConstants.java, JspParser.java, JavaCharStream.java. Le fonctionnement de ces classes est expliqué dans [12].

## 3.3 Traduire vers XML avec JTidy

JTidy est le port Java de HTML Tidy, un vérificateur de syntaxe HTML. JTidy peut être utilisé pour nettoyer un code HTML mal formé, et le traduire vers du XHTML ou du XML.

JTidy fournit également une interface DOM (Document Object Model) au document traité. DOM est une représentation sous forme d'arbre des objets d'une page web (textes, paragraphes, images…), qui va nous être utile si nous voulons accéder uniquement au contenu textuel d'une page web, sans l'HTML.

La documentation complète est disponible à [13,14].

## 3.4 Première solution : ExtractJSP.java

Une première approche de la solution à notre problème est d'écrire un programme java qui va parcourir récursivement tous les fichiers et répertoires d'un répertoire donné en paramètre, pour retrouver toutes les pages dynamiques brutes et les traduire vers du HTML, du HTML corrigé, du XHTML, du XML ou du texte sans HTML. Nous aurons alors à notre disposition le contenu statique des pages JSP, et nous pourrons utiliser le log du serveur web pour déterminer la fréquence de visite de ce contenu.

Cette solution présente de nombreux problèmes (voir *3.5 Limitations*). Une solution plus pertinente sera exposée par la suite. Toutefois, le travail déjà fourni dans cette direction sera présenté.

Ce travail prend la forme de la classe ExtractJSP.java qui, en utilisant le compilateur généré par JavaCC et JTidy, va prendre une page brute en entrée et va créer 4 fichiers : HTML, XHTML, XML et texte sans HTML.

Le code de cette classe Java est relativement simple (voir en annexe). Il y a une fonction `main(String[] args)`, qui prend en paramètres deux arguments : le fichier source (page brute JSP), le fichier résultat, exempt de script et d'HTML.

C'est la méthode `outText(String inFileName, String outFileName)`qui va faire presque tout le travail, elle va créer une nouvelle instance de notre compilateur `JSPParser`, elle va ensuite créer un objet `PrintWriter` à partir du fichier arbitraire *temp.html* et se servir de la méthode `Input(PrintWriter pw)` de `JSPParser` (cf. 3.2 Grammaire JavaCC) pour lancer la compilation du fichier source. Ensuite, elle va créer une nouvelle instance de la classe `Tidy`, et s'en servir, via la méthode `parse`, pour créer les deux fichiers *temp.xhtml* et *temp.xml*. Finalement, elle utilise la

méthode récursive `print(Node node)` et l'interface arborescente DOM fournie par l'objet `Tidy` pour écrire les nœuds de l'arbre de type TEXT (qui correspondent à du texte) dans le fichier résultat.

## 3.5 Limitations

Les limitations de ce modèle sont apparues assez rapidement :

- Il faut écrire un compilateur pour chaque technologie de pages web dynamiques (ASP, JSP, ColdFucion, PHP …).
- Il est impossible de récupérer le contenu sémantique généré par un programme CGI ou par une Servlet.
- La partie du contenu sémantique qui est extrait dynamiquement de bases de données est inaccessible lors du déformatage.
- L'explosion du XML et de ses potentialités de mise en page via XSLT et CSS font évoluer les techniques de programmation vers des pages dynamiques ne contenant plus que du code qui génère les pages à partir de contenus provenant de fichiers XML et de bases de données.
- Les statistiques sur la visualisation de contenu sont peu fiables. En effet, la génération du contenu étant traitée en temps réel, il est impossible de savoir quelles parties du contenu extrait ont été vues, ni quand, ni par qui.

# 4 Le futur des pages web dynamiques

Nous allons brièvement introduire les nouvelles technologies montantes de l'Internet, principalement basées sur XML, afin de mieux comprendre comment la génération de pages web dynamiques évolue vers le regroupement, la transformation et la mise en page de modules d'informations représentés par des fichiers XML provenant de n'importe où sur le web.

## 4.1 XML et technologies dérivées

Depuis sa sortie, en 1997, XML (eXtensible Markup Language) a été un énorme succès qui s'explique principalement par le fait qu'il permet de définir des langages à la fois faciles à traiter par un programme et faciles à lire par un humain.

C'est un sujet très large qui inclut un grand nombre de technologies. Plus d'informations peuvent être trouvées dans [1,4,7,8,15,16,17].

XML est un outil de définition de langages, où un langage est un set de tags et d'attributs ayant diverses contraintes. XML peut être défini comme un ensemble de meta-contraintes et d'un meta-langage qui servent à définir des langages balisés (markup languages).

A partir de là, il faut bien définir comment ces tags vont être affichés, et donc avoir un système de feuilles de style (stylesheet). Il faut aussi un mécanisme pour éviter les conflits entre des tags appartenant à deux langages différents mais ayant le même nom.

### 4.1.1 XML et documents XML

La notion centrale d'XML est le document XML. Il est définit dans les spécifications [18] comme étant une unité d'information pouvant être vue de deux façons : comme une séquence linéaire de caractères qui contient des données et des balises ou comme une stucture abstraite de données sous forme d'arbre de nœuds.

Un parseur est nécessaire pour passer d'une vue linéaire à une vue structurée (voir *4.1.2 Parseurs XML*).

Les documents XML sont auto-décris, et cela de deux façons :
- chaque document est balisé de manière à montrer sa structure arborescente, et
- un document peut contenir sa grammaire (DTD, voir *4.1.3 DTD et XML Schema*) ou une référence à celle-ci.

### 4.1.2 Parseurs XML

Le but de parser un document XML est de fournir une interface qui permette à une application d'utiliser ce document. Le parseur se trouve donc au milieu entre un document XML et une application qui l'utilise.

Il y deux spécifications d'interface standards :
- DOM : Document Object Model, où le document est complètement transcrit en arbre de nœud avant de pouvoir être utilisé, et
- SAX : Simple API for XML, où à chaque tag est associé un événement, auprès duquel le programme peut s'inscrire pour le traiter.

### 4.1.3 DTD et XML Schema

DTD (Document Type Definition) est le mécanisme utilisé pour définir les règles syntaxiques du langage que l'on définit. Les DTD sont en fait des grammaires non-contextuelles, mais dont la syntaxe est particulière.

Ces DTD sont écrits dans un langage spécifique, qui ne se conforme pas lui-même aux spécifications d'XML. Celui-ci est donc en passe d'être remplacé par XML Schema, qui a les avantages suivants.
- Il utilise la syntaxe XML.
- Il a un système de types de données plus détaillé (dans les DTD, tout est du texte).
- Les utilisateurs peuvent définir leurs propres types de données, structurées ou non.
- Il est plus modulaire et plus facile à utiliser.

Ce qui est important, c'est que dès à présent, nous sommes capables de définir un langage dans son entièreté : vocabulaire, syntaxe et types de donnée.

### 4.1.4 Style XML

Il y a deux langages de feuille de style qui fonctionnent avec XML : CSS (Cascading StyleSheet) et XSL (eXtensible Style Language).

XSL est séparé en deux parties, une partie concerne le formatage et l'autre les transformations avec XSLT.

Pratiquement, on utilise XSLT pour transformer un document XML en HTML et CSS pour définir la manière dont cet HTML va s'afficher.

### 4.1.5 Namespace

XML est conçu pour être modulaire et extensible, et donc il permet de réutiliser des modules d'autres DTD. Mais cela implique des problèmes de conflit entre les noms des tags.

La fonction d'XML namespace est d'empêcher ces conflits par l'ajout d'un préfixe, identifié par une URI (Universal Ressource Identifier), à tous les tags. L'URI étant le moyen d'identifier de manière univoque une ressource sur le web, elle permet, dans ce cas-ci, d'identifier de manière univoque le préfixe utilisé dans chacun des tags.

### 4.1.6 Conclusions

En conclusion, on peut voir dans XML un package complet de technologies qui permet de définir des langages de manière complètement modulaire. Cela permet à des systèmes hétérogènes de partager des informations, de les combiner, de les transformer, de les traiter et de les afficher de manière efficiente dans un navigateur.

## 4.2 Services web

Les services web sont une application des technologies XML, dont le but est de partager non seulement l'information, mais aussi les ressources.

Un service web est un ensemble de fonctions groupées en une seule entité et publiées sur le réseau pour être utilisées par d'autres applications. Les services web sont les composants de base de la création de systèmes distribués ouverts, permettant à des compagnies ou à des individus de créer des applications web réellement distribuées.

L'idée est d'utiliser un mécanisme de communication via des messages XML sur un protocole standard du web comme HTTP.

## 4.3 Nouvelles techniques de programmation

Nous voyons dès à présent comment une application va pouvoir, dans un futur de plus en plus proche, recevoir une requête d'un utilisateur, appeler des fonctions distribuées via des services web, collecter les informations (comme par exemple la météo, la disponibilité de certains articles, les résultats des recherches sur plusieurs bases de données différentes, etc.), les transformer, et les afficher dans un style choisi. Les informations ne font plus que transiter par le serveur web avant d'être transférées vers le client.

Nous arrivons à des implémentations d'applications web où le code est complètement séparé des données, elles-même complètement distinctes du style de l'affichage.

## 4.4 Nouveaux défis

La solution de déformatage des pages dynamiques brutes n'a plus de sens dans ce nouveau contexte, puisqu'elle était basée sur le principe selon lequel les applications web mixent le code (écrit en script) avec du contenu statique (le texte de la page) et avec la mise en forme (tags HTML).

Il s'impose de trouver une nouvelle solution plus indépendante de l'évolution des techniques de programmation web.

# 5 Stockage de l'information en transit

## 5.1 Introduction

La solution aux nombreux problèmes posés par ces nouvelles techniques de génération de page a été de réorienter le travail vers une architecture où les pages dynamiques sont analysées après avoir été générées et non plus avant. Le défi est dès à présent de réussir à stocker les pages HTML générées à partir de pages dynamiques au moment où le serveur web les envoie au browser.

## 5.2 Approfondissement de l'architecture du web

Il est indispensable dès à présent d'analyser plus profondément l'architecture du système client/serveur d'Internet.

### 5.2.1 Serveur web modulaire

La plupart des serveurs web sont structurellement modulaires, ce qui signifie que leurs diverses fonctionnalités sont assurées par plusieurs modules presque indépendants.

L'architecture d'Apache, par exemple, est constituée d'une série de modules noyaux auxquels il est possible de rajouter un nombre arbitraire de modules supplémentaires. Les modules de bases assurent les fonctionnalités indispensables d'un serveur web, à savoir répondre aux requêtes HTTP en fournissant les documents demandés.

Ce genre de système permet de construire un serveur web dont les fonctionnalités sont à la carte. Si nous avons besoin d'authentification, d'hôtes virtuels, d'alias ou autre, il suffit de rajouter le module correspondant. Il en va de même pour les pages dynamiques. Le traitement des JSP, par exemple, est assuré par *mod_jk* (http://jakarta.apache.org/tomcat), un module développé dans le cadre du projet Apache (http://www.apache.org).

Ces modules ont accès à l'API du serveur web pour pouvoir exécuter leur tâche et ensuite envoyer le résultat au browser.

Lorsqu'une requête est faite au server web, un module noyau se charge de la traduire en structure interne accessible depuis les API, ensuite un autre module analyse certains paramètres et détermine quel autre module va traiter la requête. Le module choisi traite la requête et envoie le résultat au client en utilisant lui aussi les API du serveur web.

### 5.2.2 Le réseau TCP-IP

Le modèle OSI (Open System Interconnection) définit par l'ISO (International Standard Organization) possède 7 couches, comme illustré sur la figure suivante ([19]).

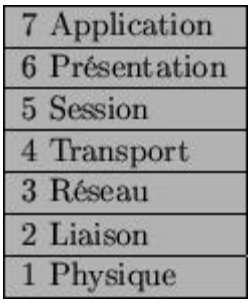


*Fig. 5.1: Les 7 couches du modèle OSI.*

Chacune de ces couches correspond à une fonctionnalité particulière du réseau. Chaque couche est constituée d'éléments matériels et/ou logiciels et offre un service à la couche située immédiatement au-dessus d'elle en lui épargnant les détails d'implémentation nécessaires.

Dans le cas de l'Internet, ce modèle est implémenté sous forme du modèle TCP/IP, comme illustré à la figure suivante ([19]).

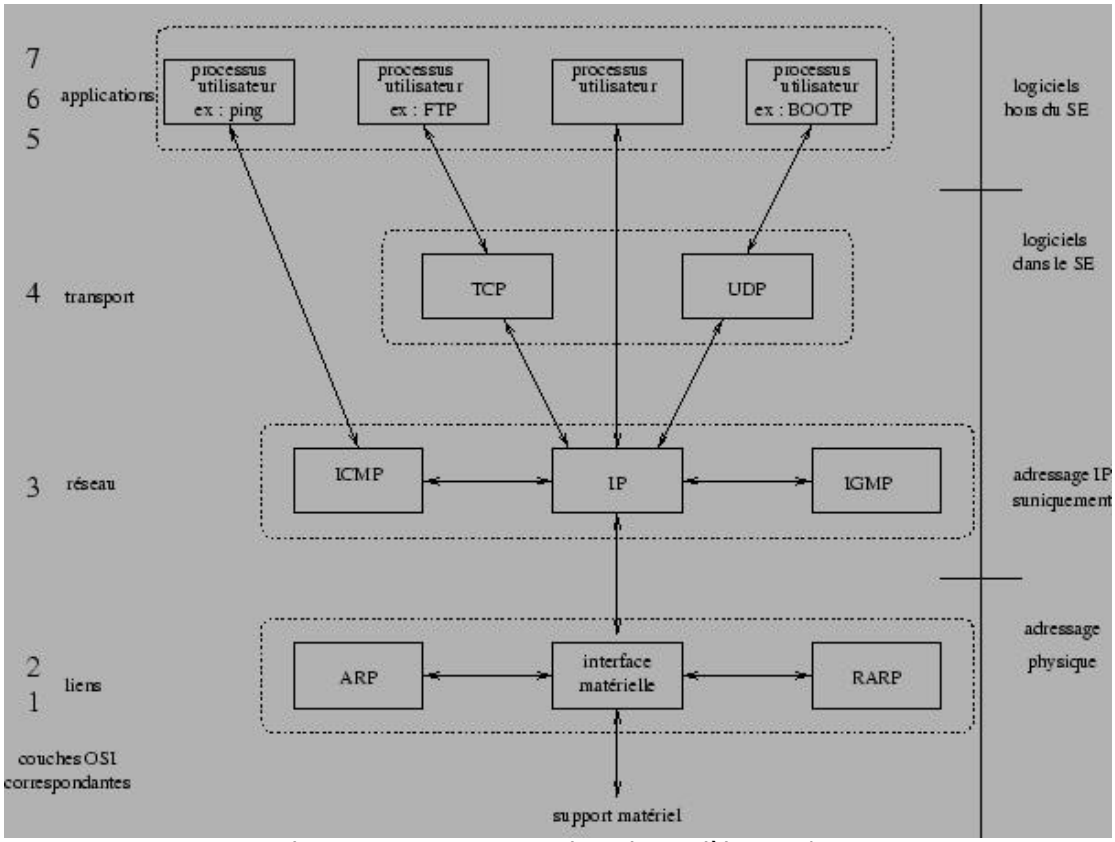


*Fig. 5.2: Les quatre couches du modèle TCP/IP.*

Les logiciels TCP/IP sont structurés en quatre couches de protocoles qui s'appuient sur une couche matérielle comme illustré dans la figure précédente.

- La couche de liens est l'interface avec le réseau et est constituée d'un driver du système d'exploitation et d'une carte d'interface de l'ordinateur avec le réseau.
- La couche réseau ou couche IP (Internet Protocol) gère la circulation des paquets à travers le réseau en assurant leur routage.
- La couche transport assure tout d'abord une communication de bout en bout en faisant abstraction des machines intermédiaires entre l'émetteur et le destinataire. Elle s'occupe de réguler le flux de données et assure un transport fiable (données transmises sans erreur et reçues dans l'ordre de leur émission) dans le cas de TCP (Transmission Control Protocol) ou non fiable dans le cas de UDP (User Datagram Protocol). Pour UDP, il n'est pas garanti qu'un paquet (appelé dans ce cas datagramme) arrive à bon port, c'est à la couche application de s'en assurer.
- La couche application est celle des programmes utilisateurs comme le serveur web et HTTP, FTP (File Transfert Protocol), etc.

## 5.3 Nouvelles solutions

Deux solutions pour obtenir le contenu au moment où il est envoyé par le serveur apparaissent lors de cette analyse. La première est d'écouter la communication au niveau de la couche réseau. La deuxième est de rajouter une fonctionnalité au server web sous la forme d'un module d'extension.

Ce sont ces deux solutions qui sont exposées et comparées ci-dessous. Nous verrons ensuite pourquoi la solution du module est préférable et comment elle répond parfaitement à toutes les exigences de l'application WASA.

### 5.3.1 Ecoute du réseau

Pour déterminer le contenu de la communication entre le serveur web et le navigateur, nous pourrions écrire un « renifleur de paquets » (packet sniffer). C'est un programme qui intercepte les paquets TCP/IP au niveau de la couche réseau.

Il existe des APIs de bas niveau qui permettent d'accéder aux basses couches du modèle OSI, ainsi que de nombreux programmes qui permettent déjà de stocker et d'analyser tous les paquets passant sur un réseau.

L'idée est d'écrire un renifleur de paquets particulier qui va stocker les paquets et réaliser le travail des couches réseaux supérieures pour obtenir en résultat la page HTML qui a été générée par le serveur.

Les étapes que le programme devra réaliser sont les suivantes.

- Stocker les paquets de manière efficiente et rapide.
- Trier les paquets stockés pour obtenir des groupes de paquets correspondant chacun à une conversation entre le serveur et le navigateur, c'est-à-dire une requête HTTP suivie d'une réponse HTTP. Cela peut se faire en se basant sur un groupe de paramètres qui identifient une conversation de manière univoque. Deux de ces paramètres sont spécifiés dans le protocole TCP et sont le port du serveur et le port du client. Le paramètre supplémentaire est l'adresse IP du client, spécifiée dans le protocole IP.
- Reconstituer le flot de données en ordonnant ces paquets grâce à trois autres paramètres spécifiés dans le protocole TCP : le numéro du paquet, le numéro du prochain paquet et le nombre dans l'accusé de réception.
- Finalement, il faut décortiquer le protocole HTTP pour séparer le document des messages et en-têtes HTTP.

### 5.3.2 Module d'extension du serveur web

Comme décrit dans *5.2.1 Serveur web modulaire*, il est possible d'ajouter une fonctionnalité au serveur web en lui ajoutant un module. Les pages dynamiques, par exemple, sont générée par des modules.

Il est possible d'écrire un module qui s'inscrirait d'une manière ou d'une autre à l'événement de l'envoi des données du serveur au client, et s'immiscerait dans la communication pour rajouter une étape de stockage de la page web en transit.

## 5.4 Comparaison des solutions

La solution de l'écoute du réseau est plus « écologique », elle respecte l'environnement dans lequel elle est introduite. Elle est indépendante du type de serveur web. En contrepartie, elle demande beaucoup de ressources machine par rapport à la solution du module. En effet, celle-ci ne fait que rajouter une étape de stockage alors que le renifleur de paquets doit stocker et puis traiter les paquets à l'aide de tris et traitements de chaînes de caractères. Or ces deux types d'opérations sont très coûteuses en terme de ressources.

La solution du module est dépendante du serveur web, il faut écrire un module par serveur sur le marché. Toutefois, il n'y a que deux serveurs web vraiment répandus (voir figure suivante disponible à l'adresse *http://www.netcraft.com/survey*):

- Apache : 54% des serveurs de l'Internet, et
- Microsoft IIS : 34%.

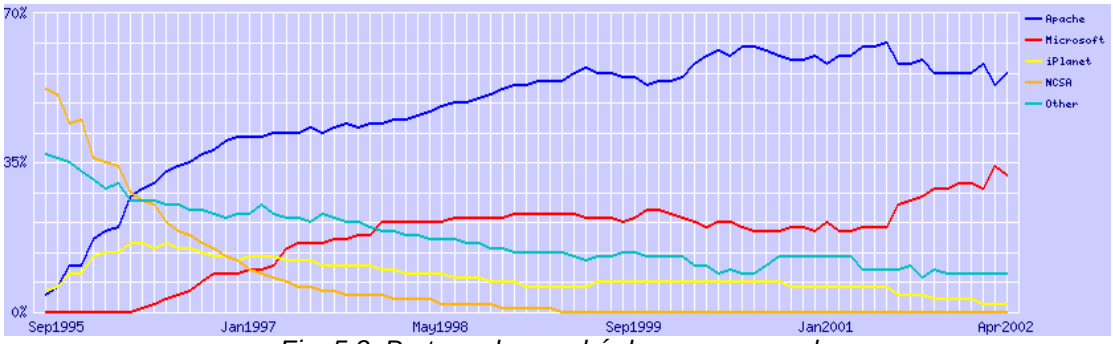

*Fig. 5.3: Partage du marché des serveurs web.*

Un autre désavantage de ce module est qu'il peut rendre le serveur instable si l'implémentation n'est pas suffisamment rigoureuse. Ce module sera exécuté dans le même processus que le serveur web, s'il bloque, le serveur peut bloquer aussi.

Par contre, un module peut accéder aux en-têtes HTTP du document, cela permet d'ignorer directement les documents qui ne sont pas intéressants du point de vue sémentique, comme par exemple les images. Dans le cas du renifleur de paquets, nous sommes obligés de traiter toutes les communications avant de se rendre compte du type des fichiers. Or il y a de nombreuses images qui transitent par un serveur web, il en résulte une consommation de ressources excessive de la part du renifleur de paquets.

Le module permet aussi d'intervenir sur le document avant qu'il ne soit envoyé au client, ce qui rend possible l'optimisation de l'utilisation de la bande passante en compressant les documents pour les navigateurs qui le supportent.

## 5.5 Conclusions

Il apparaît clairement que la solution du module d'extension du serveur web est la meilleure des deux solutions envisagées, ne fût-ce que parce qu'elle consomme moins de ressources machine.

Cette nouvelle solution est indépendante des problèmes rencontrés par la technique de déformatage des pages dynamiques brutes, à savoir qu'elle ne dépend pas de la technologie utilisée pour générer la page, ni de la provenance des données.

De plus elle est stable dans le temps, car véritablement indépendante de l'évolution des techniques de programmation des applications web.

Le module d'extension du serveur web se situe suffisamment bas dans les couches de l'application web pour accéder aux pages déjà générées, mais suffisamment haut dans les couches du modèle réseau pour ne pas avoir à recomposer les données à partir des paquets TCP/IP.

# 6 Architecture Apache et API

« Le serveur web Apache est un projet visant à développer et maintenir un serveur HTTP open-source pour diverses plateformes modernes, comme UNIX et Windows NT/2000. Le but de ce projet est de fournir un serveur sécurisé, performant et extensible qui fournit des services HTTP en totale compatibilité avec les standards HTTP actuels. »[25]

Il existe peu de documentation sur le fonctionnement interne d'Apache et sur le développement des modules, surtout pour les dernières versions. Comme décrit dans la documentation d'Apache ([20]), « Welcome to the bleeding edge ».

