Chapitre 2

Programmation Web

Sommaire

Ce premier chapitre propose une présentation générale des techniques de programmation du Web et présente, non exhaustivement, son principal langage, HTML, et le mécanisme de base pour produire des sites interactifs, le CGI.

2.1 Le Web

Le *World-Wide Web* (ou WWW, ou Web) est un grand – très grand – système d'information réparti sur un ensemble de *sites* connectés par le réseau Internet. Ce système est essentiellement constitué de *documents hypertextes*, ce terme pouvant être pris au sens large : textes, images, sons, vidéos, etc. Chaque site propose un ensemble plus ou moins important de documents qui sont transmis sur le réseau par l'intermédiaire d'un *programme serveur*. Ce programme serveur dialogue avec un *programme client* qui peut être situé n'importe où sur le réseau. Le programme client prend le plus souvent la forme d'un *navigateur*, grâce auquel un utilisateur du Web peut demander et de consulter très simplement des documents.

Le dialogue entre un programme serveur et un programme client s'effectue selon des règles précises qui constituent un *protocole*. Le protocole du Web est HTTP, mais il est souvent possible de communiquer avec un site via d'autres protocoles, comme par exemple FTP qui permet d'échanger des fichiers. Ce livre ne rentre pas dans le détail des protocoles, même si nous aurons occasionnellement à évoquer certains aspects de HTTP.

Nous revenons brièvement sur chacun de ces concepts ci-dessous.

2.1.1 Serveurs web

Un site est constitué, matériellement, d'un ordinateur connecté à Internet, et d'un programme tournant en permanence sur cet ordinateur, le *serveur*. Le programme serveur est en attente de requêtes transmises à son attention sur le réseau par un programme client. Quand une requête est reçue, le programme serveur l'analyse afin de déterminer quel est le document demandé, recherche ce document et le transmet au programme client.

Comme nous le verrons dans la section 2.3, ce schéma général admet beaucoup de variantes. Par exemple, au lieu de demander un document, le programme client peut lui-même transmettre des informations et demander au programme serveur de les conserver. Dans ce cas le message renvoyé est simplement destiné à informer le programme client que sa demande a été (ou non) satisfaite. Un autre type, important, d'interaction consiste pour le programme client à demander au programme serveur d'exécuter un programme, en fonction de paramètres, et de lui transmettre le résultat.

La figure 2.1 illustre les aspects essentiels. La communication s'effectue entre deux programmes. La requête envoyée par le programme client est reçue par le programme serveur. Ce programme se charge de rechercher le document demandé parmi l'ensemble des fichiers auxquels il a accès, et transmet ce document.

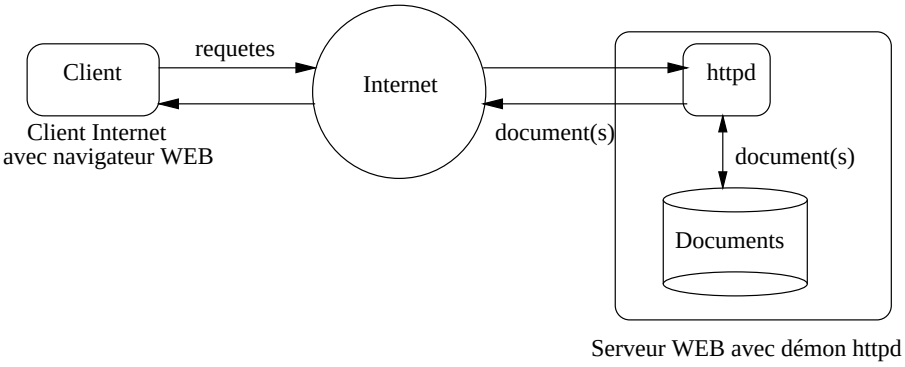

FIG. 2.1 – *Architecture Web*

Dans tout ce qui suit, le programme serveur sera simplement désigné par le terme « serveur » ou par le nom du programme particulier que nous utilisons, Apache. Les termes « navigateur » et « client » désignerons tous deux le programme client. Enfin le terme « utilisateur » (ou, parfois, « internaute »), sera réservé à la personne physique qui utilise un programme client.

Documents web

Les documents échangés sur le Web peuvent être de types très divers. De fait, afin de ne pas entretenir de confusion, la terminologie a récemment changé et on utilise le terme plus général de *ressource* pour désigner les informations disponibles sur le Web. Cela dit le principal type de ressource est le *document hypertexte*, un texte dans lequel certains mots, ou groupes de mots, sont des *liens*, ou *ancres*, donnant accès à d'autres documents. Le langage qui permet de spécifier des documents hypertextes, et donc de fait le principal langage du Web, est HTML, qui sera décrit plus loin.

À titre d'exemple, voici un court (hyper)texte dans lequel les expressions en italiques sont des liens :

Ce livre, édité aux Editions *O'Reilly*, est consacré aux techniques de création de sites à l'aide du *langage PHP* et du serveur *MySQL*

Comme on le voit, un tel texte peut être lu, classiquement, en séquence. On peut aussi le lire « en profondeur » en suivant, par exemple, le lien *Editions O'Reilly* qui donne probablement accès à la page

d'accueil du site web des Editions O'Reilly. Cette page d'accueil comprendra elle-même de nombreux liens qui permettront à d'autres documents, et ainsi de suite.

Le Web peut être vu comme un immense, et très complexe, graphe de documents reliés par des liens hypertextes. Les liens présents dans un document peuvent donner accès non seulement à d'autres documents du même site, mais également à des documents gérés par d'autres sites, n'importe où sur le réseau.

Un des principaux mécanismes du Web est le principe de localisation, dit *Universal Resource Location* (URL), qui permet de faire référence de manière unique à un document. Une URL est constituée de trois parties :

- le nom du protocole utilisé pour accéder à la ressource ;
- le nom du serveur hébergeant la ressource ;
- le numéro du port réseau sur lequel le serveur est à l'écoute ;
- le chemin d'accès, sur la machine serveur, à la ressource.

À titre d'exemple, voici l'URL de la page web de ce livre chez l'éditeur :

http://www.editions-oreilly.fr/mysqlphp.html

Cette URL s'interprête de la manière suivante : il s'agit d'un document accessible *via* le protocole HTTP, sur le serveur *www.editions-oreilly.fr* qui est l'écoute sur le port 80 – numéro par défaut, donc non précisé dans l'URL – et dont le nom est *mysqlphp.html*.

À chaque lien dans un document HTML est associée une URL qui donne la localisation de la ressource. Les navigateurs permettent à l'utilisateur de suivre à un lien par simple clic souris, et se chargent de récupérer le document correspondant grâce à l'URL. Ce mécanisme rend transparent, dans la plupart des cas, les adresses des documents web pour les utilisateurs.

2.1.2 Clients web

Le Web est donc un ensemble de serveurs connectés à Internet et proposant des ressources. L'utilisateur qui accède à ces ressources utilise en général un type particulier de programme client, le *navigateur*. Les deux principales tâches d'un navigateur consistent à :

1. dialoguer avec un serveur ;
2. afficher à l'écran les documents transmis par un serveur.

Les deux navigateurs les plus utilisés sont *Internet Explorer* de Microsoft, et Netscape. Dans la suite de ce livre, nous utiliserons ce dernier puisqu'il a l'avantage d'être disponible, gratuitement, sur un grand nombre de plates-formes. Les navigateurs offrent des fonctionnalités bien plus étendues que les deux tâches citées ci-dessus. Netscape offre par exemple des modules permettant de composer des pages HTML, d'envoyer et recevoir des courriers électroniques, d'inclure des *applets* Java, etc.

2.2 HTML

Le langage HTML (pour *HyperText Markup Language*) permet de créer des documents indépendants de toute plate-forme, et donc particulièrement bien adaptés à des échanges d'informations dans un environnement hétérogène comme le Web.

HTML repose sur quelques concepts très différents de ceux que l'on peut trouver dans un traitement de texte standard permettant de créer des documents mis en forme. Tout d'abord, dans un document HTML, les instructions de mise en forme sont nettement distinctes du texte lui-même. Un système de *balises* (d'où le terme *Markup Language*) permet d'indiquer explicitement, pour chaque partie du texte, quelle est sa fonction (titre, en-tête de section, légende de figure, etc) ou son mode de présentation. C'est ensuite à l'outil qui affiche le document HTML de déterminer la position, la taille et les attributs graphiques associés à chaque balise.

En second lieu, HTML permet de créer des documents *hypertextes* : contrairement à un texte papier standard qui se lit séquentiellement, un texte HTML comprend des *liens* qui permettent d'accéder directement à d'autres documents, situés soit sur le même serveur, soit n'importe où sur le Web. Un lien étant associé à une ressource identifiée par une URL, suivre un lien consiste en fait à demander à un serveur web

de fournir cette resource (en général un document HTML). On peut ainsi explorer de manière transparente (autrement dit, sans avoir à connaître explicitement les adresses) l'ensemble des documents qui sont mis à libre disposition sur Internet,

Il existe de nombreux éditeurs de documents HTML – on peut citer *DreamWeaver* ou *FrontPage* sous Windows, *Quanta+* sous Linux – qui facilitent le travail de saisie des balises et fournissent une aide au positionnement (ou plus exactement au pré-positionnement puisque c'est le navigateur qui sera en charge de la mise en forme finale) des différentes parties du document (images, menus, textes, etc). Notre objectif dans ce livre étant n'étant pas d'aborder les problèmes de mise en page et de conception graphique de sites web, nous nous limiterons à des documents HTML relativement simples, en mettant l'accent sur leur intégration avec PHP et MySQL.

2.2.1 Bases de HTML

Un document HTML comprend deux parties. L'*en-tête* est destiné à contenir des informations relatives à l'auteur du document et à la nature de son contenu. Ces informations ne sont pas affichées par les navigateurs, mais peuvent être utilisées par exemple par les moteurs de recherche pour faciliter l'accès au document. L'en-tête est marqué par la balise <HEAD>.

Le corps du document est marqué par la balise <BODY> et contient les informations qui seront affichées par le navigateur, ainsi que des commandes de mise en forme. Voici un premier document HTML, très simple.

Exemple 1 *ExHTML1.html* : *Un premier document HTML*

```
<HTML>
<HEAD>
<TITLE>Pratique de MySQL et PHP</TITLE>
</HEAD>
<BODY BGCOLOR="white">
<CENTER><H1>Pratique de MySQL et PHP</H1></CENTER>
<HR>
Ce site est consacré au
livre "Pratique

de MySQL et PHP", paru aux Editions
O'Reilly.
</BODY>
</HTML>
```

On distingue deux types de balises. Les *conteneurs* permettent d'encadrer une partie de texte (désignée par le terme *élément* dans la terminologie HTML) et définissent une ou plusieurs caractéristiques de mise en forme qui s'appliquent à cette partie de texte. Par exemple tout texte compris entre les balises <CENTER> et </CENTER> sera centré par le navigateur. De manière générale un conteneur CONT est ouvert par une commande <CONT> et fermé par une balise </CONT>. Un deuxième type de balise n'affecte pas le texte du document : c'est le cas par exemple de <HR> qui trace une ligne horizontale. On parlera alors *d'élément vide*.

Remarque : On peut utiliser indifféremment des majuscules ou minuscules (ou un mélange des deux) pour les balises HTML. Pour plus de clarté nous utiliserons systématiquement les majuscules.

Dans l'exemple ci-dessus, on trouve les balises de base d'un document HTML Le document lui-même est encadré par le conteneur <HTML>. On trouve ensuite successivement les conteneurs <HEAD>, puis <BODY>. Le premier est ici réduit à son contenu minimal, à savoir un titre (et un seul) défini par la balise <TITLE>. En général ce titre ne s'affiche pas dans la page, mais en tant qu'attribut de la fenêtre d'affichage. Dans Netscape par exemple, le titre est affiché dans le bord supérieur de cette fenêtre.

La balise <BODY> encadre le contenu du document proprement dit. Ici on trouve tout d'abord un titre qui doit s'afficher, centré (conteneur <CENTER>), en gros caractères (conteneur <H1>). Les balises

<H*n*>, où *n* est un chiffre pouvant aller de 1 à 6, définissent des en-têtes (*header*) de section, d'importance décroissante. Il revient au navigateur de choisir des attributs graphiques tels que l'importance (visuelle) donnée à <H1> soit maximale, puis dégressive pour les balises de <H2> à <H6>. Enfin le document de ce premier exemple contient du texte simple.

