Université libre de Bruxelles
Service de mécanique analytique & CFAO
Janvier 2001

Internet

et ses technologies

Pascal Francq, Thomas L'Eglise, Benoît Lagrange & Laurent Auquière

Table des matières

Préface

Internet connaît un essor fantastique ces dernières années. Aujourd'hui, le nombre de personnes utilisant les technologies liées à Internet augmente sans cesse.

Ce phénomène a remis en cause la vision dans de nombreux domaines :

- la communication entre personnes distantes devient de plus en plus aisée ;
- l'échange d'informations est simplifié grâce à l'utilisation de supports informatiques ;
- l'éducation et la recherche peuvent s'appuyer sur des technologies pour s'internationaliser ;
- la manière de gérer une entreprise a énormément évolué.

Si l'utilisation d'internet devient de plus en plus courante dans la vie de tous les jours, les différentes technologies mises en oeuvre sont rarement comprises.

Le but de cet ouvrage est de passer en revue les principales technologies rencontrées aujourd'hui afin et de donner une vision d'ensemble de l'aspect technique tournant autour d'internet.

Première partie

Les réseaux

Chapitre 1

Architectures de réseaux

Le partage d'informations entre plusieurs entités, qu'il s'agisse de personnes ou d'ordinateurs, nécéssite la mise en relation de supports pour acheminer celles-ci : c'est l'objet des réseaux.

1.1 Définition d'un réseau

Un réseau est composé d'un ensemble de ressources connectées entre-elles, le but principal étant de les faire communiquer. Pour qu'un réseau existe, il faut donc :
- au moins deux machines voulant communiquer ;
- une méthode permettant à des machines de se contacter ;
- un ensemble de règles permettant une communication, c'est-à-dire un échange d'informations.

1.2 Historique des réseaux

Le premier réseau date de 1833 et est dû à Morse : il s'agit bien évidemment du télégraphe. Entre 1874 et 1876, Bell inventa le téléphone. Il s'agit des prémisses de ce que l'on appelle les réseaux de télécommunications cablés.

Dans un autre domaine, le concept des ondes radios, introduit par Hertz en 1880, initia également un réseau. Marconi expérimenta en 1897 la possibilité d'échanger des informations de type télégraphique (signaux «longs» et «courts») par ondes hertziennes. Puis, en 1906, De Forest introduisit la modulation du son et donc permit la transmission de celui-ci par ondes hertziennes. C'était le début des réseaux de radio-diffusion et de télévision.

La notion de réseau dépasse donc largement le concept d'internet et même celui de la communication entre ordinateurs de manière générale.

1.3 Les types de réseaux

Il existe historiquement trois types de réseaux :
- les réseaux de télécommunications ;
- les réseaux des cablo-opérateurs ;
- les réseaux informatiques.

1.3.1 Les réseaux de télécommunications

Initialement dédiés à l'application téléphonique, les réseaux de télécommunications sont caractérisés par des contraintes très restrictives. La téléphonie est en effet une application très complexe nécessitant :
- une liaison unique et exclusive entre deux postes (liaison point-à-point) ;
- un transfert quasiment en temps réel de l'information afin d'assurer la synchronisation ;

– un temps de réponse très court afin d'éviter les échos.

La technique de transfert utilisée est la commutation de circuits : au début du dialogue, un circuit est fixé et réservé pour toute la durée de la communication.

1.3.2 Les réseaux de cablo-opérateurs

Il s'agit des réseaux utilisés pour la télévision câblée par exemple. Ils sont caractérisés par une grande bande passante, c'est-à-dire qu'ils peuvent transporter une très grande quantité de signaux et par une liaison « broadcasting », c'est-à-dire un émetteur et plusieurs récepteurs.

On commence aujourd'hui à voir apparaître l'utilisation de ces réseaux pour Internet ; en effet, de plus en plus de distributeurs de télédistribution proposent cette option parmi leurs services.

1.3.3 Les réseaux informatiques

Les réseaux informatiques sont nés du besoin de faire communiquer des terminaux distants avec un site central, puis des ordinateurs entre eux, et enfin de connecter des machines terminales telles que des stations de travail avec leur serveur. Donc, les réseaux informatiques doivent supporter aussi bien les liaisons point-à-point que les liaisons « broadcasting ».

1.3.3.1 Catégories basées sur la taille

Cinq catégories de réseaux d'ordinateurs différents sont généralement considéré en se basant sur la taille de ceux-ci :

– le bus, interne aux ordinateurs, utilisé pour relier les processeurs, les mémoires, les entrées-sorties du calculateur d'un multiprocesseur ; la distance maximale entre deux points de connexion est très faible ;
– les structures d'interconnexion qui permettent de relier, sur des distances faibles, différents calculateurs entre eux, par exemple les pré- et post- processeurs d'ordinateurs vectoriels ;
– les Local Area Network (LAN) ou réseaux locaux qui relient des ordinateurs au sein d'un bâtiment ou d'un site ;
– les Metropolitan Area Network (MAN) ou réseaux métropolitains qui permettent de connecter des ordinateurs se situant dans des sites différents ;
– les Wide Area Network (WAN) ou réseaux étendus dont le plus connu est internet.

1.3.3.2 Catégories basées sur la fonctionnalité

Les réseaux d'ordinateurs (LAN, MAN et WAN) sont aujourd'hui souvent classé suivant les fonctionnalités qu'ils implantent concernant le partage de l'information :

Intranet L'ensemble des ressources ne sont disponibles qu'à l'intérieur de l'entreprise ou de l'organisation, c'est-à-dire uniquement aux machines connectées au réseau interne.

Extranet Des personnes peuvent accéder à des ressources d'une entreprise ou une organisation en n'étant pas connecté directement sur le réseau interne.

Internet Les ressources sont disponibles pour l'ensemble des machines qu'elles soient connectées ou non directement au réseau interne de l'entreprise ou de l'organisation.

1.3.3.3 Catégories basées sur la topologie

La topologie d'une réseau désigne la manière dont les machines sont physiquement reliées entre-elles :

Bus Un câble unique ouvert (FIG. 1.1).

Ring Un câble unique fermé (FIG. 1.2).

Star Une machine centrale avec laquelle toutes les autres ont une connexion point-à-point (FIG. 1.3).

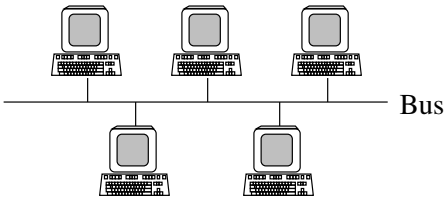


FIG. 1.1 – Topologie bus

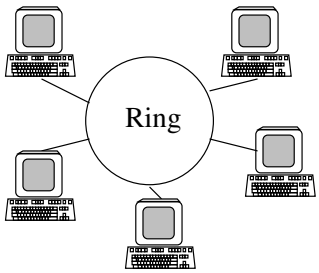


FIG. 1.2 – Topologie Ring

Mesh Toutes les machines ont une connexion point-à-point avec toutes les autres (FIG. 1.4).

Cellular Le réseau est divisé en cellules, chacune ayant sa propre topologie.

1.3.3.4 Les matériels de connexion

Afin de construire un réseau comme internet, il est nécessaire de déployer différents matériels. Les principaux sont :

Modem ou Network Interface Card (NIC) Interface permettant à un ordinateur de recevoir ou d'envoyer des informations via un réseau.

Connecteur Ils permettent de relier physiquement les ordinateurs aux différents réseaux.

Répétiteur La plupart des moyens de transport ont des limites physiques. Les répétiteurs permettent de réamplifier le signal et ainsi de rendre possible le transport d'informations sur de grandes distances .

Hub Il s'agit d'un concentrateur actif. Le hub peut être considéré comme l'élément auquel les ordinateurs sont branchés et qui implante une topologie particulière.

Bridge Il s'agit d'un concentrateur actif et intelligent capable de prendre des décisions concernant le transport des données en se basant sur une partie de l'information que celles-ci véhiculent.

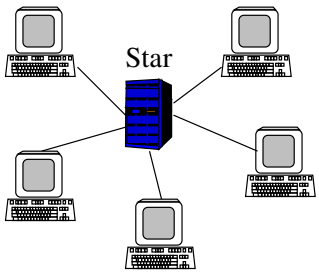


FIG. 1.3 – Topology Star

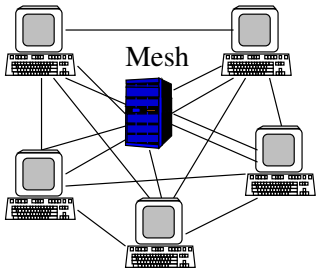


FIG. 1.4 – Topologie Mesh

Routeur Un routeur permet de connecter des réseaux différents entre-eux. En particulier, ces réseaux peuvent utiliser des topologies et des protocoles[1] différents.

La FIG. 1.5 présente un exemple de réseau avec les différents matériels introduits ci-dessus.

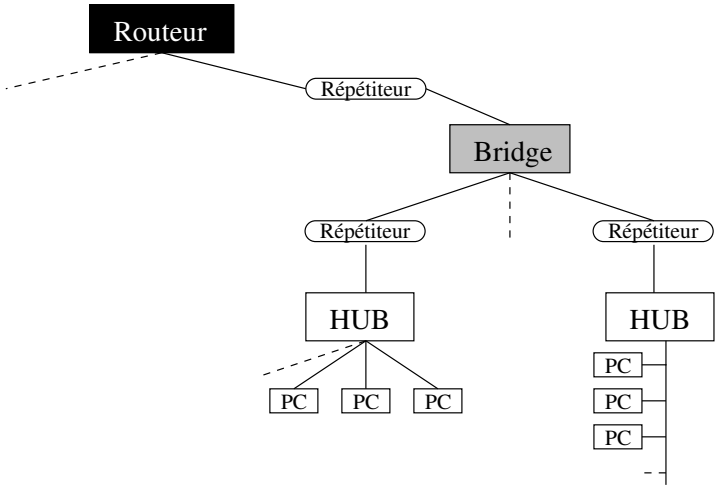


FIG. 1.5 – Exemple de réseau

1.4 Architecture client/serveur

Dans une architecture client/serveur, le serveur est une machine qui fournit un certain nombre de services. Un client fait une requête de service au serveur qui lui renvoie le résultat de sa requête, si le client possède les droits pour utiliser le service demandé (FIG. 1.6). La communication entre serveur et client se fait au moyen de réseaux et de protocoles que nous détaillerons aux chapitres suivants.

La nature des services peut varier suivant les besoins applicatifs. On peut citer comme application client/serveur :

- les serveurs web qui permettent le partage du contenu de documents à travers le réseau internet ;
- les bases de données (Oracle, Sybase, MySQL, etc.) qui permettent à différents clients de manipuler une source unique de données ;
- l'environnement graphique X-Window qui permet à des utilisateurs d'exécuter des programmes situés sur une machine distante et nécessitant une interface graphique.

[1] Un protocole est un ensemble de régles permettant à des machines de communiquer.

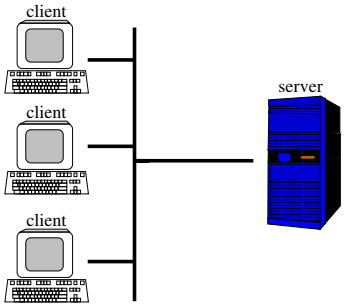


FIG. 1.6 – Architecture Client/Serveur

Remark: *Chaque application permet l'utilisation de plusieurs protocoles afin d'assurer un maximum de flexibilité quant à l'environnement dans lequel elles évolueront. Ceci dit, dans la pratique, c'est le protocole TCP/IP (voir chapitre 3) qui est utilisé comme moyen de communication dans une architecture client/serveur permettant ainsi le partage d'informations à travers des systèmes hétérogènes.*

Chapitre 2

Open Systems Interconnection

2.1 Principe du modèle OSI

Le modèle OSI, mis au point à partir de 1977, est d'une importance considérable, tant il lui est fait référence. En effet, l'ensemble des protocoles utilisés dans le monde des télécommunications, que ce soit pour le transport de la voix traditionel ou celui des données, est repris dans cette norme. Il s'agit d'un outil qui a été développé pour aider les concepteurs de protocoles à comprendre les différents éléments du problème de la communication, et à planifier une suite complète de protocoles.