Nous allons donc, dans un premier temps, synthétiser les informations obtenues des sources suivantes :
- la mailing list des développeurs de modules Apache (voir *http://httpd.apache.org/lists.html*),
- la documentation dans [21,22] et sur le web dans [20,23,24].
- le code source entièrement écrit en C d'Apache (disponible à *http://www.apache.org/dist/httpd*),
- mon expérience d'utilisation d'Apache, et finalement,
- mon expérience de développement de *mod_trace_output*.

## 6.1 Architecture Apache

Dans ce chapitre, nous allons présenter l'architecture d'Apache. Nous décrirons aussi les étapes du traitement de chaque requête et la manière dont les modules s'immiscent dans le traitement de la requête.

Cette introduction théorique va servir à comprendre le fonctionnement de *mod_trace_output*, le module que nous allons implémenter.

### 6.1.1 Fonctionnement d'Apache

Une grande partie d'Apache est évidemment dirigée par le fait que c'est un server HTTP qui tourne en tâche de fond. Il lit ses fichiers de configuration, se met en veille et se réveille lors d'une connexion TCP/IP avec un navigateur, il reconnaît les URIs (Universal Ressource Identifier) et les traduit en nom de fichiers ou de scripts et retourne une réponse au navigateur. Les modules jouent un rôle actif dans chacune de ces étapes.

Apache multiplexe ses opérations de manière à pouvoir traiter une nouvelle requête avant d'avoir fini de traiter la précédente. Sur les systèmes UNIX (ceux qui nous intéressent dans le cas présent), Apache utilise un modèle mulitprocessus dans lequel il lance un groupe de serveurs : un parent unique, qui est responsable de la supervision, et un ou plusieurs enfants, qui sont responsables du traitement des requêtes.

### 6.1.2 Protocole HTTP

Ce protocole a déjà été introduit dans *2.1.2 HTTP*. Ce chapitre va présenter les éléments principaux d'une requête HTTP et ceux d'une réponse. Ils seront utilisés par la suite dans le module *mod_trace_output*.

Voici un exemple typique d'une requête.

```
GET /index.html HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET)
Host: trace-output.sourceforge.net
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Accept-Language: fr-be
Accept-Charset: iso-8859-1,*,utf-8
Accept-Encoding: gzip, deflate
```

La première ligne contient trois composants : la méthode (GET, POST, HEAD, PUT ou DELETE), l'URI et la version du protocole.

Les champs (headers ou en-têtes) sont les suivants.

- **Connection** est une suggestion au serveur web qu'il devrait garder la connexion TCP/IP ouverte après avoir fini cette requête. C'est une optimisation qui améliore les performances sur les pages qui contiennent des images, celles-ci vont être appelées via des requêtes supplémentaires.
- **User-Agent** donne des informations sur le modèle de navigateur.
- **Host** est le nom donné dans l'URI et est utilisé par le système d'hôte virtuel (virtual host) pour sélectionner le bon arbre de documents et le bon fichier de configuration.
- **Accept** est une liste de types MIME (Multipurpose Internet Mime Extension) que le navigateur accepte.
- **Accept-Language** indique le langage de préférence de l'utilisateur.
- **Accept-Charset** indique quels sets de caractères le navigateur est capable d'afficher.
- **Accept-Encoding** est une liste de format de compression que le navigateur supporte.

Et voici un exemple typique de réponse.

```
HTTP/1.1 200 OK
Date: Mon, 06 May 2002 18:10:47 GMT
Server: Apache/1.3.20 (Unix) PHP/4.0.6
Last-Modified: Fri, 03 May 2002 23:23:05 GMT
ETag: "2b87b9-420c-3cd31bd9"
Accept-Ranges: bytes
Content-Length: 16908
Connection: Close
Content-Type: text/html
Content-Encoding: gzip

The document content goes here…
```

La réponse est similaire à la requête, la première ligne est la ligne de statut contenant trois composants : la version du protocole, le code de la réponse et une version lisible humainement du code de réponse.

Viennent ensuite les champs optionnels dont les plus importants sont les suivants.

- **Content-Length** donne la longueur du document en bytes.
- **Content-Type** donne le type MIME du document renvoyé suivit du set de caractères utilisé.
- **Content-Encoding** donne le type de compression du document renvoyé.

Ces champs sont ensuite suivis du document lui-même.

### 6.1.3 Cycle de vie d'Apache

Le cycle de vie d'Apache est illustré à la figure suivante (extraite de [21], page 57).

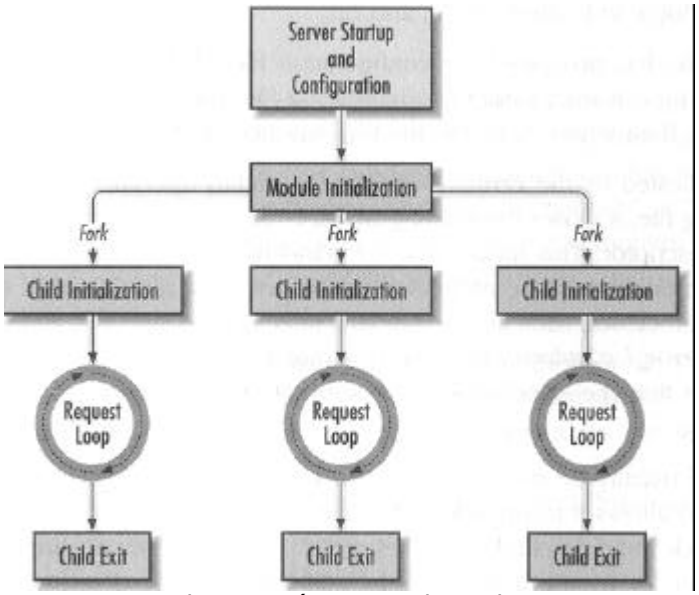


*Fig. 6.1: Démarrage d'Apache.*

Il démarre, s'initialise, crée plusieurs copies de lui-même et entre dans la boucle de traitement des requêtes. Ensuite, chacune des copies sort de cette boucle et se termine.

Les étapes les plus intéressantes du point de vue des modules sont celles qui interviennent dans la boucle de traitement des requêtes, mais ils peuvent également accéder aux autres phases.

Ils le font en inscrivant des routines de code, appelées « handlers » ou gestionnaires, que le noyau d'Apache appelle aux moments appropriés. Une phase peut avoir soit plusieurs gestionnaires qui s'y sont inscrits, soit un seul, soit aucun. Si plusieurs modules ont signifié leur volonté de gérer la même phase, Apache va les appeler dans l'ordre dans lesquels ils ont été chargés, soit statiquement à la compilation, soit dynamiquement en

utilisant la directive de configuration *LoadModule*. Si aucun module ne s'est inscrit pour une phase, elle va être gérée par une routine par défaut du module noyau d'Apache (module *http_core*).

### 6.1.4 Démarrage et la configuration

Quand le serveur démarre, Apache initialise les ressources globales et internes, et examine ses arguments de ligne de commande. Ensuite, il localise et examine ses différents fichiers de configuration.

Ces fichiers de configuration peuvent contenir des directives implémentées par des modules internes ou externes. Apache analyse chaque directive en se basant sur un prototype fourni dans la table des commandes qui fait partie de chaque module et passe les paramètres analysés aux routines de configuration du module. Apache traite ces directives dans l'ordre et ce qui peut avoir un impact important.

Le processus de configuration des modules est assez complexe. Il fait intervenir des niveaux ou contextes de directives de configuration, incluant des directives globales, des directives spécifiques à certains hôtes virtuels et des directives qui ne s'appliquent qu'à certains répertoires ou à certaines parties d'URI.

Ces contextes sont définis dans les fichiers de configuration sous forme de sections représentées par des tags. Voici un exemple pour une section correspondant à un répertoire.

```
<Directory>

#Directives spécifiques à ce répertoire
…
</Directory>
```

Il existe quatre sections :
- <VirtualHost>, pour les hôtes virtuels,
- <Directory>, pour les répertoires,
- <Location>, pour les chemins ou parties d'URI, et
- <File>, pour les fichiers.

La possibilité d'imbrication et de combinaison de ces sections permet une configuration très souple.

Une fois qu'Apache a analysé ces fichiers de configuration, il sait où sont situés les fichiers journal, il ouvre donc chacun des fichiers journaux configurés, comme par exemple *ErrorLog* et *CustomLog*.

Apache crée ensuite un fichier contenant son *PID* (Process IDentifier) conformément à la directive *PidFile*.

Il est à noter qu'Apache ferme ensuite le fichier correspondant à la sortie d'erreur standard *stderr* et le réouvre vers le fichier spécifié dans la directive *ErrorLog*. Ceci est valable pour tous les modules chargés.

### 6.1.5 Initialisation des modules

Apache initialise ensuite les modules. Chacun des modules possède une routine d'initialisation à laquelle Apache fournit une structure appelée `server_rec` et un pointeur vers un groupe de ressources (ressource pool). Cette structure contient les informations précédemment configurées, comme le nom du serveur, le port que le serveur écoute, etc. Les groupes de ressources sont utilisés pour éliminer les pertes de mémoire vive (memory leaks) et les ressources orphelines.

Quand le serveur est redémarré, ces procédures de configuration et d'initialisation sont appelées à nouveau. Et pour être sûr que tous les modules peuvent supporter un redémarrage, Apache exécute ces phases deux fois pendant son démarrage initial.

### 6.1.6 Initialisation des processus enfants

Sur le système d'exploitation choisi, à savoir UNIX, Apache crée des processus enfants qui s'occuperont de traiter les requêtes. Le processus parent s'installe en tâche de fond et s'occupe de surveiller l'activité des processus enfants et d'en créer de nouveaux si necessaire.

### 6.1.7 Terminaison des processus enfants

Après avoir traité un certain nombre de demandes, chaque processus enfants peut éventuellement se terminer, soit de « mort naturelle », soit en ayant atteint le nombre de requêtes spécifié par la directive de configuration *MaxRequestPerChild*, ou enfin parce que le serveur a reçu un ordre de redémarrage ou de terminaison.

### 6.1.8 Boucle de traitement de requêtes

Maintenant que nous avons les connaissances de bases de l'architecture d'Apache et de son processus de démarrage, analysons la partie qui nous intéresse plus particulièrement : le traitement des requêtes.

La boucle de traitement des requêtes est illustrée à la figure suivante (extraite de [21], page 60).

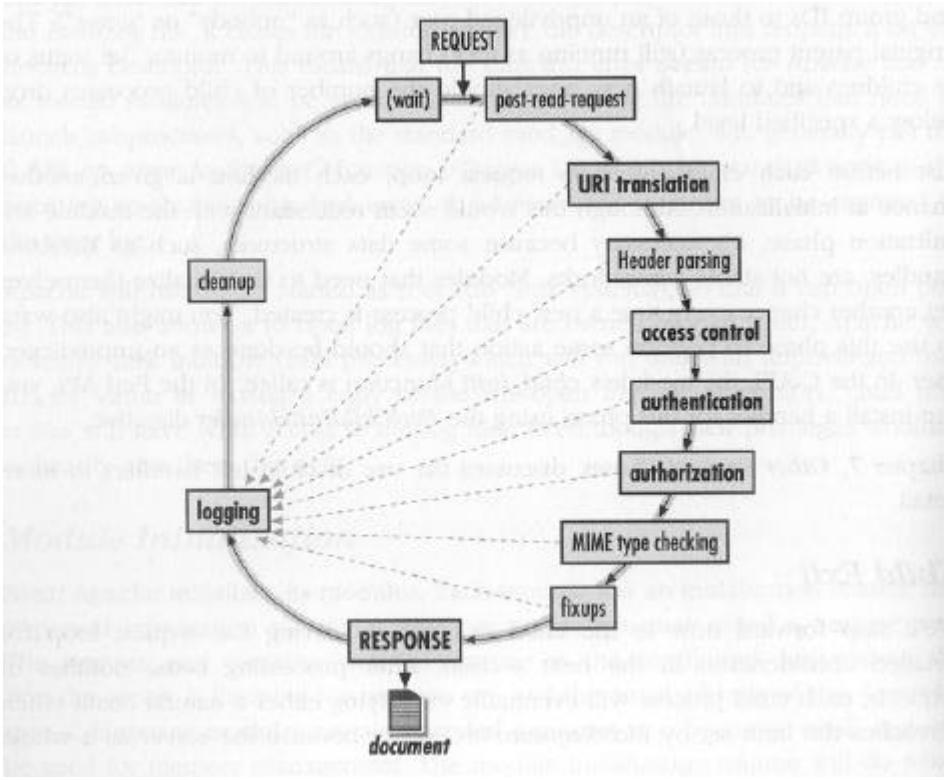


*Fig. 6.2: La boucle de traitement des requêtes.*

Le noyau d'Apache gère les aspects les plus communs d'une conversation HTTP: attente de la requête, analyse de la requête et de ses en-têtes, et composition de la réponse. A chaque boucle, il y a un certain nombre de décisions à prendre, et des modules externes peuvent définir des gestionnaires pour améliorer ou remplacer chaque décision. Si aucun gestionnaire n'est défini, Apache retourne à son comportement par défaut.

Chacune de ces décisions est représentée sous forme de phase dans la figure ci-dessus. La plupart des phases sont terminées par le premier gestionnaire qui les traite, sauf pour les phases *logging*, *fixups*, *access control* et *authentication*, où tous les gestionnaires sont toujours exécutés.

Un gestionnaire pour une phase peut typiquement faire une des trois actions suivantes :

- gérer la requête, et indiquer quand il a fini en retournant la constante OK,
- décliner la gestion de la requête en retournant la constante DECLINE,

- signaler une erreur, en retournant un code d'erreur HTTP.

S'il signale une erreur le traitement normal de la requête est terminé et seule la phase de *logging* est encore exécutée.

## URI translation

L'URI peut se référer à un fichier physique, un document généré en temps réel par un script externe ou un document généré par un module interne. Le serveur doit savoir dès le départ quel genre de document il doit gérer.

Les routines de traduction par défaut d'Apache utilisent les directives *Alias*, *ScriptAlias* et *DocumentRoot* pour traduire l'URI en chemin d'accès complet au document demandé.

Certains modules externes, comme *mod_rewrite*, permettent un contrôle plus sophistiqué de cette phase.

## Access control

Cette phase détermine si l'URI demandé par la requête est accordée en se basant sur des critères non spécifiques aux utilisateurs.

## Authentication

Cette phase permet de vérifier que l'utilisateur est bien celui qu'il prétend être.

## Authorization

Elle détermine si l'utilisateur identifié d'une telle requête est autorisé à accéder à une telle URI.

## MIME type checking

Cette phase est très importante. Apache fait une première supposition sur le type de contenu du document. Sur base de ce type de contenu, Apache va décider comment traiter le document. Il base sa décision sur le nom du document, l'extension du nom de fichier ou sur la configuration du répertoire auquel appartient le fichier.

Il existe trois types de contenu :
- un type MIME (par exemple *image/vif*),
- un codage (par exemple *httpd/unix-directory*), ou
- un libellé de gestion de contenu (par exemple *jakarta-servlet* ou *trace-output* comme nous le verrons plus tard).

Une fois que ce type est déterminé, Apache l'utilise pour sélectionner le gestionnaire de contenu (content handler) et pour générer ou transmettre le document lui-même pendant la phase de réponse.

Ce mécanisme est décrit dans *6.4 Gestionnaires de contenu et types de documents*.

### Fixups

Cette phase est utilisée pour modifier la requête en dernière limite, avant que celle-ci ne soit traitée par le gestionnaire de contenu.

*Mod_trace_output* utilise cette phase pour déterminer la suite de son action sur la requête en cours.

### Response

Une fois qu'Apache a décidé quel module va gérer le contenu, il lui passe l'URI et les informations accumulées sur le document.

Typiquement, le module modifie les champs de la réponse HTTP et dit à Apache de les envoyer au navigateur. Ensuite, il crée le contenu de la réponse et l'envoie au navigateur client.

La phase de réponse de *mod_trace_output* est sensiblement plus compliquée.

### Logging

Apache fournit un système d'enregistrement qui écrit dans des fichiers journaux. En plus des enregistrements par défaut, il est également possible de les personnaliser en fonction des besoins.

### Cleanup

La requête est terminée, mais il reste peut-être des ressources à libérer. Les modules peuvent enregistrer des gestionnaires de nettoyage pour libérer les ressources qu'ils ont utilisées, comme les connexions aux bases de données, la mémoire, etc.

## 6.1.9 Requêtes internes et sous-requêtes

Une requête interne a tout de la requête ordinaire, excepté qu'elle a été générée par Apache. Les sous-requêtes sont un cas particulier des requêtes internes. Nous allons les utiliser dans *mod_trace_output*.

## 6.2 API Apache

Ce chapitre décrit l'API (Application Programming Interface), les types de données et les structures fournis par Apache. Ne sera exposé que ce qui est nécessaire à la compréhension du module *mod_trace_output*, des renseignements supplémentaires peuvent être trouvés dans [20,21,22,23].

### 6.2.1 Structures de données

Les sections qui suivent décrivent les structures suivantes : `request_rec`, `server_rec`, `conn_rec` et les structures de configuration de module.

#### Structure de requête : request_rec

La structure `request_rec` est la structure de donnée principale par laquelle le noyau d'Apache montre les données aux différents gestionnaires intervenant dans le traitement des requêtes. Elle permet à ceux-ci d'accéder à toutes les données utiles concernant la requête HTTP.

Certains champs de cette structure sont destinés à être utilisés en interne. Seuls les champs principaux et ceux qui présentent un intérêt pour *mod_trace_output* seront décrits. La définition complète de `request_rec` est disponible dans le fichier *httpd.h* des sources d'Apache (disponibles à *http://www.apache.org/dist/httpd*).

- **ap_pool *pool** pointe sur un groupe de mémoire utilisé par les fonctions de gestion de mémoire. La mémoire du groupe est assurée de ne pas être libérée pendant la durée de vie de la requête. Voir *6.2.2 Gestion de Mémoire*.
- **conn_rec *connection** pointe sur la structure de connexion de la requête en cours.
- **server_rec *server** pointe sur la structure de serveur en cours. Voir la section suivante.
- **request_rec *next** est un pointeur vers la structure de requête d'une requête redirigée en interne.
- **request_rec *prev** est un pointeur vers la structure de requête de la requête à partir de laquelle elle a été redirigée en interne.
- **request_rec *main** est un pointeur vers la structure de requête de la requête supérieure si la requête a été redirigée en interne.
- **int header_only**, si différent de zéro, indique que la méthode de requête était *HEAD*.
- **time_t request_time** indique l'heure de démarrage de la requête comme une structure C `time_t`.
- **table *headers_in** pointe sur une table de noms/valeurs des en-têtes HTTP de la requête.
- **table *headers_out** pointe sur une table de noms/valeurs des en-têtes HTTP de la réponse.
- **table *err_headers_out** pointe sur une table de noms/valeurs des en-têtes HTTP de la réponse en cas d'erreur.

- **table *notes** pointe sur une table noms/valeurs. Il n'y a pas d'utilisation prédéfinie de ce champ. Les gestionnaires peuvent l'utiliser à leur convenance.
- **const char *content_type** contient le type de contenu MIME de la réponse.
- **const char *handler** contient le libellé de gestion du contenu Apache (par exemple *trace-output*).
- **char *unparsed_uri** contient l'URI brute de la requête, avant traduction.
- **char *uri** contient la partie chemin de l'URI.
- **char *filename** contient l'URI traduite de la requête.
- **void *per_dir_config** pointe sur la structure de configuration de répertoire.

## Structure de serveur : server_rec

La structure `server_rec` contient des informations sur le serveur en cours, en particulier concernant l'hôte virtuel actuel. La structure `server_rec` est accessible par l'intermédiaire du champ `server` de la structure `request_rec` ou en la passant directement au gestionnaire dans les arguments.

Comme `request_rec`, certains champs sont destinés à être utilisés en interne. Seuls les principaux et ceux qui nous seront utiles dans *mod_trace_output* seront présentés.

- **char *server_hostname** contient le nom du serveur, virtuel ou non.
- **unsigned short port** contient le port TCP/IP que le serveur écoute.
- **FILE error_log** contient un descripteur de fichier ouvert sur le fichier journal d'erreur.
- **int is_virtual** indique un nombre différent de zéro si le serveur est virtuel.
- **void *module_config** pointe sur la structure de configuration des modules.

## Structure de connexion : conn_rec

La structure `conn_rec` contient les informations spécifiques à la connexion client/serveur en cours. Comme les autres structures, certains champs sont destinés à être utilisés en interne. Seuls les principaux et ceux qui nous seront utiles dans *mod_trace_output* seront présentés.

- **ap_pool *pool** pointe sur un groupe de mémoire, utilisé par les fonctions de gestion de mémoire. Voir *6.2.2 Gestion de Mémoire*, pour plus d'informations.
- **server_rec *server** pointe sur le serveur actuellement en service.
- **BUFF *client** pointe sur le tampon de connexion au client.
- **struct sockaddr_in local_addr** indique la socket TCP/IP de l'adresse locale.
- **struct sockaddr_in remote_addr** indique la socket TCP/IP de l'adresse distante.

### Structure de tampon client : BUFF

La structure `BUFF` contient les informations spécifiques du tampon client de la connexion. On y accède généralement via le champ `client` de la structure `conn_rec`. Comme les autres structures, certains champs sont destinés à être utilisés en interne. Seuls les principaux et ceux qui nous seront utiles dans *mod_trace_output* seront présentés.

- **int flags** est utilisé en interne par Apache.
- **int fd** est le descripteur de fichier du tampon client.
- **void *callback_data** est un pointeur générique. Il n'y a pas d'utilisation prédéfinie de ce champ. Les gestionnaires peuvent l'utiliser à leur convenance.
- **void (*filter_callback)(BUFF *, const void *, int )** est un pointeur vers une fonction qui va recevoir une copie des données envoyées au tampon client. Le deuxième paramètre est un pointeur vers la copie des données et le troisième est la longueur de ces données.

Les deux derniers champs sont une nouveauté d'Apache 1.3.23. Nous allons les utiliser dans *mod_trace_output* (voir *7.4.2 Boucle de traitement de requêtes*) et c'est pourquoi il ne fonctionne qu'avec Apache 1.3.23 et Apache 1.3.24 qui sont les dernières versions d'Apache avant la version 2.0.

### Structure de configuration de module

La structure de données de configuration n'est pas une structure prédéfinie d'Apache, mais une structure à définir par le module pour utiliser des directives de configuration. Le noyau d'Apache ne connaît donc pas son `typedef` et doit traiter les données de façon générique, par un pointeur générique, `void *`.

## 6.2.2 Gestion de mémoire

L'API d'Apache comporte un mécanisme de gestion de mémoire. Il permet de contrôler l'allocation et la désallocation méthodiques de la mémoire, évitant les fuites de mémoire qui pourraient détruire la fiabilité et la stabilité du serveur.