Remarque : En principe les caractères accentués dans un texte HTML devraient être spécifiés par des symboles (appelés *entités*) commençant par un '&' et finissant par ';'. Un 'é' est par exemple rendu par le symbole `&eacute;`. C'est la méthode la plus sûre puisque les caractères accentués sont codés sur 8 bits (au lieu de 7 pour les caractères ascii les plus simples). Il n'y a pas de garantie absolue qu'un document codé sur 8 bits sera interprété avec le bon jeu de caractères (caractères latins) par le navigateur, mais nous utiliserons quand même directement les caractères accentués qui sont plus faciles à manipuler que les entités HTML.

La manière dont ce texte est disposé dans le fichier contenant le document HTML n'a aucune incidence sur la présentation qu'en fera le navigateur. En particulier les blancs et sauts de lignes multiples entre les mots sont ramenés un à seul espace de séparation au moment de l'affichage. De plus le navigateur est libre de déterminer la largeur de la page, la police utilisée et la taille des caractères, etc. En fait toute mise en forme doit être *explicitement* spécifiée dans le document par la balise appropriée.

La figure 2.2 montre la présentation de ce premier document avec Netscape. Notez la manière dont sont rendues les instructions de mise en forme, ainsi que l'affichage du titre dans le cadre de la fenêtre.

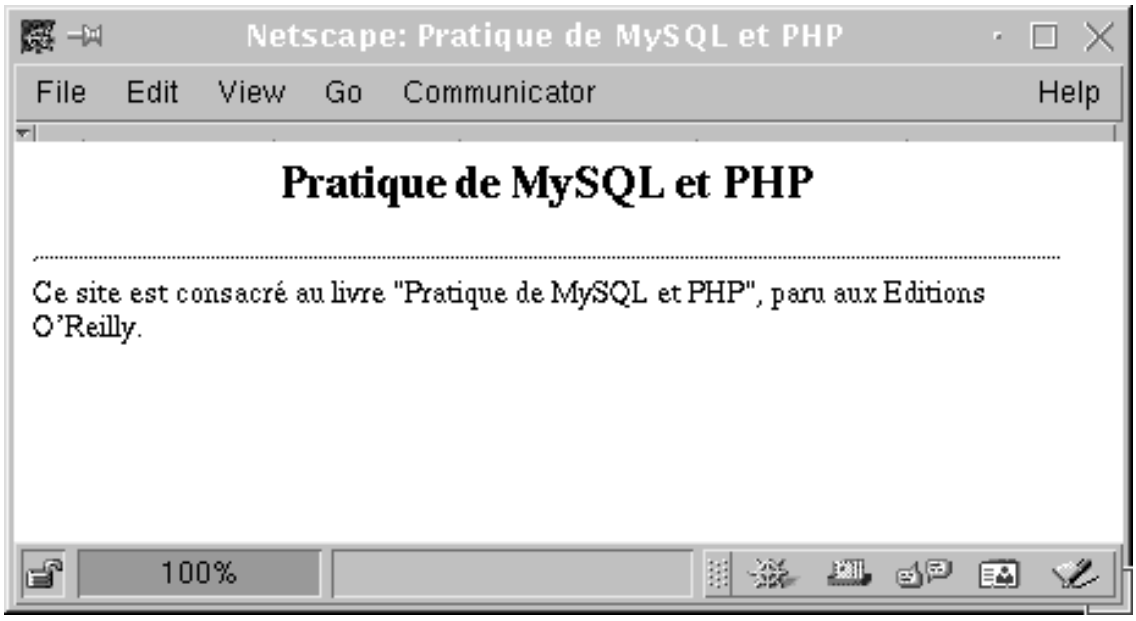

FIG. 2.2 – *Présentation avec Netscape*

2.2.2 Liens hypertexte

Pour l'instant le document HTML que nous avons créé est très faiblement interactif. Afin d'en faire un véritable document hypertexte, nous allons lui ajouter des liens qui permettront aux client de l'utiliser comme un point de départ pour accéder à des sites apparentés. Il est possible d'enrichir même dans un texte aussi court que celui-ci en transformant les mots importants en liens. Par exemple :

1. le mot « MySQL » peut être un lien vers le site *http://www.mysql.com* ;
2. « PHP » peut servir de lien vers *http://www.php.net* ;
3. enfin « Editions O'Reilly » peut servir de lien vers le site des Éditions O'Reilly.

La balise qui permet de créer des liens est le conteneur <A> pour *anchor* – « ancre » en français . L'URL qui désigne le lien est un *attribut* de la balise. Voici par exemple comment on associe le mot MySQL à son URL.

```
<A HREF="http://www.mysql.com">MySQL</A>
```

Les attributs dans les balises sont des paires *nom=valeur*. Les attributs peuvent notamment être utilisés pour spécifier explicitement des propriétés graphiques (couleur, police, etc) associées à une balise. Dans le cas des liens, l'attribut HREF doit prendre pour valeur l'URL. En principe la valeur doit être encadrée par des guillemets doubles (") ou simples ('), même si les navigateurs tolèrent l'absence de ces guillemets quand il n'y a pas d'ambiguité.

Voici le document avec les trois liens : noter que la disposition du texte (blancs et sauts de lignes) a été changée, sans que cela influe sur la présentation finale produite par Netscape. La seule modification notable faite par le navigateur consistera à mettre en valeur l'élément délimité par les balises <A> et </A>, en général en le soulignant et en lui attribuant une couleur spécifique. La figure 2.3 montre le résultat obtenu.

Exemple 2 *ExHTML2.html* : *Utilisation des ancres*

```
<HTML>
<HEAD>
<TITLE>Pratique de MySQL et PHP</TITLE>
</HEAD>
<BODY BGCOLOR="white">
<CENTER>
<H1>
Pratique de <A HREF="http://www.mysql.com">MySQL</A>
et <A HREF="http://www.php.net">PHP</A>
</H1></CENTER>
<HR>
Ce site est consacré au
livre "Pratique de <A HREF="http://www.mysql.com">MySQL</A>
et <A HREF="http://www.php.net">PHP</A>", paru aux
<A HREF="http://www.editions-oreilly.fr">Editions O'Reilly</A>.
</BODY>
</HTML>
```

La balise <A> peut être utilisée de manière un peu plus générale. On peut par exemple définir une position interne à un document en plaçant des « marques », puis utiliser une ancre pour se positionner directement sur une marque. Ces ancres « internes » permettent de se déplacer rapidement à l'intérieur d'un même document sans avoir à utiliser le défilement séquentiel du texte, et sont donc particulièrement utiles dans les très longs documents. Par exemple :

```
<A NAME='Liens'>Liens utiles</A>
```

définit une position de de nom Liens. On peut alors utiliser une ancre donnant directement accès à cette position :

```
<A HREF='Liens'>Aller directement aux liens utiles</A>
```

Autre utilisation de la balise <A> : l'envoi de courriers électroniques. Par exemple :

```
<A HREF=MAILTO:rigaux@cnam.fr>Philippe Rigaux</A>
```

Quand l'utilisateur clique sur « Philippe Rigaux », la fenêtre du navigateur permettant d'envoyer des emails est automatiquement appelée.

2.2.3 Principales balises HTML

HTML est un langage très simple : l'essentiel des principes permettant de construire un document a été exposé ci-dessus. Nous donnons maintenant une présentation des balises les plus couramment utilisées, avant d'aborder plus en détail deux aspects importants : l'utilisation des feuilles de style, et les formulaires.

FIG. 2.3 – *Présentation d'un document avec ancres.*

Balises de niveau bloc

Une première catégorie de balises permet de structurer le texte en sections et sous-sections, paragraphes, listes, tableaux, etc. Dans la liste qui suit, chaque balise est un conteneur comprenant une ouverture et une fermeture, à moins que l'inverse ne soit explicitement spécifié [1].

- <H*n*> introduit un titre de niveau *n* compris entre 1 et 6, par ordre d'importance décroissant.
- <P> permet de commencer un nouveau paragraphe.
- <HR> insère une ligne horizontale (c'est un élément vide : pas de balise de fermeture).
- <UL> et <OL> permettent de créer des listes d'items, chacun étant marqué par <LI>. Par exemple :

```
<OL>
  <LI> Un premier item. </LI>
  <LI> Un second.       </LI>
  <LI> ...              </LI>
 </OL>
```

  Chaque item est marqué par une puce dans le cas de <UL>, et par un numéro dans le cas de <OL>. Les listes peuvent être imbriquées.
- <PRE> indique un texte préformaté : le navigateur doit garder les caractères blancs et les sauts de lignes, et employer une police à chasse fixe de type Courier. Cette balise peut notamment être utilisée pour afficher du code informatique.
- <TABLE> permet d'insérer un tableau. Il s'agit d'une fonctionnalité importante de HTML, notamment parce qu'elle permet de contrôler assez précisément le positionnement relatif des éléments dans une page. La balise comprend de nombreuses options et attributs, et est présentée en détail ci-dessous.
- <FORM> permet d'insérer un *formulaire*. Les formulaires constituent le moyen privilégié pour interagir avec l'utilisateur en lui demandant de saisir des informations : voir section 2.3.2.

1. Les navigateurs sont en général assez tolérants pour accepter l'omission d'une balise de fermeture s'il n'y a pas d'ambiguïté. Par exemple la balise </P> peut être oubliée sans dommage.

– <DIV> permet de diviser un texte, et de changer le style de présentation, notamment avec l'attribut ALIGN qui peut prendre les valeurs LEFT, CENTER ou RIGHT. Le conteneur <CENTER> est équivalent à <DIV> avec un alignement centré.

Tableaux

La balise <TABLE> permet d'insérer un tableau. Chaque ligne est marquée par <TR> et </TR>, et chaque colonne par <TD> et </TD>. Il existe de nombreuses balises permettant d'indiquer une légende (<CAPTION>), des en-têtes de colonnes (<TH>), etc.

Les attributs permettent de contrôler l'apparence d'un tableau, à chaque niveau (tableau, ligne, colonne). Voici les principaux attributs de la balise <TABLE>.

– BORDER indique l'épaisseur des traits bordant les cellules. Une valeur de 0 supprimera toute bordure. On obtient alors une grille « invisible » dont l'intérêt principal est de permettre l'alignement régulier d'un ensemble d'éléments dans une page. Nous utiliserons cette possibilité pour la mise en page de formulaires.
– CELLSPACING définit l'espace (en pixels) entre les cellules.
– CELLPADDING définit l'espace entre le bord de la cellule et son contenu.
– ALIGN définit l'alignement du contenu des cellules. Cet attribut peut prendre les valeurs LEFT, CENTER et RIGHT.
– WIDTH indique la largeur occupée par le tableau dans la fenêtre d'affichage du navigateur. On peut donner la largeur en pourcentage de cette fenêtre, ce qui est recommandé puisque cela permet d'adapter l'affichage proportionnelement à la taille choisie par l'utilisateur. Par exemple WITDH="80%" indique que la largeur du tableau est limitée à 80 % de celle de la fenêtre. On peut aussi indiquer une valeur absolue (en pixels). Par défaut (autrement dit, quand WIDTH n'est pas défini), le navigateur calcule automatiquement la largeur de chaque colonne et celle du tableau en fonction du contenu.

L'exemple 3 montre les principales options. Le tableau est décrit par une liste d'éléments <TR>, eux-mêmes constitués de cellules marquées par <TH> pour les en-têtes de colonnes et par <TD> pour les cellules. La balise </TR> peut toujours être omise puisque <TR> indique le début d'une nouvelle ligne. C'est vrai aussi des balises <TD> et <TH>.