Il est composé de sept couches indépendantes (FIG. 2.1), chacune fournissant à la couche supérieure un certain nombre de services tout en utilisant les services de la couche inférieure. Pour un étude complète su modèle OSI, le lecteur est invité à consulter [4].

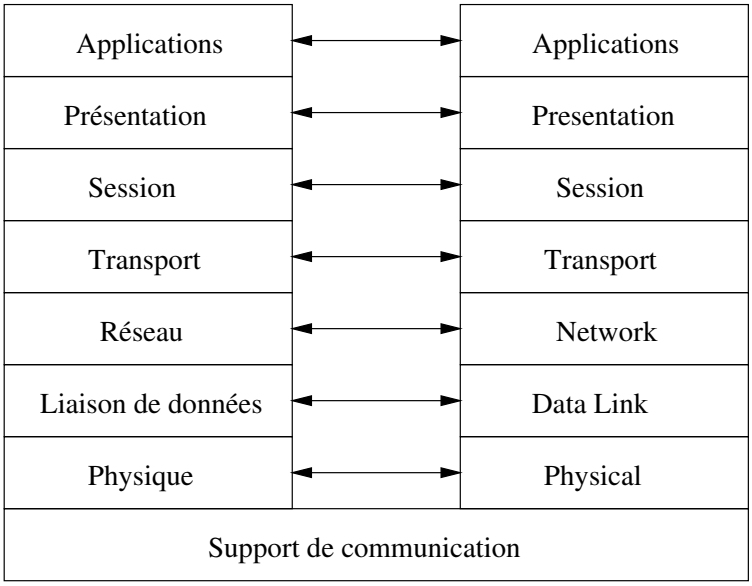


FIG. 2.1 – Le Modèle OSI

Même si certains protocoles récents ne correspondent plus exactement à ce modèle, sa terminologie reste cependant toujours en vigueur et, lorsque les spécialistes des réseaux parlent de la « couche i », ils font généralement référence à la couche i du modèle OSI.

Chaque couche est composée d'une partie service et d'une partie protocole, c'est-à-dire l'ensemble des règles permettant à la partie service d'être réalisée. Schématiquement, on peut imaginer que chaque

couche reprend le protocole de la couche inférieure et rajoute de l'information au début et à la fin du paquet d'informations [1] pour définir son propre protocole.

2.2 Utilité de la structuration en couches

L'intérêt de l'organisation en couches, de manière générale, résulte d'un principe scientifique simple : le message que le logiciel destinaire de la couche i reçoit, doit être exactement le même que celui envoyé par le logiciel émetteur de la couche i. D'un point de vue mathématique, si la transformation qu'un protocole applique à une trame avant de l'envoyer est représenté par T, elle doit pouvoir s'appliquer de manière inverse lorsque cette trame est reçue, c'est-à-dire que la transformation doit être T^{-1}.

Ce principe est puissant, car il simplifie la conception et le tests des protocoles. Il évite en particulier qu'une modification dans une couche n'affecte les autres couches, ce qui permet de développer des protocoles spécifiques à une couche donnée, sans devoir se soucier des protocoles des couches supérieures et inférieures.

2.3 Techniques utilisées par les protocole

De manière générale, un protocole implante un ensemble des règles permettant le transport de l'information entre deux ressources. Certains protocoles ne se contentent pas uniquement de détecter les erreurs, mais essaient, dans la mesure du possible, de corriger ou de contourner les problèmes complexes de communication.

2.3.1 Remise en séquence

Certains protocoles permettent que les paquets d'une même trame puisse emprunter des chemins différents et donc arriver dans le désordre.

Pour gérer ce problème, ces protocoles se servent d'une numérotation. En particulier, lorsque le logiciel destinataire se rend compte que le paquet reçu est hors séquence, il l'ajoute à une liste interne, tandis que si le paquet reçu est celui attendu, il le fournit immédiatement à la couche supérieure.

2.3.2 Eliminiation des paquets dupliqués

Il arrive souvent que, à la suite d'un fonctionnement défecteux du matériel, un logiciel d'émission soit amené à réémettre des paquets.

La numérotation permet également au logiciel destinataire de le remarquer et d'ainsi ne pas fournir de paquets dupliqués à la couche supérieure.

2.3.3 Retransmission des paquets perdus

Il arrive également que des paquets soient perdus ou corrompus lors de leur transfert. Un système « d'accusé de réception » est implanté pour résourde ce type de problème : il doit être envoyé par le logiciel destinataire lorsque le paquet est arrivé correctement. Le logiciel émetteur enclenche un temporisateur chaque fois qu'un paquet est envoyé et si aucun accusé de réception n'est arrivé avant la fin de la temporisation, le paquet est réémis.

2.3.4 Répétitions dues à des délais excessifs

Dans un systèmes basé sur la communication par paquets, le mécanisme de stockage temporaire et de retransmission peut constituer une source de délai. Lorsqu'un paquet arrive dans un commutateur, il est stocké dans une file d'attente. Si les paquets arrivent plus vite que le commutateur ne peut les propager, la

[1] Lorsqu'il faut transporter de l'information, celle-ci est divisée en paquets, chaque paquet représentant l'unité de données transportable.

taille de la file d'attente augmente et les délais peuvent devenir longs. Des délais excessifs peuvent conduire à des erreurs de répétition : un ancien paquet perturbe les communications qui suivent. Considérons par exemple un ancien paquet 3 qui arrive, à cause d'une duplication, après l'établissement d'une nouvelle communication entre les deux machines. Dès lors, la machine destinataire recevra deux paquets numérotés 3 de l'émétteur sans pouvoir dire à laquelle des communications chacun d'eux appartient.

Afin de résoudre ce problème, les protocoles utilisent un identificateur unique par session, comme l'heure d'établissement par exemple. Le protocole logiciel écarte tout paquet entrant qui ne contient pas un identificateur correct, c'est-à-dire qui n'appartient à aucune communication ouverte.

2.3.5 Contrôle de flux

Les ordinateurs ne sont pas capables de travailler tous au même rythme, et un débordement de données peut se produire lorsqu'un émetteur envoie plus vite les données à travers le réseau que le récepteur n'est capable de traiter, ce qui provoque la perte de données.

Pour éviter cela, les protocoles utilisent la notion de « fenêtre glissante ». Au début de la communication, l'émetteur et le récepteur fixent une taille de fenêtre, qui est la taille maximum de données pouvant être envoyée avant qu'arrive le premier accusé de réception.

Le mécanisme de fenêtre glissante permet d'augmenter considérablement le débit. Il est évident que le débit ne peut être augmenté indéfiniment par ce système, car la capacité du réseau sous-jacent en fixe une borne supérieure.

2.3.6 Prévention de congestion

La congestion – un nœud du réseau reçoit tellement de paquets qu'il est incapable de traiter, que l'ensemble du réseau s'en trouve ralenti, – est un problème majeur dans les systèmes à commutation par paquets. Les protocoles essaient de l'éviter, en surveillant le réseau et en réagissant au plus vite en début de congestion. Il existe deux approches pour détecter une telle congestion :

- faire en sorte que les commutateurs de paquets avertissent les émetteurs, lorsqu'une congestion se produit ;
- se servir de la perte de paquets comme estimation de la congestion.

Dans les réseaux modernes, la perte des paquets est généralement utilisée comme estimation, car la fiabilité des différents matériels permet de considérer que la proportion de pertes de paquets due à une panne matérielle est faible.

2.4 Descriptions des niveaux

2.4.1 Niveau 1 : la couche physique

C'est la norme ISO10022 qui définit les services devant être rendus par cette couche afin de transporter correctement les éléments binaires. Parmi ces services, on trouve :

- les interfaces de connexion des équipements informatiques que l'on appelle des jonctions ;
- les modems qui transforment les signaux binaires produits par les ordinateurs en des signaux binaires de forme sinusoidale pour une propagation de meilleure qualité ;
- les multiplexeurs qui concentrent plusieurs émissions distinctes vers une ligne unique en utilisant toute la bande passante de celle-ci ;
- les nœuds de commutation qui forment le matériel intermédiaire entre l'émetteur et le récepteur ;
- les différents équipements spécifiques du réseau permettant d'assurer la continuité du chemin physique.

On caractérise la qualité d'un réseau en Bit Error Rate (BER) qui représente la moyenne du nombre de bits envoyés pour un bit envoyé par erreur.

2.4.2 Niveau 2 : la couche liaison

Cette couche fournit les moyens fonctionnels et procéduraux nécessaires à l'établissement, au maintient et à la libération des connexions entre entités du réseau, et au transfert des unités de données du service de liaison. Elle est définie par la norme ISO8886.

Le but de cette couche est de corriger autant que possible les erreurs intervenues dans la couche 1. Il n'est évidemment pas possible de corriger toutes les erreurs, mais la norme spécifie que le taux d'erreurs résiduelles doit être négligeable.

C'est également dans la couche 2 que sont définies les règles pour qu'un même support physique puisse être partagé entre plusieurs stations.

Il existe plusieurs autres normes définies. L'ISO a aussi fourni une normalisation dans le domaine des réseaux locaux, en particulier les normes ISO8802.x.

2.4.3 Niveau 3 : la couche réseau

Le rôle de la couche réseau est de fournir d'une part les moyens d'établir, de maintenir et de libérer des connexions de réseau entre des systèmes ouverts et, d'autre part, de donner les moyens fonctionnels et les procédures nécessaires pour échanger, entre les entités de transport, des unités de réseau.

En d'autres termes, cette couche est chargée d'acheminer correctement les paquets jusqu'à l'utilisateur final. Pour cela, elle possède trois fonctions principales :

- le contrôle de flux, afin notamment d'éviter la congestion, c'est-à-dire l'embouteillage des paquets dans le réseau ;
- le routage, qui est là pour guider les paquets à travers le maillage des nœuds de commutation ; il peut-être centralisé ou distribué suivant les stratégies choisies ;
- l'adressage, qui est nécessaire pour pouvoir identifier de manière unique tous les terminaux connectés sur le réseau. L'ISO a dû prévoir une norme d'adressage permettant de répertorier l'ensemble des équipements terminaux.

Pour développer et mettre en œuvre cette couche, il existe deux grandes philosophies :

- le mode connecté ;
- le mode non connecté.

2.4.3.1 Le mode connecté

Dans ce mode, l'émetteur et le récepteur se mettent d'accord avant la communication : les paramètres et valeurs à utiliser sont fixés une fois pour toute. Parmi les normes les plus connues, citons les normes ISO8208 et X.25 utilisées par les grands réseaux publics.

C'est ce type de mode qui est utilisé par le réseau téléphonique. Lorsqu'un utilisateur forme le numéro d'un poste, avant que celui-ci n'émette de sonneries, toute les ressources nécessaires à la communication sont bloquées. Elles seront libérées lorsqu'un des deux utilisateurs raccrochera son téléphone.

2.4.3.2 Le mode non connecté

Dans ce mode, il n'y a aucune contrainte sur l'émetteur vis-à-vis du récepteur. Un exemple est la norme ISO8473 qui implante le protocole IP du réseau internet.