Son fonctionnement repose sur le fait que la mémoire est répartie en groupes de ressources (pools), utilisés pour tout ce qui concerne la mémoire. Ils sont rattachés aux durées de vies des entités spécifiques à Apache, ils sont créés et détruits à la naissance et à la mort de ces entités.

Deux méthodes permettent d'accéder à un groupe de ressources de l'intérieur des gestionnaires, via un argument de fonction ou via une structure de données, comme pour `request_rec` et `conn_rec` avec leurs champs `ap_pool`.

Les gestionnaires Apache peuvent aussi créer des sous-groupes de groupes de ressources et autoriser ainsi un contrôle affiné de la mémoire. Les gestionnaires peuvent détruire ces sous-groupes sans danger et chaque groupe libérera la mémoire des sous-groupes lorsqu'il sera libéré.

Voici quelques fonctions qui serviront pour *mod_trace_output*.

- **char *ap_pcalloc(struct pool *p, int nbytes)** alloue `nbytes` octets de mémoire, remplie de `'\0'`. Equivaut à la fonction C standard `calloc()`.
- **char *ap_pstrdup(struct pool *p, const char *s)** copie la chaîne `s` dans une nouvelle mémoire du groupe `p`, et renvoie un pointeur vers cette mémoire.
- **char *ap_pstrcat(struct pool *p, ...)** concatène plusieurs chaînes en une nouvelle depuis la mémoire du groupe et renvoie un pointeur vers cette chaîne concaténée.
- **char *ap_psprintf(struct pool *p, const char *fmt, ...)** utilise la mémoire du groupe pour exécuter une `sprintf()` (voir C standard).

### 6.2.3 Reste de l'API

L'API d'Apache est très riche, elle contient de nombreuses autres structures et fonctions opérationnelles comme des utilitaires TCP/IP, des fonctions d'URI et d'URL, des utilitaires de directives de configuration, ainsi que d'autres que nous allons décrire plus en détails.

#### Enregistrement dans les journaux

La fonction suivante sera aussi utilisée.

- **void ap_log_error(const char *file, int line, int level, const server_rec *s, const char *fmt, ...)** enregistre une erreur des le fichier `file`, à la ligne `line` de sévérité d'erreur `level`. L'argument `server_rec` est utilisé pour trouver le bon journal dans lequel écrire. La chaîne `fmt` permet de formater le reste de l'argument selon les règles de formatage de `sprintf()` (Voir C standard).

#### Gestion de fichiers

Afin de protéger le serveur des ressources orphelines, l'API Apache se montre très pointilleux sur la façon de créer et de détruire des fichiers, en particulier, les descripteurs de fichiers sont attachés à des groupes afin de pourvoir les fermer en même temps que le groupe. L'association entre un groupe et son fichier constitue une fonction de nettoyage.

Les fonctions qui se chargent des fichiers sont les suivantes.

- **int ap_popenf(pool *p, const char *name, int flag, int mode)** équivaut à la fonction C `open()`, excepté que le descripteur de fichier ouvert est rattaché au groupe indiqué.
- **FILE ap_fopen(pool *p, const char *name, const char *mode)** équivaut à `fopen()`, excepté que le fichier est attaché au groupe indiqué.

- **int ap_closef(pool *p, int fd)** équivaut à `close()`. La fonction de nettoyage du fichier est supprimée du groupe.

## HTTP

Les fonctions suivantes seront utilisées par la suite.

- **void ap_send_http_header(request_rec *r)** envoie les en-têtes de la réponse HTTP au client.
- **int ap_rwrite(const void *buffer, int nbytes, request_rec *r)** envoie `nbytes` octets au client et retourne `-1` en cas d'échec.
- **int ap_rflush(request_rec *r)** envoie les données actuellement en tampon au client. Retourne `-1` en cas d'échec.

## Structures de mémoire

Ces structures ne concernent pas le protocole HTTP ou le fonctionnement interne d'Apache, elles ont été ajoutées à l'API pour leur utilité générale.

Il y en a deux, l'API Array et l'API Table. Les *arrays* sont des tableaux de longueur et de types variables, et les *tables* sont des structures qui font correspondre un nom à une valeur. Ces valeurs doivent être des chaînes de caractères.

Les fonctions suivantes seront utilisées par la suite.

- **void ap_table_set(table *t, const char *key, const char *value)** crée une entrée de valeur unique dans la table indiquée, de nom/valeur `key` et `value`. Les valeurs de `key` et `value` sont copiées.
- **const char *ap_table_get(table *t, const char *key)** récupère la valeur associée à `key` dans la table indiquée.

## Divers

La fonction suivante sera utilisée par la suite.

- **int ap_strcmp_match(const char *str, const char *exp)** compare la chaîne pointée par `str` et celle pointée par `exp`. `'*'` peut être utilisée pour n'importe quel nombre de caractères et `'?'` pour n'importe quel caractère unique.

## 6.3 Architecture d'un module

Le C n'est pas un langage orienté objet, il faut donc qu'Apache fournisse un mécanisme aux modules pour leur permettre d'inscrire des gestionnaires pour chacune des phases de son cycle de vie. Apache a résolu ce problème en définissant une série de structures statiques que les modules vont implémenter.

### 6.3.1 Structure module

La structure `module` reflète le cycle de vie d'un module Apache.

```
module MODULE_VAR_EXPORT example_module =
{
    STANDARD_MODULE_STUFF,
    example_init,               /* module initializer */
    example_create_dir_config,  /* per-directory config creator */
    example_merge_dir_config,   /* dir config merger */
    example_create_server_config,       /* server config creator */
    example_merge_server_config,        /* server config merger */
    example_cmds,               /* command table */
    example_handlers,           /* list of handlers */
    example_translator,         /* filename-to-URI translation */
    example_check_user_id,      /* check/validate user_id */
    example_auth_checker,       /* check user_id is valid *here* */
    example_access_checker,     /* check access by host address */
    example_type_checker,       /* MIME type checker/setter */
    example_fixer_upper,        /* fixups */
    example_logger,             /* logger */
    example_header_parser,      /* header parser */
    example_child_init,         /* process initializer */
    example_child_exit,         /* process exit/cleanup */
    example_post_read_request   /* post read_request handling */

};
```

Les labels, exceptés `example_cmds` et `example_handlers`, sont des pointeurs vers les fonctions du module qui seront les gestionnaires des phases correspondantes déterminées par leur ordre dans la structure. Nous verrons dans le chapitre suivant les signatures attendues par Apache pour ces fonctions.

`MODULE_VAR_EXPORT` est une macro utilisée par les systèmes Win32.

`STANDARD_MODULE_STUFF` est une macro qui passe dans des champs de modules standards comme le numéro de version de l'API, le nom du module, le numéro d'ordre d'exécution du module, le gestionnaire DSO et le pointeur de liste chaînée des modules. Ces champs sont générés et utilisés en interne par le noyau d'Apache.

### 6.3.2 Structure command_rec

Le label `example_cmds` est un pointeur vers un tableau de structures `command_rec`. La structure `command_rec` définit une directive de configuration du module. Elle se présente comme suit.

```
typedef struct command_struct {
      const char *name;       /* Name of this command */
      const char *(*func) (); /* Function invoked */
      void *cmd_data;         /* Extra data, for functions which
                              * implement multiple commands...
                              */
      int req_override;       /* What overrides need to be allowed to
                              * enable this command.
                              */
      enum cmd_how args_how;  /* What the command expects as
                              * arguments.
                              */

      const char *errmsg;     /* 'usage' message, in case of syntax
                              * errors.
                              */
} command_rec;
```

Voici à quoi correspondent les champs.

- **char *name** est le nom de la directive. Il s'agit du label utilisé dans le fichier de configuration.
- **const char *(*func) ()** est un pointeur de fonction à invoquer pour cette directive. La fonction accepte des arguments et change la configuration de ce module en conséquence. Normalement, la fonction renvoie NULL, sinon le retour est considéré comme une erreur.
- **void *cmd_data** est un pointeur vers un bloc de données. Il sert à partager des informations entre les directives de ce module.
- **int req_override** définit la visibilité de cette directive, utilisée par Apache pour déterminer les endroits de la configuration où il est utile de faire appel à cette directive. Il s'agit du `OR` logique de macros définies par Apache dont `RSRC_CONF`, qui définit une directive qui n'est utilisable dans le fichier de configuration qu'en dehors des sections <Directory>, <Location> et <Files>, et `ACCESS_CONF`, qui définit une directive qui n'est utilisable que dans les sections <Directory>, <Location> et <Files> du fichier de configuration.
- **enum cmd_how args_how** décrit le format de la liste d'arguments de la directive dans le fichier de configuration. Ce sont des constantes comme par exemple `NO_ARGS`, la directive ne prend aucun argument, `TAKE1`, elle prend un argument, et `TAKE3`, elle prend trois arguments.
- **Const char *errmsg** est le message que doit afficher Apache s'il rencontre une erreur lors du traitement de cette directive au démarrage.

Les fonctions pointées par `const char *(*func) ()` reçoivent toutes l'argument `mconfig`, qui est un pointeur générique vers les données de configuration par répertoire de ce module, voir *6.2.1 Structures de données* pour plus d'information.

### 6.3.3 Structure handler_rec

La phase de réponse du cycle de traitement des requêtes a la spécificité de permettre à un module de définir plusieurs gestionnaires différents parmi lesquels Apache peut faire son choix en fonction du type de document demandé (voir *6.4 Gestionnaires de contenu et types de documents*).

C'est la structure `handler_rec` qui signifie à Apache l'existence d'un gestionnaire de contenu du module.

Le label `example_handlers` dans la structur `module` vue plus haut, est un pointeur vers un tableau de structures `handler_rec` représentant tous les gestionnaires de contenu du module.

La structure `handler_rec` est définie comme suit.

```
typedef struct {
    const char *content_type; /* MUST be all lower case */
    int (*handler) (request_rec *);
} handler_rec;
```

Les deux champs de `handler_rec` sont les suivants.

- **const char *content_type** est le nom du type de contenu : un type MIME, un codage ou un type de gestion de contenu Apache. Voir *6.4 Gestionnaires de contenu et types de documents*.
- **int (*handler) (request *)** est la fonction qui sera appelée pour ce type particulier.

Comme l'indique la structure `typedef` précédente, la signature des fonctions `handler` doit être la suivante.

```
int handler (request_rec *r)
```

## 6.4 Gestionnaires de contenu et types de documents

Grâce à cette étude de l'architecture modulaire d'Apache, de la manière dont il se configure et de son API, il est possible de comprendre le lien entre les gestionnaires de contenu et le type des documents demandés. Le mécanisme est un peu compliqué, mais aura une grande importance par la suite.

Sa complexité découle de sa flexibilité. Lorsque Apache entre dans la phase de *MIME types checking*, le gestionnaire de types par défaut (module standard *mod_mime*) compare l'extension du fichier avec une table reprennant les types MIME les plus courants. S'il en trouve un, il le place dans le champ `content_type` de la structure `request_rec` de la requête en cours, sinon le type MIME est indéfini et le `content_type` reste NULL.

Il est possible de rajouter des types MIME à la table de *mod_mime* en utilisant la directive *AddType* dans le fichier de configuration.

```
AddType dynamic/php .php
```

Cet exemple fait correspondre le type MIME *dynamic/php* à l'extension de fichier.*php*.

Deux cas un peu plus particuliers peuvent se présenter. Si le fichier est un répertoire physique, Apache lui assigne en interne un type « magic » MIME, défini par la constante DIR_MAGIC_TYPE comme *httpd/unix-directory*. Le second cas apparaît quand le module optionnel *mod_mime_magic* est installé et que le fichier existe physiquement. Dans ce cas, Apache va analyser les premiers octets du fichier et tenter de déterminer son type.

Ensuite, une fois que le type MIME est déterminé, nous avons vu qu'Apache va décider vers quel gestionnaire de contenu envoyer la requête. Pour ce faire, il se base sur les informations qu'il trouve dans les tableaux de structures `handler_rec` (voir *6.3.3 Structure handler_rec*) codés dans les modules. Nous avons vu qu'une structure `handler_rec` associe un type MIME ou un codage (équivalant au type « magic » MIME) avec un gestionnaire de contenu. Par exemple, le module *mod_cgi* associe le type « magic » MIME *application/x-httpd-cgi* avec son gestionnaire de contenu *cgi_handler()* .

Il existe encore une autre méthode, plus flexible, dans laquelle les gestionnaires de contenu sont associés aux fichiers au moment même de l'exécution grâce à des noms explicites. Dans cette méthode, que nous allons utiliser plus tard pour *mod_trace_output*, le module déclare un ou plusieurs noms de gestionnaires de contenu dans son tableau de structures `handler_rec`, au lieu ou en plus des types MIME. Par exemple, *cgi-script*, *trace-output*, *server-info* et *imap-file*. Lorsque cette méthode

est utilisée, Apache place le nom du gestionnaire dans le champ `handler` de la structure `request_rec` de la requête en cours.

Ces noms de gestionnaires de contenu peuvent être associés à leur tour à des fichiers à l'aide des directives *AddHandler* ou *SetHandler* du module standard *mod_actions*, ou encore manuellement pendant le traitement de la requête en plaçant soi-même la valeur désirée dans le champ `handler` de la structure `request_rec` de la requête en cours. *Mod_trace_output* utilisera cette dernière méthode.

La directive *AddHandler* permet d'associer un nom de gestionnaire de contenu à une extension de fichier, comme dans l'exemple suivant.

```
AddHandler cgi-script .cgi
```

La directive *SetHandler* permet quant à elle d'associer tous les fichiers d'un répertoire ou une location avec un nom de gestionnaire de contenu, comme dans l'exemple suivant.

```
<Location /cgi>
      SetHandler cgi-script
</Location>
```

En bref, lorsque Apache doit déterminer vers quel gestionnaire de contenu il doit envoyer la requête, il regarde d'abord dans le champ `handler` de la structure `request_rec` de la requête en cours, s'il trouve quelque chose, il envoie vers le gestionnaire en question, sinon, il regarde dans le champ `content_type` et choisit le gestionnaire sur base du type MIME ou du type « magic » MIME qu'il y trouve.

# 7 Mod_trace_output : le module Apache

Nous allons maintenant exposer une implémentation de la solution du module d'extension pour le serveur web Apache et sur un système d'exploitation de type UNIX/POSIX, qui est le plus courant pour Apache.

La solution choisie est donc un module Apache baptisé *mod_trace_output*, écrit en C, qui va s'immiscer dans le cycle du traitement des requêtes, de manière à obtenir le résultat du traitement fait par les autres modules, c'est-à-dire les données qui vont être envoyées au navigateur.

Deux types de support de stockage sont fournis :
- o des fichiers sur le disque dur du serveur physique, et
- o une base de données MySQL, locale ou distante.

Il y a aussi une option de compression au format gzip tant pour les fichiers que pour les enregistrements en base de données.

Une description de l'installation du module est fournie en annexe.

Ce chapitre traitera en outre des points suivants :
- o première ébauche de la boucle de traitement de requêtes par *mod_trace_output*,
- o fonctionnalités et configuration du module,
- o détails de l'implémentation des structures statiques de *mod_trace_output*,
- o détails de l'implémentation des gestionnaires et des fonctions du module,
- o différents problèmes survenus au cours de l'implémentation et leur résolution,
- o tests et performance du *mod_trace_output*, et finalement
- o acceptation de *mod_trace_output* comme nouveau projet sourceforge.

## 7.1 Boucle de traitement simplifiée

Le schéma simplifié de la boucle du traitement d'une requête avec *mod_trace_output* est illustrée à la figure suivante.

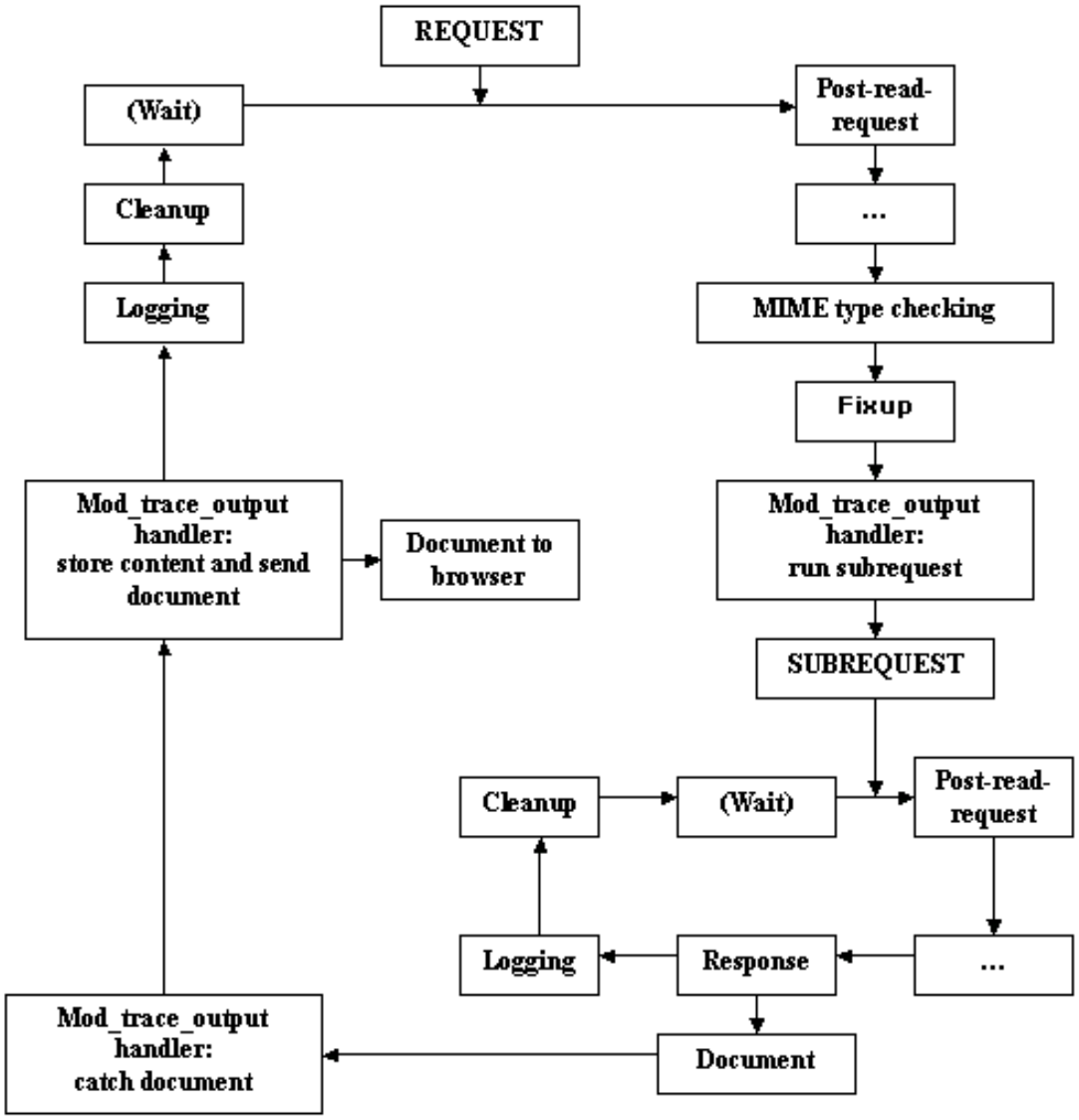


*Fig. 7.1: Schéma du cycle d'une requête gérée par mod_trace_output.*

L'idée est d'utiliser la phase de fixup du cycle de traitement des requêtes pour forcer toutes les requêtes à être gérées par le gestionnaire de contenu de mod_trace_output. Quand le gestionnaire est exécuté dans la phase de response qui suit, il crée une sous-requête à partir de la requête actuelle et lui laisse suivre son cours normal, comme si mod_trace_output n'était pas là, mais en fait mod_trace_output intercepte le résultat de cette sous-requête, il le stocke et l'envoie lui-même au navigateur client.

## 7.2 Fonctionnalités et configuration

Le module *mod_trace_output* possède de nombreuses fonctionnalités et une configuration par répertoire très flexible.

Bien que le but premier de ce module soit d'obtenir une copie du contenu dynamique envoyé au navigateur et de le stocker pour être traité plus tard par l'application WASA, j'ai volontairement conçu *mod_trace_output* pour qu'il puisse être utilisé dans d'autres cas de figure.

Le stockage peut se faire soit dans une base de données MySQL, locale ou distante, soit dans des fichiers sur le disque du serveur. L'utilisation d'une base de donnée MySQL distante est très intéressante car elle permet d'exporter directement les données vers une autre machine où elles peuvent être traitées au fur et à mesure.

Les données stockées peuvent être préalablement compressées au format gzip, afin d'améliorer les performances tant du point de vue de la place occupée sur le disque que du point de vue de la bande passante (lorsque ces données devront être exportées ou dans le cas d'une base de donnée MySQL distante).

De plus, cette compression sert un autre aspect de la communication avec le navigateur. En effet, presque tous les navigateurs actuels supportent le format de compression gzip, permettant, le cas échéant, à *mod_trace_output* d'envoyer les données compressées aux navigateurs. Cela améliore considérablement les performances du serveur en terme de temps de réponse et de nombre de requêtes traitées par seconde. Nous verrons des résultats concrets plus loin (voir *7.7 Benchmarks et résultats*).

*Mod_trace_output* fournit également la possibilité d'enregistrer les actions de stockage qu'il effectue dans un fichier journal spécifique.

Nous allons voir comment configurer toutes ces fonctionnalités à l'aide des directives de *mod_trace_output*.

Les paramètres se rapportant à la base de donnée MySQL (nom, utilisateur, mot de passe, hôte, port TCP/IP et fichier socket) et au fichier journal sont définis au niveau du server virtuel, et sont valables pour toutes les sections <Directory>, <Location> et <File> dans le champ d'application du serveur virtuel.

Par contre, les configurations se rapportant au fait de stocker ou non, aux types de document à traiter, aux types de document à stocker, au fait de compresser ou non, et au répertoire de stockage (dans le cas des fichiers), sont définies dans des sections <Directory>, <Location> ou <File> permettant une configuration très flexible. Par exemple, on peut choisir certains types de pages dynamiques à stocker compressées dans une base de données MySQL pour un répertoire, et d'autres types de

pages à stocker non compressées dans des fichiers pour un autre répertoire.

Les directives du module sont les suivantes.

## ToMySQLInfo

Cette directive prend trois arguments : le nom de la base de données MySQL, le nom d'utilisateur et le mot de passe.

```
ToMySQLInfo dbname user password
```

## ToMySQLHost

Cette directive prend un argument : l'hôte de la base de donnée. Sa valeur par défaut est *localhost*.

```
ToMySQLHost trace-output.sourceforge.net
```

Lorsque l'hôte est *localhost*, MySQL utilise des sockets UNIX au lieu d'une connexion TCP/IP.

## ToMySQLPort

Elle prend un argument : le port TCP/IP que le serveur MySQL écoute sur la machine hôte. Il n'est nécessaire qui si l'hôte est différent de *localhost*. Sa valeur par défaut est le port TCP/IP par défaut de MySQL.

```
ToMySQLPort 3306
```

## ToMySQLSocket

L'unique argument de cette directive est le chemin d'accès complet au fichier socket de MySQL.

```
ToMySQLSocket /var/lib/mysql/mysql.sock
```

Ce n'est nécessaire que s'il n'est pas à sa place par défaut.