Exemple 3 *ExHTML3.html* : *Un tableau HTML.*

```
<HTML> <HEAD>
<TITLE>Exemple d'un tableau HTML</TITLE>
</HEAD>
<BODY BGCOLOR="white">

<H1>Une table HTML</H1>

    <TABLE BORDER=4 CELLSPACING=2 CELLPADDING=2>
    <CAPTION ALIGN=bottom><B>Quelques films</B></CAPTION>
    <TR><TH>Titre
        <TH>Année
        <TH>Nom Réalisateur
        <TH>Prénom
        <TH>Année naissance
    </TR>
    <TR><TD>Alien<TD>1979<TD>Scott<TD>Ridley<TD>1943
    <TR><TD>Vertigo<TD>1958<TD>Hitchcock<TD>Alfred<TD>1899
    <TR><TD>Van Gogh<TD>1991<TD>Pialat<TD>Maurice<TD>1925
    <TR><TD>Pulp Fiction<TD>1994<TD>Tarantino<TD>Quentin<TD>1963
    <TR><TD>Psychose<TD>1960<TD>Hitchcock<TD>Alfred<TD>1899
    <TR><TD>Kagemusha<TD>1980<TD>Kurosawa<TD>Akira<TD>1910
    <TR><TD>Volte-face<TD>1997<TD>Woo<TD>John<TD>1946
    <TR><TD>Sleepy Hollow<TD>1999<TD>Burton<TD>Tim<TD>1958
```

```
    <TR><TD>Titanic<TD>1997<TD>Cameron<TD>James<TD>1954
    <TR><TD>Sacrifice<TD>1986<TD>Tarkovski<TD>Andrei<TD>1932
    </TABLE>

</BODY></HTML>
```

Le navigateur déterminera automatiquement la largeur du tableau et de chacune de ses colonnes, ainsi que la disposition du contenu (voir la copie d'écran de la figure 2.4). Il est possible d'influer sur cette présentation par défaut à l'aide des attributs de la balise <TABLE>, déjà vus ci-dessus, et des balises <TR> et <TD>. Voici quelques uns des principaux attributs de <TD>.

`ALIGN` est équivalent, au niveau d'une cellule individuelle, à l'attribut de même nom au niveau du tableau.

`WIDTH` permet d'indiquer (en pourcentage ou en pixels) la largeur d'une colonne.

`COLSPAN=`*`c`* permet d'indiquer qu'un élément couvre plusieurs colonnes.

`ROWSPAN=`*`r`* permet d'étendre un élément sur plusieurs lignes.

`NOWRAP` force un élément à s'étendre sur une seule ligne au lieu de revenir à la ligne quand le bord de la cellule est atteint.

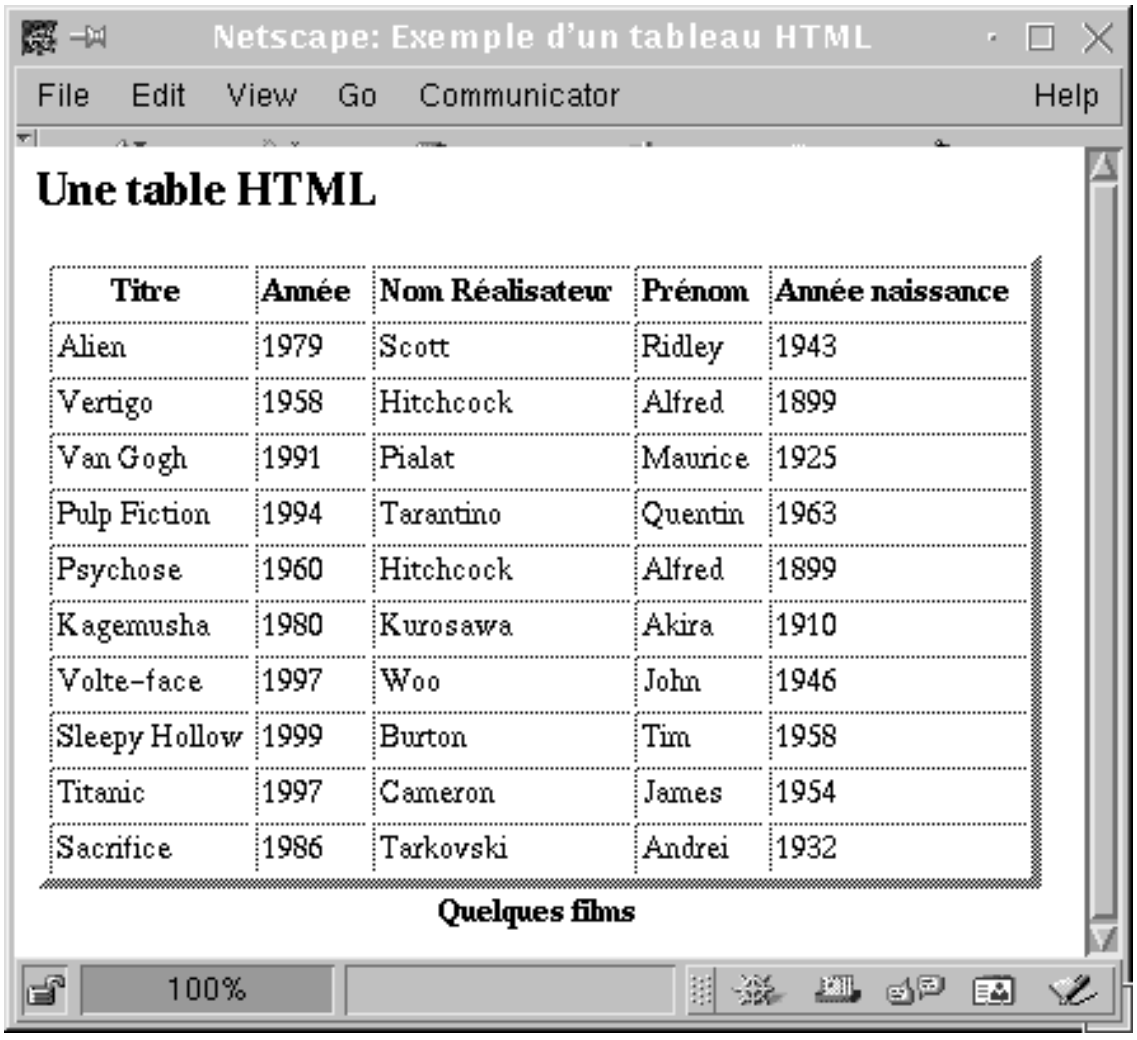

FIG. 2.4 – *Présentation avec Nestcape de l'exemple 3*

Balises de niveau texte

À la différence des balises de niveau bloc, celles de niveau texte n'introduisent pas de saut de ligne dans la présentation de la page HTML, et ne servent donc, pour l'essentiel, qu'à modifier le style de présentation du texte. Comme précédemment, chaque balise dans ce qui suit est un conteneur comprenant une ouverture et une fermeture, sauf mention contraire.

- <B> (pour *boldface*) met en gras tout le texte jusqu'à la balise de fermeture </B>. On peut aussi utiliser <STRONG> qui a le même effet.
- <I> (pour *italic*) met en italiques tout le texte jusqu'à la balise de fermeture </I>.
- <TT> permet de passer dans une police à chasse constante, de type Courier, pour présenter par exemple du code informatique.
- <EM> (pour *emphasize*) est destinée à mettre en valeur l'élément. Le style choisi pour la mise en valeur est à la discrétion du navigateur (en général en italiques).
- <STRONG> est également destinée à une mise en valeur, le style attribué par le navigateur étant en principe le gras.

Il est possible de changer la police de caractère, la taille ou la couleur avec la balise <FONT> qui propose de nombreux attributs de mise en forme. À titre d'exemple, voici comment passer, avec <FONT>, en police times, couleur rouge, et grands caractères :

```
<FONT SIZE=2 COLOR=red FACE='times'> ..... </FONT>
```

L'attribut SIZE peut prendre pour valeur un entier de 1 à 7, 1 représentant la taille la plus grande.

L'inclusion d'images se fait avec la balise <IMG> (il n'y a pas de fermeture) qui doit contenir un attribut SRC donnant l'URL du fichier image. Par exemple :

```
<IMG SRC='http://cartier.cnam.fr/redbar.gif'>
```

demande au serveur *cartier* le fichier *redbar.gif* (correspondant à un trait horizontal de couleur rouge). Voici quelques autres attributs de la balise <IMG> :

- BORDER=*b*, où *b* indique la taille, en pixels, de la bordure.
- ALIGN=*option*, où *option* peut être RIGHT, LEFT, MIDDLE, BOTTOM ou TOP, permet de spécifier l'alignement de l'image.
- HEIGHT=*h* WITDH=*w* sont les attributs qui permettent de forcer la taille d'une image (en pourcentage ou en pixels).

Il existe de très nombreux attributs associés aux balises HTML, la plupart destinés à indiquer des options de présentation (couleur, alignement, taille etc). Jusqu'à la version 3.2 de HTML, ces attributs constituaient le seul moyen pour contrôler le graphisme et la représentation des pages HTML.

L'inconvénient principal de cette approche est de multiplier des spécifications identiques (par exemple pour dire que toutes les balises <H1> sont en police Arial et de couleur bleue), ce qui est à la fois fastidieux et rend très difficile une modification des choix de présentation après coup (comment faire pour passer tous les éléments <H1> en couleur rouge, quand le site en contient déjà quelques dizaines, voire quelques centaines ?).

Ces options on perdu de leur intérêt avec la version 4 de HTML qui introduit les *feuilles de style*, une méthode beaucoup plus puissante pour gérer la présentation des documents HTML.

2.2.4 Feuilles de style

Les feuilles de style (en anglais *Cascading Style Sheets*, d'où l'acronyme CSS), introduites avec la version 4 de HTML, permettent de spécifier, *indépendamment du document HTML lui-même*, les attributs de présentation. Au lieu de qualifier chaque balise par un ensemble d'attributs définissant sa présentation graphique, et d'appliquer répétitivement les mêmes attributs à toutes les balises identiques, les feuilles de style définissent ces attributs, dans un document séparé, à charge pour le navigateur d'utiliser le style approprié. Les avantages sont nombreux :

- la garantie d'une présentation homogène et cohérente des pages HTML ;

– la possibilité de changer globalement cette présentation en modifiant un seul document ;
– la compatibilité multi-navigateurs (à la différence des bricolages souvent utilisés par les programmeurs HTML) ;
– enfin, et surtout, moins de code, mieux structuré, et donc des sites plus faciles à maintenir et à faire évoluer.

Il faut quand même moduler quelque peu les avantages des CSS. Tout d'abord il s'agit d'une norme assez récente, et assez pauvrement supportée par les navigateurs. Plus grave : Netscape et Internet Explorer ne semblent pas adopter tout à fait les mêmes parties de la norme, et ne semblent pas non plus interpréter de la même manière certaines des spécifications... Conclusion : il vaut mieux ne pas utiliser les techniques les plus avancées, du moins à l'heure actuelle.

Exemple d'une feuille de style

Il existe plusieurs manières d'associer des styles à un document. On peut définir les styles directement dans l'en-tête du document, ou utiliser un fichier séparé qui est transmis par le serveur avec le document principal. C'est cette dernière solution qui est préférable puisqu'elle permet de factoriser les styles pour un ensemble de documents.

L'exemple ci-dessous montre une feuille CSS très simple qui donne quelques directives de présentation pour les pages HTML de notre site.

Exemple 4 *films.css* : Exemple de feuille de style

```
/*-----------------------------------
* La feuille de style du site Films
*-----------------------------------
*/

/* Le fond est toujours en blanc    */

BODY {background-color: white}

/* On utilise une couleur rouge pour les ancres
 * et les entêtes
*/

A, H1, H2, H3 {color: #ca0000}

CAPTION {font-size:large; color: #ca0000}

/* Alternance des couleurs pour les lignes des tables */

TR.A0 {background-color:white}
TR.A1 {background-color:yellow}
```

Ce fichier est un fichier ascii très ordinaire, contenant des instructions de mise en forme qui peuvent être disposées arbitrairement. Les commentaires sont encadrés par « /* » et « */ », comme en C.

On trouve ensuite les noms des balises, suivis par un ensemble d'affectation de valeurs à certains attributs de ces balises. La balise <BODY> a pour attribut `background-color` à laquelle on affecte la valeur `white`. De même les balises <A> et <H*n*> ont pour attribut `color` qui est ici affectée à une variante de rouge. L'énumération complète de tous les attributs pour toutes les balises sort du cadre de ce livre, mais nous citons les plus usités dans ce qui suit.