Lorsqu'un terminal veut échanger des informations avec un autre terminal via le réseau internet, il va découper l'ensemble des informations en paquets qui seront envoyés sur le réseau. Chaque paquet est susceptible d'utiliser un autre « chemin ». Ainsi, les paquets peuvent arriver dans un ordre différent de celui dans lequel ils ont été envoyés.

2.4.4 Niveau 4 : la couche transport

Cette couche doit assurer un transfert de données entre les entités de session de manière transparente, c'est-à-dire indépendamment de la succession des éléments binaires transportés.

Les principales normes existantes sont :

- ISO8072 qui définit le service transport ;

– ISO8073 qui définit le protocole de transport orienté connexion ;
– ISO8602 qui définit le protocole de transport en mode non connecté.

En particulier dans le cas du transport en mode non connecté, le protocole de cette couche doit assurer que les paquets qui peuvent être reçus de manière désordonnée, se retrouvent dans le bon ordre.

2.4.5 Niveau 5 : la couche session

La couche session doit fournir les services nécessaires afin d'organiser et de synchroniser un dialogue entre un émetteur et un récepteur. Il s'agit de l'établissement d'une connexion, de son maintien et de sa libération, ainsi que des moyens pour contrôler les interactions entre les entités de présentation.

C'est la première couche, qui se trouve en dehors de la commutation proprement dite, qui doit ouvrir et fermer les sessions entre utilisateurs. En effet, il est par exemple inutile d'envoyer des informations si personne n'est là pour les récupérer à l'autre extrémité.

Afin d'ouvrir une connexion avec une machine distante, la couche session doit posséder un langage compréhensible par l'autre extrémité, c'est la raison pour laquelle il y a passage obligatoire par la couche application avant l'ouverture d'une session.

Remark: *Certaines architectures utilisent la couche application pour l'ouverture et la fermeture des sessions.*

Un certain nombre de services supplémentaires peuvent être implantés en dehors de ceux de base. Ainsi, la pose de points de resynchronisation, qui permet de fixer un endroit où une conversation pourrait reprendre en cas de problèmes, ferait également partie des services de la couche session.

La normalisation comprend les documents suivants :

– ISO8326 qui définit le service orienté connexion ;
– ISO8327 qui définit le protocole orienté connexion ;
– ISO9548 qui définit le protocole de session dans un mode sans connexion.

Toutes les questions relatives à la sécurité (authentification à l'aide de mots de passe, par exemple), concernent généralement aussi la couche session.

2.4.6 Niveau 6 : la couche présentation

La couche présentation se charge de la syntaxe des informations que les entités d'application se communiquent. Celle-ci comprend deux aspects complémentaires :

– la représentation des données transférées entre entités d'applications ;
– la représentation de la structure de données et la représentation de l'ensemble des actions effectuées sur cette structure de données.

La couche présentation s'intéresse donc à la syntaxe, alors que la couche application se chargera de la sémantique.

La couche présentation procurant un langage syntaxique commun à l'ensemble des utilisateurs connectés, elle est d'une importance cruciale dans un environnement hétérogène. Les normes suivantes existent aujourd'hui :

– ISO8824 qui définit la syntaxe ASN.1 (Abstract Syntax Notation 1) ;
– ISO8326 qui définit le service orienté connexion ;
– ISO8327 qui donne les spécifications du protocole de présentation orienté connexion ;
– ISO9548 qui définit le protocole de présentation sans connexion.

Cette couche est nécessaire car les ordinateurs utilisent des représentations internes pour les nombres entiers et réels ainsi que pour les caractères alphanumériques.

2.4.7 Niveau 7 : la couche application

Cette couche donne aux entités d'application le moyen d'accéder à l'environnement OSI. En plus de la norme de définition de la couche – la norme ISO9545 – il existe une multitude de normes. La couche application a fortement évolué ces dernières années, et elle est souvent découpée en sous-couches dont la description sort largement du cadre du présent ouvrage.

A titre d'exemple de protocoles se situant au niveau de cette couche , citons la spécification du transfert de fichiers entre deux applications ou encore le protocole qui définit la requête effectuée par une application qui s'exécute sur une machine ainsi que la réponse fournie par l'application sur l'autre machine.

Chapitre 3

TCP/IP et le réseau internet

3.1 Historique du TCP/IP

Au début des années 70, le gouvernement américain voulait interconnecter les différentes universités et administrations fédérales entre elles. Le problème qui se posa directement était l'hétérogenéité des réseaux. En effet, chaque institution utilisait ces propres protocoles, qui dépendaient souvent de la nature des machines utilisées.

Afin de remédier à cela, le protocole TCP/IP fut créé. Le réseau Internet démarra en 1980 quand le DARPA (Defense Advanced Research Projects Agency) commença à convertir les différents protocoles de réseaux en TCP/IP.

Le système d'exploitation UNIX utilisé alors dans toutes les institutions implanta TCP/IP de manière formelle, ce qui contribura largement au succès de ce dernier. Depuis, le protocole TCP/IP a été normalisé, ce qui permet aujourd'hui à tous les systèmes d'exploitation (Linux/UNIX, Windows, Macintosh, OS/2, ...) de pouvoir être interconnectés entre-eux. Pour un étude complète, il faut consulter [4].

3.2 Modèle TCP/IP

TCP/IP se présente sous la forme d'une architecture en couches qui inclut également, sans qu'elle ne soit définie, une interface d'accès au réseau et définit plusieurs protocoles différents (FIG. 3.1).

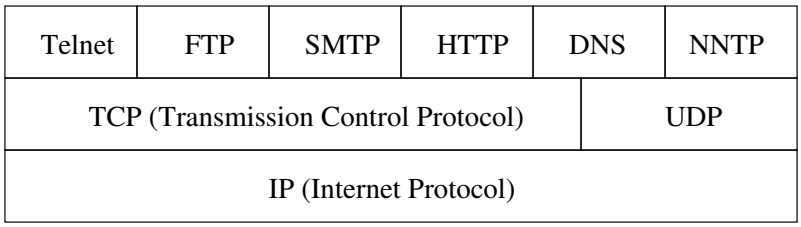


FIG. 3.1 – Le modèle TCP/IP

Le modèle TCP/IP étant antérieur au modèle OSI, il n'utilise pas une structuration en 7 couches. Depuis lors, le modèle OSI a introduit dans sa représentation l'ensemble des spécifications liées au protocoles TCP/IP afin de permettre facilement le développement d'applications interagissant avec des réseaux TCP/IP et d'autres types de réseaux plus récents, comme NetBIOS. La FIG. 3.2 illustre la comparaison entre le modèle OSI et le modèle TCP/IP.

La machine réceptrice est capable de décider de la manière de traiter les différents protocoles car l'adresse IP qui conduit la requête à son interface est suivie par un numéro de port de deux octets.

Les principaux protocoles utilisés sont :

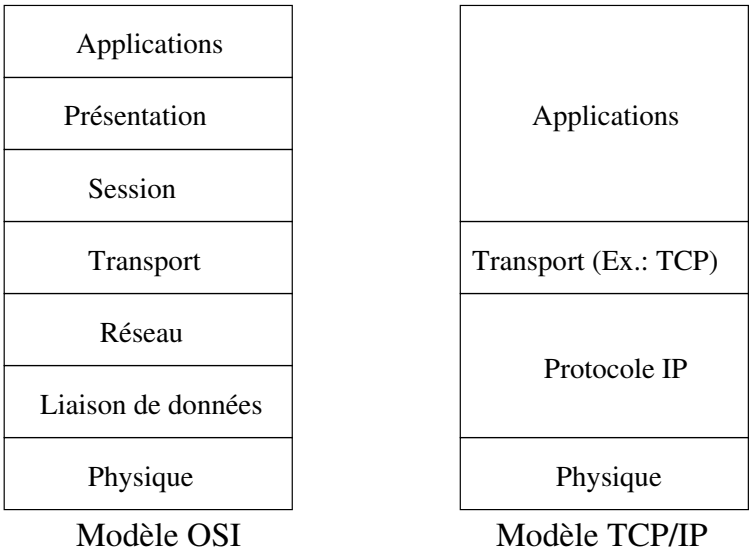


FIG. 3.2 – Comparaison entre le modèle OSI et le modèle TCP/IP

- MIL-STD-1777 : Internet Protocol (IP) qui est un protocole du niveau réseau assurant un service sans connexion ;
- MIL-STD-1778 : Transmission Control Protocol (TCP) est un protocole de niveau transport qui fournit un service fiable avec connexion ;
- MIL-STD-1780 : File Transfert Protocol (FTP) est un protocole de transfert de fichiers ASCII, EBCDIC et binaires ;
- MIL-STD-1781 : Simple Mail Transfer Protocol (SMTP) est un protocole de messagerie électronique utilisant le port 25 ;
- MIL-STD-1782 : TELNET Protocol est un protocole de présentation d'écran.
- Le protocole HyperText Transfert Protocol (HTTP) permet le transfert de texte utilisant le port 80.
- Le protocole Domain Name Service (DNS) fournit un service de noms (voir section 3.5) utilisant le port 53.
- Le protocole Network News Transfert Protocol (NNTP) sert pour les nouvelles (Usenet) utilisant le port 119.

L'architecture est basée sur le protocole IP qui correspond au niveau 3 du modèle OSI. Il a pour but de transporter les paquets, appelés datagrammes, d'une extrémité à l'autre du réseau. La sécurisation apportée est très faible : pas de détection de paquets perdus ou de possibilités de reprise sur erreur.

Le protocole TCP regroupe les services devant être proposés par le niveau 4 du modèle OSI. C'est un protocole très complexe censé pouvoir résoudre les problèmes d'erreurs du protocole IP. Il s'agit d'un mode connecté.

Il existe un mode non connecté représenté par le protocole User Datagram Protocol (UDP) mais qui n'est quasiment jamais utilisé car il ne garantit pas la livraison des paquets et ne délivre pas d'accusé de réception.

Toute la puissance de cette architecture apparaît lors de sa mise en place au-dessus de tous les réseaux différents existants. Ainsi, deux réseaux quelconques dont toutes les machines implantent le protocole IP peuvent communiquer via un équipement spécifique : le routeur.

Comme montré à la FIG. 3.1, il existe de nombreuses applications proposées par le modèle TCP/IP. Dans cette section, seules les applications propres à une utilisation internet orientée web seront détaillées.

3.3 Le Protocole IP

3.3.1 L'adressage IP

Chaque machine est identifié par une «adresse IP» unique qui est composée par une série de quatre octets[1] séparées par un point (Exemple : 127.15.12.31)[2]. C'est une institution mondiale qui gère ces numéros de manière à éviter les conflits ; de plus ces adresses IP sont payantes.

Bien que cela ne soit pas prévu par le protocole, une séparation peut être définie dans cette adresse : à sa gauche se situe le numéro de réseau, et à sa droite le numéro d'hôte. Deux machines situées sur le même réseau physique, généralement un LAN, possèdent le même numéro de réseau.

La séparation est déterminée par le premier des quatres octets :

- 0 à 127, la ligne de séparation se trouve après le premier octet, et il s'agit d'un réseau de classe A. Il peut y en avoir 125 utilisables, chacun d'eux pouvant abriter jusqu'à 16.777.214 hôtes.
- 128 à 191, la ligne de séparation se trouve après le deuxième octet, et il s'agit d'un réseau de classe B. Il peut y en avoir 16.382 utilisables, chacun d'eux pouvant abriter jusqu'à 65.534 hôtes.
- 192 à 223, la ligne de séparation se trouve après le troisième octet, et il s'agit d'un réseau de classe C. Il peut y en avoir 2.097.150 utilisables, chacun d'eux pouvant abriter jusqu'à 254 hôtes.