## ToMimeTypesIn

Il faut pouvoir spécifier quels types de documents *mod_trace_output* doit traiter. Cela se fait via l'intermédiaire de la directive *ToMimeTypesIn* qui prend une liste composée de quatre types d'éléments :

- des types MIME,
- des types « magic » MIME,
- des noms de gestionnaire de contenu, et
- la chaîne « HANDLER », pour traiter toutes les requêtes dont le champ `handler` est non NULL (voir *6.4 Gestionnaires de contenu et types de documents*).

Les éléments dans la liste doivent être séparés par une virgule. La valeur par défaut est une chaîne vide, ce qui signifie que *mod_trace_output* ne traite aucun document.

```
ToMimeTypesIn text/php,application/x-httpd-cgi,server-paresed,HANDLER
```

## ToMimeTypesOut

Le type MIME du document retourné par la sous-requête (voir *7.1 Boucle de traitement simplifiée*) peut être modifié par celle-ci. Il est donc impossible de le connaître à l'avance, comme par exemple une page JSP qui renvoie une image, de type MIME *image/gif,* générée dynamiquement par *mod_jk*.

La directive *ToMimeTypesOut* permet de spécifier une liste des types MIME des documents dont on veut stocker le contenu. Les éléments de cette liste doivent être séparés par des virgules. La valeur par défaut est *text/html*, ce qui signifie que *mod_trace_output* ne va stocker que les documents HTML renvoyés aux navigateurs.

```
ToMimeTypesIn text/html,text/plain
```

## ToDataGroup

Cette directive prend un argument : une chaîne de caractères qui va permettre de regrouper les enregistrements dans la base de données MySQL. Par défaut, elle prend la valeur du répertoire du document demandé.

```
ToDataGroup MyGroupOfData
```

## ToStoreInMySQL

L'unique argument de cette directive peut prendre deux valeurs, `ON` et `OFF`, respectivement pour dire à *mod_trace_output* de stocker les documents dans la base de données MySQL ou non.

Sa valeur par défaut est `OFF`.

```
ToStoreInMySQL ON
```

## ToGzipInMySQL

L'unique argument de cette directive peut prendre deux valeurs, `ON` et `OFF`, respectivement pour dire à *mod_trace_output* de compresser les données qu'il stocke dans la base de données MySQL ou non.

Sa valeur par défaut est `ON`.

```
ToGzipInMySQL ON
```

### ToStoreInFiles

L'unique argument de cette directive peut prendre deux valeurs, ON et OFF, respectivement pour dire à *mod_trace_output de* stocker les documents dans des fichiers ou non.

Sa valeur par défaut est OFF.

```
ToStoreInFiles ON
```

### ToGzipInFiles

L'unique argument de cette directive peut prendre deux valeurs, ON et OFF, respectivement pour dire à *mod_trace_output* de compresser les données qu'il stocke dans des fichiers ou non.

Sa valeur par défaut est ON.

```
ToGzipInFiles ON
```

### ToDir

Cette directive prend un seul argument : le chemin complet du répertoire où il faut stocker les fichiers si *ToStoreInFiles* est ON.

```
ToDir /var/trace-output
```

Sa valeur par défaut est */tmp.*

## 7.3 Structures statiques

Nous savons que le noyau d'Apache et les modules « communiquent » par l'intermédiaire de structures statiques. Voici celles de *mod_trace_output*.

### 7.3.1 Structure module et gestionnaires

Voici la structure *module* de mod_trace_output.

```
module MODULE_VAR_EXPORT mod_trace_output_module=
{
      STANDARD_MODULE_STUFF,
      to_init,              /* initialize */
      to_create_dir_config, /* per-directory configuration creator */
      NULL,
      NULL,
      NULL,
      to_cmds,              /* command handlers */
      to_handlers,          /* handlers */
      NULL,
      NULL,
      NULL,
      NULL,
      NULL,
```

```
    to_fixup,            /* fixup */
    NULL,
    NULL,
    NULL,
    NULL,
    NULL
};
```

Comme il a été vu dans *6.3.1 Structure module*, c'est elle qui reflète le cycle de vie du module en fournissant au noyau d'Apache les pointeurs vers les fonctions des gestionnaires.

Mod_trace_output a les 3 gestionnaires suivants.

- **static void to_init(server_rec *s, pool *p)** pour la phase d'initialisation du module.
- **static void *to_create_dir_config(pool *p, char *path)** pour la phase de création et l'initialisation de la configuration de répertoire.
- **static int to_fixup(request_rec *r)** pour la phase de fixup du cycle de traitement de requêtes.

Leur implémention sera expliquée dans le chapitre suivant.

### 7.3.2 Structure to_handlers et gestionnaire de contenu

Les gestionnaires de contenu sont signalés au noyau d'Apache grâce à `to_handlers` qui se présente comme suit.

```
static const handler_rec to_handlers[] =
{
        {"trace-output", to_handler},
        {NULL}
};
```

Comme nous l'avons vu plus haut (voir *6.3.3 Structure handler_rec*), c'est un tableau de structures `handler_rec` dont la dernière doit toujours être `{NULL}`. Il n'y a qu'un seul gestionnaire qui gère les documents dont le type est *trace-output* et dont la fonction est la suivante.

- **int to_handler(request_rec *r)** qui va gérer toute la phase de réponse du cycle du traitement de requêtes.

Son implémention sera expliquée dans le chapitre suivant.

### 7.3.3 Structure to_cmds et directives

La configuration du module à l'aide des directives a été présentée dans le chapitre précédent. Ce chapitre présente leurs définitions dans des structures `command_rec` qui sont placées dans le tableau `to_cmds` (voir *6.3.2 Structure command_rec*). Il se présente comme suit.

```
static command_rec to_cmds[] = {
       {"ToMySQLInfo", mysql_info, NULL, RSRC_CONF, TAKE3, "Db, user
                        and password of the MySQL database." },
       {"ToMySQLHost", mysql_host, NULL, RSRC_CONF, TAKE1, "Host of
                        the MySQL database." },
```

```
        {"ToMySQLPort", mysql_port, NULL, RSRC_CONF, TAKE1, "TCP-IP
                        port to connect to the MySQL database." },
        {"ToMySQLSocket", mysql_socket, NULL, RSRC_CONF, TAKE1, "Path
                        to socket (ex:\"/var/llib/mysql/mysql.sock\")
                        to connect to the MySQL database." },
        {"ToMimeTypesIn", set_mime_types_in, NULL, ACCESS_CONF,
                        TAKE1, "Mime types of files requested that
                        mod_trace_output will handle (seperated by
                        comas)." },
        {"ToMimeTypesOut", set_mime_types_out, NULL, ACCESS_CONF,
                        TAKE1, "Mime types of files that
                        mod_trace_output will store {"ToDataGroup",
                        set_data_group, NULL, ACCESS_CONF, TAKE1,
                        "Used to group data in the db by directory
                        (default = directory path)." },
        {"ToStoreInMySQL", set_store_in_mysql,NULL, ACCESS_CONF,
                        TAKE1, "Flag to allow / disallow storing of
                        the pages in MySQL." },
        {"ToGzipInMySQL", set_gzip_in_mysql,NULL, ACCESS_CONF, TAKE1,
                        "Flag to allow / disallow to gzip compress
                        data in MySQL." },
        {"ToStoreInFiles", set_store_in_files,NULL, ACCESS_CONF,
                        TAKE1, "Flag to allow / disallow storing of
                        the pages in a specified directory." },
        {"ToGzipInFiles", set_gzip_in_files,NULL, ACCESS_CONF, TAKE1,
                        "Flag to allow / disallow to gzip compress
                        data in files." },
        {"ToDir", set_dir, NULL, ACCESS_CONF, TAKE1, "Directory where
                        to store pages (default /tmp)." },
        {"ToLogFile", set_log_file, NULL, RSRC_CONF, TAKE1, "File
                        where to log actions." },
        {"ToLog", set_log,NULL, RSRC_CONF, TAKE1, "Flag to allow /
                        disallow the log." },
        { NULL }
};
```

L'implémentation des fonctions associées à ces directives sera exposée dans le chapitre suivant.

## 7.4 Cycle de vie d'Apache et mod_trace_output

Ce chapitre expose et analyse le code des différents gestionnaires et fonctions utiles.

*Mod_trace_output* est constitué des six fichiers suivants, dont le contenu complet est en annexe.

- *mod_trace_output.c* qui est le fichier principal contenant la structure module, les définitions des directives et des gestionnaires.
- *mod_trace_output.h* contient les définitions de la structure de configuration de répertoire de *mod_trace_output* et d'une structure `cb_data` dont il sera question dans *7.4.2 Boucle de traitement de requêtes* section *B. Gestionnaire de contenu*.
- *filter.c* contient la fonction `output_filter()` (voir *7.4.2 Boucle de traitement de requêtes* section *C. Fonction output_filter*).
- *mysql.c* contient tout le code qui gère la base de données MySQL.
- *gzipcomp.c* contient une fonction qui compresse au format gzip.

Voici maintenant les étapes du cycle de vie d'Apache avec en parallèle l'analyse du code de *mod_trace_output*.

### 7.4.1 Démarrage d'Apache

Le cycle de vie d'Apache est illustré dans la figure ci-dessous.

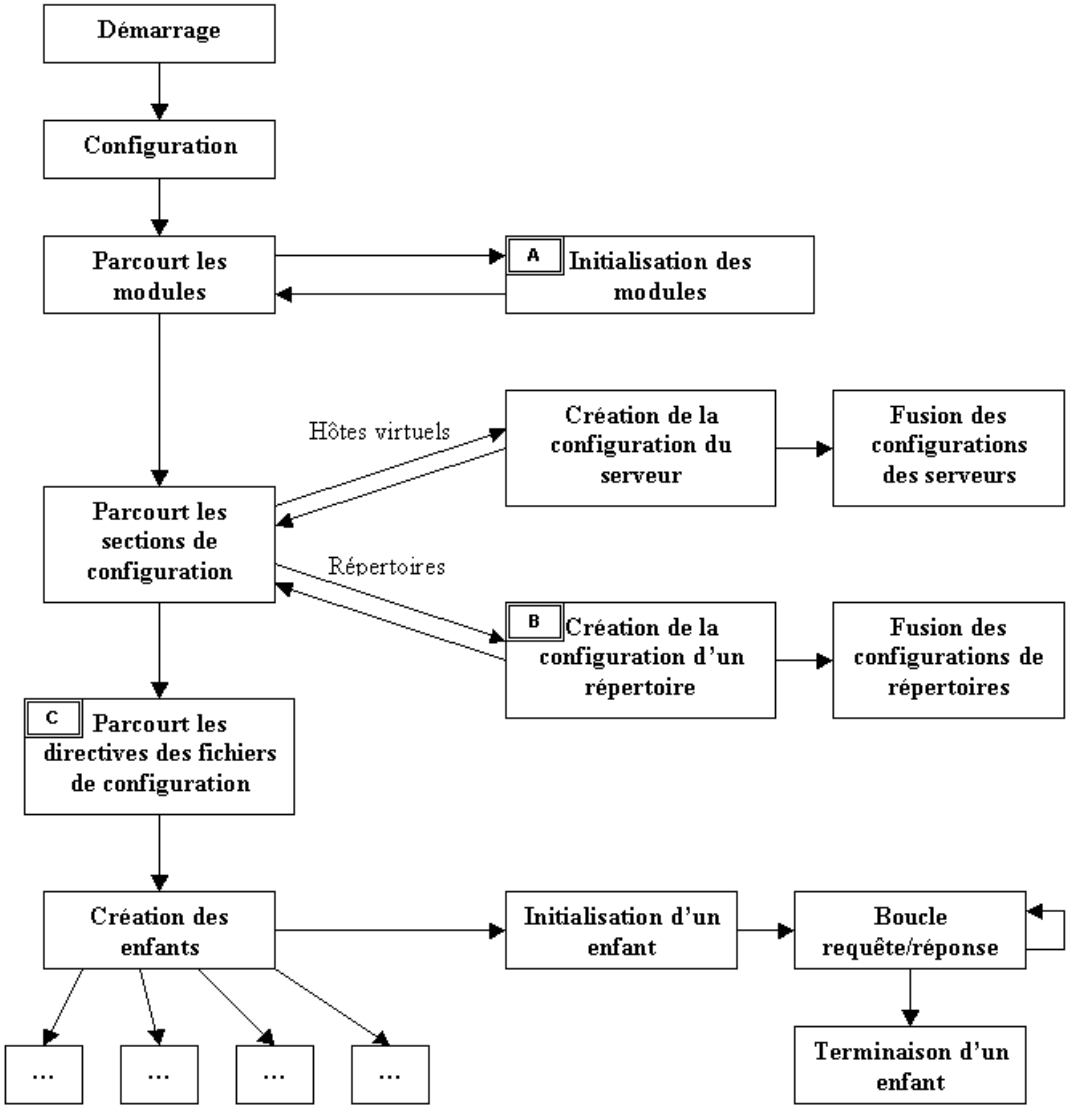


*Fig. 7.2: Cycle de vie d'Apache.*

Il y a trois phases du démarrage pour lesquelles *mod_trace_output* a défini des actions : initialisation du module, création de la configuration de répertoire et parcours des directives des fichiers de configuration.

## A. Initialisation du module

*Mod_trace_output* définit un gestionnaire pour la phase d'initialisation de module (voir *7.3.1 Structure module et gestionnaires*). Il est assez simple et se présente comme suit.

```
static int unid;

static void to_init(server_rec *s, pool *p)
{
        unid = 1;
        return;
}
```

Il consiste à initialiser la variable statique `unid`, qui est une variable globale qui va servir à créer un nom de fichier unique par document (voir *7.4.2 Boucle de traitement de requêtes* section *B. Gestionnaire de contenu*) dans le cas d'un stockage des contenus de documents dans des fichiers sur le disque.

## B. Création de la configuration de répertoire

Le module définit aussi un gestionnaire pour la phase de création de configuration de répertoire.

Il a été vu dans *6.2.1 Structures de données* que la définition de la structure de configuration de répertoires est laissée au module. La voici pour *mod_trace_output*.

```
typedef struct
{
      char  *mime_types_in;
      char  *mime_types_out;

      char  *data_group;
      int   store_in_mysql;
      int   gzip_in_mysql;

      char  *dir;
      int   store_in_files;
      int   gzip_in_files;
}dir_config;
```

Les champs de cette structure vont conserver la configuration par répertoire fournis par l'utilisateur par l'intermédiaire des directives de configuration (voir section suivante et *7.2 Fonctionnalités et configuration*).

Le gestionnaire de création de configuration de répertoire est responsable de la création de cette structure pour chaque section <Directory> ou <Location> rencontrée, ainsi que de l'initialisation des valeurs de ses champs. Il se présente comme suit.

```
static  void *to_create_dir_config(pool *p, char *path)
```

```
{
      dir_config *cfg = (dir_config *) ap_pcalloc(p,
                        sizeof(dir_config));

      cfg->mime_types_in = ap_pstrdup(p,"");
      cfg->mime_types_out = ap_pstrdup(p,"text/html");

      cfg->data_group = path;
      cfg->store_in_mysql = OFF;
      cfg->gzip_in_mysql = ON;

      cfg->dir = ap_pstrdup(p,"/tmp");
      cfg->store_in_files = OFF;
      cfg->gzip_in_files = ON;

      return (void *) cfg;
}
```

Il commence par allouer la mémoire nécessaire en utilisant la fonction `ap_calloc()` fournie par Apache pour la gestion de mémoire (voir *6.2.2 Gestion de mémoire*). Ensuite Il initialise chaque champ à sa valeur par défaut. Les valeurs `ON` et `OFF` sont des macros définies de la manière suivante.

```
#define OFF (0)
#define ON (1)
```

La fonction `ap_strdup()` est également une fonction fournie par Apache pour la gestion de mémoire qui permet de dupliquer une chaîne de caractères dans le groupe de ressources `p` de manière à être sûr que la mémoire va être libérée dès que possible (voir *6.2.2 Gestion de mémoire*).

## C. Parcours des directives des fichiers de configuration

Lorsque Apache parcourt les directives dans les fichiers de configuration, il appelle la fonction correspondante à chaque directive en lui passant les paramètres de la directive (voir *7.2 Fonctionnalités et configuration*).

### *Directives des paramètres MySQL*

Les fonctions correspondant aux directives en rapport avec les paramètres de la base de donnée MySQL sont les suivantes.

```
static const char *mysql_info(cmd_parms * parms, void *dummy,
                              char *db, char *user, char *pwd)
{
      return set_mysql_info(db, user, pwd);
}

static const char *mysql_host(cmd_parms * parms, void *dummy,
                              char *host)
{
      return set_mysql_host(host);
}
```

```
static const char *mysql_port(cmd_parms * parms, void *dummy,
                              char *port)
{
        return set_mysql_port(port);
}

static const char *mysql_socket(cmd_parms * parms, void *dummy,
                              char *socket)
{
      return set_mysql_socket(socket);
}
```

Elles reçoivent les paramètres suivants :

- Une structure `cmd_parms * parms` qui est une structure de paramètres générique de directives, mais nous n'en avons pas besoin.
- Un pointeur générique `void *dummy` vers la structure de configuration de répertoire, mais ici, les directives sont étendues à tout le serveur virtuel (voir *7.2 Fonctionnalités et configuration*), ils n'ont pas leur place dans cette structure, mais bien dans les variables globales définies ci-dessous.
- Une ou plusieurs chaînes de caractères, qui sont les arguments des directives de configuration (voir *7.2 Fonctionnalités et configuration*).

Elles utilisent les fonctions suivantes du fichier *mysql.c* qui, à leur tour, donnent leurs valeurs aux variables statiques globales qui vont être utilisées pour la gestion de la base de donnée.

```
static  char    *mysql_db_host = NULL, *mysql_db_name = NULL,
                *mysql_db_user = NULL, *mysql_db_pwd = NULL,
                *mysql_db_socket = NULL;

static  int     mysql_db_port=0;

const char *set_mysql_info(char *db, char *user, char *pwd)
{
      mysql_db_name = db;
      mysql_db_user = user;
      mysql_db_pwd = pwd;
        return NULL;
}

const char *set_mysql_host(char *host)
{
      mysql_db_host = host;
        return NULL;
}

const char *set_mysql_port(char *port)
{
        mysql_db_port = atoi(port);
        return NULL;
}

const char *set_mysql_socket(char *socket)
{
      mysql_db_socket = socket;
        return NULL;
}
```

### *Directives de fichier journal*

Les fonctions correspondant aux directives se rapportant au fichier journal sont les suivantes.

```
static int log=0;
static FILE *log_file;

static  const char *set_log(cmd_parms *cmd, void *dummy, char *flag)
{
      if(ap_strcmp_match(flag, "ON") == 0)
            log = ON;
      else
            log = OFF;
      return NULL;
}

static  const char *set_log_file(cmd_parms *cmd, void *dummy,
                                 char *logfilename)
{
      if ((log_file = fopen(logfilename, "a")) == NULL)
      {
            fprintf(stderr,"[TRACE OUTPUT Error]: Cannot open trace
                  output log file : %s\n", logfilename);
      }
      return NULL;
}
```

Elles servent toutes les deux à donner leurs valeurs aux variables statiques globales qui vont servir à la gestion du fichier journal (voir *7.2 Fonctionnalités et configuration*).

La première utilise les macros `ON` et `OFF` et la fonction `ap_strcmp_match` de l'API Apache (voir *6.2.3 Reste de l'API*). La deuxième reçoit le nom du fichier journal et l'ouvre. Si une erreur se produit, elle est écrite sur la sortie d'erreur standard qui n'est autre que le fichier journal d'erreur d'Apache.

### *Directives de configuration de répertoires*

Les fonctions correspondant aux directives de configuration de répertoires sont les suivantes.

```
static  const char *set_mime_types_in(cmd_parms *cmd, void *mconfig,
                                      char *mime_types)
{
        dir_config *cfg = (dir_config *) mconfig;
        cfg->mime_types_in = mime_types;
        return NULL;
}

static  const char *set_mime_types_out(cmd_parms *cmd, void *mconfig,
                                       char *mime_types)
{
        dir_config *cfg = (dir_config *) mconfig;
        cfg->mime_types_out = mime_types;
```

```
        return NULL;
}

static  const char *set_data_group(cmd_parms *cmd, void *mconfig,
                                        char *data_group)
{
        dir_config *cfg = (dir_config *) mconfig;
        cfg->data_group = data_group;
        return NULL;
}

static  const char *set_store_in_mysql(cmd_parms *cmd, void *mconfig,
                                        char *flag)
{
        dir_config *cfg = (dir_config *) mconfig;
        if(ap_strcmp_match(flag, "ON") == 0)
                cfg->store_in_mysql = ON;
        else
                cfg->store_in_mysql = OFF;
        return NULL;
}

static  const char *set_gzip_in_mysql(cmd_parms *cmd, void *mconfig,
                                        char *flag)
{
        dir_config *cfg = (dir_config *) mconfig;
        if(ap_strcmp_match(flag, "ON") == 0)
                cfg->gzip_in_mysql = ON;
        else
                cfg->gzip_in_mysql = OFF;
        return NULL;
}

static  const char *set_store_in_files(cmd_parms *cmd, void *mconfig,
                                        char *flag)
{
        dir_config *cfg = (dir_config *) mconfig;
        if(ap_strcmp_match(flag, "ON") == 0)
                cfg->store_in_files = ON;
        else
                cfg->store_in_files = OFF;
        return NULL;
}

static  const char *set_gzip_in_files(cmd_parms *cmd, void *mconfig,
                                        char *flag)
{
        dir_config *cfg = (dir_config *) mconfig;
        if(ap_strcmp_match(flag, "ON") == 0)
                cfg->gzip_in_files = ON;
        else
                cfg->gzip_in_files = OFF;
        return NULL;
}

static  const char *set_dir(cmd_parms *cmd, void *mconfig, char *dir)
{
      dir_config *cfg = (dir_config *) mconfig;
      char *str;

      str = dir + strlen(dir) - 1;
```

```
        if(strcmp(str,"/"))
        {
                cfg->dir = dir;
        }
        else
        {
                bzero(str,1);
                cfg->dir = dir;
        }
        return NULL;
}
```

Remarquons que cette fois-ci, le pointeur générique `void *mconfig` est utilisé pour accéder à la structure de configuration de répertoire.

Chacune de ces fonctions donne sa valeur au champ de la structure de configuration de répertoire correspondant à la directive.

`set_dir()` a un petit traitement particulier visant à permettre de spécifier le répertoire dans le fichier de configuration avec ou sans `"/"` à la fin.

### 7.4.2 Boucle de traitement de requêtes

La boucle de traitement de requêtes par *mod_trace_output* est illustrée dans la figure ci-dessous.