Une instruction de la forme `H1 {color : white}` est une *règle* dans la terminologie CSS, `H1` est le *sélecteur*, `color` est la *propriété*, et `white` est la *valeur*. Toutes les balises HTML peuvent être utilisées comme sélecteur. Il est possible de grouper plusieurs sélecteurs et de leur associer des déclarations communes (voir exemple 4 pour <A> et <H*n*>), et, inversement, de grouper plusieurs propriétés (voir exemple 4 pour <CAPTION>).

Quand le sélecteur est un conteneur, ses propriétés sont *héritées* par tout le code HTML compris entre les balises d'ouverture et de fermeture, même si ce code contient lui-même des conteneurs imbriqués. Par exemple la couleur de fond définie pour <BODY> est valable pour tout le document, du moins tant qu'il n'y a pas redéfinition.

Dans ce que nous avons vu jusqu'à présent, les directives d'une feuille de style s'appliquent à toutes les balises d'un même type. Donc toutes les balises <H1> seront affectées indifféremment de la couleur #ca0000. Que faire quand on souhaite utiliser des critères plus fins que le simple nom de la balise pour déterminer la présentation à adopter ?

Un exemple très concret nous est donné avec la présentation des tableaux. Afin de faciliter la lecture, il est courant d'alterner la couleur de fond pour chaque ligne. La propriété définissant la couleur de fond s'applique à la balise <TR> : le mécanisme que nous avons vu jusqu'à présent ne permet d'associer à <TR> qu'une couleur et s'avère donc insuffisant pour gérer la couleur en fonction du contexte.

Les *classes* CSS permettent d'associer aux balises un contexte d'application qui peut être référencé dans le document HTML. Dans l'exemple 4, les lignes suivantes utilisent cette notion de classe.

```
TR.A0 {background-color:white}
TR.A1 {background-color:yellow}
```

On a donc deux classes pour <TR>. La classe A0 spécifie une couleur de fond blanche, la classe A1 une couleur de fond jaune. Voici maintenant comment on peut faire référence à ces classes dans le document HTML.

```
<TR CLASS='A0'><TD>Alien<TD>1979<TD>Scott<TD>Ridley<TD>1943
<TR CLASS='A1'><TD>Vertigo<TD>1958<TD>Hitchcock<TD>Alfred<TD>1899
```

Les classes offrent un mécanisme vraiment puissant pour dissocier les balises HTML des attributs de présentation. À l'extrême, on peut définir une classe sans faire référence à une balise. Par exemple l'instruction suivante crée une classe remarque qui peut être utilisée, dans le document HTML, en association avec n'importe quelle balise.

```
.remarque {background-color:yellow}
```

Un dernier mot sur les feuilles de style : pourquoi le terme *cascading* ? En fait il peut y avoir, pour un même document, plusieurs spécifications contradictoires. On peut imaginer des styles placés dans l'en-tête du document, qui entrent en conflit avec une feuille de style externe liée au document. Le terme *cascading* fait référence aux règles de priorité que le navigateur doit suivre pour choisir la spécification à appliquer.

Application à un document

Il reste à appliquer cette feuille de style à un document HTML. Il suffit d'ajouter, dans le conteneur <HEAD>, la ligne suivante.

```
<LINK REL=stylesheet HREF='films.css' TYPE='text/css'>
```

Exemple 5 *ExHTML4.html* : *Document HTML avec feuille de style*

```
<HTML><HEAD>
<TITLE>Exemple d'un tableau HTML</TITLE>
<LINK REL=stylesheet HREF="films.css" TYPE="text/css">
</HEAD>
<BODY>

<H1>Une table HTML</H1>

    <TABLE BORDER=4 CELLSPACING=2 CELLPADDING=2>
    <CAPTION ALIGN=bottom>Quelques films</CAPTION>
    <TR><TH>Titre
        <TH>Année
        <TH>Nom Réalisateur
```

```
        <TH>Prénom
        <TH>Année naissance
    </TR>
    <TR CLASS=A0><TD>Alien<TD>1979<TD>Scott<TD>Ridley<TD>1943
    <TR CLASS=A1><TD>Vertigo<TD>1958<TD>Hitchcock<TD>Alfred<TD>1899
    <TR CLASS=A0><TD>Van Gogh<TD>1991<TD>Pialat<TD>Maurice<TD>1925
    <TR CLASS=A1><TD>Pulp Fiction<TD>1994<TD>Tarantino<TD>Quentin<TD>1963
    <TR CLASS=A0><TD>Psychose<TD>1960<TD>Hitchcock<TD>Alfred<TD>1899
    <TR CLASS=A1><TD>Kagemusha<TD>1980<TD>Kurosawa<TD>Akira<TD>1910
    <TR CLASS=A0><TD>Volte-face<TD>1997<TD>Woo<TD>John<TD>1946
    <TR CLASS=A1><TD>Sleepy Hollow<TD>1999<TD>Burton<TD>Tim<TD>1958
    <TR CLASS=A0><TD>Titanic<TD>1997<TD>Cameron<TD>James<TD>1954
    <TR CLASS=A1><TD>Sacrifice<TD>1986<TD>Tarkovski<TD>Andrei<TD>1932
    </TABLE>

</BODY></HTML>
```

La figure 2.5 montre la présentation de ce document avec Netscape. Vous pouvez constater, entre autres, l'alternance des couleurs dans la présentation du tableau.

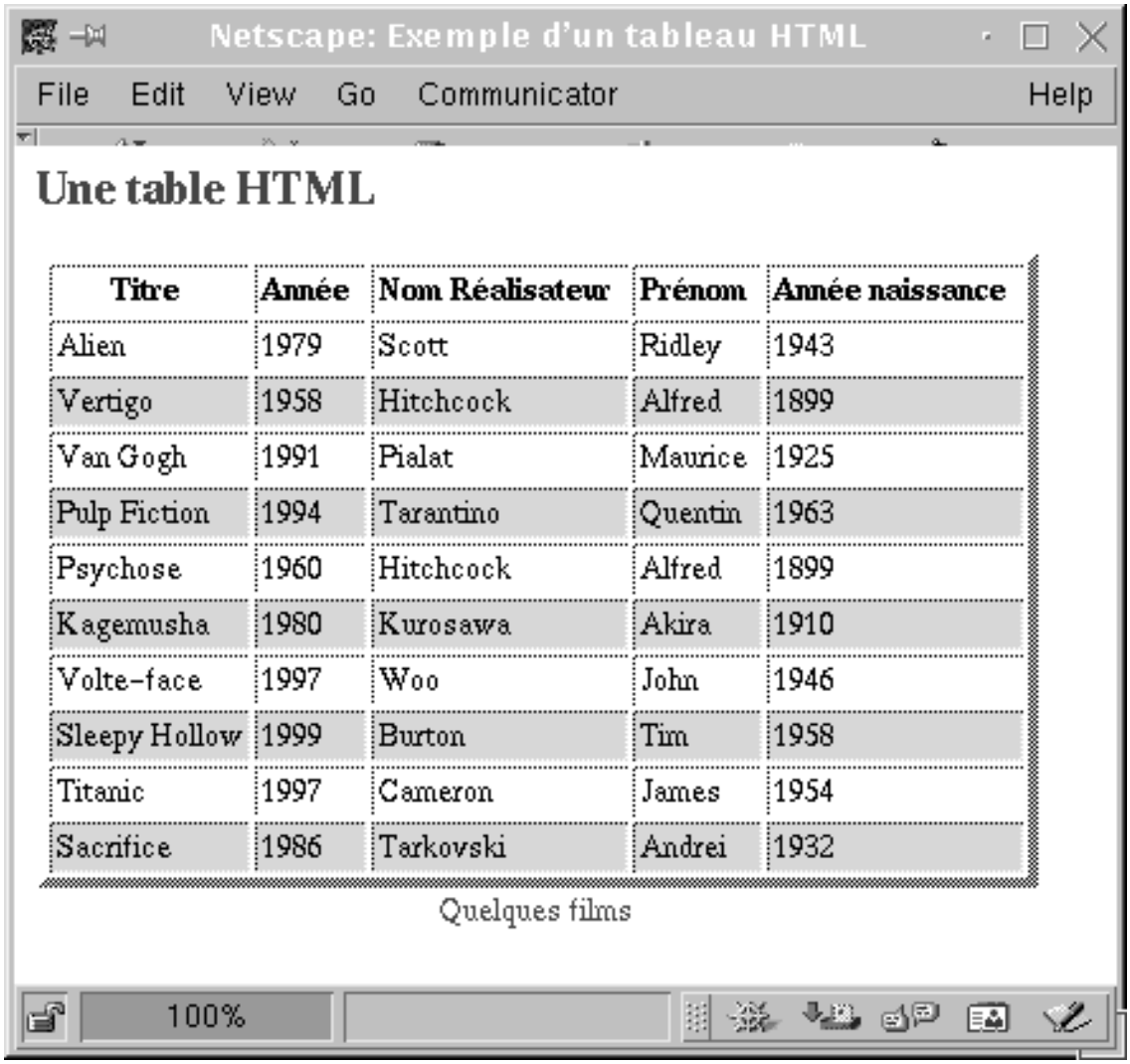

FIG. 2.5 – *Présentation d'un tableau avec feuille de style*

Compléments CSS

Nous donnons ci-dessous une sélection de quelques propriétés importantes qui peuvent être utilisées dans les feuilles de style. Si vous souhaitez en savoir plus, référez vous à la bibliographie ou aux liens donnés sur le site. Toutes ces propriétés font partie de la norme CSS, mais il est toujours bon de tester leur effet avec les deux navigateurs principaux.

font-family Le plus prudent est d'utiliser un nom générique comme `serif` (Times), `sans-serif` (Arial ou Helvetica) ou `monospace` (Courier).

font-size Les valeurs peuvent être exprimées en *points* (par exemple `font-size:16pt`), en pixels (`font-size:20px`), ou par des termes génériques (`xx-small, x-small, small, medium, large, x-large, xx-large`).

font-style Permet de mettre en italiques (`italic`), en `oblique`, ou de revenir à `normal`.

font-weight Les valeurs principales sont `bold` pour mettre en caractères gras, et `normal`.

text-decoration Valeurs principales : `underline`, `overline` et `none`. On peut indiquer au navigateur que les ancres ne devront pas être soulignées avec l'instruction `A:link {text-decoration : none}`. Noter que `A:link` fait référence à une classe prédéfinie de la balise <A>.

text-align Permet de contrôler l'alignement horizontal du texte à l'intérieur des balises définissant un paragraphe, avec les valeurs `center`, `left`, `right` et `justify`.

vertical-align Permet de contrôler l'alignement vertical, avec les valeurs `top`, `bottom`, `middle`, `sub` (pour mettre en indice) et `super` (pour mettre en exposant).

margin-top, margin-bottom, margin-left, margin-right Les CSS permettent de contrôler la marge autour des éléments du document, ce qui est très difficile en HTML simple. Les valeurs peuvent être indiquées en pixels (`10px`) ou en points (`20pt`).

color Cette propriété correspond à la couleur du texte. Il y a beaucoup de possibilités pour spécifier une couleur. On peut utiliser un code hexadécimal (comme `#ca0000`), une combinaison rouge-vert-bleu, ou plus simplement un nom comme `white, black, red, blue, aqua, olive`. Les noms ou codes couleurs peuvent se trouver partout sur le Web, par exemple à *www.webreference.com*.

background-color Correspond à la couleur de fond.

background-image Permet d'utiliser une image comme fond de l'élément. Cette propriété accepte pour valeur `url(`*`MonURL`*`)` où *MonURL* peut être une URL quelconque, y compris, comme c'est souvent le cas, un nom de fichier local.

position Avec les CSS il devient possible d'indiquer explicitement la position des éléments dans une page. Cette propriété peut prendre deux valeurs. `absolute` indique un positionnement absolu, par rapport à la fenêtre d'affichage et indépendamment des autres éléments. `relative` signifie que les paramètres de positionnement s'interprètent relativement à la position normale de l'élément, et influent sur le positionnement des autres objets.

width, left, top Ces propriétés sont à utiliser en association avec *position*. Par exemple

```
H1 {position:relative; left: 100px; top:50px}
```

indique que les éléments <H1> doivent être décalés de 100 pixels sur la gauche et 50 pixels vers le bas.