Les autres valeurs du premier octet ne sont pas importantes ici. On peut cependant contourner les règles des réseaux de classes A,B et C en utilisant des masques de sous-réseaux. Le principe est de segmenter plus en avant le réseau en utilisant davantage de bits pour le numéro de réseau et moins pour le numéro d'hôte, mais ceci sortirait du cadre du présent ouvrage.

3.3.2 Routage IP

Afin de permettre l'échange de données entre ordinateurs, on découpe les données à transmettre en datagrammes ou paquets. Chaque datagramme contient des informations, comme l'adresse IP de la machine émettrice et l'adresse de la machine réceptrice. Ainsi, lorsque dans un navigateur comme Netscape ou Internet Explorer, on entre «www.ulb.ac.be/uae», un certain nombre de paquets sont envoyés à la machine identifiée par l'adresse IP 164.15.59.200.

Comme la machine émettrice qui envoie ne connaît que l'adresse IP de la machine réceptrice, les paquets sont envoyés sur toutes les machines du réseau. En d'autres termes, chaque machine du réseau reçoit le paquet et le traite si l'adresse IP spécifiée est la sienne.

Dans la réalité, le réseau est organisé de manière hiérarchique en sous-réseaux. Chaque sous-réseaux est «responsable» d'une série d'adresses IP. Ces sous-réseaux sont interconnectés entre-eux au moyen de «routeurs». Les routeurs et les bridges sont des appareils capables de déterminer si un paquet est destiné à l'une des machines de son sous-réseau.

Il existe deux cas de figures :

1. Le routage direct : Les deux machines voulant communiquer appartiennent au même réseau. Par exemple, quelqu'un de l'ULB envoie un mail à une autre personne de l'ULB. Dans ce cas, le routeur expédie les paquets directement vers la machine réceptrice (En effet, les deux machines possède le même numéro de réseau).
2. Le routage indirect : Les deux machines voulant communiquer n'appartiennent pas au même réseau. Par exemple, quelqu'un de l'ULB envoie un mail à une personne de l'ULg. Dans ce cas, le routeur le détecte et fait parvenir les paquets à tous les routeurs auxquels il est connectés. Les paquets sont ainsi passés de routeurs à routeurs jusqu'au moment où le paquet arrive au routeur de la machine réceptrice.

3.4 Protocole TCP

Le rôle du protocole TCP est de fournir un transport fiable de données. Il implémente toutes les techniques permettant de résoudre les principaux problèmes trouvés sur les réseaux, comme la pertes de pa-

[1] Un octet est un nombre dont la valeur est comprise entre 0 et 255.

[2] Il s'agit de la norme d'adressage IPv4. La prochaine norme IPv6 sera composée d'une série de huit octets séparées par un point.

quets, la duplication de ceux-ci ou encore la congestion du réseau.

Du point de vue du programmeur d'application, le service proposé par TCP possède les propriétés suivantes :

Orientation connexion TCP offre un service orienté connexion dans lequel une application commence par demander l'établissement d'une connexion avec la destination visée, puis transmet ses données sur cette connexion.

Communication point à point Toutes les connexions TCP ne possèdent que deux extrémités.

Fiabilité totale TCP offre la garantie que les données envoyées sur une connexion sont remises exactement comme elles ont été emises : il n'y pas de données manquantes ni hors séquence.

Communication bidirectionnelle simultanée Sur une connexion TCP, les données peuvent circuler dans les deux sens, et chacune des applications peut émettre à tout moment.

Interface de flot de données TCP fournit une interface de flot de données («stream interface» en anglais), car l'application envoie une suite continue d'octets sur une connexion.

Etablissement fiable des connexions Lorsque deux applications démarrent une connexion TCP, elles doivent s'accorder sur le fait qu'il s'agit d'une nouvelle connexion : des paquets dupliqués provenant de connexions antérieures ne peuvent être pris pour des réponses valides, ni interférer avec la nouvelle connexion.

Clôture sécurisée des connexions Un programme d'application peut ouvir une connexion, transmettre une quantité quelconque de données, puis demander la clôture de la connexion. TCP garantit la remise fiable de toutes les données avant la clôture effective de la connexion.

3.5 Domaine Name Service

Il est possible d'associer à chaque machine, donc à chaque adresse IP, un nom. Par exemple, le nom «www.ulb.ac.be» correspond à l'adresse IP 164.15.59.200.

Le DNS est un service qui établit la correspondance entre les adresses physiques, c'est-à-direr les adresses IP, et les adresses logiques, c'est-à-dire les noms de domaine.

Bien qu'il n'y aucune régle officielle en vigueur, les adresses logiques sont choisies en respectant une certaine hiérarchie de celles-ci. Par exemple, toutes les adresses finissant par «.be» désignent des machines physiquement présentes en Belgique ; les adresses finissant par «.gov» désignent des machines appartenant au gouvernement américain. Enfin l'adresse logique «cfao30.ulb.ac.be» désigne un ordinateur se trouvant sur le réseau nommé «ulb.ac.be».

L'attribution de ces noms, gérée au niveau national par plusieurs sociétés autorisées, a été recemment liberalisée en Belgique.

3.6 HyperText Transfert Protocol

Ce protocole gère la demande et la réception de n'importe quel type de documents à travers un réseau basé sur les protocoles TCP/IP. Un serveur reçoit une requête concernant un document par le client, regarde si celui-ci peut consulter le document et lui renvoit ensuite le contenu.

L'application HTTP prévoit notamment que lors de la réception d'un document, celle-ci soit précédée d'un en-tête contenant des informations notamment sur sa nature par exemple. Ainsi, une demande peut être faite sans que la nature du document soit nécessairement connue avant.

3.6.1 Les seveurs web et les browsers

Les serveurs web et les browsers sont un cas particulier d'une architecture client/serveur utilisant TCP/IP et plus particulièrement le protocole HTTP. Le site web joue le rôle du serveur et les browsers ceux des clients. Le serveur web est donc une application permettant à des clients d'accéder au contenu de documents disponibles sur ce serveur, généralement stockés dans un répertoire particulier d'un disque dur.

Afin de permettre à un client d'accéder à plusieurs documents se trouvant sur plusieurs serveurs web, le client adresse ses requêtes en utilisant une URL (Voir section 3.7). Pour comprendre l'architecture client/serveur, nous traiterons un petit exemple. Soit un browser et l'URL «http ://www.ulb.ac.be/ulb/uae» :

1. Le browser de la machine client demande d'abord l'adresse IP de la machine serveur désignée par le nom «www.ulb.ac.be», et reçoit 164.15.59.200 en retour (Voir section 3.5).
2. Ensuite, il envoie une requête vers la machine serveur via son adresse IP en utilisant le protocole HTTP et en lui disant : «je voudrais avoir l'information concernant /ulb/uae».
3. La machine serveur reçoit la requête. Elle examine si le document demandé existe et éventuellement si le client peut y accéder. Dans ce cas, elle sait que «/ulb/uae» correspond à un document dont elle renvoie le contenu à la machine client.
4. Le browser de la machine client interprète le contenu du document reçu et affiche à l'écran la représentation visuelle de celui-ci.

Il existe de nombreux serveurs dans l'industrie qui se différencient par leurs différentes possibilités ainsi que par l'évolution de leurs performances lorsque le nombre de clients se connectant simultanément augmente. Actuellement, le serveur web le plus utilisé est Apache. Pour une étude complète sur l'utilisation de ce serveur web, il faut se reporter à [13].

3.6.2 En-tête HTTP

L'en-tête HTTP est composé d'un ensemble d'intitulés généralement séparés par un retour de chariot (séquence d'échappement '\n' dans la plupart des langages de programmation). Le TAB. 21.2 présente les intitulés HTTP les plus utiles.

TAB. 3.1 – Intitulés HTTP usuels

Intitulé	Description
Content-length	Taille du flux de sortie (en octets), considéré comme étant du binaire, c'est-à-dire une succession de caractères alphanumériques.
Content-type	Type MIME du flux de sortie.
Expires	Date et heure de fin de validité du document, lequel devra alors être rechargé à partir du serveur.
Location	Redirection du serveur (ne doit en aucun cas figuré dans un intitulé complet).
Pragma	Document caché / non-caché.
Status	Etat de la requête (ne doit en aucun cas figuré dans un intitulé complet).

Le TAB. 21.3 présente les intitulés HTTP introduits par Netscape et actuellement supportés par la plupart des browsers, même s'ils ne font pas partie de la spécification HTTP officielle.

TAB. 3.2 – Intitulés HTTP reconnus par la plupart des browsers

Intitulé	Description
Refresh	Le client charge à nouveau le document mentionné.
Set-Cookie	Le client stocke les données spécifiées sous forme d'un cookie sur sa machine.

L'orde dans lequel sont placés les éléments n'a aucune importance. L'exemple suivant montre un en-tête minimaliste indiquant que le document renvoyé est au format HTML et utilise la vesrion 1.0 du protocole HTTP :

```
HTTP/1.0
Content-type : text/html
```

Remark: *Il est nécessaire de laisser une ligne vide, c'est-à-dire de rajouter deux retours de chariot («\n\n» dans la plupart des langages de programmation), entre l'en-tête et la suite des informations.*

En fait, l'intitulé **Content-type** référence le type Multipurpose Internet Mail Extensions (MIME) (Voir section 3.6.3) des données qui seront envoyées par la suite. Cependant, il ne s'agit là que d'un type d'intitulé pouvant se retrouver dans un en-tête HTTP, on peut également indiquer :

– la taille des données ;
– le nom d'un document existant ;
– les codes d'état HTTP.

3.6.2.1 Intitulés «Expires» et «pragma»

La plupart des browsers placent automatiquement les documents consultés dans un «cache» afin d'éviter de devoir les recharger systèmatiquement du serveur lorsque l'utilisateur en a besoin. Ceci permet aux documents les plus souvent consultés d'être chargés beaucoup plus rapidement par le browser puisqu'il va chercher ceux-ci sur son propre disque dur et non sur internet. Il est évident que cette technique est très performante pour les documents statiques par contre pour les documents dynamiques cela présente peu d'intérêt.

Les intitulés **Expires** et **pragma** permettent de gérer la validité d'un document. En rajoutant à l'en-tête HTTP :

```
Pragma : no cache
```

On évite que le browser mette le document dans son cache, c'est-à-dire que celui-ci sera systématiquement rechargé à partir du serveur chaque fois que l'utilisateur désirera le consulter.

Par contre, en rajoutant dans l'en-tête HTTP :

```
Expires : Friday 27-Dec-01 05 :29 :10 GMT
```

On spécifie une date à partir de laquelle le document ne sera plus valable. Tant que cette date n'est pas atteinte, le browser pourra utiliser le document se trouvant dans son cache. Dès que la date est atteinte, il est forcé de recharger le document à partir du serveur.

3.6.2.2 Codes d'état

Un certain nombre de codes sont utilisés par HTTP afin de représenter l'état de la requête. L'en-tête HTTP contient parfois ce type de renseignement. L'intitulé **Status** se compose de trois chiffres formant un code d'état, lequel est ensuite décrit dans une chaîne de caractères. Les codes les plus usuels se trouvent repris dans le TAB.21.4.

TAB. 3.3 – Codes d'état HTTP

Code	Message
200	Réussite
204	Aucune réponse
301	Document déplacé
401	Autorisation non valable
403	Accès interdit
404	Introuvable
500	Erreur système
501	Non implémenté

3.6.3 Multipurpose Internet Mail Extensions

Comme expliqué, le serveur peut renvoyer différents types de documents :
- des pages webs contenant du textes et des images ;
- des documents pouvant être téléchargés, comme des fichiers Microsoft Word ou PDF ;
- des programmes pouvant être directement exécutés par l'utilisateur.