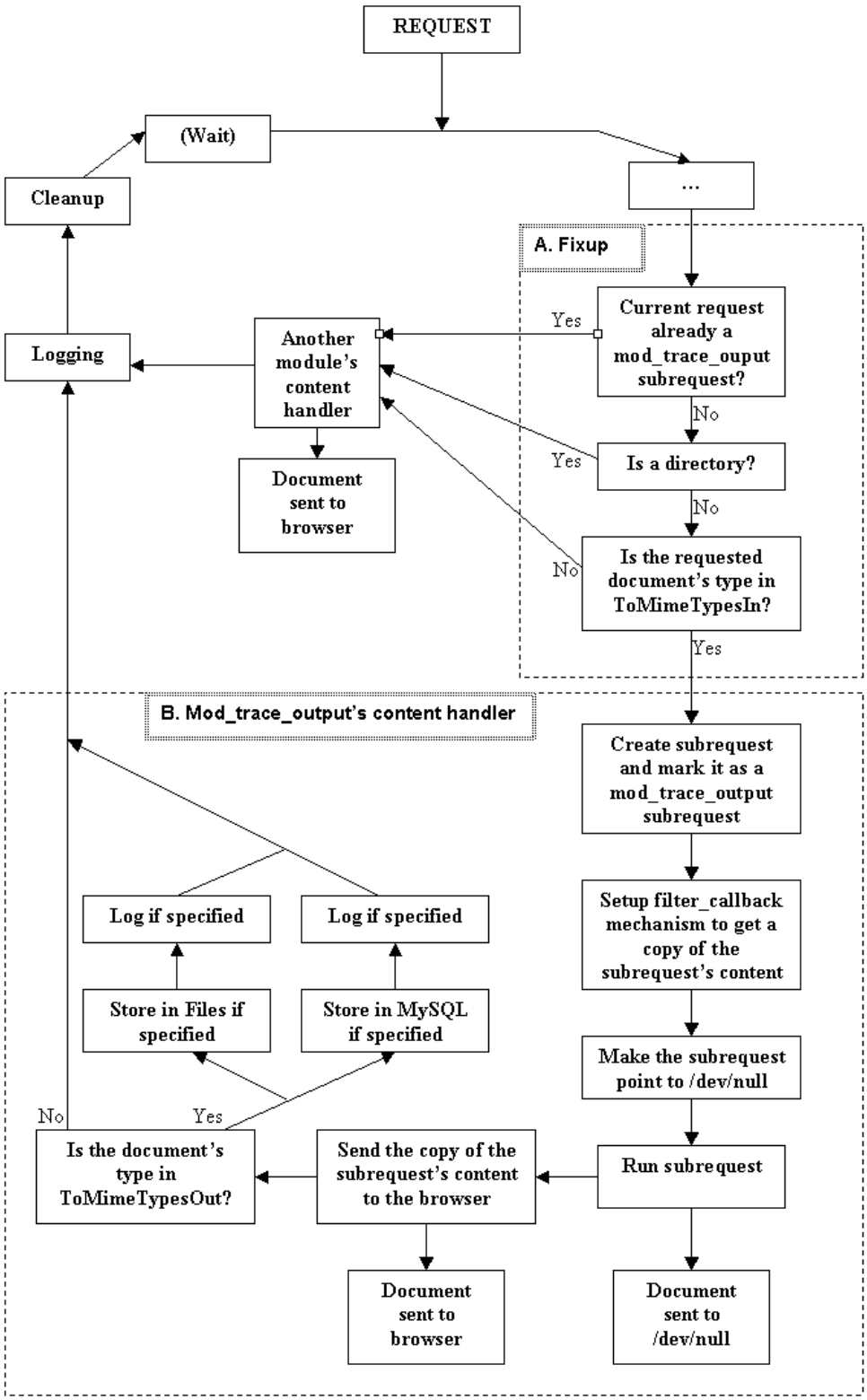


*Fig. 7.3: Boucle de traitement des requêtes par mod_trace_output.*

*Mod_trace_output* fournit un gestionnaire pour la phase de *fixup* et un gestionnaire de contenu. Le schéma ci-dessus va maintenant être expliqué dans les sections suivantes.

## A. Fixup

Le gestionnaire de la phase de fixup reçoit la structure `request_rec` correspondant à la requête en cours.

```
static int to_fixup(request_rec *r)
{
      dir_config *cfg;
```

Ensuite, nous déclarons une variable `cfg` qui pointera vers la structure de configuration de répertoire spécifique au répertoire du document demandé.

```
      if(r->main)
      {
            if(ap_table_get(r->main->notes, "TRACE_OUTPUT_SUBR"))
                  return DECLINED;
      }
```

Après, nous allons vérifier si la requête en cours n'est pas déjà une sous-requête créée par *mod_trace_output*. Nous verrons plus loin que dès que *mod_trace_output* crée une sous-requête, il place une information dans la table `notes` de la requête parente sous forme de la paire clé/valeur suivante : *"TRACE_OUTPUT_SUBR"/"yes"*.

Si la requête en cours est une sous-requête, `r->main` est différent de NULL, nous testons la requête supérieure pour trouver la paire *"TRACE_OUTPUT_SUBR"/"yes"*. Si elle est là, nous ne traitons pas la requête en cours avec *mod_trace_output* mais nous lui laissons suivre son cours normal, sinon nous entrons dans une boucle infinie où chaque sous-requête créée par *mod_trace_output* est à nouveau traitée par *mod_trace_output* et puis sa sous-sous-requête, et ainsi de suite.

```
      if(r->header_only)
            return DECLINED;
```

Si la requête est une requête sans contenu qui ne concerne que des champs ou en-tête HTTP, *mod_trace_output* n'a rien à faire et nous laissons la requête suivre son cours normal.

```
      if(ap_is_directory(r->filename))
            return DECLINED;
```

Si le fichier demandé est un répertoire, on doit laisser le module *mod_dir* faire le travail de trouver la page d'index de ce répertoire. De toute façon, après l'avoir trouvé, *mod_dir* va créer une sous-requête qui sera traitée à son tour par *mod_trace_output.*

```
cfg = (dir_config *) ap_get_module_config(r->per_dir_config,
      &mod_trace_output_module);

//Not going to do anything if no need
if((isOff(cfg->store_in_files) && isOff(cfg->store_in_mysql))
      || !(strcmp(cfg->mime_types_in,"")))
      return DECLINED;
```

Après récupération de la configuration de répertoire grâce à la fonction `ap_get_module_config()` de l'API Apache, la necessité de faire quelque chose est vérifiée à l'aide des conditions suivantes :

- `isOff(cfg->store_in_files)` : faut-il stocker dans des fichiers?
- `isOff(cfg->store_in_ mysql)` : faut-il stocker dans MySQL?
- `!(strcmp(cfg->mime_types_in,""))` : existe-t-il des types de document à traiter?

On utilise la macro `isOff()` qui est définie de la manière suivante : `#define isOff(a) ((a == ON) ? 0 : 1 )`.

```
//Not going to do anything for types not in mime_types_in
if(r->handler)
{
      if(!strstr(cfg->mime_types_in, r->handler) &&
            !strstr(cfg->mime_types_in, "HANDLER"))
            return DECLINED;
}
else
{
      if(!strstr(cfg->mime_types_in, r->content_type))
            return DECLINED;
}
```

Ensuite, nous vérifions si le type de document doit être géré par *mod_trace_output*. Nous avons vu dans *6.4 Gestionnaires de contenu et types de documents* qu'Apache mettait le type du document demandé dans `r->content-type` ou bien le nom du gestionnaire associé au document dans `r->handler` nous testons donc si `r->handler` est différent de NULL.

Si oui, nous testons si le nom du gestionnaire et/ou la chaîne de caractères « HANDLER » est spécifié dans `mime_types_in` (voir *7.2 Fonctionnalités et configuration*). Si ce n'est pas le cas, la requête poursuit son cours normal.

Si `r->handler` est NULL, nous testons si le type du document est spécifié dans `mime_types_in` (voir *7.2 Fonctionnalités et configuration*). Si ce n'est pas le cas, la requête poursuit son cours normal.

Nous avons utilisé la fonction C standard `strstr()` qui renvoie NULL si la deuxième chaîne de caractères n'apparaît pas dans la première.

```
r->handler = "trace-output";
```

la requête doit être traitée par *mod_trace_output* et le nom du gestionnaire de contenu `"trace-output"` est assigné à `r->handler`. De cette manière, la requête est forcée de passer par le gestionnaire de contenu de *mod_trace_output*. Il n'est pas grave d'écraser la valeur précédente, celle-ci va être reconstruite dans la sous-requête crée dans le gestionnaire de contenu (voir section suivante).

```
      return DECLINED;
}
```

Finalement, nous retournons `DECLINED` pour permettre à des gestionnaires d'autres modules de faire aussi quelque chose pendant la phase de fixup.

## B. Gestionnaire de contenu

La structure `cb_data` est définie de la manière suivante.

```
typedef struct
{
      char  *data;
      int   bytes;
} cb_data;
```

Cette structure contient un champ `data` qui pointe vers une chaîne de caractères et un champ `bytes` qui est un entier. Elle va servir à stocker le contenu de la sous-requête et sa longueur.

Le gestionnaire de contenu reçoit aussi la structure `request_rec` correspondant à la requête en cours.

```
int to_handler(request_rec *r)
{
      FILE                    *output_file;
      char                    devnull[] = "/dev/null";
      char                    *dataptr, *filename, *timestamp,
                              *timefile, *encoding, *mime_type;
      int                     rc,len,fd,clifd,cliflags,gzipped;
      struct tm               *newtime;
      time_t                  long_time;
      request_rec             *subr;
      dir_config              *cfg;
      cb_data                 *cb;
      long                    datalen;
```

Ensuite il déclare toute une série de variables qui seront décrites au fur et à mesure de leurs apparitions dans le code.

```
      cfg = (dir_config *) ap_get_module_config(r->per_dir_config,
                                        &mod_trace_output_module);
```

Nous récupérons la configuration du répertoire du document demandé pour la requête en cours.

```
      cb = (cb_data *) ap_palloc(r->pool, sizeof(cb_data));
      cb->data="";
```

```
cb->bytes = 0;
```

On alloue de la mémoire et on initialise la variable `cb` qui est une structure `cb_data` qui va servir à garder le contenu de la sous-requête et sa longueur.

```
if(r->main)
{
      ap_table_set(r->main->notes, "TRACE_OUTPUT_SUBR", "yes");
}
else
{
      ap_table_set(r->notes, "TRACE_OUTPUT_SUBR", "yes");
}
```

Nous testons si la requête courante possède une requête supérieure, auquel cas, nous plaçons une information dans la table `notes` de la requête supérieure sous forme de la paire clé/valeur suivante : *"TRACE_OUTPUT_SUBR"/"yes"*, sinon nous plaçons cette information dans la requête courante qui va devenir la requête supérieure de la sous-requête que nous créerons juste après. Cette information est utilisée dans la phase de *fixup* pour éviter de créer une boucle infinie ou chaque sous-requête créée par *mod_trace_output* est à nouveau traitée par *mod_trace_output* et ensuite sa sous-sous-requête, et ainsi de suite.

```
subr = (request_rec*) ap_sub_req_lookup_uri(r->unparsed_uri,r);

//Backup fd and flags witch can be modified by subrequest
clifd = r->connection->client->fd;
cliflags = r->connection->client->flags;
```

Ensuite, la sous-requête est crée à partir de la requête actuelle, sans toutefois la faire s'exécuter, cela se fera plus tard. Cette sous-requête a le champ `connection` qui pointe vers la même structure `conn_rec`. Nous avons vu dans *6.2.1 Structure de données* que le champ `client` de `r->connection` est la structure de tampon de client. Etant donné que les données de la sous-requête sont envoyées vers `/dev/null`, il faut faire une copie de sauvegarde du descripteur de fichier du tampon client `fd` et du champ `flags`.

```
r->connection->client->filter_callback = (void *)output_filter;
r->connection->client->callback_data = (cb_data *) cb;
```

Nous faisons pointer le champ `filter_callback` vers la fonction `output_filter` qui recevra une copie des données (voir *6.2.1 Structures de données* et *C. Fonction output_filter* plus loin). Cette fonction reçoit la structure de tampon client en paramètre, nous utilisons donc le champ `callback_data` pour lui passer un pointeur vers la variable `cb`. Nous verrons dans *C. Fonction output_filter* que nous utiliserons `cb` pour concaténer et stocker les paquets de données dont nous recevons les copies.

```
/* Sending subrequest response to /dev/null, so that we send
*  the data ourself later.
*/
```

```
if(!(fd = ap_popenf(r->pool,devnull, O_RDWR, S_IRWXU)))
{
      ap_log_error(APLOG_MARK, APLOG_ERR, r->server , "Cannot
                  serve :%s\n", r->uri);
      return DECLINED;
}
```

Le fichier */dev/null* est ouvert et son descripteur est assigné à la variable `fd`.

```
subr->connection->client->fd = fd;
```

Nous changeons le descripteur de fichier du tampon client pour le descripteur de */dev/null*, ce qui a pour effet que le contenu que la sous-requête pense envoyer au navigateur n'est en fait envoyé nulle part.

```
//Run subrequest
rc = ap_run_sub_req(subr);
ap_bflush(subr->connection->client);
ap_pclosef(r->pool,fd);
```

La sous-requête est exécutée, la mémoire tampon est vidée et le fichier */dev/null* est fermé. Après ce traitement, nous avons une copie du contenu généré par la sous-requête dans `cb`.

```
r->connection->client->filter_callback = NULL;
```

Le champ `filter_callback` mis à NULL pour ne pas avoir une deuxième copie des mêmes données lorsque nous allons les envoyer au navigateur plus tard.

```
// Restore fd and flags
r->connection->client->fd = clifd;
r->connection->client->flags = cliflags;
```

Le descripteur de fichier du tampon client `client->fd` et le champ `client->flags` sont restaurées.

```
if(rc != 0)
{
      fprintf(stderr,"[TRACE OUTPUT Error] Error generating
            page : %s", r->uri);
      return rc;
}
```

Si `rc` est différent de zéro, cela signifie qu'il y a eu une erreur lors de l'exécution de la sous-requête, nous écrivons l'erreur dans le fichier journal d'erreur d'Apache et nous la retournons.

```
//Serve the page
r->content_type = subr->content_type;
```

Nous allons maintenant servir la page au navigateur. Nous mettons le type MIME de la requête à la même valeur que celui de la sous-requête.

```
encoding = (char *)ap_table_get(r->headers_in,
```

```
                                    "Accept-Encoding");
```

Nous allons chercher dans la table des champs HTTP d'entrée de la requête la valeur du champ *Accept-Encoding* (voir *6.1.2 Protocole HTTP*), pour voir si le navigateur accepte le contenu compressé au format gzip ou non.

```
r->headers_out = subr->headers_out;
r->err_headers_out = subr->err_headers_out;
```

Nous mettons les tables des champs HTTP de sortie aux mêmes valeurs que ceux de la sous-requête.

```
//Check if client accept encoding
gzipped = 0;
if(strstr((encoding==NULL?"":encoding), "gzip"))
{
      gzipped = 1;
      ap_table_set(r->headers_out, "Content-Encoding", "gzip");
}
```

Nous testons si le navigateur accepte le contenu compressé au format gzip, auquel cas, nous mettons la variable `gzipped` à 1 et nous mettons la valeur du champ HTTP *Content-Encoding* (voir *6.1.2 Protocole HTTP*) à *gzip*. Cela permet au navigateur de savoir que le contenu qui suit dans la réponse HTTP est compressé au format gzip.

```
//If there is a need for compression, compress it!
if(gzipped || isOn(cfg->gzip_in_files) ||
   isOn(cfg->gzip_in_mysql))
{
      dataptr = gzipcompress(cb->data,cb->bytes,&datalen,r);
}
```

Nous avons besoin de compresser les données si le navigateur le supporte (`gzipped=1`), s'il est spécifié de compresser les données dans la configuration de répertoire pour le stockage dans des fichiers (`isOn(cfg->gzip_in_files)`) ou s'il est spécifié de compresser les données dans la configuration de répertoire pour le stockage dans MySQL (`isOn(cfg->gzip_in_` mysql)). Dans ce cas, nous compressons en utilisant la fonction `gzipcompress()` qui prend en paramètres un pointeur vers les données à compresser, la longueur de ces données, un pointeur d'entier dans lequel la fonction va mettre la longueur des données après compression et la requête r, et qui retourne un pointeur vers les données compressées (voir *E. Compression gzip* plus loin).

Nous utilisons aussi la macro `isOn()` qui est définie de la manière suivante : `#define isOn(a) ((a == ON) ? 1 : 0 )`.

```
//If client accepts gzip, send it compressed
if(gzipped)
{
      ap_set_content_length(r, datalen);
      ap_send_HTTP_header(r);
      ap_rwrite(dataptr, datalen, r);
```

```
}
else
{
      ap_set_content_length(r, cb->bytes);
      ap_send_HTTP_header(r);
      ap_rwrite(cb->data, cb->bytes, r);
}
```

Le contenu est finalement envoyé au navigateur, compressé ou non. Nous mettons d'abord la longueur des données dans le champ HTTP *Content-Length* (voir *6.1.2 Protocole HTTP*) via la fonction `ap_set_content_length()`. Ensuite nous envoyons les champs HTTP au navigateur via la fonction `ap_send_HTTP_header()`. Et puis nous envoyons les données au navigateur via la fonction `ap_rwrite()`. Voir *6.2.3 Reste de l'API,* pour plus d'information sur ces fonctions.

```
//Not gonna store for mime types not in mime_types_out
mime_type = ap_pstrdup(r->pool,subr->content_type);
strtok(mime_type,";");
if(!strstr(cfg->mime_types_out, mime_type))
      return OK;
```

Nous passons ensuite au stockage de ce contenu. Nous ne stockons que si le type du document renvoyé par la sous-requête fait partie de la liste spécifiée dans `cfg->mime_types_out` (voir *7.2 Fonctionnalités et configuration*).

Or le champ `subr->content_type` contient le type MIME du document, mais celui-ci peut être suivi par `";Charset=…"` et nous ne voulons que le type MIME. Donc, nous créons d'abord une copie de `subr->content_type` dans `mime_type` et nous utilisons la fonction C standard `strtok()` qui remplace la première occurrence de `";"` par `"/0"` dans `mime_type`. Et nous utilisons la fonction C standard `strstr()` qui renvoie NULL si la deuxième chaîne de caractères n'apparaît pas dans la première.

```
//Store the necessary info and do log
time( &long_time );
newtime = localtime( &long_time );
timestamp = ap_palloc(r->pool, sizeof(char) * 20);
strftime(timestamp,20,"%Y-%m-%d %H:%M:%S",newtime);
```

Nous allons maintenant stocker le contenu. Nous créons d'abord une chaîne de caractères contenant la date au format « YYYY-mm-dd HH:MM:SS » qui est le format de date MySQL. Nous l'utiliserons également comme date pour l'enregistrement dans le fichier journal.

Nous utiliserons les fonctions C `time()` et `localtime()` pour obtenir la variable `newtime` qui est de type `time_t` et qui contient alors la date et l'heure locale. Nous utiliserons la fonction C `strftime()` pour formater cette date et placer le résultat dans la variable `timestamp`.

```
if (isOn(cfg->store_in_files))
{
```

S'il est spécifié dans la configuration qu'il faut stocker dans des fichiers (voir *7.2 Fonctionnalités et configuration*), continuons.

```
timefile = ap_palloc(r->pool, sizeof(char) * 15);
strftime(timefile,15,"%Y%m%d%H%M%S",newtime);
```

Nous créons une autre chaîne de caractères contenant la date, mais avec un format différent, plus approprié pour faire partie du nom du fichier créé : « YYYYmmddHHMMSS ».

```
filename = ap_psprintf(r->pool,"%s/%s_%d_%d.html%s",
          cfg->dir,timefile,r->connection->child_num,
          (unid++), (cfg->gzip_in_files?".gz":""));
```

Ensuite, un nom de fichier unique est créé. Le problème de la création des noms de fichiers semble simple à priori, mais il a posé quelques problèmes. En effet, nous avons vu qu'après l'initialisation, Apache crée des enfants qui sont des copies exactes de lui-même. Il en résulte que plusieurs processus indépendants ont `unid` initialisé à 1! De plus il est impossible d'obtenir un timestamp plus précis qu'à la seconde près, et donc si nous combinons le timestamp et `unid`, nous avons un problème quand des requêtes sont traitées par deux enfants différents dans la même seconde. La solution a été de rajouter une information supplémentaire au nom du fichier sous la forme du champ `child_num` de la structure `connection` de la requête en cours (`r->connection->child_num`). Ce champ est unique par connexion pour l'instant donné.

On utilise la fonction Apache `ap_psprintf()` pour mettre en forme le chemin d'accès complet au nouveau fichier. Le répertoire est donné par la configuration (voir *7.2 Fonctionnalités et configuration*) dans `cfg->dir` et le nom du fichier est formé par la concaténation de `timefile`, `r->connection->child_num`, `unid` (que l'on incrémente en même temps) et de l'extension du fichier qui est soit *.html* soit *.html.gz* si le document est gzippé.

```
if (!(output_file = ap_pfopen(r->pool,filename, "a")))
      ap_log_error(APLOG_MARK, APLOG_ERR, r->server ,
                  "Cannot open %s\n", filename);
```

Nous ouvrons le fichier en écriture avec `ap_pfopen()` (voir *6.2.3 Reste de l'API*). En cas d'erreur, nous l'enregistrons dans le fichier journal d'erreur d'Apache avec `ap_log_error()`(voir *6.2.3 Reste de l'API*).

```
if(isOn(cfg->gzip_in_files))
{
      fwrite(dataptr,sizeof(char),datalen,output_file);
}
else
{
      fwrite(cb->data,sizeof(char),
             cb->bytes,output_file);
}
```

Nous écrivons les données dans le fichier, compressées ou non, selon la configuration (voir *7.2 Fonctionnalités et configuration*). Nous utilisons la fonction C `fwrite()` car elle permet de spécifier la longueur des données à écrire. Si nous avions utilsé `write()`, l'écriture se serait arrêtée dès le premier `'/0'` rencontré or il y en a plusieurs dans les données compressées au format gzip.

```
            ap_pfclose(r->pool,output_file);
```

Le fichier est fermé.

```
            if(isOn(log) && log_file)
            {
                  fprintf(log_file,"[TRACE OUTPUT store file]: %s %s
                        %s\n",r->unparsed_uri,filename,timestamp);
                  fflush(log_file);
            }
}
```

Nous enregistrons l'action dans le fichier journal selon la configuration (voir *7.2 Fonctionnalités et configuration*).

```
      if (isOn(cfg->store_in_mysql))
      {
```

S'il est spécifié dans la configuration qu'il faut stocker dans la base de donnée MySQL (voir *7.2 Fonctionnalités et configuration*) , continuons.

```
            if(isOn(cfg->gzip_in_mysql))
            {
                  safe_mysql_query(r, timestamp, dataptr, datalen,
                              cfg->data_group,cfg->gzip_in_mysql);
            }
            else
            {
                  safe_mysql_query(r, timestamp, cb->data,
                              cb->bytes, cfg->data_group,
                              cfg->gzip_in_mysql);
            }
```

Nous écrivons les données vers la base de données, compressées ou non, selon la configuration (voir *7.2 Fonctionnalités et configuration*), via la fonction `safe_mysql_query()`. Celle-ci sera détaillée plus bas dans *D. Gestion de MySQL*.

```
            if(isOn(log) && log_file)
            {
                  fprintf(log_file,"[TRACE OUTPUT store in mysql]: %s
                        %s %s\n", r->unparsed_uri,
                        cfg->data_group,timestamp);
                  fflush(log_file);
            }
      }
```

Nous enregistrons l'action dans le fichier journal selon la configuration (voir *7.2 Fonctionnalités et configuration*).

```
      return OK;
}
```

Nous avons terminé, nous retournons la valeur `OK` au noyau d' Apache, ce qui signifie que nous avons géré la requête avec succès.