2.3 Programmation CGI

Jusqu'à présent les pages HTML que nous avons créées sont totalement statiques. Leur contenu est fixé à l'avance et l'utilisateur ne dispose d'aucun moyen d'action pour sélectionner les informations qui l'intéressent. Un autre inconvénient, du point de vue du webmestre cette fois, est que la maintenance du site implique la manipulation d'un ensemble de pages HTML qui peut rapidement devenir très volumineux.

Reprenons l'exemple de l'affichage d'un ensemble de films dans une page HTML. Dans une application réelle, la gestion d'une petite dizaine de films ne présente aucun intérêt : l'ordre de grandeur à envisager est de quelques centaines, voire quelques milliers de références. Dans ce cas (1) l'utilisateur doit pouvoir

choisir les films qui lui sont proposés (par exemple tous les films de l'année 1999), (2) l'administrateur doit disposer d'un outil pratique pour engendrer à la demande une sous-liste des films disponibles.

2.3.1 Principes de base du CGI

Le *Common Gateway Interface* (CGI) constitue la technique traditionnelle pour pallier les deux inconvénients ci-dessus. L'idée de base est de produire les documents HTML par un programme qui est associé au serveur web. Ce programme reçoit en outre des paramètres (comme l'année de parution des films) saisis par l'utilisateur.

Le CGI est la solution la plus ancienne, et sans doute encore la plus utilisée, pour la gestion de sites web dynamiques. La programmation de sites web avec PHP s'appuie d'ailleurs, pour tous les échanges client/serveur, sur le protocole CGI.

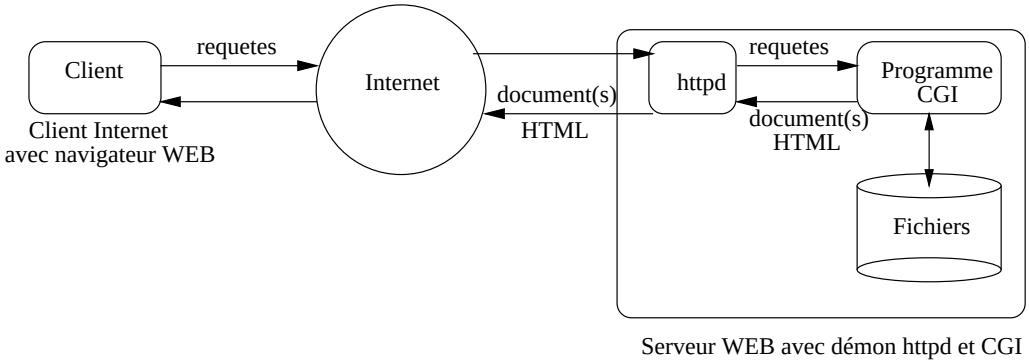

FIG. 2.6 – *Architecture CGI*

La figure 2.6 illustre les composants classiques d'une architecture CGI. Le navigateur (client) envoie une requête (souvent à partir d'un *formulaire* HTML) qui est plus complexe que la simple demande de transmission d'un document. Cette requête consiste à faire déclencher une *action* (que nous désignins par « programme CGI » dans ce qui suit) sur le serveur. En d'autres termes, l'URL dans la page web référence, au lieu d'une page HTML, le nom du programme CGI qui doit se trouver (du moins sous UNIX) dans le répertoire *cgi-bin*.

L'exécution du programme CGI par le serveur web se déroule en trois phases :

1. *Requête du client au serveur* : le programme serveur récupère les informations transmises par le navigateur, soit le nom du programme CGI accompagné, le plus souvent, de paramètres saisies par l'internaute dans un formulaire ;
2. *Exécution du programme CGI* : le serveur déclenche l'exécution du programme CGI, en lui fournissant les paramètres reçus ci-dessus ;
3. *Transmission du document HTML* : le programme CGI renvoie le résultat de son exécution au serveur sous la forme d'un fichier HTML, le serveur se contentant alors de faire suivre au client.

Le programme CGI peut être écrit en n'importe quel langage (C, C++, Perl, script shell, etc) et est libre de faire toutes les opérations nécessaires pour satisfaire la demande (dans la limite de ses droits d'accès bien sûr). Il peut notamment rechercher et transmettre des fichiers ou des images, effectuer des contrôles, des calculs, créer des rapports, etc. Il peut aussi accéder à une base de données pour insérer ou rechercher des informations. C'est ce dernier type d'utilisation, dans sa variante PHP/MySQL, que nous étudions dans ce livre.

Avec le CGI, les communications entre serveur et client deviennent plus riches. Il faut décrire d'une part comment le client (navigateur) transmet des informations plus compliquées qu'une simple URL au serveur, et d'autre part comment le serveur joue son rôle de transmission entre le programme CGI et le navigateur.

2.3.2 Formulaires

Les formulaires constituent un moyen privilégié d'interaction puisqu'ils permettent à l'utilisateur d'entrer ses demandes par l'intermédiaire de champs de saisie, et de transmettre ces demandes au serveur. HTML propose un ensemble de balises pour définir des champs de saisie qui offrent la possibilité appréciable de créer très facilement une interface.

La figure 2.7 montre un exemple de formulaire permettant la saisie de la description d'un film. Différents types de champs sont utilisés :

- Le titre et l'année sont des champs simples de saisie. L'utilisateur est libre d'entrer toute valeur alphanumérique de son choix.
- Le pays producteur est proposé sous la forme d'une liste de valeurs prédéfinies. Le choix est de type exclusif : on ne peut cocher qu'une valeur à la fois.
- Le genre est lui aussi présenté sous la forme d'une liste de choix imposés, mais ici il est possible de sélectionner plusieurs choix simultanément.
- L'internaute peut transmettre au serveur un fichier contenant l'affiche du film, grâce à un champ spécial qui offre la possibilité de choisir (bouton *Browse*) le fichier sur le disque local.
- Une liste présentée sous la forme d'un menu déroulant propose une liste des metteurs en scène.
- On peut entrer le résumé du film dans une fenêtre de saisie de texte.
- Enfin, les boutons « Valider » ou « Annuler » sont utilisés pour, au choix, transmettre les valeurs saisies au progamme CGI, ou réinitialiser le formulaire.

Cet exemple couvre pratiquement l'ensemble des types de champs disponibles. Nous décrivons en détail dans ce qui suit les balises de création de formulaires.

La balise <FORM>

C'est un conteneur délimité par <FORM> et </FORM> qui, outre les champs de saisie, peut contenir n'importe quel texte ou balise HTML. Les trois attributs suivants sont essentiels pour la communication du programme serveur avec un programme CGI :

- ACTION est la référence au programme qui doit être exécuté par le serveur ;
- METHOD indique le mode de transmission des paramètres au programme CGI. Il y a essentiellement deux valeurs possibles, GET ou POST ;
- ENCTYPE indique quel est le type d'encodage des données du formulaire qui doit être utilisé pour la transmission au serveur. Il y a deux valeurs possibles.
  1. *application/x-www-form-urlencoded*.
     Il s'agit de l'option par défaut, utilisée même quand on ne donne pas d'attribut ENCTYPE. Les champs du formulaire sont transmis sous la forme d'une liste de paires *nom=valeur*, séparées par des '&'.
  2. *multipart/form-data*.
     Cette option doit être utilisée pour les transmissions comprenant des fichiers. Le mode de transmission par défaut est en effet inefficace pour les fichiers binaires à cause de la codification assez volumineuse qui est utilisée pour les caractères non-alphanumériques. Quand on utilise *multipart/form-data*, les fichiers sont transmis séparément des champs classiques, avec une représentation plus compacte que ces derniers.
     Le formulaire des saisie des films utilise cette option pour transmettre efficacemnt le fichier contenant l'affiche du film (voir chapitre 5, section 5.3.2).

Voici une illustration avec le code HTML donnant le début du formulaire de la figure 6. Le service associé à ce formulaire est le programme Films qui se trouve sur le serveur cartier.cnam.fr (port 8080). La méthode POST indique un mode particulier de passage des paramètres.

```
<FORM ACTION='http://cartier.cnam.fr:8080/cgi-bin/Films' METHOD=POST>
```

À l'intérieur d'un formulaire, on peut placer plusieurs types de champs de saisie, incluant les valeurs numériques ou alphanumériques simples saisies par l'utilisateur, les choix multiples ou exclusifs parmi un ensemble de valeurs prédéfinies, du texte libre ou la transmission de fichiers.

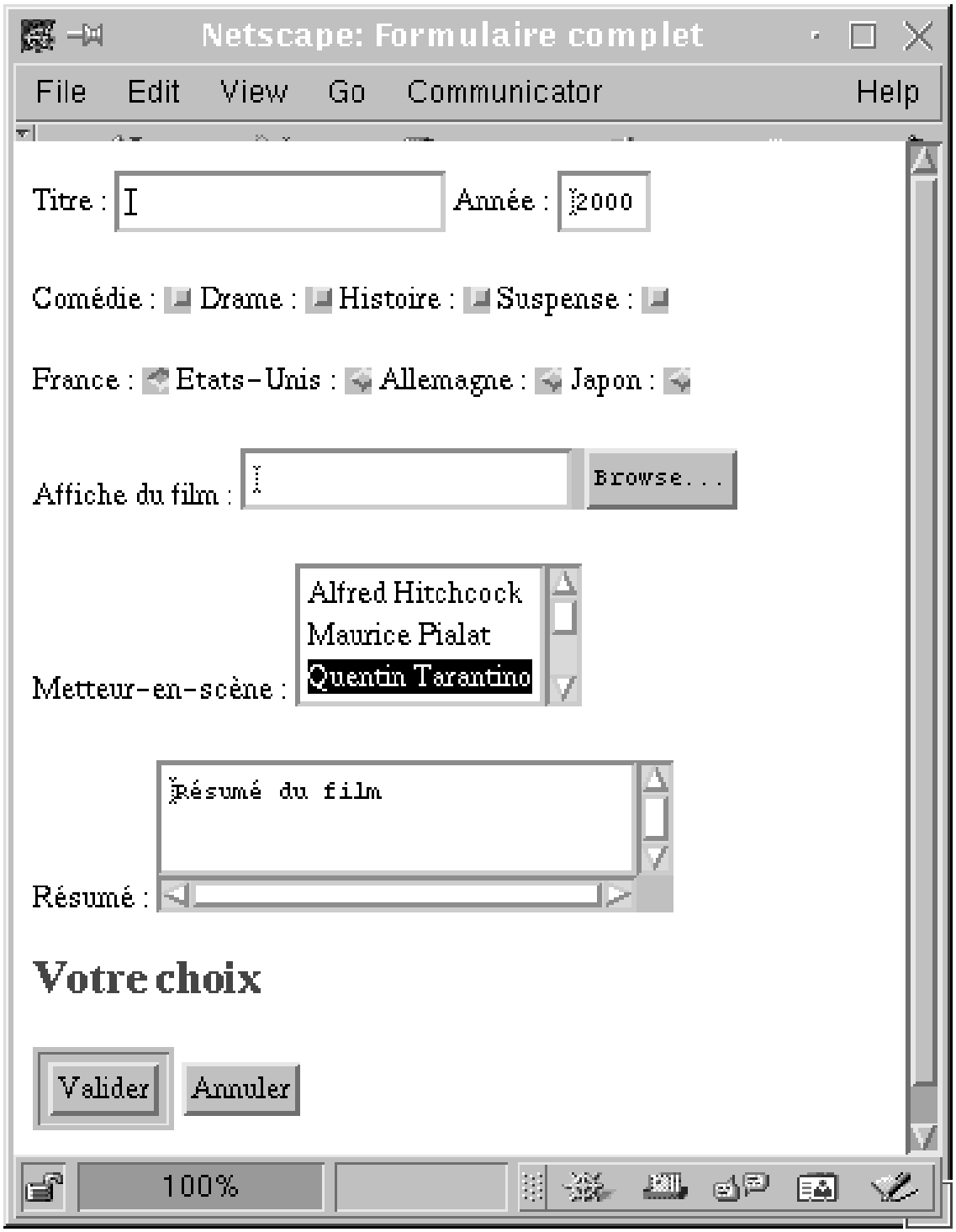

FIG. 2.7 – *Présentation d'un formulaire avec Netscape*

La balise <INPUT>

La balise <INPUT> est la plus générale. Elle permet de définir tous les champs de formulaires, à l'exception des listes de sélection et des fenêtres de saisie de texte.