Afin de permettre aux clients de pouvoir réagir correctement aux types de documents renvoyés, on utilise la méthode MIME. Le but est de pouvoir associer des logiciels installés sur la machine client avec des types de documents. Ainsi, lorsque vous téléchargez un document PDF depuis un browser, l'Adobe Acrobat Reader s'ouvre ; ou encore, lorsque vous consultez des fichiers sons, le browser lance une application permettant d'écouter le son sur la machine client. Par exemple, lorsque le document est un document HTML (Voir partie II), l'en-tête contient une chaîne de caractères «text/html» signifiant que le document est sous format texte et doit être interprété comme du code HTML. Il n'existe aucune spécification concernant la chaîne

TAB. 3.4 – Quelques type MIME

Type MIME	Description
text/plain	Document texte au format ASCII contenant simplement du texte.
text/html	Document HTML.
msword	Document de Microsoft Word.
x-mp3	Fichiers sons au format mp3.
mpeg	Fichiers vidéo au format MPEG.
jpeg	Fichiers images au format JPEG.
pdf	Document PDF.

de caractères associée à un type de document donné, mais des conventions se sont imposées depuis des années, facilitant ainsi une utilisation globale. Le TAB. 3.4 donne quelques exemples de types MIME.

C'est la machine client qui est responsable de créer les liens MIME, bien que ceux-ci se mettent en places sans intervention de l'utilisateur. En effet, lorsque les logiciels s'installent, ils créent automatiquement les liens MIME qui leurs sont associés.

Remark: *Lorsque le browser ne trouve pas de liens MIME pour un type de document qui lui est renvoyé, il propose automatiquement à l'utilisateur de pouvoir le sauvegarder quelque part en local permettant ainsi un traitement ultérieur, ou de l'ouvrir avec une application que l'utilisateur choisit à ce moment.*

3.7 Universal Ressource Locator

3.7.1 Concepts de base

L'adressage Universal Ressource Locator (URL) est un moyen uniforme et universel de référencer des objets ou des services avec un réseau basé sur le protocole TCP/IP. L'objectif de la syntaxe de l'adressage URL est de définir une notation simple et claire.

Pour accéder à un objet quelconque sur un réseau, il faut que trois conditions soient remplies.

1. Il faut pouvoir accéder à la machine, c'est-à-dire connaître son adresse IP ou son nom, et avoir un droit d'accès à l'objet.
2. Il faut savoir quel protocole va devoir être utilisé.
3. Il faut connaître le nom de l'objet. Généralement, le nom correspond à l'endroit où se trouve le fichier.

Le format général d'une URL est :

```
protocole ://adresse du site/nom
protocole ://adresse du site/répertoire/fichier
```

Par exemple :

```
http ://www.ulb.ac.be/ulb/uae/index.html
```

Dans ce cas, on utilise le protocole «http». On désire accéder à un objet se trouvant sur la machine «www.ulb.ac.be». Le nom de l'objet est «ulb/uae/index.html», il s'agit d'un fichier «index.html» se trouvant dans le répertoire «/ulb/uae» de la machine.

3.7.2 Les protocoles

Les URLs supportent plusieurs types de protocoles qui seront repris dans le TAB. 3.5.

3.7.3 Les URLs relatives

Les URL utilisée jusqu'ici le sont de manière absolue, c'est-à-dire que le contenu de l'URL permet d'accéder à l'objet quelque soit l'endroit où l'on se trouve. Il existe aussi des URL relatives. En effet, lorsqu'un lien existe entre deux objets se trouvant sur la machine mais dans des répertoires différents, on peut utiliser des références relatives comme le montre le TAB. 3.6.

TAB. 3.5 – URLs et protocoles

Protocole	Description	Exemple
http	Permet de se connecter à un serveur web.	http ://www.site.com/index.html
file	Permet d'accéder un fichier local.	file :///home/pfrancq/index.html
ftp	Permet d'accéder à un fichier via FTP. Si aucun utilisateur n'est spécifié, le browser essaye d'y accéder via le user anonymous, sinon le browser demande un mot de passe.	ftp ://user@ftp.site.com
mailto	Permet d'envoyer un mail.	mailto :user@www.site.com
telnet	Permet d'ouvrir une session active d'un terminal. Le browser demande un mot de passe si nécessaire.	telnet ://user@www.site.com

TAB. 3.6 – Exemple d'URL relatifs

Objet courant	Objet destination	URL relatif
http ://www.essai.com/index.html	http ://www.essai.com/staff.html	staff.html
http ://www.essai.com/index.html	http ://www.essai.com/staff/staff.html	staff/staff.html
http ://www.essai.com/staff/staff.html	http ://www.essai.com/index.html	../index.html

En fait, les logiciels transforment l'URL relative en URL absolue, en rajoutant les informations nécessaires trouvées dans l'URL courante. Par exemple, en prenant la première URL du TAB. 3.6, en utilisant comme URL relative «staff.html», le logiciel, par exemple un browser, voit qu'il manque le protocole ainsi qu'une référence du serveur où l'objet doit se trouver. Il va donc chercher ses informations dans l'URL courante, en l'occurence le protocole «http» et le serveur «www.essai.com», pour reconstruire l'URL absolue de l'objet destination.

Deuxième partie

HTML

Chapitre 4

Les concepts de base du HTML

4.1 Pourquoi HTML ?

Avec l'introduction du TCP/IP, et de l'ensemble de ses protocoles (voir section 3.2), l'interconnexion des ordinateurs de natures différentes est devenue une réalité : n'importe quel ordinateur est suceptible de recevoir ou d'envoyer des informations à n'importe quel autre ordinateur, à partir du moment où chacun implante le protocole TCP/IP. Le principal problème réside dans la nature de ces informations, c'est-à-dire le format utilisé pour leur représentation. Même au niveau du format de fichier «texte», le plus simple, des divergences existent entre les ordinateurs et les systèmes d'exploitation[1]. Dès lors, la mise en place de conventions est devenue une nécessité.

C'est la raison pour laquelle le langage HyperText Markup Language (HTML), «Langage hypertexte utilisant des balises», a été mis au point. L'objectif était de représenter le contenu d'un document en utilisant un système pouvant être compris par tous les ordinateurs. Tous les browsers sont capables de décoder et d'afficher du code HTML, c'est pourquoi la plupart des documents sur Internet utilisent ce type de codage.

Il existe de nombreux outils, comme Netscape Composer ou Microsoft FrontPage, qui permettent de créer des documents HTML sans que l'utilisateur ait à manipuler le code HTML directement. Malheureusement, il n'est pas possible de tout faire avec ces outils, c'est pourquoi, pour faire des choses complexes ou pour affiner la présentation, l'utilisateur est obligé d'utiliser directement du code HTML. Il est donc non seulement important de pouvoir écrire des pages en utilisant le langage HTML, mais en plus il faut également apprendre à coder clairement les pages afin de permettre des modifications rapides. De plus, un document clairement structuré faciletera un traitement informatique ultérieur.

Le langage HTML est dérivé du Standard Generic Markup Language (SGML). Il s'agit d'un fichier texte reposant sur l'utilisation de balises afin d'indiquer des informations sur la manière d'afficher du contenu. Le Language LaTeX est un exemple bien contenu de langage SGML.

La personne désirant publier des informations sur internet, rajoutera des balises au texte original afin d'expliciter la manière dont le texte sera traité. Le browser de l'utilisateur désirant consulter cette information interprétera les balises afin de présenter le résultat.

HTML est un langage très complexe. Pour plus des informations complète, on peut consulter [29] et [20].

[1] Les ordinateurs fonctionnant sous DOS/Windows supposent qu'une fin de ligne est composée de deux caractères (13 et 10 pour code ASCII). Les autres ordinateurs, en particulier ceux utilisant MacOS ou UNIX/Linux, utilisent un seul caractère (le code ASCII 13).

4.2 Les caractères d'espacement

Pour créer un document au format HTML, il suffit de créer un fichier texte[2] et de respecter les conventions concernant le nom des balises, leurs paramètres ainsi que leur ordre d'apparition.

Le designer[3] peut organiser son fichier texte comme bon lui semble en utilisant autant de sauts de ligne ou de caractères de tabulation qu'il souhaite. En effet, le langage HTML considère tous ces caractères comme des caractères d'espacement : qu'il y ait un ou plusieurs caractères d'espacement, HTML considère toujours qu'il y a un seul espace. C'est pourquoi les trois formes suivantes seront interprétées de la même manière.

Forme 1 :

```
Ceci est du contenu.
```

Forme 2 :

```
Ceci     est    du      contenu.
```

Forme 3 :

```
Ceci est
du
contenu.
```

Pour forcer un retour à la ligne ou plusieurs espaces dans la présentation du document HTML, il existe des balises spécifiques.

4.3 Les balises

Le langage HTML utilise des balises afin de séparer le contenu du contenant. La forme générale d'une balise est :

```
<TAG ATTR1="quelque chose" ATTR2="quelque chose d'autre">
  Ceci est du contenu.
</TAG>
```

ou

```
<TAG ATTR="quelque chose">
```

Dans la première forme, la balise fait référence au texte «Ceci est du contenu.» et est utilisée par exemple pour mettre du texte en italique. La deuxième forme est utlisée lorsque la balise est auto-suffisante, comme par exemple le code utilisé pour insérer un saut de ligne.

Le code suivant affiche par exemple deux lignes dont la première est en italique.

```
<I>Ceci est la 1ère en italique.</I><BR>
Ceci est déjà la 2ème ligne.
```

Le nom des balises ainsi que celui des attributs peuvent être écrits en utilisant des minuscules, des majuscules ou un mélange des deux. Cependant, il existe une convention implicite consistant à écrire les noms des balises et des attributs en majuscules, et les valeurs en minuscules.

Remark: *La valeur des attributs peut être mise entre guillements ou entre apostrophes, permettant ainsi d'utiliser dans ceux-ci des apostrophes ou des guillements. Mais il faut utiliser le même caractère pour fermer et pour ouvrir une même chaîne de caractères.*

[2] Les deux «formats» de fichiers texte, ceux du type UNIX et ceux du type DOS, sont acceptés et compris par l'ensemble des browsers. Il est donc possible d'écrire un document HTML avec un éditeur sous UNIX et le visualiser sans problème avec un browser sous Windows.

[3] Un designer est une personne qui crée des documents HTML destinés à être publiés sur le web.

4.4 Les séquences d'échappement

Comme une balise commence par le caractère '<', comment-on peut faire pour afficher un tel caractère dans le contenu sans qu'il soit interprété comme une marque de début d'une balise ? C'est pour répondre à cette problématique que le langage HTML introduit les séquences d'échappement.

Une séquence d'échappement est composée d'une série de caractères commencant par '&' et terminant par ' ;'. Les principales séquences sont montrées au TAB. 4.1. Une forme spéciale est '&#*code*;', où *code* est le code ASCII du caractère que l'on désire imprimer.

TAB. 4.1 – Les séquences d'échappement

Séquence	Valeur numérique	Description
&lt ;	&#060 ;	Plus petit
&gt ;	&#062 ;	Plus grand
;	;	Espace
&quot ;	&#034 ;	Guillemet
&amp ;	&#038 ;	Et commercial
&copy ;	&#169 ;	Symbol copyright
&reg ;	&#174 ;	Symbole trademark registré
N/A	&#153 ;	Symbol trademark

4.5 Structure générale d'un document HTML

La structure générale d'un document au format HTML est montrée au listing 4.1.