## C. Fonction output_filter

Cette fonction reçoit une copie des données. Cette fonction reçoit en paramètres la structure de tampon client `BUFF *client`, un pointeur vers les données envoyées au tampon `const void *voidptr` et un entier donnant la longueur de ces données.

```
void *output_filter(BUFF *client, const void *voidptr, int i)
{
      char    *dataptr;
      char    *srcptr;
      int     *ret;
      cb_data *cb;
```

Nous déclarons deux pointeurs vers des chaînes de caractères, un entier et un pointeur vers une structure `cb_data`.

```
      srcptr = (char *) voidptr;
```

Nous utilisons `srcptr` pour convertir le pointeur générique `voidptr` en un pointeur de chaîne.

```
      cb = (cb_data *) client->callback_data;
```

Comme vu plus haut dans *B. Gestionnaire de contenu*, nous utilisons le champ `callback_data` du tampon client de la connexion pour passer un pointeur vers les données déjà reçues et stockées dans une structure `cb_data`. Nous utilisons donc `cb` pour convertir le pointeur générique `client->callback_data` en un pointeur vers une structure `cb_data`.

```
      dataptr = ap_palloc(client->pool,
                          sizeof(char) * (cb->bytes + i + 1));
```

Nous allouons la mémoire nécessaire pour concaténer les données déjà contenues dans `cb` et les nouvelles données de longueur `i`.

```
      memcpy(dataptr,cb->data,cb->bytes);
```

Nous copions les données de `cb` dans la nouvelles chaîne de données `dataptr`. Nous utilisons la fonction C `memcopy()` car elle nous permet de spécifier la longueur des données à copier et elle ne s'arrête pas au premier `'/0'` rencontré.

```
      cb->data = dataptr;
```

Nous faisons pointer `cb->data` vers la nouvelles chaîne de données.

```
        dataptr += cb->bytes;
        memcpy(dataptr,srcptr,i);
```

Nous incrémentons le pointeur `dataptr` du nombre d'octets qu'il y avait dans `cb->data`. Ce pointeur pointe alors sur la fin des données déjà stockées dans `cb`. Nous copions alors les nouvelles données à la suite, réalisant ainsi une concaténation.

```
        dataptr += i;
        dataptr = '\0';
```

Nous incrémentons le pointeur `dataptr` du nombre d'octets rajoutés de manière à rajouter un caractère de fin de chaîne `'/0'` à la fin des données.

```
        cb->bytes +=i;

        return;
}
```

Nous incrémentons finalement `cb->bytes` du nombre d'octets rajoutés et nous avons fini.

## D. Gestion de MySQL

Voici maintenant les fonctions qui s'occupent de la gestion de la base de donnée. L'API MySQL met plusieurs fonctions et structures à notre disposition, un manuel complet est disponible sur Internet à l'adresse *http://www.mysql.com/documentation/index.html*.

La structure `MYSQL` représente un descripteur d'une base de données MySQL. Elle est utilisée par presque toutes les fonctions de l'API.

Les fonctions de l'API MySQL qui vont être utilisées sont les suivantes.

- **MYSQL *mysql_init(MYSQL *mysql)** alloue de la mémoire et initialise une structure `MYSQL`.
- **MYSQL *mysql_real_connect(MYSQL *mysql, const char *host, const char *user, const char *passwd, const char *db, unsigned int port, const char *unix_socket, unsigned int client_flag)** qui tente d'établir une connexion à un moteur de base de données MySQL qui tourne sur l'hôte `host`. Lorsqu'une valeur NULL ou 0 est passée à la place d'un paramètre, MySQL utilise alors les paramètres par défaut. Elle retourne le pointeur vers `mysql` si la connexion est réussie ou NULL dans le cas contraire.
- **int mysql_select_db(MYSQL *mysql, const char *db)** fait en sorte que la base de données spécifiée par `db` soit la base de données par défaut dans la connexion `mysql`. Et ce, y compris pour les requêtes qui vont suivre, si celles-ci ne spécifient pas une base de données explicitement.

- **char *mysql_error(MYSQL *mysql)** retourne le message d'erreur pour la fonction de l'API la plus récemment employée. Elle retourne "" si cette fonction n'a pas fait d'erreur.
- **unsigned int mysql_real_escape_string(MYSQL *mysql, char *to, const char *from, unsigned int length)** est utilisée pour créer une chaîne de caractères légale utilisable dans une requête MySQL. Elle permet de spécifier la longueur `length` des données, ce qui nous permet d'utiliser des données binaires qui peuvent contenir le caractère `'/0'`.
- **int mysql_real_query(MYSQL *mysql, const char *query, unsigned int length)** exécute la requête SQL pointée par `query`. Elle permet de spécifier la longueur `length` de `query`, ce qui nous permet d'envoyer des données binaires qui peuvent contenir le caractère `'/0'`.

Voyons maintenant le code.

```
static  MYSQL   mysql_server, *mysql_db = NULL;
```

En plus des variables statiques de configuration (voir *7.4.1 Démarrage d'Apache* Section *C. Parcours des directives des fichiers de configuration*), nous déclarons une structure `MYSQL` et un pointeur vers cette structure.

```
static void open_dblink()
{
      if (mysql_db_name != NULL)
      {
            mysql_init(&mysql_server);
            mysql_db = mysql_real_connect(&mysql_server,
                        mysql_db_host, mysql_db_user, mysql_db_pwd,
                        mysql_db_name, mysql_db_port,
                        mysql_db_socket,0);
      }
}
```

Cette première fonction va nous servir à ouvrir une connexion vers le moteur de base de données MySQL, en utilisant les paramètres spécifiés dans la configuration du module (voir *7.2 Fonctionnalités et configuration*). Si le nom de la base de donnée choisie est non NULL, nous initialisons la structure `mysql_server` et nous nous connectons. Comme vu plus haut, la fonction renvoie dans `mysql_db` soit un pointeur vers `mysql_server`, soit NULL, en cas d'erreur. L'erreur est alors accessible via `mysql_error(&mysql_server)`.

```
void safe_mysql_query(request_rec *r, char *timestamp, char *data,
                        int data_len,char *data_group, int is_gzipped)
{
      int error = 1, gone = 0;
      char *str,*esc_data,*query,*str_data_len;
      char *esc_data_group, *esc_uri;
      int was_connected = 0, data_group_len = 0, uri_len = 0,
           query_len = 0, esc_data_len = 0, str_data_len_len = 0;

      void (*sigpipe_handler)();

      sigpipe_handler = signal(SIGPIPE, SIG_IGN);
```

Nous avons utilisé cette fonction dans le gestionnaire de contenu, voir *B. Gestionnaire de contenu*. Elle reçoit en paramètres la requête en cours, une chaîne de caractère contenant la date formatée au format MySQL « YYYY-mm-dd HH:MM:SS », les données, la longueur de ces données, la chaîne de caractère représentant le groupe des données (voir *7.2 Fonctionnalités et configuration*) et un flag disant si les données sont compressées ou non.

Nous déclarons une série de variables qui seront expliquées par la suite, et un pointeur vers une fonction `sigpipe_handler`, ce qui nécessite quelques explications. Le système possède en interne une série de gestionnaires qui vont gérer les signaux correspondants à des événements exceptionnels qui se produisent durant l'exécution. Un de ces signaux est défini par `SIGPIPE`. Il est possible que MySQL envoie ce signal au système, mais nous ne voulons pas que le système s'arrête. Nous utilisons alors la fonction C `signal()` qui prend 2 arguments : le code du signal et la fonction qui va gérer ce signal à l'avenir (lorsque `SIG_IGN` est utilisé, le signal n'est pas pris en compte), et qui retourne le gestionnaire défini précédemment pour ce signal, ce qui nous permet d'en faire une copie de sauvegarde dans `sigpipe_handler` que nous restaurerons après le traitement de MySQL.

```
if (mysql_db)
{
      mysql_select_db(mysql_db, mysql_db_name);
}
```

Si `mysql_db` est non NULL, cela signifie que nous sommes déjà connectés au moteur de MySQL. Nous sélectionnons alors la base de données `mysql_db_name` à l'aide de la fonction `mysql_select_db()`

```
else
{
      open_dblink();
```

Sinon, nous ouvrons la connexion avec `open_dblink()`.

```
      if (mysql_db == NULL) // unable to link
      {
            signal(SIGPIPE, sigpipe_handler);
            str = ap_pstrcat(r->pool, "[TRACE OUTPUT Error]:
                  connect failed:  ",
                  mysql_error(&mysql_server), NULL);
            fprintf(stderr, str);
            return;
      }
}
```

Si `mysql_db` est NULL après `open_link()`, cela signifie que la connexion n'a pas pu être établie. Nous rétablissons le gestionnaire du signal système `SIGPIPE`, nous créons un message d'erreur à partir du message d'erreur de `mysql_server` et nous l'enregistrons dans le journal d'erreur d'Apache.

A partir de là, nous sommes sûrs d'avoir une connexion avec le moteur MySQL et que `mysql_db` pointe sur une structure `MYSQL` valide et initalisée. Nous allons donc créer la requête et l'exécuter.

```
esc_data_group = ap_palloc(r->pool,
                  sizeof(char) * (strlen(data_group) * 2 + 1));
data_group_len = mysql_real_escape_string(mysql_db,
                  esc_data_group, data_group,
                  strlen(data_group));

esc_uri = ap_palloc(r->pool,
                  sizeof(char) * (strlen(r->uri) * 2 + 1));
uri_len = mysql_real_escape_string(mysql_db, esc_uri, r->uri,
                  strlen(r->uri));
```

Nous allons créer deux nouvelles chaîne de caractères `esc_data_group` et `esc_uri` en leur allouant la mémoire suffisante et nous utilisons `mysql_real_escape_string()` pour obtenir des chaînes utilisables par MySQL. Nous stockons leurs longueurs dans `data_group_len` et `uri_len`.

```
str_data_len = ap_psprintf(r->pool,"%d",data_len);
str_data_len_len = strlen(str_data_len);
```

Nous convertissons la longueur des données en chaîne de caractère `str_data_len` et nous stockons la longueur de cette chaîne dans `str_data_len_len`.

```
if(is_gzipped)
{
      esc_data = ap_palloc(r->pool,
                  sizeof(char) * (data_len * 2 + 1));
      esc_data_len = mysql_real_escape_string(mysql_db,
                  esc_data, data, data_len);
      query_len = 26 + 19 + 3 + data_group_len + 3 + uri_len+
                  3 + esc_data_len + 2 + str_data_len_len + 3;
      str = ap_palloc(r->pool, sizeof(char) * (query_len+1));
      query = str;
      memcpy(str,"INSERT INTO data VALUES ('",26);
      str += 26;
      memcpy(str,timestamp,19);
      str += 19;
      memcpy(str,"','",3);
      str += 3;
      memcpy(str,esc_data_group,data_group_len);
      str += data_group_len;
      memcpy(str,"','",3);
      str += 3;
      memcpy(str,esc_uri,uri_len);
      str += uri_len;
      memcpy(str,"','",3);
      str += 3;
      memcpy(str,esc_data,esc_data_len);
      str += esc_data_len;
      memcpy(str,"',",2);
      str += 2;
      memcpy(str,str_data_len,str_data_len_len);
      str += str_data_len_len;
      memcpy(str,",1)",3);
      str += 3 + 1;
```

```
        str = '\0';
    }
```

Si les données sont compressées, il est possible qu'elles contiennent des caractères `'\0'`, et nous ne pouvons pas utiliser les fonctions qui concatènent des chaînes de caractères, mais bien celles qui copient de la mémoire sans regarder le contenu. Nous allouons la mémoire nécessaire à recevoir la requête SQL et utilisons donc un pointeur `str` pour traverser cet espace mémoire et copier les différentes chaînes de caractères nécessaires aux endroits adéquats.

```
    else
    {
        esc_data = ap_palloc(r->pool,
                    sizeof(char) * (data_len * 2 + 1));
        mysql_real_escape_string(mysql_db, esc_data,
                    data,data_len);
        query = ap_pstrcat(r->pool, "INSERT INTO data VALUES ('",
                    timestamp, "','", esc_data_group, "','",
                    esc_uri, "','", esc_data, "',", str_data_len,
                    ",0);", NULL);
        query_len = strlen(query);
    }
```

Si les données ne sont pas compressées, nous utilisons `ap_pstrcat()` (voir *6.2.3 Reste de* l'API) pour concaténer toutes les chaînes de caractères.

```
    error = mysql_real_query(mysql_db, query, query_len);
```

Nous exécutons la requête SQL avec `mysql_real_query()`.

```
    if (error)
        gone = !strcasecmp(mysql_error(mysql_db), "mysql server
                            has gone away");
```

Si une erreur se produit, et que cette erreur est « mysql server has gone away », nous avons besoin de nous reconnecter au moteur MySQL.

```
    if (gone)
    {
        // we need to restart the server link
        mysql_db = NULL;

        open_dblink();
```

Donc, si nous avons perdu la connexion, nous appellons `open_dblink()` pour nous reconnecter.

```
        // unable to link
        if (mysql_db == NULL)
        {
            signal(SIGPIPE, sigpipe_handler);
            str = ap_pstrcat(r->pool, "[TRACE OUTPUT Error]:
                MySQL connect failed:  ",
                mysql_error(&mysql_server), NULL);
            fprintf(stderr, str);
            return;
        }
```

Si `mysql_db` est NULL, nous n'avons pas réussi à nous connecter, nous rétablissons alors le gestionnaire du signal système `SIGPIPE`, nous créons un message d'erreur à partir du message d'erreur de `mysql_server` et nous l'enregistrons dans le journal d'erreur d'Apache.

```
            error = mysql_select_db(mysql_db, mysql_db_name) ||
                        mysql_real_query(mysql_db, query, query_len);
      }
```

Finalement, si `mysql_db` est non NULL, nous avons réussi à nous reconnecter, nous sélectionnons la base de données à nouveau et nous réessayons d'exécuter la requête.

```
      signal(SIGPIPE, sigpipe_handler);
```

Nous rétablissons de toute façon le gestionnaire du signal système `SIGPIPE`.

```
      if (error)
      {
            str = ap_pstrcat(r->pool, "[TRACE OUTPUT Error]: MySQL
                  query failed:  ", query, NULL);
            fprintf(stderr, str);
            str = ap_pstrcat(r->pool, "[TRACE OUTPUT Error]: MySQL
                  failure reason:  ", mysql_error(mysql_db), NULL);
            fprintf(stderr, str);
      }
      return;
}
```

Et s'il se produit une erreur lors du deuxième essai, nous créons un message d'erreur à partir du message d'erreur de `mysql_db` et nous l'enregistrons dans le journal d'erreur d'Apache.

## E. Compression gzip

La fonction que nous utilisons plus haut pour compresser les données se présente comme suit.

```
char *gzipcompress(const char *source, long length,
                   long *dest_length, request_rec *r)
```

Le code complet est disponible en annexe.

Cette fonction a été trouvée dans le module *mod_mmap_dynamic* (disponible à l'adresse *http://sourceforge.net/projects/mmap-dynamic*) qui est sous licence *Apache Software License* (voir annexe J), ce qui signifie que les sources sont libres. Le nom de l'auteur n'était pas spécifié.

Cette fonction n'étant pas documentée dans le module *mod_mmap_dynamic,* nous allons brièvement expliquer son fonctionnement.

Elle utilise la fonction `compress2()` la librairie `zlib` pour compresser les données mais, elle retire l'en-tête du résultat et sa somme de contrôle pour les remplacer par l'en-tête et la somme de contrôle correspondant au format gzip.

## 7.5 Problèmes rencontrés

Tout ne s'est pas passé si facilement. Ce module, tel qu'il est, est le résultat d'un travail acharné, au cours duquel il a fallu modifier la structure du module plusieurs fois pour gérer toujours plus de cas de figure. Ce type de développement est caractéristique des projets pour lesquels il existe peu de documentation, l'apprentissage vient en pratiquant, jusqu'au moment où les connaissances sont suffisantes pour pouvoir revenir en arrière et créer un code plus robuste, plus flexible et plus général.

Le fonctionnement du module est assez complexe, et si l'architecture modulaire d'Apache a de nombreux avantages, elle a aussi des inconvénients. Il y a peu de mécanismes de communication entre modules, les gestionnaires de tous les modules doivent s'ordonner pour chaque phase, et plus il y a de modules, plus cela devient complexe de déterminer exactement quel est le chemin suivi par une requête, avant d'aboutir au navigateur du client.

Voyons quelques exemples de problèmes qui sont survenus et de cas particuliers qui ont dû être traités.

La première version de *mod_trace_output* ne gérait que les types MIME généraux et stockait le contenu des documents dans un unique fichier dans lequel les différents processus enfants d'Apache écrivaient en même temps.

A partir du moment ou le concept de récupération des données par la fonction `output_filter` a été mis au point, le problème du nom des fichiers est survenu. Les variables « globales » ne le sont pas. En effet, chaque enfant possède sa propre copie de ces variables. La solution à ce problème a été exposée dans *7.4.2 Boucle de traitement de requêtes* section *B. Gestionnaire de contenu*.

Ensuite, l'étape suivante consiste à stocker ces fichiers dans un format compressé. Le contenu n'était pas complet parce que les données compressées au format gzip peuvent contenir des caractères `'/0'`, ce qui les rend inutilisables par les fonctions qui traitent les chaîne de caractères, `'/0'` étant le caractère de fin de chaîne.

Un stockage dans une base de données MySQL semblait intéressant pour améliorer les performances et pour permettre d'exporter directement les données vers un serveur distant. La plupart des problèmes qui ont suivi ont été dus au fait que la plateforme de travail était Windows couplé avec l'interface Cygwin (disponible à l'adresse *http://www.cygwin.com*), qui permet d'utiliser tous les outils de base de Linux sous Windows.

En effet, les performances du stockage dans les fichiers n'étaient pas très bonnes et cela pour deux raisons, Windows gère assez mal la création de fichiers et l'interface Cygwin rajoute une étape supplémentaire qui prend un certain temps. D'où l'idée de la base de données MySQL, mais malheureusement, celle-ci n'a jamais voulu compiler avec Cygwin. La seule solution a été de changer de plateforme et d'utiliser directement Linux. MySQL a alors compilé sans problème et les performances de la création de fichiers se sont sensiblement améliorées.

L'architecture générale de *mod_trace_output* commençant à apparaître, celui-ci a subi quelques tests qui ont surtout servi, dans un premier temps, à découvrir des cas particuliers.

A ce moment-là, le module *mod_trace_output* s'insérait dans le traitement des requêtes de manière à être toujours le premier module exécuté. Il gérait toutes les requêtes principales et laissait les sous-requêtes suivre leurs cours. Mais lorsque le document demandé dans l'URI est un répertoire, *mod_trace_output* ne peut plus passer en premier, il doit laisser *mod_dir* faire son travail et le laisser trouver le document index de ce répertoire. *Mod_dir* crée alors une sous-requête qui renvoie le document index au navigateur et *mod_trace_output* doit gérer cette sous-requête, sous peine de perdre l'information correspondante.

Le test avec *mod_jk* et les pages JSP a montré que *mod_trace_output* n'était en fait jamais appelé, comme s'il était inexistant. La raison en était que celui-ci utilise la phase de traduction d'URI pour forcer les requêtes à passer par son gestionnaire de contenu en plaçant directement le nom de ce gestionnaire dans le champ `handler` de la requête (voir les sources de mod_jk disponibles à l'adresse *http://jakarta.apache.org/tomcat*). Dès lors, une partie du code de *mod_trace_output* a été déplacée vers la phase de fixup (voir *7.4.2 Boucle de traitement de requêtes* section *A. Fixup*) afin de reprendre le contrôle sur le gestionnaire de contenu.

## 7.6 Sourceforge.net

Le travail réalisé sur *mod_trace_output* est original, il utilise une fonctionnalité des toutes dernières versions d'Apache, très peu documentée et peu connue (voir *6.2.1 Structures de données* section *Structure de tampon client*). De plus, la mailing list des développeurs de modules Apache a mis en évidence l'intérêt de ses capacités en termes de stockage de contenu.

C'est pourquoi *mod_trace_output* a été placé sous licence *Apache Software License* (voir annexe J) et sa candidature a été posée chez sourceforge.net (plus d'informations peuvent être trouvées à l'adresse *http://sourceforge.net*) afin d'être hébergé sur sourceforge avec toutes les avantages que cela implique : environnement de travail, mailing list, forum, etc.

Le projet a été accepté par sourceforge le 26/04/2002, il a été annoncé sur plusieurs sites dont *http://www.linuxlinks.com* et *http://freshmeat.net* et jusqu'à aujourd'hui dimanche 12/05/2002, il y a eu près de 2000 visites de la homepage et près de 150 downloads.

La homepage est visible à l'adresse *http://trace-output.sourceforge.net* et l'espace de travail de *mod_trace_output* sur sourceforge peut être trouvé à *http://sourceforge.net/projects/trace-output.*

C'est un accomplissement d'autant plus satisfaisant que sourceforge est très sélectif par rapport aux projets qu'ils acceptent, ceux-ci doivent être originaux, nouveaux et bien entendu, non-triviaux.

## 7.7 Benchmarks et résultats

Plusieurs tests ont été réalisés avec le module *mod_trace_output* intégré à un serveur Apache 1.3.23, installé sur un pentium II 400 Mhz avec 128 Mb de RAM et un système d'exploitation Linux Debian 2.2. La version du moteur de base de données MySQL est 3.23.49.

La machine cliente est un pentium III 1 Ghz avec 256 Mb de RAM disposant d'une connexion adsl.

Les tests ont été réalisés à l'aide de Jakarta JMeter (*http://jakarta.apache.org/jmeter*).

### 7.7.1 Tests de comparaison

Chaque test consistait à simuler 10 navigateurs faisant 4 requêtes par seconde pour des documents dont les tailles étaient successivement: 1, 20, 40 et 80 kb. Chaque test dure une minute.

Les navigateurs web peuvent ou non supporter les documents dont le contenu est compressé. Ces deux cas de figure seront testés pour cinq configurations du serveur:

- sans *mod_trace_output*,
- stockage dans des fichiers non compressés,
- stockage dans une base de données MySQL, données non compressées,
- stockage dans des fichiers compressés, et
- stockage dans une base de données MySQL, données compressées.

## Contenu non compressé pour le navigateur

Le navigateur ne supporte pas le contenu compressé, les quatres fichiers de 1, 20, 40 et 80 kb sont envoyés en entier à chaque requête. Les résultats de ces tests ont présentés très peu de différences.