Chaque champ <INPUT> a un attribut NAME qui permet, au moment du passage des paramètres au

programme CGI, de référencer les valeurs saisies sous la forme de couples *nom=valeur*. La plupart des champs ont également un attribut VALUE qui permet de définir une valeur par défaut (voir exemple ci-desssous). Les valeurs de NAME ne sont pas visibles dans la fenêtre du navigateur : elles ne servent qu'à référencer les valeurs respectives de ces champs au moment du passage des paramètres au programme CGI.

Le type d'un champ est défini par un atribut TYPE qui peut prendre les valeurs suivantes :

TYPE=TEXT Correspond à un champ de saisie permettant à l'utilisateur d'entrer n'importe quelle chaîne de caractères. La taille de l'espace de saisie est fixée par l'attribut SIZE, et la longueur maximale par l'attribut MAXLENGTH. Voici le champ pour la saisie du titre du film.

```
Titre : <INPUT TYPE=TEXT SIZE=20 NAME="titre">
```

Un paramètre titre=*valeur* sera passé par le serveur au programme CGI.

TYPE=PASSWORD Identique au précédent, mais le texte entré au clavier n'apparaît pas en clair (une étoile '*' sera affichée par le navigateur en lieu et place de chaque caractère). Ce type de champ est principalement utile pour la saisie de mots de passe.

TYPE=HIDDEN Un champ de ce type n'est pas visible. Il est principalement destiné à définir un paramètre dont la valeur est fixée, et à passer ce paramètre au programme CGI en même temps que ceux saisis par l'utilisateur.

Par exemple le champ ci-dessous permet de passer systématiquement un paramètre monNom ayant la valeur ExForm1, pour indiquer au programme CGI le nom du formulaire qui lui transmet les données saisies.

```
<INPUT TYPE=HIDDEN NAME='monNom' VALUE='ExForm1'>
```

Il est important de noter que « caché » ne veut pas dire « secret » ! Rien n'empêche un utilisateur de consulter la valeur d'un champ caché en regardant le code source du document HTML.

TYPE=CHECKBOX Ce type crée les boutons associés à des valeurs, ce qui permet à l'utilisateur de cocher un ou plusieurs choix, sans avoir à entrer explicitement la valeur. En associant le *même* nom à un ensemble de champs CHECKBOX, on indique au navigateur que ces champs doivent être groupés dans la fenêtre d'affichage.

L'exemple ci-dessous montre comment donner le choix (non exclusif) entre les genres des films.

```
Comédie   : <INPUT TYPE='CHECKBOX' NAME='genre' VALUE='C'>
Drame     : <INPUT TYPE='CHECKBOX' NAME='genre' VALUE='D'>
Histoire  : <INPUT TYPE='CHECKBOX' NAME='genre' VALUE='H'>
Suspense  : <INPUT TYPE='CHECKBOX' NAME='genre' VALUE='S'>
```

Contrairement aux champs de type TEXT ou apparenté, les valeurs (champ VALUE) ne sont pas visibles. On peut donc utiliser une codification ('C', 'D', ...) plus concise que les libellés descriptifs (Comédie, Drame, ...). Au moment où le formulaire sera validé, une information genre=*valeur* sera passée au programme CGI pour chaque bouton sélectionné par l'utilisateur. Ce programme est bien entendu supposé connaître la signification des codes qu'il reçoît.

TYPE=RADIOBOX Comme précédemment, on donne le choix entre plusieurs valeurs, mais ce choix est maintenant exclusif. Par exemple on n'autorise qu'un seul pays producteur.

```
France     : <INPUT TYPE=RADIO NAME='pays' VALUE='FR' CHECKED>
Etats-Unis : <INPUT TYPE=RADIO NAME='pays' VALUE='US'>
Allemagne  : <INPUT TYPE=RADIO NAME='pays' VALUE='DE'>
Japon      : <INPUT TYPE=RADIO NAME='pays' VALUE='JP'>
```

L'attribut CHECKED (sans valeur associée) permet de pré-sélectionner un des choix. Il est particulièrement utile pour les champs RADIO mais peut aussi être utilisé avec les champs CHECKBOX.

TYPE=SUBMIT Ce champ correspond en fait à un bouton qui valide la saisie et déclenche le programme CGI sur le serveur. En principe il n'y a qu'un seul bouton SUBMIT, mais on peut en utiliser plusieurs, chacun étant caractérisé par un attribut NAME auquel on associe une valeur spécifique.

```
<INPUT TYPE=SUBMIT VALUE='Valider' NAME='bouton1'>
```

La valeur de l'attribut VALUE est ici le texte à afficher. Au lieu de présenter un bouton simple, on peut utiliser un image quelconque, avec le type IMAGE. Par exemple :

```
<INPUT TYPE=IMAGE SRC='bouton.gif'>
```

TYPE=RESET Ce type est complémentaire de SUBMIT. Il indique au navigateur de réinitialiser le formulaire.

TYPE=FILE Il est possible de transmettre des fichiers par l'intermédiaire d'un formulaire. Le champ doit alors contenir le chemin d'accès au fichier sur l'ordinateur du client. Le navigateur associe au champ de type FILE un bouton permettant de sélectionner le fichier à transmettre au serveur pour qu'il le passe au programme. Les attributs sont identiques à ceux du type TEXT.
Voici la définition du bouton permettant de transmettre un fichier contenant l'affiche du film.

```
Fichier : <INPUT TYPE=FILE SIZE=20 NAME='affiche'>
```

La balise <SELECT>

Le principe de ce type de champ est similaire à celui des champs RADIO ou CHECKBOX. On affiche une liste d'options parmi lesquelles l'utilisateur peut faire un ou plusieurs choix. Le champ SELECT est surtout utile quand le nombre de valeurs est élevé.

<SELECT> est un conteneur dans lequel on doit énumérer, avec les balises <OPTION>, tous les choix possibles. La balise <OPTION> a elle-même un attribut VALUE qui indique la valeur à envoyer au programme CGI quand le choix correspondant est sélectionné par l'utilisateur. Voici par exemple un champ <SELECT> proposant une liste de réalisateurs :

```
Metteur en scène :
 <SELECT NAME='realisateur' SIZE=3>
 <OPTION VALUE=1>Alfred Hitchcock
 <OPTION VALUE=2>Maurice Pialat
 <OPTION VALUE=3 SELECTED>Quentin Tarantino
 <OPTION VALUE=4>Akira Kurosawa
 <OPTION VALUE=5>John Woo
 <OPTION VALUE=6>Tim Burton
 </SELECT>
```

L'attribut SIZE indique le nombre de choix à visualiser simultanément. Par défaut, <SELECT> propose un choix exclusif. L'attribut MULTIPLE (sans valeur associée) donne la possibilité de séléctionner plusieurs choix.

Au niveau de la balise <OPTION>, l'attribut SELECTED permet de pré-sélectionner un des choix (ou plusieurs si le champ <SELECT> est de type MULTIPLE). Noter que si on sélectionne 'John Woo', la valeur 5, pour le paramètre nommé realisateur, sera envoyée au programme CGI. Ce dernier est supposé averti de la signification de ces codes.

La balise <TEXTAREA>

Enfin la dernière balise des formulaires HTML est <TEXTAREA> qui permet à l'utilisateur de saisir un texte libre sur plusieurs lignes. Les principaux attributs, outre NAME qui permet de référencer le texte saisi, sont COLS et ROWS qui indiquent respectivement le nombre de colonnes et de lignes de la fenêtre de saisie.

<TEXTAREA> est un conteneur : tout ce qui est placé entre les balises de début et de fin est proposé comme texte par défaut à l'utilisateur. Voici le code HTML du champ destiné à saisir le résumé du film :

```
<TEXTAREA NAME='resume' COLS=30 ROWS=3>Résumé du film
</TEXTAREA>
```

L'exemple 6 donne le code HTML complet pour la création du formulaire de la figure 2.7. Une première remarque est que le code est long, relativement fastidieux à lire, et assez répétitif. En fait la plus grande partie du code est constituée de balises HTML (ouvertures, fermetures, attributs) et pas par le texte spécifique à l'application. HTML est un langage « bavard », et une page HTML devient rapidement très volumineuse,

confuse, et donc difficile à faire évoluer. Un des buts que nous nous fixerons avec PHP est de se doter des moyens d'obtenir un code beaucoup plus concis.

Exemple 6 *ExForm1.html* : *Le formulaire complet*

```
<HTML><HEAD>
<TITLE>Formulaire complet</TITLE>
<LINK REL=stylesheet HREF="films.css" TYPE="text/css">
</HEAD>
<BODY>

<FORM ACTION="http://cartier.cnam.fr:8080/cgi-bin/Films" METHOD=POST>

  <INPUT TYPE="HIDDEN" NAME="monNom" VALUE="ExForm1">

  Titre : <INPUT TYPE=TEXT SIZE=20 NAME="titre">
  Année : <INPUT TYPE=TEXT SIZE=4 MAXLENGTH=4 NAME="annee" VALUE="2000">
  <P>
  Comédie   : <INPUT TYPE=CHECKBOX NAME='genre' VALUE='C'>
  Drame     : <INPUT TYPE=CHECKBOX NAME='genre' VALUE='D'>
  Histoire  : <INPUT TYPE=CHECKBOX NAME='genre' VALUE='H'>
  Suspense  : <INPUT TYPE=CHECKBOX NAME='genre' VALUE='S'>
  <P>
   France     : <INPUT TYPE=RADIO NAME='pays' VALUE='FR' CHECKED>
   Etats-Unis : <INPUT TYPE=RADIO NAME='pays' VALUE='US'>
   Allemagne  : <INPUT TYPE=RADIO NAME='pays' VALUE='DE'>
   Japon      : <INPUT TYPE=RADIO NAME='pays' VALUE='JP'>
  <P>
  Affiche du film : <INPUT TYPE=FILE SIZE=20 NAME='affiche'>
  <P>
  Metteur-en-scène :
  <SELECT NAME='realisateur' SIZE=3>
  <OPTION VALUE=1>Alfred Hitchcock
  <OPTION VALUE=2>Maurice Pialat
  <OPTION VALUE=3 SELECTED>Quentin Tarantino
  <OPTION VALUE=4>Akira Kurosawa
  <OPTION VALUE=5>John Woo
  <OPTION VALUE=6>Tim Burton
  </SELECT>
  <P>
  Résumé :  <TEXTAREA NAME='resume' COLS=30 ROWS=3>Résumé du film
            </TEXTAREA>

  <H1>Votre choix</H1>
  <INPUT TYPE=SUBMIT VALUE='Valider'>
  <INPUT TYPE=RESET VALUE='Annuler'>
</FORM>

</BODY></HTML>
```

Deuxième remarque, qui vient à l'appui de la précédente : malgré tout le code employé, le résultat en terme de présentation graphique est peu attrayant. Pour bien faire, il faudrait (au minimum) aligner les libellés et les champs de saisie, ce qui peut se faire avec un tableau à deux colonnes. Il est facile d'imaginer le surcroît de confusion introduit par l'ajout des balises <TABLE>, <TR>, etc. Là encore l'utilisation de PHP permettra de produire du HTML de manière plus raisonnée et mieux organisée.

Enfin il n'est pas inutile de signaler que l'interface créée par un formulaire HTML est assez incomplète en terme d'ergonomie et de sécurité. Il faudrait pouvoir aider l'utilisateur dans sa saisie, et surtout contrôler que les valeurs entrées respectent certaines règles. Rien n'interdit, dans l'exemple donné ci-dessus, de ne pas entrer de titre pour le film, ou d'indiquer -768 pour l'année. Ce genre de contrôle peut être fait par

le serveur, *après* que l'utilisateur a validé sa saisie, mais ce mode de fonctionnement a l'inconvénient de multiplier les échanges sur le réseau. Une solution plus séduisante est d'effectuer les contrôles par le navigateur, à l'aide d'un langage comme Javascript qui permet de faire de la programmation au niveau du client.