Listing 4.1 – Structure générale d'un fichier HTML

```
<!DOCTYPE html
2       PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "DTD/xhtml1-transitional.dtd">
4 <HTML>
  <HEAD>
6     <TITLE> A simple web page </TITLE>
  </HEAD>
8 <BODY>
    Contents of the document.
10 </BODY>
</HTML>
```

Tout document HTML devrait au moins posséder toutes les balises présentes dans ce document type. Les deux premières lignes donnent des informations génériques concernant le type de document, et en particulier la version du langage HTML utilisé. C'est une bonne pratique que de l'inclure systématiquement, même si cela n'est pas obligatoire. La FIG. 4.1 illustre le résultat de l'exemple.

La balise **<HTML>** (Lignes 4 et 11) permet d'indiquer le début et la fin du document et est obligatoire dans tout document HTML. La balise **<HEAD>** (Lignes 5 et 7) permet de spécifier des informations d'entêtes, comme par exemple le titre (Balise **<TITLE>** de la ligne 6) devant apparaître dans la fenêtre du navigateur.

Enfin , la balise **<BODY>** (Lignes 8 et 10) spécifie le contenu réel du document. La forme générale de la balise **<BODY>** est :

<**BODY TEXT**=*c1* **BACKGROUND**=*c2* **LINK**=*c3* **ALINK**=*c4* **VLINK**=*c5*></**BODY**>

La signification des différents attributs est :

TEXT La couleur du texte.

BACKGROUND La couleur de fond.

LINK La couleur dans laquelle les liens s'affichent.

FIG. 4.1 – Exemple d'un document HTML de base

ALINK La couleur dans laquelle s'affichent les liens actifs.

VLINK La couleur dans laquelle s'affichent les liens déjà visités.

Remark: *Les couleurs peuvent être encodés soit sous forme d'une chaîne de caractères pour les couleurs de bases (Par exemple «red» ou «black»), soit sous la forme #NNNNNN dans laquelle on utilise la représentation RGB (Red-Green-Blue) pour coder la couleur en utilisant la notation héxadécimale (Par exemple, «#000000» représente le noir, «#FFFFFF» représente le blanc et «#0000FF» représente le bleu).*

4.6 Les attributs d'identification

On peut associer à n'importe quelle balise deux atributs d'identification différents :

ID=*name* Un nom unique dans le document identifiant l'élément.

CLASS=*classnames* Un nom de classe ou un ensemble de noms de classe. Plusieurs noms de classes seront séparés par des espaces.

Ces informations sont utilisées dans différentes situations comme par exemple dans une référence, dans une balise **<OBJECT>**, dans un style ou encore dans des scripts. Dès que l'on doit accéder d'une manière ou d'une autre à une balise ou à son contenu, il faudra peut-être inclure au moins l'un des deux attributs. Par la suite, l'existence de ces deux attributs ne sera plus présentée pour les différentes balises rencontrées.

Dans le listing 4.2, qui est un extrait de «Hamlet», on a utilisé une classe par intervenant, ainsi qu'un identifiant pour chaque réplique. Cela pourra par exemple servir à mettre toutes les répliques d'un personnage donné, dans une certaine police de caractères (Voir partie III).

Listing 4.2 – Utilisation de classes

```
<HTML>
  <HEAD>
    <TITLE> Hamlet, excerpt from act II </TITLE>
  </HEAD>
  <BODY>
    <P CLASS="POLONIUS" ID="R1">
      Polonius: Do you know me, my lord?
    </P>
    <P CLASS="HAMLET" ID="R2">
      Hamlet: Excellent well, you are a fishmonger.
    </P>
    <P CLASS="POLONIUS" ID="R3">
      Polonius: Not I, my lord.
    </P>
    <P CLASS="HAMLET" ID="R4">
      Hamlet: Then I would you were so honest a man.
    </P>
  </BODY>
</HTML>
```

Chapitre 5

Les balises communes

Ce chapitre présente les balises les plus courantes du langage HTML :
- les séparations ;
- les titres ;
- le texte ;
- les listes ;
- les images.

5.1 Les séparations

Il existe des balises permettant de créer des séparations dans un document HTML.

5.1.1 Le saut de ligne

Comme expliqué dans la section 4.2, un saut de ligne est considéré comme un espace par le langage HTML qui n'effectue donc pas de saut de ligne dans la présentation visuelle du document. Afin de permettre le saut de ligne, le langage HTML introduit la balise **
**. Un exemple est :

```
Ligne 1<BR>Ligne 2
Toujours ligne 2<BR><BR>Ligne 4
```

5.1.2 Les séparations horizontales

Pour séparer différentes parties dans un document HTML, il est possible d'utiliser de nombreuses balises **
**. Cependant, il existe une autre alternative consistant à afficher une ligne horizontale. Le format est :

```
<HR ALIGN=align WIDTH=width SIZE=size NOSHADE>
```

La signification des différents attributs est :

ALIGN Cet attribut permet de spécifier la manière dont le texte sera justifié. Les valeurs possibles sont «**Left**», «**Right**», «**Center**» et «**Align**». Par défaut, la justification est à gauche.

WIDTH La longueur de la séparation. Elle peut être directement donnée en pixels (Par exemple ”50”) ou en pourcentage de la longueur de la fenêtre du browser (Par exemple ”75%”). Par défaut, la longueur est de 100%.

SIZE Cet attribut permet de préciser l'épaisseur de la ligne de séparation. Par défaut, l'épaisseur est de 1 pixel.

NOSHADE En précisant cet attribut, l'effet de relief de la ligne est supprimé.

5.2 Les titres

Le langage HTML, initialement créer pour la présentation d'articles scientifiques, permet d'afficher du texte. Celui-ci est structuré à l'aide de titres et de paragraphes. Les balises utilisées sont **<H1>**, **<H2>**, **<H3>**, **<H4>**, **<H5>** et **<H6>**. La balise **<H1>** correspond au niveau le plus élévé alors que la balise **<H6>** correspond au niveau le plus bas. La forme générale est :

```
<H1 ALIGN=align>titre</H1>
```

L'attribut **ALIGN** permet de spécifier la manière dont le texte sera justifié. Ses valeurs possibles sont **«Left»** (Alignement à gauche), «**Right**» (Alignement à droite), «**Center**» (Texte centré) et «**Justify**» (texte justifié). Par défaut, la justification est à gauche.

Un exemple montrant l'utilisation des titres est montré au listing 5.1 ainsi qu'à la FIG. 5.1.

Listing 5.1 – Exemple de titre

```
<HTML>
  <HEAD>
    <TITLE> Headings sizes </TITLE>
  </HEAD>
  <BODY>
    <H1> Level 1 heading &lt;h1&gt; </H1>
    <H2> Level 2 heading &lt;h2&gt; </H2>
    <H3> Level 3 heading &lt;h3&gt; </H3>
    <H4> Level 4 heading &lt;h4&gt; </H4>
    <H5> Level 5 heading &lt;h5&gt; </H5>
    <H6> Level 6 heading &lt;h6&gt; </H6>
  </BODY>
</HTML>
```

FIG. 5.1 – Exemple de titres

Ces balises correspondent aux types de structuration des logiciels de traitements de texte comme Microsoft Word («Headings 1», «Headings 2», etc...) ou KWord («Head 1», «Head 2», etc...).

5.3 Le texte

Il existe de nombreuses balises définies pour manipuler du texte.

5.3.1 Les paragraphes

La balise **<P>** définit un nouveau paragraphe. Son format est le suivant :

```
<P ALIGN=align>Contenu du paragraphe</P>
```

où le attribut «**ALIGN**» a la même signification que pour les titres.

Le listing 5.2 et la FIG. 5.2 illustrent les effets de cet attribut.

Listing 5.2 – Exemple de justifications possibles

```
<HTML>
  <HEAD>
    <TITLE> Paragraph alignements </TITLE>
  </HEAD>
  <BODY BGCOLOR="#ffffff" TEXT="#000080">
    <P ALIGN="left">
      This paragraph is left aligned. This paragraph is left aligned.
      This paragraph is left aligned.
    </P>
    <P ALIGN="center">
      This paragraph is center aligned. This paragraph is center aligned.
      This paragraph is center aligned.
    </P>
    <P ALIGN="right">
      This paragraph is right aligned. This paragraph is right aligned.
      This paragraph is right aligned.
    </P>
    <P ALIGN="justify">
      This paragraph is justifed. This paragraph is justifed.
      This paragraph is justifed.
    </P>
  </BODY>
</HTML>
```

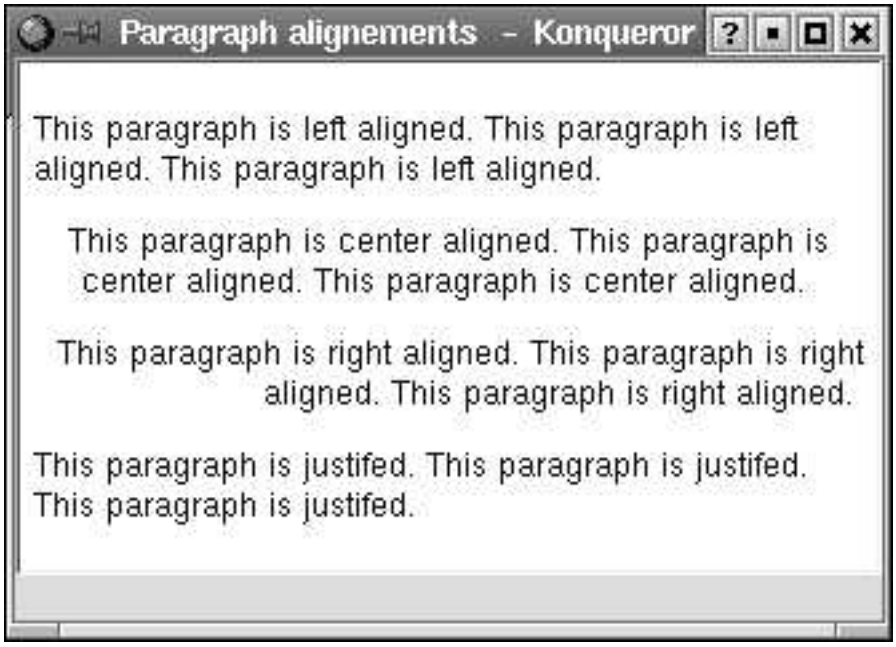


FIG. 5.2 – Exemple de justifications possibles

5.3.2 Les polices de caractères

Les premières versions de HTML ne permettait l'affichage du texte que dans une seule police de caractères possible. Actuellement, les caractéristiques de la police de caractères utilisée pour afficher un texte donné peuvent être spécifiées à l'aide de la balise **<FONT>** dont la forme générale est :

```
<FONT FACE=face COLOR=col SIZE=size>Texte</FONT>
```

FACE Cet attribut permet de spécifier la police de caractère utilisée.

COLOR Cet attribut permet de spécifier la couleur dans laquelle le texte sera affiché.

SIZE Cet attribut permet de spécifier la taille de la police de caractère.

Cette taille de la police de caractères peut être donnée soit en absolu, comme dans le listing 5.3 et la FIG. 5.3, ou en relatif, comme dans le listing 5.4 et la FIG. 5.4.