La seule conclusion intéressante est que le temps perdu lors du traitement de la requête par *mod_trace_output* est négligeable par rapport au temps de transfert de la réponse au navigateur(3360 ms en moyenne).

Par exemple, le graqhique suivant montre les temps de réponse pour les quatres pages groupées par couleur, dans le cas où m*od_trace_output* est configuré pour stocker le contenu des documents non compressé dans une base de données MySQL.

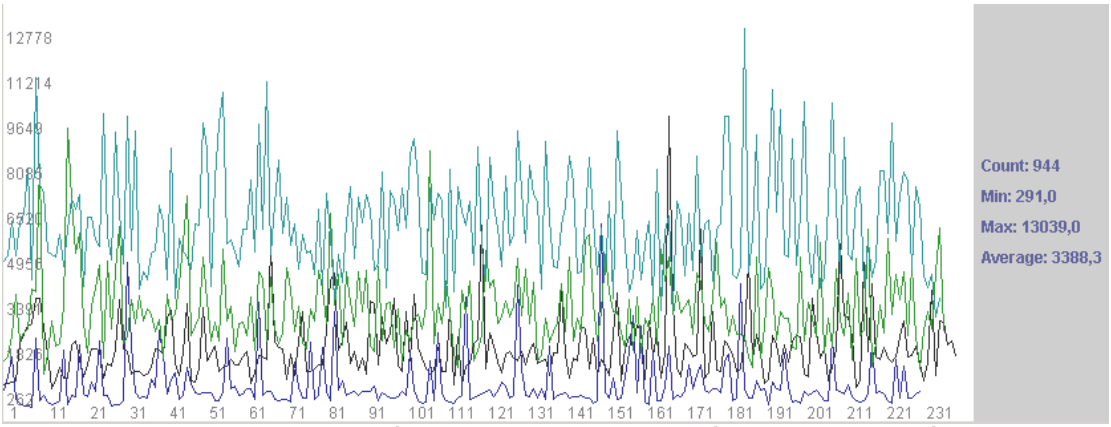


*Fig. 7.4 : Temps de réponse pour MySQL, données non compressées.*

| | |
|---|---|
| Nombre de requêtes par seconde (r/s) : | 15,73 |
| Temps moyen de réponse (ms): | 3388 |

## Contenu compressé pour le navigateur

Le navigateur supporte le contenu compressé, les fichiers de 1, 20, 40 et 80 Koctets au départ ont une taille de, respectivement, 839, 1016, 1153 et 1413 octets. Les résultats sont plus intéressants.

| | Sans le module | Fichiers non compressés | MySQL non compressés | Fichiers compressés | MySQL compressés |
|---|---|---|---|---|---|
| Nombre de requêtes par seconde (r/s) | 23,9 | 20,7 | 14,7 | 22,9 | 23,6 |
| Temps de réponse moyen (ms) | 147 | 234 | 436 | 185 | 173 |

Lorsque le contenu est compressé avant d'être stocké, le nombre de requêtes par seconde est pratiquement identique à celui obtenu lorsque le module est désactivé. C'est un très bon résultat, étant donné que peu de serveurs dépassent une activité de 10 requêtes par seconde.

Par contre, lorsque le contenu n'est pas compressé avant d'être stocké, les performances sont moins bonnes. Le temps nécessaire à l'écriture de plusieurs dizaines de Koctets dans des fichiers n'est plus négligeable face au temps de transfert d'1 Koctet vers le navigateur. C'est encore plus marqué pour la base de données, probablement à cause de la taille réduite des paquets transférés entre *mod_trace_*output et MySQL par l'intermédiaire de sockets.

Par exemple, le graphique ci-dessous montre les temps de réponse pour les quatres pages groupées par couleur, dans le cas où *mod_trace_output* est configuré pour stocker le contenu des documents non compressé dans une base de données MySQL.

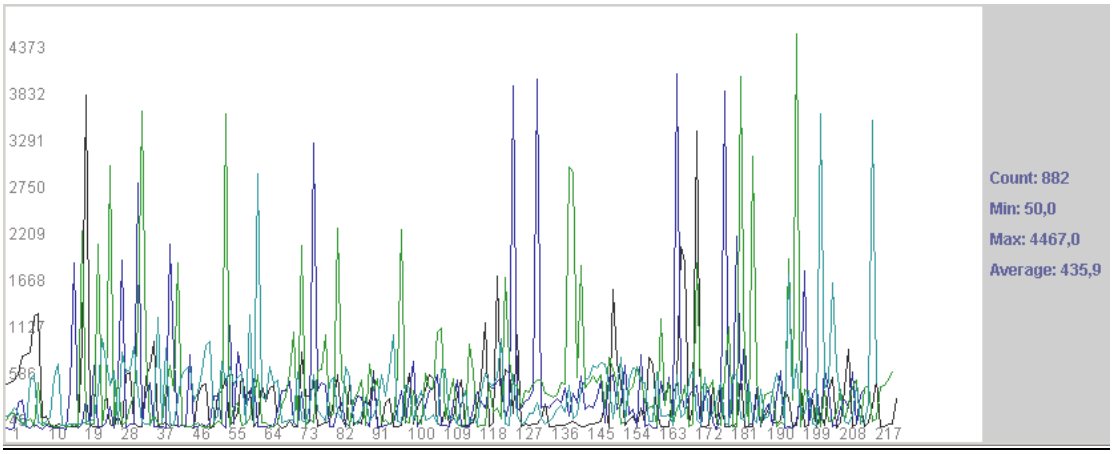


*Fig. 7.5 : Temps de réponse pour des fichiers non compressés.*

### 7.7.2 Tests de stabilité

Plusieurs tests de 30 minutes ont été conduits pour un navigateur supportant le contenu compressé et pour deux configurations de *mod_trace_output* : stockage dans des fichiers compressés et stockage des données compressées dans MySQL. Chaque test consistait à simuler 10 navigateurs faisant 4 requêtes par seconde pour des documents dont les tailles étaient successivement: 1, 20, 40 et 80 kb.

| | Fichiers compressés | MySQL compressés |
|---|---|---|
| Nombre total de documents | 33971 | 31537 |
| Nombre de requêtes par seconde (r/s) | 18,9 | 17,5 |
| Temps de réponse moyen (ms) | 259 | 368 |

Le serveur reste stable et les temps de réponse sont réguliers.

### 7.7.3 Conclusions

Les performances du module *mod_trace_output* sont très bonnes dans toutes les configurations en charge normale (<10 r/s).

En charge plus intense (>15 r/s), la compression des données stockées est indispensable pour garder un bonne tenue.

*Mod_trace_output* reste stable, même sous des charges de près de 20 r/s.

# Conclusions

Le problème posé était de mettre le contenu textuel et les meta-informations de consultation de pages web dynamiques à disposition de l'application WASA, en vue de leur analyse sémantique et statistique.

Une première approche de la solution de ce problème a consisté à extraire le contenu des pages brutes par déformatage, en utilisant un compilateur écrit en Java. Ce dernier traduit un langage source, constitué de l'HTML et du script, en un langage cible, constitué du seul HTML. L'application WASA n'aurait plus alors qu'à récupérer ce contenu et combiner cette information avec l'information recueillie dans les fichiers journaux du serveur web.

Cette solution a présenté de nombreux défauts, dont le principal est sans doute d'être éphémère face à l'évolution actuelle des technologies de programmation web.

L'étude de ces technologies et de leur évolution rapide nous a montré qu'il était préférable d'aborder le problème sous un nouvel angle. Au lieu de décomposer les couches des applications pour trouver et extraire le contenu sémantique, nous avons analysé la communication entre ces applications et les utilisateurs pour intercepter les données générées, sans nous soucier des technologies utilisées.

Le défi était dès lors de réussir à stocker les pages HTML générées à partir de pages dynamiques au moment où le serveur web les envoie au navigateur.

Deux solutions ont été envisagées dans ce sens. La première est d'écouter la communication entre le serveur et le navigateur au niveau de la couche réseau. La seconde est de rajouter une fonctionnalité au server web sous la forme d'un module d'extension.

Cette seconde solution a été retenue en raison de ses performances. Elle a été réalisée sous forme d'un module d'extension, baptisé *mod_trace_output*, pour le serveur web le plus répandu : Apache. Il résout entièrement le problème posé et possède également de nombreuses fonctionnalités utiles de manière générale. Il a été soumis et accepté comme nouveau projet sourceforge, avec plus de 2000 visiteurs et 150 downloads en 15 jours.

En ce qui concerne l'application WASA, les perspectives sont l'implémentation d'une interface entre l'application et les informations stockées par *mod_trace_output*. Une application de test de validation est en cours de développement, son but est de vérifier de manière systématique les informations recueillies par *mod_trace_output* par recoupement avec les fichiers journaux classiques du serveur web.

Les perspectives du module *mod_trace_output* sont très larges, il est impossible de prévoir de quelles façons il va être utilisé par la communauté des développeurs de modules Apache.

En guise de conclusion personnelle, ce travail m'a permis d'approfondir, tant théoriquement que pratiquement, les mécanismes de l'Internet et les technologies de pointe dans le domaine des applications web. Il m'a permis également d'acquérir une bonne connaissance dans divers autres domaines : le langage C, le système d'exploitation Linux, la base de données MySQL, le serveur Apache et son API.

# Bibliographie


[1] *Professional ASP XML.*
M. Baartse, R. Blair, L. Bolognese, D. Dalvi, S. Hahn, C. Haines, A. Homer, B. Kropog, B. Loesgen, S. Mohr, J. Slater, K. Williams, M. Zucca. Wrox Press, 2000.

[2] *ASP in a nutshell.*
A. Keyton Weissinger. O'Reilly & Associates, Second edition, 2000.

[3] *Professional JSP.*
K. Avedal, D. Ayers, T. Briggs, C. Burnham, A. Halberstadt, R. Haynes, P. Henderson, M. Holden, S. Li, D. Malks, T. Myers, A. Nakhimovsky, S. Osmont, G. Palmer, J. Timney, S. Tyagi, G. Van Damme, M. Wilcox, S. Wilkinson, S. Zeiger, J. Zukowski. Wrox Press, 2000.

[4] *Professional JSP, 2nd Edition.*
S. Brown, R. Burdick, J. Falkner, B. Galbraith, R. Johnson, L. Kim, C. Kochmer, T. Kristmundsson, S. Li, D. Malks, M. Nelson, G. Palmer, B. Sullivan, G. Taylor, J. Timney, S. Tyagi, G. Van Damme, S. Wilkinson. Wrox Press, 2001.

[5] *Professional Active Server Page 2.0.*
A. Federov, B. Francis, R. Harrison, A. Homer, S. Murphy, R. Smith, D. Sussman, S. Wood. Wrox Press, 1998.

[6] *Professional Java Server Programming.*
D. Ayers, H. Bergsten, M. Bogovitch, J. Diamond, M. Ferris, M. Fleury, A. Halberstadt, P. Houle, P. Mohseni, A. Patzer, R. Phillips, S. Li, K. Vedati, M. Wilcox, S. Zeiger. Wrox Press, 1999.

[7] *Professional Java Server Programming, J2EE Edition.*
S. Allamaraju, K. Avedal, R. Browett, J. Diamond, J. Griffin, M. Holdsen, A. Hoskinson, R. Johnson, T. Karsjens, L. Kim, A. Longshaw, T. Myers, A. Nakhimovsky, D O'Connor, S. Tyagi, G. Van Damme, G. van Huizen, M. Wilcox, S. Zeiger. Wrox Press, 2000.

[8] *Professional Java E-Commerce.*
S. Allamaraju, R. Ashri, C. Darby, R. Flenner, T. Karsjens, M. Kezner, A. Krotov, A. Linde, J. MacIntosh, J. McGovern, T. Mirchandani, B. Plaster, D. Reamy, Dr P. G. Sarang, D. Writz. Wrox Press, 2001.

[9] *Web Application Development with PHP 4.0.*
T. Ratschiller, T. Gerken. New Riders, 2000.

[10] *Compilateurs : Principes, techniques et outils.*
A. Aho, R. Sethi, J. Ullman. Dunod, 2000.

[11] *Description of the JavaCC Grammar File.*
http://www.webgain.com/products/java_cc/javaccgrm.html

[12] *The TokenManager MiniTutorial.*
http://www.webgain.com/products/java_cc/tokenmanager.html

[13] *HTML Tidy Library Project.*
http://tidy.sourceforge.net

[14] *JTidy : HTML Parser and Pretty-Printer in Java.*
http://lempinen.net/sami/jtidy

[15] *XML and Java.*
H. Maruyama, K. Tamura, N. Uramotot. Addison-Wesley, 1999.

[16] *Professional Java XML Programming.*
A. Nakhimovsky, T. Myers. Wrox Press, 1999.

[17] *Professional XML Databases.*
K. Willliams, M. Brundage, P. Dengler, J. Gabriel, A. Hoskinson, M. Kay, T. Maxwell M. Ochoa, J. Papa, M. Vanlane. Wrox Press, 2000.

[18] *Réseaux.*
A. Tanenbaum. Dunod / Prentice Hall, 3ème edition, 1999.

[19] *Réseaux TCP/IP.*
http://www.info.univ-angers.fr/pub/pn/poly/poly.html

[20] *Notes on Apache Web server 1.3 API.*
http://httpd.apache.org/dev/API.html

[21] *Writing Apache Module with Perl and C.*
L. Stein, D. MacEachern. O'Reilly & Associates, 1999.

[22] *Apache Server.*
R. Bowen, K. Coar. Campus Press 2000.

[23] *Apache Web server 1.3 API Dictionary.*
http://httpd.apache.org/dev/apidoc

[24] *Apache Web server 1.3 Documentation.*
http://httpd.apache.org/docs

[25] *Apache project.*
http://www.apache.org

# Annexes

## A. JspParser.jj

```
options {
      Java_UNICODE_ESCAPE = true;
}

PARSER_BEGIN(JspParser)

package parser;

import java.io.*;

public class JspParser
{

      public static void main(String args[]) throws Exception
      {
            JspParser parser;

            if (args.length == 2)
            {
                  System.out.println("JspParser:  Reading from file "
                        + args[0] + " . . .");

                  try {
                        parser = new JspParser(new
                              FileInputStream(args[0]));
                  } catch (FileNotFoundException e) {
                        System.out.println("JspParser:  File " +
                              args[0] + " not found.");
                        return;
                  }
            }
            else
            {
                  System.out.println("JspParser:  Usage is \"java
                              JspParser inputfile outputfile\"");
                  return;
            }

            try {
                  PrintWriter ostr = new PrintWriter(new
                                    FileWriter(args[1]));
                  parser.Input(ostr);
                  ostr.close();
                  System.out.println("JspParser:  Transformation
                                    completed successfully.");
            } catch (ParseException e) {
                  System.out.println("JspParser:  Encountered errors
                                    during parse : " + e.toString());
            } catch (IOException e) {
                  System.out.println("JspParser:  Could not create
                                    file " + args[1]);
            }
      }
}
```

```
PARSER_END(JspParser)

TOKEN :
{
      <ELETEMENT : ["\u0000"-"\uffff"]>
}

SKIP :
{
      <"<%" (~["%"])* "%" (~[">"] (~["%"])* "%")* ">">
}

void Input(PrintWriter pw) :
{}
{
      (Element(pw))* <EOF>
}

void Element(PrintWriter pw) :
{
      Token t;
}
{
      t = <ELETEMENT>
      {
            pw.print(t.image);
      }
}
```

## B. ExtractJSP.java

```
import java.io.*;

import org.w3c.dom.*;
import org.w3c.tidy.Tidy;

import parser.*;

public class ExtractJSP
{

      private static PrintWriter out;

      public ExtractJSP()
      {
      }

      public static void main( String[] args )
      {
            try {
                  ExtractJSP exjsp = new ExtractJSP();
                  exjsp.outText(args[0], args[1]);
            } catch ( Exception e ) {
                  System.out.println( e.toString() );
            }
      }

      public static void outText(String inFileName, String
                              outFileName) throws Exception
```

```
{
      JspParser comp;
      System.out.println("JspParser:  Reading from file " +
                        inFileName + " . . .");
      try {
            comp = new JspParser(new
                        FileInputStream(inFileName));
      } catch (FileNotFoundException e) {
            System.out.println("JspParser:  File " + inFileName
                              + " not found.");
            return;
      }

      try {
            PrintWriter ostr = new PrintWriter(new
                              FileWriter("temp.html"));
            comp.Input(ostr);
            ostr.close();
            System.out.println("JspParser:  Extracting HTML to
                        \"temp.html\".");
      } catch (ParseException e) {
            System.out.println("JspParser:  Encountered errors
                        during parse : " + e.toString());
      } catch (IOException e) {
            System.out.println("JspParser:  Could not create
                        file temp.html");
      }

      Tidy tidy = new Tidy();

      try {
            tidy.setErrout(new PrintWriter(new
                        FileWriter("errors.txt"), true));
            tidy.setXHTML(true);

            System.out.println("JTidy:  creating
                        \"temp.xhtml\".");
            tidy.parse(new FileInputStream("temp.html"), new
                        FileOutputStream("temp.xhtml"));
            tidy.setXHTML(false);
            tidy.setXmlOut(true);
            System.out.println("JTidy:  creating
                        \"temp.xml\".");
            tidy.parse(new FileInputStream("temp.html"), new
                        FileOutputStream("temp.xml"));
      } catch ( IOException e ) {
            System.out.println("JTidy:  error creating
                        \"temp.xhtml\" and \"temp.xml\".");
            System.out.println( e.toString() );
      }


      try {
            out = new PrintWriter(new
                        FileOutputStream(outFileName));
      } catch (FileNotFoundException e) {
            System.out.println("JTidy:  Could not create file
                        "+outFileName);
            return;
      }
```

```
            try {
                  tidy.setMakeClean(true);
                  tidy.setXmlTags(true);
                  System.out.println("JTidy:  Exctracting text to
                              "+outFileName);
                  print(tidy.parseDOM(new
                              FileInputStream("temp.html"), null));
                  out.close();
            } catch ( IOException e ) {
                  System.out.println("JTidy:  error );
                  System.err.println( e.toString() );
            }
            System.out.println("JspParser & JTidy:  Exctraction
                              completed.");
      }

      public static void print(Node node)
      {
            if ( node == null )
            {
                  return;
            }
            int type = node.getNodeType();
            switch ( type )
            {
                  case Node.DOCUMENT_NODE:
                        print(((Document)node).getDocumentElement());
                        break;

                  case Node.ELEMENT_NODE:
                        NodeList children = node.getChildNodes();
                        if ( children != null )
                        {
                              int len = children.getLength();
                              for ( int i = 0; i < len; i++ )
                              {
                                    print(children.item(i));
                              }
                        }
                        break;
                  case Node.TEXT_NODE:
                        out.print(node.getNodeValue());
                        break;
            }
            out.flush();
      }
}
```