2.3.3 Echanges client/serveur

Tous les paramètres saisis dans un formulaire doivent maintenant être transmis au serveur web ou, plus précisément, au programme CGI sur ce serveur. Inversement ce programme doit produire un document (généralement un document HTML) et le transmettre au navigateur *via* le serveur web.

Un exemple simple

Prenons un exemple très simple qui nous permettra d'illustrer les divers aspects des échanges CGI. Afin de rendre plus souple l'affichage de la liste des films dans une page HTML, on va stocker cette liste dans un simple fichier texte sur le serveur et permettre à l'utilisateur le choisir les films qu'il faut extraire de ce fichier pour les afficher dans le navigateur. Pour cela on va créer un formulaire qui permet d'indiquer les critères de sélection des films à afficher, puis on va associer à ce formulaire un programme CGI qui va :

1. récupérer les paramètres du formulaire ;
2. accéder au fichier des films, et retenir ceux qui satisfont les critères de sélection ;
3. produire une page HTML présentant la liste des films sélectionnés.

L'exemple 7 montre un (petit !) fichier de films qui sera utilisé pour nos exemples.

Exemple 7 *films.txt* : le fichier des films

```
Alien 1979 Scott Ridley 1943
Vertigo 1958 Hitchcock Alfred 1899
Psychose 1960 Hitchcock Alfred 1899
Kagemusha 1980 Kurosawa Akira 1910
Volte-face 1997 Woo John 1946
Titanic 1997 Cameron James 1954
Sacrifice 1986 Tarkovski Andrei 1932
```

De même nous nous contenterons, en attendant de faire beaucoup mieux avec PHP et MySQL, d'un formulaire qui permet de saisir un titre, un intervalle de dates, et d'appliquer un ET ou un OU pour combiner ces deux critères. Voici le code de ce formulaire.

Exemple 8 *ExForm2.html* : *Formulaire pour les critères de recherche*

```
<HTML><HEAD>
  <TITLE>Formulaire pour programme CGI</TITLE>
  <LINK REL=stylesheet HREF='films.css' TYPE='text/css'>
</HEAD><BODY>

<H1>Formulaire pour programme CGI</H1>

<FORM ACTION='http://cartier.cnam.fr:8080/cgi-bin/ExCGI1' METHOD=POST>

Ce formulaire vous permet d'indiquer des paramètres pour
la recherche de films :
  <P>
  Titre :  <INPUT TYPE=TEXT SIZE=20 NAME = 'titre'> <P>
  Année début : <INPUT TYPE=TEXT SIZE=4 NAME='anMin' VALUE=1900>
  Année fin : <INPUT TYPE=TEXT SIZE=4 NAME='anMax' VALUE=2100> <P>

  <B>Comment combiner ces critères. </B>
  ET <INPUT TYPE=RADIO NAME='comb' VALUE='ET' CHECKED>
```

```
  OU <INPUT TYPE=RADIO NAME='comb' VALUE='OU'> ?
  <P>
  <INPUT TYPE=SUBMIT VALUE='Rechercher'>
</FORM>

</BODY></HTML>
```

La figure 2.8 montre l'affichage de ce formulaire avec Netscape.

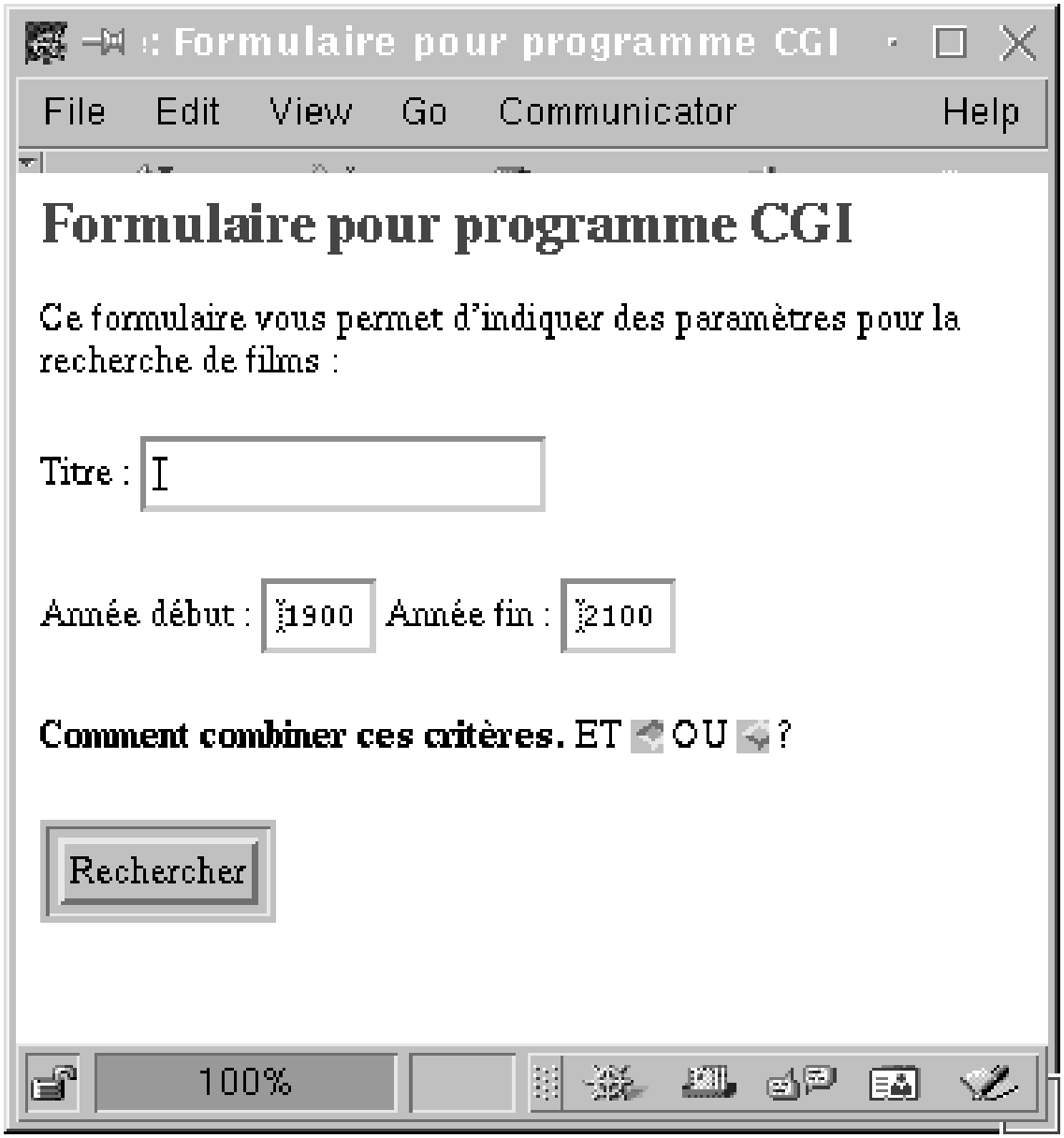

FIG. 2.8 – *Le formulaire de recherche des films*

Transmission au serveur

Au moment où l'utilisateur déclenche la recherche, la navigateur concatène dans une chaîne de caractères une suite de descriptions de la forme `nomChamp=val` où `nomChamp` est le nom du champ dans

le formulaire et *val* la valeur saisie. Les différents champs sont séparés par '&' et les caractères blancs sont remplacés par des '+'.

Par exemple, si on a saisi « Le Saint » dans `titre`, 1978 dans `anMin`, 1980 dans `anMax` et le `ET` logique, la chaîne est constituée par :

```
titre=Le+Saint&anMin=1978&anMax=1980&comb=ET
```

Il existe alors deux manières de transmettre cette chaîne au serveur, selon que la méthode utilisée est `GET` ou `POST`.

1. **Méthode** `GET` : la chaîne est placée à la fin de l'URL appelée, après un caractère '?'. Donc on obtiendrait dans ce cas :

   ```
   cgi-bin/ExCGI1?titre=Le+Saint&anMin=1978&anMax=1980&comb=ET
   ```

2. **Méthode** `POST` : la chaîne est transmise séparément de l'URL.

La méthode `POST` est généralement préférée car elle évite d'avoir à gérer des URL très longues.

Actions du serveur

Quand le serveur reçoit une requête CGI, il lance le programme, et lui transmet un ensemble de paramètres correspondant non seulement aux champs du formulaire, mais également à diverses informations relatives au client qui a effectué la requête. On peut par exemple savoir si on a affaire à Netscape ou à Internet Explorer. Ces transmissions de paramètres se font essentiellement par l'une des trois méthodes suivantes.

- **Variables d'environnement**. Le serveur, avant d'exécuter le programme CGI, initialise des variables d'environnement qui peuvent être récupérées par ce programme. Quand la méthode utilisée est `GET`, une de ces variables (`QUERY_STRING`) contient la liste des paramètres issus du formulaire. Les variables les plus importantes sont décrites dans la table 2.1.
- **Entrée standard** (*stdin*). Quand la méthode est `POST`, les paramètres sont transmis sur l'entrée standard du programme.
- **Sortie standard** (*stdout*). Le programme CGI est responsable de la production d'un document, HTML ou autre, qui doit être renvoyé au navigateur du client. Ce document, y compris les en-têtes du protocole HTTP qui sont habituellement pris en charge par le serveur, doit être transmis sur la sortie standard du programme.

Une des premières tâches du programme est de décoder la chaîne contenant les paramètres pour en extraire les valeurs des champs.

Transmission du serveur au client

Le programme CGI doit écrire le résultat sur sa sortie standard *stdout* qui, par un mécanisme de redirection, communique directement avec l'entrée standard *stdin* du serveur. Le serveur transmet alors ce résultat au client.

Ce résultat peut être n'importe quel document multimédia, ce qui représente beaucoup de choses, depuis le simple texte ascii jusqu'à la vidéo. Dans le cas où la requête d'un client se limite à demander au serveur de lui fournir un fichier, le serveur se base sur l'extension de ce fichier pour déterminer son type.

Conformément au protocole HTPP, il faut alors transmettre ce type dans l'en-tête, avec la clause `Content-type:` *typeDocument*, pour que le navigateur sache comment décrypter les informations qui lui proviennent par la suite. Pour un fichier HTML par exemple, l'extension est le plus souvent *.html*, et la valeur de *typeDocument* est `text/html`.

Dans le cas d'une communication CGI, le serveur reste totalement neutre – il n'a pas d'extension de fichier sur laquelle s'appuyer – et transmet tel quel au client le résultat que lui fournit le programme CGI.

Variable d'environnement	Description
REQUEST_METHOD	Méthode de transmission des paramètres (GET, POST, etc).
QUERY_STRING	Une chaîne de caractères contenant tous les paramètres de l'appel en cas de méthode GET. Cette chaîne doit être décodée (voir ci-dessus), ce qui constitue l'aspect le plus fastidieux du traitement.
CONTENT_LENGTH	Longueur de la chaîne transmise sur l'entrée standard, en cas de méthode POST.
PATH_INFO	Informations sur les chemins d'accès menant par exemple vers des fichiers que l'on souhaite utiliser.
HTTP_USER_AGENT	Type et version du navigateur utilisé par le client.
REMOTE_ADDR	Adresse internet du client.
REMOTE_HOST	Nom de la machine du client.
REMOTE_USER	Nom du client, pour les sites sécurisés avec htaccess (voir annexe A).
REMOTE_PASSWORD	Mot de passe du client, pour les sites sécurisés.

TAB. 2.1 – *Variables d'environnement*

Ce dernier doit prendre en charge l'en-tête du document. Un début typique pour un programme CGI est donc d'écrire l'en-tête d'un fichier HTML sur *stdout*. Par exemple (langage C) :

```
printf ("Content-type: text/html\n\n");
printf ("<TITLE>Resultat</TITLE>");
printf ("<BODY BGCOLOR='white'>");
printf ("<H1><CENTER>Resultat</CENTER></H1>");
```

Noter les deux \n qui, pour d'obscures raisons, semblent nécessaires pour insérer une ligne vide après la clause Content-type. Pour le reste, le programme doit continuer à afficher la suite de la page HTML avec la fonction *printf*.