Listing 5.3 – Utilisation des tailles absolues

```
<HTML>
  <HEAD>
    <TITLE> Absolute font sizes </TITLE>
  </HEAD>
  <BODY>
    <P><FONT SIZE="7"> This is font size 7 </FONT></P>
    <P><FONT SIZE="6"> This is font size 6 </FONT></P>
    <P><FONT SIZE="5"> This is font size 5 </FONT></P>
    <P><FONT SIZE="4"> This is font size 4 </FONT></P>
    <P><FONT SIZE="3"> This is font size 3 (default size) </FONT></P>
    <P><FONT SIZE="2"> This is font size 2 </FONT></P>
    <P><FONT SIZE="1"> This is font size 1 </FONT></P>
  </BODY>
</HTML>
```

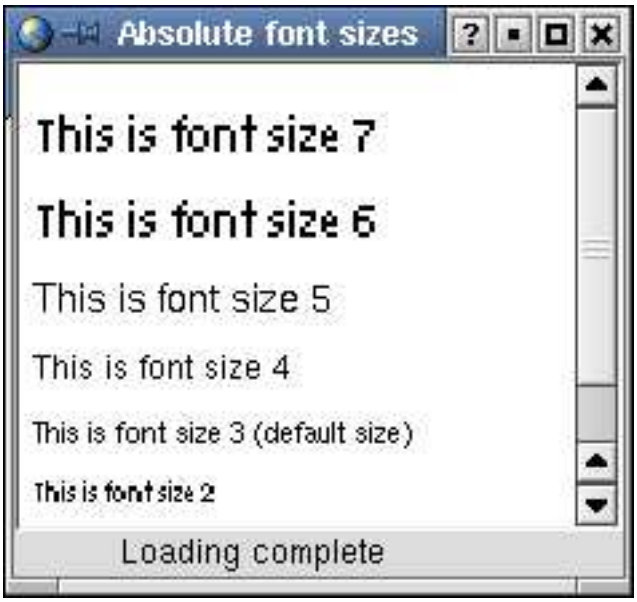


FIG. 5.3 – Utilisation des tailles absolues

Il est évident que pour que le browser puisse affiché correctement la police de caractères à l'écran, il faut que celle-ci soit disponible sur la machine client. Cependant, les browsers proposent tous la possibilité de pouvoir utiliser une autre police lorsque celle définie dans le document HTML. Le problème principal avec l'usage des polices de caractères est qu'elles ne sont pas standard à travers les différentes plate-formes.

Listing 5.4 – Utilisation des tailles relatives

```
<HTML>
  <HEAD>
    <TITLE> Relative font sizes </TITLE>
  </HEAD>
  <BODY>
    <P>
      This is the default font size, actually "3".
      The following font sizes are relative to this, so that "+3"
      means "3 sizes bigger than the default font size
    </P>
    <P><FONT SIZE="+4"> This is font size +4 </FONT></P>
    <P><FONT SIZE="+3"> This is font size +3 </FONT></P>
    <P><FONT SIZE="+2"> This is font size +2 </FONT></P>
    <P><FONT SIZE="+1"> This is font size +1 </FONT></P>
    <P><FONT SIZE="-1"> This is font size -1 </FONT></P>
    <P><FONT SIZE="-2"> This is font size -2 </FONT></P>
  </BODY>
</HTML>
```

Ainsi, la police de caractères «Arial» est disponible par défaut sous le système d'exploitation Windows, alors qu'elle n'est pas disponible dans le monde UNIX. C'est pourquoi, le concepteur de page doit faire très attention aux polices qu'il utilise dans ses pages HTML.

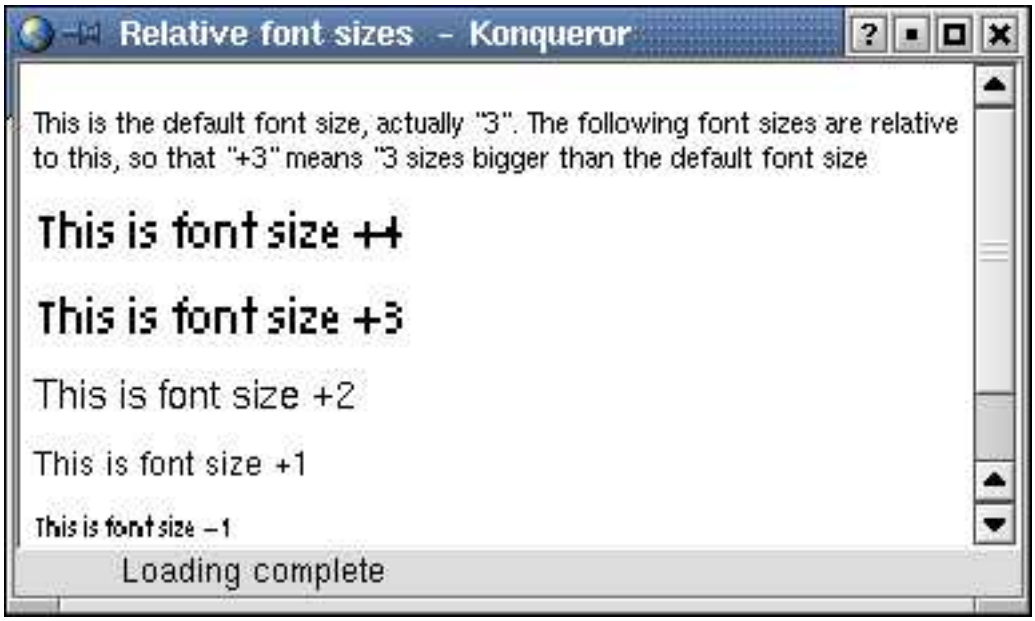


FIG. 5.4 – Utilisation des tailles relatives

Afin de permettre plus de souplesse pour les concepteurs de pages, la notion de police de caractères customisée est apparue ; l'idée étant qu'un auteur de pages puisse définir une police de caractères indépendante des différentes plate-formes, et que le browser puisse aller les chercher sur un site donné. Bien que cette solution paraisse simple, dans la pratique cela n'est pas le cas. Actuellement, les sociétés Microsoft et Netscape proposent chacun leur propre solution.

Netscape a développé un nouveau format, le Portable Font Resource (PFR), permettant de décrire des polices de caractères. Le principe est de rajouter dans le fichier HTML, une référence à chaque fichier PFR définissant une police de caractères utilisée dans le document.

Microsoft propose une autre solution. En utilisant l'un des de ses outils, le responsable du site web crée pour chaque document HTML des fichiers contenant la définition de toutes les polices de caractères utilisées. Ensuite, une référence à ces fichiers est créée par l'intermédiaire des styles (Voir la partie III). Il

faut savoir que la solution de Microsoft se conforme à la spécification initiale W3C concernant les polices téléchargeables.

Ceci dit, malgré l'intérêt que suscite l'idée de pouvoir utiliser des polices de caractères orignales dans les documents HTML, il n'y a aucun standard qui a été définit et qui serait supporté par la plupart des browsers.

Il est également possible de spécifier d'autre caractéristiques de la police de caractères. Pour une étude détaillée, il faut se référer à [29].

5.3.3 Le formatage simple

Le formatage simple à du texte s'applique en utlisant les balises du TAB. 5.1, permettant par exemple de mettre du texte en gras ou/et en italique. Ces balises permettent donc de changer le style de la police de caractères et non ses caractéristiques.

TAB. 5.1 – Le formatage simple

Balise	Influence sur le texte
<I>...</I>	Met le texte en italique
<B>...</B>	Met le texte en gras
<TT>...</TT>	Utilise une police de caractères mono-espacée
<U>...</U>	Souligne le texte
<STRIKE>...</STRIKE>	Barre le texte
<S>...</S>	Une alternative pour barrer du texte
_{...}	Met le texte en indice
^{...}	Met le texte en exposant
<BIG>...</BIG>	Augmente de 1pt la taille de la police de caractères
<SMALL>...</SMALL>	Diminue de 1pt la taille de la police de caractères

Les différentes balises peuvent être combinées afin de mélanger les genres. Pour obtenir du texte en italique, gras et souligné, il suffit d'utiliser :

```
<B><I><U>Texte en gras, italique et souligné</U></I></B>
```

L'ordre des balises n'a aucune importance, du moment qu'elles soient fermées dans l'ordre inverse de celui dans lequel elles ont été ouvertes.

Il est important de signaler que le texte contenu entre ces balises n'est pas considéré comme un paragraphe. Dés lors, il n'y a pas de passage à la ligne entre deux balises successives. De même, il n'est pas possible d'ajouter des paramètres, par exemple d'alignement, a ces balises. Pour cela, il faut donc combiner celles-ci avec une balise du type paragraphe.

Remark: *Bien que certaines balises permettent l'écriture d'exposants et d'indices, il n'est pas possible, en utilisant le langage HTML normal, d'écrire des formules mathématiques compliquées. La seule manière pour afficher des équations est d'utiliser un outil à part, et de sauvegarder le résultat sous forme d'images qui seront ensuite insérées dans le document.*

5.3.4 Les éléments logiques

Les entités logiques permettent de structurer le texte en utilisant les balises du TAB. 5.2, chaque balise logique ayant son propre style d'affichage. L'utilisation de ces balises permet de créer des documents clairs et structurés sans perdre du temps à utiliser de nombreuses balises de formatage.

Outre le fait que l'utilisation de ces balises évite d'introduire trop de mise en page, elle assure également une certaine consistance dans l'ensemble des documents HTML. En effet, si tous les documents HTML contenant des extraits de code source utilisaient la balise **<CODE>**, le formatage de ce code serait partout identique sur le browser de l'utilisateur.

Bien que les browsers interprètent chacune de ces entités différentes, il y a de grande similitude.

TAB. 5.2 – Les éléments logiques

Balise	Type d'élément logique
<ABBR>...</ABBR>	Une abréviation
<CITE>...</CITE>	Une citation
<CODE>...</CODE>	Du code source
<DFN>...</DFN>	Une définition
<EM>...</EM>	Un élément important
<KBD>...</KBD>	Un raccourci clavier
<SAMP>...</SAMP>	Un exemple
<STRONG>...</STRONG>	Un élément très important
<VAR>...</VAR>	Une variable de programmation

5.4 Les listes

Il est également possible de créer des listes dans un document HTML. Il existe en fait trois types de listes : les listes ordonnées (Balise **<OL>**), les listes non ordonnées (Balise **<UL>**) et des listes de définition (Balise **<DL>**). La balise **<LI>** permet d'ajouter un nouvel élément dans une liste.

Les listes peuvent être imbriquées de n'importe quelle manière. Le listing 5.5 et la FIG. 5.5 présentent un exemple d'utilisation de listes.