## C. mod_trace_output.c

```
/*************************/
/** Version : 1.1-beta  **/
/** Update : 09/05/2002 **/
/*************************/

#include "httpd.h"
#include "HTTP_config.h"
#include "HTTP_core.h"
#include "HTTP_log.h"
#include "HTTP_main.h"
```

```
#include "HTTP_protocol.h"
#include "util_script.h"
#include "multithread.h"
#include "ap_compat.h"

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>

#include "mod_trace_output.h"

#define TO_VERSION "1.1-Beta"

#define OFF (0)
#define ON (1)
#define isOff(a) ((a == ON) ? 0 : 1 )
#define isOn(a) ((a == ON) ? 1 : 0 )
#define WATCHPOINT fprintf(log_file,"WATCHPOINT %s %d\n", __FILE__,
                           __LINE__);fflush(log_file);

module MODULE_VAR_EXPORT mod_trace_output_module;


static void to_init(server_rec *s, pool *p);
static int to_handler(request_rec *r);
static void *to_create_dir_config(pool *p, char *path);

extern void safe_mysql_query(request_rec *r, char *timestamp, char
             *data, int data_len, char *data_group, int is_gzipped);
extern const char *set_mysql_info(const char *db, char *user, char
             *pwd);
extern const char *set_mysql_host(const char *host);
extern const char *set_mysql_port(const char *port);
extern const char *set_mysql_socket(const char *socket);

extern char *gzipcompress(const char *source, long length, long
             *dest_length, request_rec *r);
extern void *output_filter(BUFF *client, const void *voidptr, int i);

static int log=0;
static FILE *log_file;

static int unid;

static void to_init(server_rec *s, pool *p)
{
      unid = 1;
      return;
}

static const handler_rec to_handlers[] =
{
      {"trace-output", to_handler},
      {NULL}
};

int to_handler(request_rec *r)
{
      FILE                    *output_file;
      char                   devnull[] = "/dev/null";
```

```
char                    *dataptr, *filename, *timestamp,
                        *timefile, *encoding, *mime_type;
int                     rc,len,fd,clifd,cliflags,gzipped;
struct tm               *newtime;
time_t                  long_time;
request_rec             *subr;
dir_config              *cfg;
cb_data                 *cb;
long                    datalen;

cfg = (dir_config *) ap_get_module_config(r->per_dir_config,
                                    &mod_trace_output_module);

cb = (cb_data *) ap_palloc(r->pool, sizeof(cb_data));
cb->data="";
cb->bytes = 0;

if(r->main)
{
      ap_table_set(r->main->notes, "TRACE_OUTPUT_SUBR", "yes");
}
else
{
      ap_table_set(r->notes, "TRACE_OUTPUT_SUBR", "yes");
}

subr = (request_rec*) ap_sub_req_lookup_uri(r->unparsed_uri,r);

//Backup fd and flags witch can be modified by subrequest
clifd = r->connection->client->fd;
cliflags = r->connection->client->flags;

r->connection->client->filter_callback = (void *)output_filter;
r->connection->client->callback_data = (cb_data *) cb;

/* Sending subrequest response to /dev/null, so that we send
*  the data ourself later.
*/
if(!(fd = ap_popenf(r->pool,devnull, O_RDWR, S_IRWXU)))
{
      ap_log_error(APLOG_MARK, APLOG_ERR, r->server , "Cannot
                  serve :%s\n", r->uri);
      return DECLINED;
}

subr->connection->client->fd = fd;

//Run subrequest
rc = ap_run_sub_req(subr);
ap_bflush(subr->connection->client);
ap_pclosef(r->pool,fd);

r->connection->client->filter_callback = NULL;

// Restore fd and flags
r->connection->client->fd = clifd;
r->connection->client->flags = cliflags;

if(rc != 0)
{
```

```
      fprintf(stderr,"[TRACE OUTPUT Error] Error generating
             page : %s", r->uri);
      return rc;
}

//Serve the page
r->content_type = subr->content_type;

encoding = (char *)ap_table_get(r->headers_in,
                                "Accept-Encoding");

r->headers_out = subr->headers_out;
r->err_headers_out = subr->err_headers_out;

//Check if client accept encoding
gzipped = 0;
if(strstr((encoding==NULL?"":encoding), "gzip"))
{
      gzipped = 1;
      ap_table_set(r->headers_out, "Content-Encoding", "gzip");
}

//If there is a need for compression, compress it!
if(gzipped || isOn(cfg->gzip_in_files) ||
   isOn(cfg->gzip_in_mysql))
{
      dataptr = gzipcompress(cb->data,cb->bytes,&datalen,r);
}

//If client accepts gzip, send it compressed
if(gzipped)
{
      ap_set_content_length(r, datalen);
      ap_send_HTTP_header(r);
      ap_rwrite(dataptr, datalen, r);
}
else
{
      ap_set_content_length(r, cb->bytes);
      ap_send_HTTP_header(r);
      ap_rwrite(cb->data, cb->bytes, r);
}

//Not gonna store for mime types not in mime_types_out
mime_type = ap_pstrdup(r->pool,subr->content_type);
strtok(mime_type,";");
if(!strstr(cfg->mime_types_out, mime_type))
      return OK;

//Store the necessary info and do log
time( &long_time );
newtime = localtime( &long_time );
timestamp = ap_palloc(r->pool, sizeof(char) * 20);
strftime(timestamp,20,"%Y-%m-%d %H:%M:%S",newtime);

if (isOn(cfg->store_in_files))
{
      timefile = ap_palloc(r->pool, sizeof(char) * 15);
      strftime(timefile,15,"%Y%m%d%H%M%S",newtime);

      filename = ap_psprintf(r->pool,"%s/%s_%d_%d.html%s",
```

```
                        cfg->dir,timefile,r->connection->child_num,
                        (unid++), (cfg->gzip_in_files?".gz":""));

            if (!(output_file = ap_pfopen(r->pool,filename, "a")))
                  ap_log_error(APLOG_MARK, APLOG_ERR, r->server ,
                              "Cannot open %s\n", filename);

            if(isOn(cfg->gzip_in_files))
            {
                  fwrite(dataptr,sizeof(char),datalen,output_file);
            }
            else
            {
                  fwrite(cb->data,sizeof(char),
                         cb->bytes,output_file);
            }

            ap_pfclose(r->pool,output_file);

            if(isOn(log) && log_file)
            {
                  fprintf(log_file,"[TRACE OUTPUT store file]: %s %s
                         %s\n",r->unparsed_uri,filename,timestamp);
                  fflush(log_file);
            }
      }

      if (isOn(cfg->store_in_mysql))
      {

            if(isOn(cfg->gzip_in_mysql))
            {
                  safe_mysql_query(r, timestamp, dataptr, datalen,
                            cfg->data_group,cfg->gzip_in_mysql);
            }
            else
            {
                  safe_mysql_query(r, timestamp, cb->data,
                            cb->bytes, cfg->data_group,
                            cfg->gzip_in_mysql);
            }

            if(isOn(log) && log_file)
            {
                  fprintf(log_file,"[TRACE OUTPUT store in mysql]: %s
                         %s %s\n", r->unparsed_uri,
                         cfg->data_group,timestamp);
                  fflush(log_file);
            }
      }

      return OK;
}

static  void *to_create_dir_config(pool *p, char *path)
{
      dir_config *cfg = (dir_config *) ap_pcalloc(p,
                        sizeof(dir_config));

      cfg->mime_types_in = ap_pstrdup(p,"");
      cfg->mime_types_out = ap_pstrdup(p,"text/html");
```

```
      cfg->data_group = path;
      cfg->store_in_mysql = OFF;
      cfg->gzip_in_mysql = ON;

      cfg->dir = ap_pstrdup(p,"/tmp");
      cfg->store_in_files = OFF;
      cfg->gzip_in_files = ON;

      return (void *) cfg;
}

static const char *mysql_info(cmd_parms * parms, void *dummy,
                              char *db, char *user, char *pwd)
{
      return set_mysql_info(db, user, pwd);
}

static const char *mysql_host(cmd_parms * parms, void *dummy,
                              char *host)
{
      return set_mysql_host(host);
}

static const char *mysql_port(cmd_parms * parms, void *dummy,
                              char *port)
{
        return set_mysql_port(port);
}

static const char *mysql_socket(cmd_parms * parms, void *dummy,
                              char *socket)
{
      return set_mysql_socket(socket);
}

static  const char *set_mime_types_in(cmd_parms *cmd, void *mconfig,
                                      char *mime_types)
{
        dir_config *cfg = (dir_config *) mconfig;
        cfg->mime_types_in = mime_types;
        return NULL;
}

static  const char *set_mime_types_out(cmd_parms *cmd, void *mconfig,
                                       char *mime_types)
{
        dir_config *cfg = (dir_config *) mconfig;
        cfg->mime_types_out = mime_types;
        return NULL;
}

static  const char *set_data_group(cmd_parms *cmd, void *mconfig,
                                       char *data_group)
{
        dir_config *cfg = (dir_config *) mconfig;
        cfg->data_group = data_group;
        return NULL;
}
```

```
static  const char *set_store_in_mysql(cmd_parms *cmd, void *mconfig,
                                       char *flag)
{
        dir_config *cfg = (dir_config *) mconfig;
        if(ap_strcmp_match(flag, "ON") == 0)
                cfg->store_in_mysql = ON;
        else
                cfg->store_in_mysql = OFF;
        return NULL;
}

static  const char *set_gzip_in_mysql(cmd_parms *cmd, void *mconfig,
                                       char *flag)
{
        dir_config *cfg = (dir_config *) mconfig;
        if(ap_strcmp_match(flag, "ON") == 0)
                cfg->gzip_in_mysql = ON;
        else
                cfg->gzip_in_mysql = OFF;
        return NULL;
}

static  const char *set_store_in_files(cmd_parms *cmd, void *mconfig,
                                       char *flag)
{
        dir_config *cfg = (dir_config *) mconfig;
        if(ap_strcmp_match(flag, "ON") == 0)
                cfg->store_in_files = ON;
        else
                cfg->store_in_files = OFF;
        return NULL;
}

static  const char *set_gzip_in_files(cmd_parms *cmd, void *mconfig,
                                       char *flag)
{
        dir_config *cfg = (dir_config *) mconfig;
        if(ap_strcmp_match(flag, "ON") == 0)
                cfg->gzip_in_files = ON;
        else
                cfg->gzip_in_files = OFF;
        return NULL;
}

static  const char *set_dir(cmd_parms *cmd, void *mconfig, char *dir)
{
      dir_config *cfg = (dir_config *) mconfig;
      char *str;

      str = dir + strlen(dir) - 1;

      if(strcmp(str,"/"))
      {
            cfg->dir = dir;
      }
      else
      {
            bzero(str,1);
            cfg->dir = dir;
      }
      return NULL;
```

```
}

static  const char *set_log(cmd_parms *cmd, void *dummy, char *flag)
{
      if(ap_strcmp_match(flag, "ON") == 0)
            log = ON;
      else
            log = OFF;
      return NULL;
}

static  const char *set_log_file(cmd_parms *cmd, void *dummy,
                                 char *logfilename)
{
      if ((log_file = fopen(logfilename, "a")) == NULL)
      {
            fprintf(stderr,"[TRACE OUTPUT Error]: Cannot open trace
                  output log file : %s\n", logfilename);
      }
      return NULL;
}

static int to_fixup(request_rec *r)
{
      dir_config *cfg;

      if(r->main)
      {
            if(ap_table_get(r->main->notes, "TRACE_OUTPUT_SUBR"))
                  return DECLINED;
      }

      if(r->header_only)
            return DECLINED;

      if(ap_is_directory(r->filename))
            return DECLINED;

      cfg = (dir_config *) ap_get_module_config(r->per_dir_config,
            &mod_trace_output_module);

      //Not going to do anything if no need
      if((isOff(cfg->store_in_files) && isOff(cfg->store_in_mysql))
            || !(strcmp(cfg->mime_types_in,"")))
            return DECLINED;

      //Not going to do anything for types not in mime_types_in
      if(r->handler)
      {
            if(!strstr(cfg->mime_types_in, r->handler) &&
                  !strstr(cfg->mime_types_in, "HANDLER"))
                  return DECLINED;
      }
      else
      {
            if(!strstr(cfg->mime_types_in, r->content_type))
                  return DECLINED;
      }

      r->handler = "trace-output";
```

```
        return DECLINED;
}

static command_rec to_cmds[] = {
        {"ToMySQLInfo", mysql_info, NULL, RSRC_CONF, TAKE3, "Db, user
                        and password of the MySQL database." },
        {"ToMySQLHost", mysql_host, NULL, RSRC_CONF, TAKE1, "Host of
                        the MySQL database." },
        {"ToMySQLPort", mysql_port, NULL, RSRC_CONF, TAKE1, "TCP-IP
                        port to connect to the MySQL database." },
        {"ToMySQLSocket", mysql_socket, NULL, RSRC_CONF, TAKE1, "Path
                        to socket (ex:\"/var/llib/mysql/mysql.sock\")
                        to connect to the MySQL database." },
        {"ToMimeTypesIn", set_mime_types_in, NULL, ACCESS_CONF,
                        TAKE1, "Mime types of files requested that
                        mod_trace_output will handle (seperated by
                        comas)." },
        {"ToMimeTypesOut", set_mime_types_out, NULL, ACCESS_CONF,
                        TAKE1, "Mime types of files that
                        mod_trace_output will store {"ToDataGroup",
                        set_data_group, NULL, ACCESS_CONF, TAKE1,
                        "Used to group data in the db by directory
                        (default = directory path)." },
        {"ToStoreInMySQL", set_store_in_mysql,NULL, ACCESS_CONF,
                        TAKE1, "Flag to allow / disallow storing of
                        the pages in MySQL." },
        {"ToGzipInMySQL", set_gzip_in_mysql,NULL, ACCESS_CONF, TAKE1,
                        "Flag to allow / disallow to gzip compress
                        data in MySQL." },
        {"ToStoreInFiles", set_store_in_files,NULL, ACCESS_CONF,
                        TAKE1, "Flag to allow / disallow storing of
                        the pages in a specified directory." },
        {"ToGzipInFiles", set_gzip_in_files,NULL, ACCESS_CONF, TAKE1,
                        "Flag to allow / disallow to gzip compress
                        data in files." },
        {"ToDir", set_dir, NULL, ACCESS_CONF, TAKE1, "Directory where
                        to store pages (default /tmp)." },
        {"ToLogFile", set_log_file, NULL, RSRC_CONF, TAKE1, "File
                        where to log actions." },
        {"ToLog", set_log,NULL, RSRC_CONF, TAKE1, "Flag to allow /
                        disallow the log." },
        { NULL }
};

module MODULE_VAR_EXPORT mod_trace_output_module=
{
      STANDARD_MODULE_STUFF,
      to_init,              /* initialize */
      to_create_dir_config, /* per-directory configuration creator */
      NULL,
      NULL,
      NULL,
      to_cmds,              /* command handlers */
      to_handlers,          /* handlers */
      NULL,
      NULL,
      NULL,
      NULL,
      NULL,
      to_fixup,             /* fixup */
      NULL,
```

```
      NULL,
      NULL,
      NULL,
      NULL
};
```

## D. mod_trace_output.h

```
/************************/
/** Version : 1.1-beta  **/
/** Update : 30/04/2002 **/
/************************/

typedef struct
{
      char    *data;
      int     bytes;
} cb_data;

typedef struct
{
      char  *mime_types_in;
      char  *mime_types_out;

      char  *data_group;
      int   store_in_mysql;
      int   gzip_in_mysql;

      char  *dir;
      int   store_in_files;
      int   gzip_in_files;
}dir_config;
```

## E. filter.c

```
/************************/
/** Version : 1.1-beta  **/
/** Update : 30/04/2002 **/
/************************/

#include "httpd.h"
#include "HTTP_core.h"
#include "HTTP_main.h"

#include "mod_trace_output.h"

void *output_filter(BUFF *client, const void *voidptr, int i);

void *output_filter(BUFF *client, const void *voidptr, int i)
{
      char    *dataptr;
      char    *srcptr;
      int     *ret;
      cb_data *cb;

      srcptr = (char *) voidptr;

      cb = (cb_data *) client->callback_data;
```

```
      dataptr = ap_palloc(client->pool,
                          sizeof(char) * (cb->bytes + i + 1));

      memcpy(dataptr,cb->data,cb->bytes);

      cb->data = dataptr;

      dataptr += cb->bytes;
      memcpy(dataptr,srcptr,i);

      dataptr += i;
      dataptr = '\0';

      cb->bytes +=i;

      return;
}
```

## F. mysql.c

```
/***********************/
/** Version : 1.1-beta  **/
/** Update : 09/05/2002 **/
/*************************/

#include "httpd.h"
#include "HTTP_config.h"
#include "HTTP_core.h"
#include "HTTP_log.h"
#include "HTTP_main.h"
#include "HTTP_protocol.h"
#include "util_script.h"
#include "multithread.h"
#include "ap_compat.h"

#include "mysql.h"

#include "mod_trace_output.h"

static  MYSQL   mysql_server, *mysql_db = NULL;
static  char    *mysql_db_host = NULL, *mysql_db_name = NULL,
                *mysql_db_user = NULL, *mysql_db_pwd = NULL,
                *mysql_db_socket = NULL;
static  int     mysql_db_port=0;

void safe_mysql_query(request_rec *r, char *timestamp, char *data,
                      sint data_len, char *data_group, int
                      is_gzipped);
const char *set_mysql_info(char *db, char *user, char *pwd);
const char *set_mysql_host(char *host);
const char *set_mysql_port(char *port);
const char *set_mysql_socket(char *socket);

static void open_dblink()
{
      if (mysql_db_name != NULL)
      {
            mysql_init(&mysql_server);
```

```
            mysql_db = mysql_real_connect(&mysql_server,
                        mysql_db_host, mysql_db_user, mysql_db_pwd,
                        mysql_db_name, mysql_db_port,
                        mysql_db_socket,0);
      }
}

void safe_mysql_query(request_rec *r, char *timestamp, char *data,
                        int data_len,char *data_group, int is_gzipped)
{
      int error = 1, gone = 0;
      char *str,*esc_data,*query,*str_data_len;
      char *esc_data_group, *esc_uri;
      int was_connected = 0, data_group_len = 0, uri_len = 0,
          query_len = 0, esc_data_len = 0, str_data_len_len = 0;

      void (*sigpipe_handler)();

      sigpipe_handler = signal(SIGPIPE, SIG_IGN);

      if (mysql_db)
      {
            mysql_select_db(mysql_db, mysql_db_name);
      }
      else
      {
            open_dblink();

            if (mysql_db == NULL) // unable to link
            {
                  signal(SIGPIPE, sigpipe_handler);
                  str = ap_pstrcat(r->pool, "[TRACE OUTPUT Error]:
                        connect failed:  ",
                        mysql_error(&mysql_server), NULL);
                  fprintf(stderr, str);
                  return;
            }
      }

      esc_data_group = ap_palloc(r->pool,
                        sizeof(char) * (strlen(data_group) * 2 + 1));
      data_group_len = mysql_real_escape_string(mysql_db,
                        esc_data_group, data_group,
                        strlen(data_group));

      esc_uri = ap_palloc(r->pool,
                        sizeof(char) * (strlen(r->uri) * 2 + 1));
      uri_len = mysql_real_escape_string(mysql_db, esc_uri, r->uri,
                        strlen(r->uri));

      str_data_len = ap_psprintf(r->pool,"%d",data_len);
      str_data_len_len = strlen(str_data_len);

      if(is_gzipped)
      {
            esc_data = ap_palloc(r->pool,
                        sizeof(char) * (data_len * 2 + 1));
            esc_data_len = mysql_real_escape_string(mysql_db,
                        esc_data, data, data_len);
            query_len = 26 + 19 + 3 + data_group_len + 3 + uri_len +
                        3 + esc_data_len + 2 + str_data_len_len + 3;
```

```
      str = ap_palloc(r->pool, sizeof(char) * (query_len+1));
      query = str;
      memcpy(str,"INSERT INTO data VALUES ('",26);
      str += 26;
      memcpy(str,timestamp,19);
      str += 19;
      memcpy(str,"','",3);
      str += 3;
      memcpy(str,esc_data_group,data_group_len);
      str += data_group_len;
      memcpy(str,"','",3);
      str += 3;
      memcpy(str,esc_uri,uri_len);
      str += uri_len;
      memcpy(str,"','",3);
      str += 3;
      memcpy(str,esc_data,esc_data_len);
      str += esc_data_len;
      memcpy(str,"',",2);
      str += 2;
      memcpy(str,str_data_len,str_data_len_len);
      str += str_data_len_len;
      memcpy(str,",1)",3);
      str += 3 + 1;
      str = '\0';
}
else
{
      esc_data = ap_palloc(r->pool,
                  sizeof(char) * (data_len * 2 + 1));
      mysql_real_escape_string(mysql_db, esc_data,
                  data,data_len);
      query = ap_pstrcat(r->pool, "INSERT INTO data VALUES ('",
                  timestamp, "','", esc_data_group, "','",
                  esc_uri, "','", esc_data, "',", str_data_len,
                  ",0);", NULL);
      query_len = strlen(query);
}

error = mysql_real_query(mysql_db, query, query_len);

if (error)
      gone = !strcasecmp(mysql_error(mysql_db), "mysql server
                        has gone away");

if (gone)
{
      // we need to restart the server link
      mysql_db = NULL;

      open_dblink();

      // unable to link
      if (mysql_db == NULL)
      {
            signal(SIGPIPE, sigpipe_handler);
            str = ap_pstrcat(r->pool, "[TRACE OUTPUT Error]:
                  MySQL connect failed:  ",
                  mysql_error(&mysql_server), NULL);
            fprintf(stderr, str);
            return;
```

```
            }

            error = mysql_select_db(mysql_db, mysql_db_name) ||
                        mysql_real_query(mysql_db, query, query_len);
      }

      signal(SIGPIPE, sigpipe_handler);

      if (error)
      {
            str = ap_pstrcat(r->pool, "[TRACE OUTPUT Error]: MySQL
                  query failed:  ", query, NULL);
            fprintf(stderr, str);
            str = ap_pstrcat(r->pool, "[TRACE OUTPUT Error]: MySQL
                  failure reason:  ", mysql_error(mysql_db), NULL);
            fprintf(stderr, str);
      }
      return;
}

const char *set_mysql_info(char *db, char *user, char *pwd)
{
      mysql_db_name = db;
      mysql_db_user = user;
      mysql_db_pwd = pwd;
      return NULL;
}

const char *set_mysql_host(char *host)
{
      mysql_db_host = host;
      return NULL;
}

const char *set_mysql_port(char *port)
{
        mysql_db_port = atoi(port);
        return NULL;
}

const char *set_mysql_socket(char *socket)
{
      mysql_db_socket = socket;
      return NULL;
}
```

## G. gzipcomp.c

```
/* this implementation was taken from mod_mmap_dynamic release*/

#include <zlib.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "httpd.h"
#include "HTTP_config.h"
#include "HTTP_log.h"
#include "HTTP_protocol.h"
#include "HTTP_request.h"
```

```
#include "HTTP_core.h"
/* Added the apache header files to allocate memory */

char    *gzipcompress(const char *source, long length, long
                      *dest_length, request_rec *r);

/* This function is a little bit of a hack. Basically zlib returns an
 * two byte header and an Adler32 checksum when using in memory
 * compression. The header and checksum are removed and a gzip header
 * and checksum are put on the result. Enough memory is allocated to
 * allow this and memory allocated in case the file is already
 * compressed. This is done through the Apache API so it is not freed
 */

static char gz_magic[2] = {0x1f, 0x8b}; /* gzip magic header - from
gzio.c in zlib-1.1.3 */
static unsigned char    CM = 8;
static unsigned char    FLAGS = 0;
static unsigned int     MTIME = 0;
static unsigned char    XFL = 2;
static unsigned char    OS = 3;

char *gzipcompress(const char *source, long length, long
                      *dest_length, request_rec *r)
{
      char *blah;
      char *blib;
      unsigned long crc;
      int   rc;
      int   gzip_padding;

      gzip_padding = 12 + (length * 1.2);
      //blah = malloc(sizeof(char) * gzip_padding);
      //bzero(blah, sizeof(char) * gzip_padding);
      blah = ap_palloc(r->pool, sizeof(char) * gzip_padding);
      bzero(blah, sizeof(char) * gzip_padding);
      blib = blah;
      *dest_length = gzip_padding;

      crc = crc32(0L, Z_NULL, 0);
      crc = crc32(crc, source, length);
      blah = memcpy(blah, gz_magic, 2);
      blah += 2;
      memcpy(blah, &CM, 1);
      blah++;
      memcpy(blah, &FLAGS, 1);
      blah++;
      memcpy(blah, &MTIME, 4);
      blah += 4;
      rc = compress2(blah, dest_length, source, length, 9);
      if(rc != Z_OK)
            return NULL;
      blah +=  (*dest_length - 4);
      memcpy(blah, &crc, 4);
      blah += 4;
      memcpy(blah, &length, 4);
      blah = blib;
      blah += 8;
      memcpy(blah, &XFL, 1);
      blah++;
      memcpy(blah, &OS, 1);
```

```
      blah++;
      *dest_length += 12;
      return blib;
}
```

## H. Installation

In previous versions, mod_trace_output had no handler for the fixup phase, so the module needed to be executed first, but this is not a requirement anymore.

I'll give the installation steps if you compile mod_trace_output statically with Apache. But you can also use DSO and apxs to compile it out of Apache.

Here we go. Create a directory mod_trace_output in the modules directory of the source distribution of apache, copy all the files provided in the mod_trace_output release (mod_trace_output.c, mod_trace_output.h, filter.c, gzipcomp.c, mysql.c and MakeFile.tmpl), then add the following line in the configuration file.

AddModule modules/mod_trace_output/mod_trace_output.o
It uses libmysqlclient (.a or .so) so you need to set extra options in the Configuration file.

Add -L/path/to/libmysqlclient/folder -lz -lmysqlclient (if you have libmysqlclient.so) or -L/path/to/libmysqlclient/folder/libmysqlclient.a (if you have libmysqlclient.a) to EXTRA_LDFLAGS.

Add -I/path/to/mysql/include to EXTRA_INCLUDES.

Then compile apache as usual. See HTTP://httpd.apache.org/ for more information on how to do it.

## I. Makefile.tmpl

```
OBJS = mod_trace_output.o gzipcomp.o filter.o mysql.o
#Dependencies

$(OBJS) $(OBJS_PIC): Makefile

# DO NOT REMOVE
mod_trace_output.o: mod_trace_output.c gzipcomp.o filter.o mysql.o\
 $(INCDIR)/httpd.h \
 $(INCDIR)/ap_config.h $(INCDIR)/ap_mmn.h \
 $(INCDIR)/ap_config_auto.h $(OSDIR)/os.h \
 $(INCDIR)/ap_ctype.h $(INCDIR)/hsregex.h \
 $(INCDIR)/ap_alloc.h $(INCDIR)/buff.h $(INCDIR)/ap.h \
 $(INCDIR)/util_uri.h $(INCDIR)/HTTP_config.h \
 $(INCDIR)/HTTP_core.h $(INCDIR)/HTTP_log.h \
 $(INCDIR)/HTTP_main.h $(INCDIR)/HTTP_protocol.h \
 $(INCDIR)/util_script.h
```

# J. Apache Software License

Apache Software License

Version 1.1



Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment:

"This product includes software developed by the Apache Software Foundation (HTTP://www.apache.org/)."

Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

4. The names "Apache" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.