Un progamme CGI écrit en C

Voici maintenant un exemple complet de service basé sur CGI. Ce service – très simple – est basé sur le formulaire HTML de l'exemple 8. On peut donc saisir le titre du film et la tranche d'années souhaitée, puis demander l'exécution. Ces paramètres sont alors transmis au programme *ExCGI1* qui se charge d'extraire les films du fichier *films.txt*. Voici le code complet de ce programme C.

Exemple 9 *ExCGI1.c* : Programme CGI en C

```
#include <stdio.h>
#include <stdlib.h>
#define MAXL 132

char*  ExtraitArg (char* ligne, char *arg);
```

```
int main(int argc, char* argv[])
{
  int nbFilms=0, anMin, anMax=0, anneeNaissance;
  char titre[40], film[40], nom[40], prenom[40], comb[3];
  int annee, lg;
  FILE *films;
  char ligne [MAXL], arg[MAXL];
  short bonTitre=0, bonnePeriode=0;

  /* On annonce qu'on va retourner un fichier HTML */

  printf ("Content-type: text/html\n\n");
  printf ("<HEAD><TITLE>Résultat de la recherche</TITLE>");
  printf ("</HEAD><BODY bgcolor=white>");
  printf ("<H1><CENTER>Résultat de la recherche</CENTER></H1>");

  /************************************************
              Phase 1: extraction des paramètres
  ************************************************/

  /* Dans la chaîne 'ligne', on place les paramètres
     provenant du formulaire                        */

  lg    = atoi (getenv("CONTENT_LENGTH"));
  fgets  (ligne, lg+1, stdin);
  printf ("<B>Paramètres :</B> %s<P>\n", ligne);

  /* Avec la fonction ExtraitArg, on décrypte la chaîne
     pour en extraire les valeurs saisies par l'utilisateur */

  strcpy (ligne, ExtraitArg (ligne, titre));
  strcpy (ligne, ExtraitArg (ligne, arg));  anMin = atoi(arg);
  strcpy (ligne, ExtraitArg (ligne, arg)); anMax = atoi(arg);
  strcpy (ligne, ExtraitArg (ligne, comb));

  printf ("<HR><B>Recherche:</B>Titre= %s %s année de %d à %d<br><HR>",
            titre, comb, anMin, anMax);

  /************************************************
             Phase 2: recherche et affichage
  ************************************************/

  /* Ouverture du fichier des films, et boucle sur les lignes */

  films = fopen ("films.txt", "r");
  while (fgets (ligne, MAXL, films))
  {
    /* Décryptage de chaque ligne    */

    sscanf (ligne, "%s %d %s %s %d",
              film, &annee, nom, prenom, &anneeNaissance);

    /* Test du fil courant pour voir s'il satisfait les
        critères de sélection
    */

    if (!strcmp (film, titre)) bonTitre = 1;
    if (annee >= anMin && annee <= anMax) bonnePeriode=1;
```

```
    /* Si oui : affichage du film dans la page HTML  */

    if (  (!strcmp ("ET", comb) && bonTitre && bonnePeriode)
       || (!strcmp ("OU", comb) && (bonTitre || bonnePeriode)))
    {
      nbFilms++;
      printf ("<B>Film : </B> %s, %d, de %s %s <br>",
                        film, annee, prenom, nom);
    }
    bonTitre=0; bonnePeriode=0;
  }
  fclose (films);

  if (!nbFilms) printf ("<B>Désolé : aucun film !</B>");

  return 1;
}
/** Fin du programme    */

  /*********************************************
            Fonction ExtraitArg
  *********************************************/

char*  ExtraitArg (char* ligne, char *arg)
{
  int pos = 0, posArg=0;

  /* Recherche du signe '=', puis extraction jusqu''&' */

  while (ligne[pos++] != '=');
  while ( ligne[pos] != '&' &&  ligne[pos] != '\0')
      arg[posArg++] =  ligne[pos++];

  arg[posArg] = '\0';

  return &(ligne[pos+1]);
}
```

Beaucoup de détails sont laissés de côté, et notamment tous les contrôles qui seraient souhaitables pour vérifier que les valeurs passées sont correctes.

Pour l'essentiel, ce programme écrit sur la sortie standard, avec des fonctions `printf`, un fichier HTML. Ce fichier comprend des parties statiques pour la mise en forme, et une partie dynamique qui présente le résultat de la requête. Voici quelques explications sur les principaux aspects.

Tout d'abord on va stocker dans une chaîne de caractères `ligne` la liste des paramètres qui est fournie sur l'entrée standard (`stdin` en C). On récupère la valeur de la variable d'environnement `CONTENT_LENGTH`.

```
lg    = atoi (getenv("CONTENT_LENGTH"));
fgets  (ligne, lg+1, stdin);
```

Vient ensuite la phase la moins agréable : il faut extraire les arguments qui sont dans la chaîne `ligne`. Ces arguments constituent une liste de paires *`nom=valeur`* (voir ci-dessus). Nous utilisons une fonction très simple, *ExtractArg*, pour récupérer les valeurs en question.

```
strcpy (ligne, ExtraitArg (ligne, titre));
strcpy (ligne, ExtraitArg (ligne, arg));  anMin = atoi(arg);
strcpy (ligne, ExtraitArg (ligne, arg)); anMax = atoi(arg);
strcpy (ligne, ExtraitArg (ligne, comb));
```

Finalement on parcourt les lignes du fichier des films en effectuant le test par rapport aux valeurs saisies dans le formulaire HTML.

```
if (!strcmp (film, titre)) bonTitre = 1;
if (annee >= anMin && annee <= anMax) bonnePeriode=1;

if (  (!strcmp ("ET", comb) && bonTitre && bonnePeriode)
   || (!strcmp ("OU", comb) && (bonTitre || bonnePeriode)))
{
  nbFilms++;
  printf ("<B>Film : </B> %s, %d, de %s %s <br>",
                   film, annee, prenom, nom);
}
```

Une des caractéristiques essentielles des programmes CGI est d'écrire sur la sortie standard (ici avec `printf`) un document HTML qui comprend des données provenant de sources extérieures (ici un fichier). Bien entendu il est possible d'utiliser des commandes HTML arbitrairement complexes, ce qui permettrait par exemple de présenter les films sélectionnés sous forme de tableau comme dans l'exemple 3. Cela entrainerait une lourdeur accrue du code qui est déjà assez touffu.

Le programme donné ci-dessus fonctionne et peut être testé sur notre site. Cela dit, s'il est suffisant pour illustrer les principaux aspects de l'interface CGI, il présente beaucoup trop de faiblesses pour être utilisé dans une application réelle. Par exemple il faudrait prendre en compte les caractères blancs dans les titres des films qui sont codés par des '+' dans le passage au programme CGI, ainsi que les possibles caractères spéciaux ou accentués. Il serait bon également de tester que la méthode utilisée est bien `POST`, avec un test comme :

```
if (strcmp (getenv("REQUEST_METHOD"), "POST"))
{
  printf ("<B>Seule la méthode POST est acceptée !</B>\n");
  exit (1);
}
```

De plus la fonction *ExtraitArg* ne présente pas les plus grandes garanties de robustesse. D'ailleurs le langage C n'est pas le plus approprié pour la programmation CGI, Perl paraissant plus adapté. Quel que soit le langage utilisé, si vous souhaitez vraiment faire du CGI (ce qui ne devrait pas être nécessaire puisque nous verrons plus loin que la solution PHP offre une solution plus simple et tout aussi puissante), il faut de toutes manières utiliser une des nombreuses bibliothèques déjà disponibles, qui fournissent tout un ensemble de fonctions prêtes à l'emploi pour la récupération des paramètres et la production de HTML.

2.3.4 Quelques limites de la programmation CGI

Il n'y a pas vraiment de limites à ce que l'on peut faire avec un programme CGI puisque tous les outils (bases de données, logiciels bureautique, génération d'images) interfacés avec un langage de programmation comme le C ou Perl peuvent être intégrés. Cela dit il est bon de souligner quelques inconvénients de cette architecture, qui justifient le recours à une solution plus souple basée sur un langage comme PHP.

Passage des paramètres

Tout d'abord, comme nous l'avons vu avec l'exemple précédent, le passage des paramètres au programme est relativement pénible à gérer, même si les nombreuses fonctions existantes amoindrissent cet inconvénient. Idéalement, on souhaiterait disposer, directement dans le programme, des variables correspondant aux champs du formulaire HTML : c'est ce que fait PHP.

Faible intégration avec HTML

Ensuite le passage d'une solution HTML à une solution CGI est de type « tout ou rien ». La programmation CGI n'est pas une extension de HTML qui permettrait de recourir à ce langage pour la partie des pages qui correspond à du texte fixe. Au contraire, PHP s'intègre aux documents HTML et permet de passer souplement d'un langage à un autre.

Sessions

Enfin une caractéristique essentielle de la programmation CGI (et de la programmation web en général) est son mode *déconnecté*. Le serveur ne mémorise pas les demandes successives effectuées par un client particulier, et ne peut donc pas tenir compte de l'historique des échanges pour améliorer la communication avec le client.

Un exemple familier qui illustre bien cette limitation est *l'identification* d'un visiteur sur un site. Cette identification se base sur un formulaire pour fournir un nom et un mot de passe et le serveur, s'il valide ce code d'accès, peut alors donner le droit au visiteur de consulter des parties sensibles du site. Le problème est que lors de la prochaine connexion (qui peut avoir lieu dans les secondes qui suivent la première !) le serveur ne sera pas en mesure de reconnaître que le client s'est déjà connecté, et devra lui demander à nouveau nom et mot de passe.

L'absence de connexion permanente entre client et serveur web est un gros obstacle pour la gestion de *sessions* qui se dérouleraient dans un contexte riche, constitué d'un ensemble d'informations persistant tout au long des communications client/serveur. Par exemple on voudrait stocker non seulement le nom et le mot de passe, mais aussi l'historique des accès au site afin de guider plus facilement un visiteur habitué. Plusieurs solutions, plus ou moins satisfaisantes, existent pour tenter de résoudre ce problème.

1. **Variables d'environnement**. Le protocole HTTP s'est enrichi d'un ensemble de variables qui sont systématiquement transmises du navigateur au serveur. Par exemple les variables `REMOTE_USER` et `REMOTE_PASSWORD` peuvent être définies au moment d'un accès à un site sécurisé. Cette méthode est globalement insatisfaisante : les variables disponibles sont en nombre limité, fixées à l'avance, et en partie dépendantes du navigateur utilisé.
2. **Paramètres dans l'URL**. Les informations constituant le contexte de la session peuvent être placées systématiquement dans l'URL, après le caractère '?' qui indique le début des paramètres. Le programme peut ensuite les récupérer dans la variable `QUERY_STRING`. Cette technique est limitée par la taille des informations qui peuvent être placées dans une URL, ainsi que par le manque de confidentialité.
3. **Cookies**. La dernière possibilité, et la plus utilisée, est de recourir aux *cookies*. Essentiellement, un *cookie* est une donnée, représentable sous la forme habituelle *`nom=valeur`*, que le navigateur conserve pour une période déterminée à la demande du serveur. Cette demande doit être effectuée dans un en-tête HTTP avec une instruction `Set-Cookie`. Par la suite, les cookies stockés par un navigateur sont envoyés au serveur dans une variable d'environnement `HTTP_COOKIE`. Nous ne détaillons pas le format d'échange qui est relativement complexe à décrypter.

Il faut noter qu'un *cookie* ne disparaît pas quand le navigateur est stoppé puisque il est stocké dans un fichier. Netscape par exemple écrit les *cookies* dans un fichier *cookies* du répertoire *netscape* : il est toujours intéressant de consulter ce fichier pour voir qui a laissé traîner des informations chez vous !

On peut considérer comme relativement indélicat cette technique qui consiste à écrire des informations sur le disque dur d'un client, et à son insu, mais les *cookies* offrent le moyen le plus simple, et aussi le plus puissant, pour créer un contexte persistant aux différents échanges client/serveur. Nous les utiliserons le moment venu pour gérer les sessions, mais par l'intermédiaire de fonctions PHP qui fournissent une interface simple et facile à utiliser.