Listing 5.5 – Exemple de listes

```
<HTML>
  <HEAD>
    <TITLE> Lists </TITLE>
  </HEAD>
  <BODY>
    <P> There are two main sections in an HTML file: </P>
    <OL>
      <LI> HEAD </LI>
      <LI> BODY </LI>
    </OL>
    <P> Some HTML elements we have used are: </P>
    <UL>
      <LI> headings: h1 and h2 </LI>
      <LI> paragraphs: p </LI>
      <LI> lists:
        <UL>
          <LI> unordered lists: ul </LI>
          <LI> ordered lists: ol </LI>
        </UL>
      </LI>
    </UL>
  </BODY>
</HTML>
```

Remark: *La norme HTML spécifie que la balise* ***<LI>*** *ne doit pas forcément être refermées. En effet, HTML estime que la fin de l'élément est représenté soit par le début d'un nouvel élément, et donc d'une balise* ***<LI>*** *ouverte, ou bien par la fermeture de la balise de la liste. Cependant, c'est une excellente habitude que de refermer systématiquement toute les balises ouvertes.*

5.4.1 Les Listes ordonnées

Dans une liste ordonnée, chaque élément de la liste est précédé par un caractère ou chiffre précisant l'ordre. Le format général est :

```
<OL START=start TYPE="type1">
  <LI TYPE=type2> Elément nř1 de ma liste. </LI>
  <LI TYPE=type3> Elément nř2 de ma liste.</LI>
  <LI VALUE=value> Elément nřvalue de ma liste.</LI>
</OL>
```

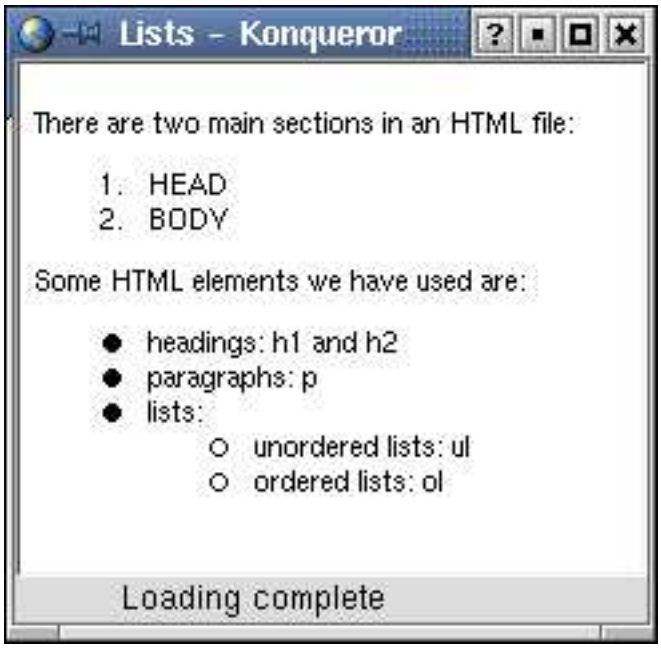


FIG. 5.5 – Exemple de listes

La signification des attributs est la suivante :

START Désigne l'indice de départ de la liste. Par défaut, l'indice de départ est 1.

TYPE Désigne la nature de l'indice. Comme indiqué, les types peuvent varier au sein d'un même liste. Les possibilités sont :

a Cet attribut spécifie des petits caractères (a,b,c,d).

A Cet attribut spécifie des grands caractères (A,B,C,D).

i Cet attribut spécifie des petits caractères romains (i,ii,iii,iv).

I Cet attribut spécifie des grands caractères romains (I,II,III,IV).

1 Cet attribut spécifie des nombres arabes (1,2,3,4) (Type par défaut).

VALUE Cet attribut permet de spécifier un indice différent que celui qui devrait suivre logiquement.

5.4.2 Les listes non ordonnées

Dans une liste non ordonnée, les éléments sont précédés d'un signe particulier, qui peut varier si les listes sont imbriquées. Le format général est :

```
<UL TYPE=type1>
  <LI TYPE=type2> Un élément.</LI>
  <LI TYPE=type3> Un autre élément.</LI>
</UL>
```

L'attribut **TYPE** permet de choisir le sigle qui sera utilisé. Les principales possibilités sont :

Square Cet attribut spécifie un carré.

Disc Cet attribut spécifie un cercle plein.

Circle Cet attribut spécifie un cercle.

Remark: *Il existe des browsers qui supportent d'autres valeurs pour l'attribut **TYPE**, mais cela ne fait pas partie du standard HTML.*

5.4.3 Les listes de définition

Ces listes permettent d'associer des noms avec des définitions. La forme générale est :

```
<DL>
  <DT> Nom 1 </DT>
  <DD> Définition correspondante à 'Nom 1'.</DD>

  <DT> Nom 2 </DT>
  <DD> Définition correspondante à 'Nom 2'. </DD>
</DL>
```

Ce type de liste ne prend aucun attribut.

5.5 Les images

5.5.1 Formats d'image

Dans un document HTML, il est possible d'insérer des images stockées dans un fichier, du moment que le format du fichier soit défini. En effet, à partir du moment où les images doivent être visualisées sur n'importe quelle machine avec n'importe quel système d'exploitation il est nécessaire de restreindre le nombre de formats circulant sur internet.

Actuellement, les deux formats reconnus par tous les browsers sont le Graphics Interchange Format (GIF) et le Joint Photographic Experts Group (JPEG). Un nouveau format Portable Network Graphics (PNG) a été introduit par le World Wide Web Consortium (W3C), qui a déjà mis au point le standard HTML, et il est probable que ce format s'imposera dans l'avenir, même si actuellement peu de browsers le supportent. Vu les liens historiques liant internet et le monde Unix, la plupart des browsers supportent également de manière native les formats X BitMaps (XBM) et X PixelMaps (XPM).

Le format GIF, outre un bon taux de compréssion, offre deux avantages supplémentaires :

- la couleur transparente, c'est-à-dire que la couleur de fond de l'image peut être fixé par la couleur de fond du document HTML qui la contient ;
- la possibilité de créer des GIF animé, c'est-à-dire de stocker dans un seul fichier plusieurs images qui apparaîtront les unes après les autres comme un petit film d'animation.

Il existe de nombreux autres types de fichiers d'images existants sur internet, comme les fichiers AutoCAD par exemple. Ces formats ne sont pas supportés de manière native par les browsers, mais peuvent généralement être visualisés à l'aide de plugins[1] installés sur les machines.

5.5.2 Insertion d'images

L'insertion d'une image se fait grâce à la balise **<IMG>.** Le listing 5.6 et la FIG. 5.6 illustrent un premier exemple.

Listing 5.6 – Insertion d'images (1)

```
<HTML>
  <HEAD>
    <TITLE> Images (1) </TITLE>
  </HEAD>
  <BODY>
    <IMG SRC="telex.gif" WIDTH="137" HEIGHT="150" ALT="Telex">
    <P> Sample text. Sample text. Sample text. </P>
  </BODY>
</HTML>
```

La balise **IMC** complète est définie par :

```
<IMG SRC=url WIDTH=w HEIGHT=h BORDER=b ALIGN=a HSPACE=hs VSPACE=vs ALT=alt>
```

[1] Un plugin est une extension software à un certain browser.

FIG. 5.6 – Insertion d'images (1)

La signification des différents attributs est :

SRC Cet attribut permet de spécifier l'URL du fichier contenant l'image.

WIDTH Cet attribut permet de spécifier quelle largeur l'image doit occuper (en pixels) sur l'écran. Si ce paramètre est omis, la largeur réelle de l'image est utilisée.

HEIGHT Cet attribut permet de spécifier quelle hauteur l'image doit occuper (en pixels) sur l'écran. Si ce paramètre est omis, la hauteur réelle de l'image est utilisée.

BORDER Cet attribut permet de spécifier l'épaisseur éventuelle (en pixels) du cadre entourant l'image.

ALIGN Cet attribut permet de mettre du texte autour de l'image insérée.

HSPACE Cet attribut permet de spécifier un espace horizontal entre la zone réservée pour l'image et le texte sui l'entoure.

VSPACE Cet attribut permet de spécifier un espace vertical entre la zone réservée pour l'image et le texte qui l'entoure.

ALT Cet attribut permet d'associer un titre alternatif à l'image qui apparaîtra si le browser ne peut afficher des images ou si l'utilisateur reste quelques secondes sur l'image avec le curseur.

5.5.3 Ajouter du texte à une image

L'attribut **ALIGN** permet de mettre du texte à coté de l'image. En plus des valeurs normales, l'attribut **ALIGN** permet d'utiliser les valeurs «**top**», «**middle**» et «**bottom**» de sorte que le texte qui sera affiché à coté de l'image soit aligné en haut, au milieu ou en bas. Un exemple est montrè au listing 5.7 dont le résultat est illustré à la FIG. 5.7.

En combinant l'attribut **ALIGN** avec les attributs **HSPACE** et **VSPACE**, il est possible d'entourer les images de texte. Un exemple est montré au listing 5.8 et est illustré à la FIG. 5.8.

Il arrive que le designer veuille qu'à partir d'un certain endroit le texte cesse de se mettre autour de l'image, même s'il reste de la place. Pour cela, il faut utiliser la balise **
** avec la forme suivante :

```
<BR CLEAR=c>
```

Listing 5.7 – Insertion d'images (2)

```
<HTML>
  <HEAD>
    <TITLE> Images (2) </TITLE>
  </HEAD>
  <BODY BGCOLOR="#ffffff">
    <IMG SRC="telex.gif" WIDTH="137" HEIGHT="150" ALT="Telex" ALIGN="left">
    <P> Sample text. Sample text. Sample text. </P>
  </BODY>
</HTML>
```

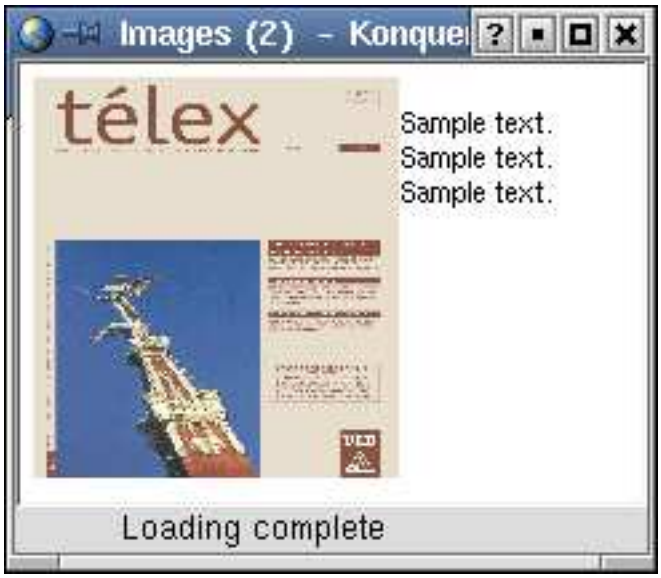


FIG. 5.7 – Insertion d'images (2)

L'attribut **CLEAR** peut prendre les valeurs «**Left**», «**Right**», «**All**» et «**None**». Par exemple, dans l'exemple précédent, l'attribut **ALIGN** de l'image est utilisée avec la valeur «**Left**». En utilisant la balise **
** avec la valeur «**Left**» pour l'attribut **CLEAR**, le texte qui suit cette balise doit recommencer en dessous de l'image.

Listing 5.8 – Insertion d'images (3)

```
<HTML>
  <HEAD>
    <TITLE> Images (3) </TITLE>
  </HEAD>
  <BODY>
    <P>
      Sample text. Sample text. Sample text. <BR>
      <IMG SRC="telex.gif" WIDTH="137" HEIGHT="150" HSPACE="10" VSPACE="10"
        ALT="Telex" ALIGN="left">
      Sample text. Sample text. Sample text. Sample text. Sample text.
      Sample text. Sample text. Sample text. Sample text. Sample text.
      Sample text. Sample text. Sample text. Sample text. Sample text.
      Sample text. Sample text.
    </P>
  </BODY>
</HTML>
```

FIG. 5.8 – Insertion d'images (3)

5.5.4 L'attribut ALT

L'attribut **ALT** permet d'associer un texte à une image qui sera affiché si le browser n'affiche pas l'image, soit parce que le transfert a été interrompu, ou lorsque le support ne permet pas d'afficher des images (comme pour les Personal Digital Assistant (PDA) par exemple). Certains browsers, comme Microsoft Internet Explorer, utilise ce texte dans une bulle d'aide lorsque l'utilisateur point quelques secondes sur l'image avec sa souris.

La spécification HTML n'impose aucune limite quant à la longueur du texte associé à l'attribut **ALT**, mais il faut signaler que certains browsers, de moins en moins nombreux, n'acceptent qu'une longueur maximale de 1024 caractères.

Cet attribut est très important, non seulement il permet de renseigner sur le contenu de l'image lorsque celle-ci n'est pas disponible, mais en plus certain browser, comme pmWebSpeak (http ://www.prodworks.com) par exemple, intègre un synthétiseur vocal qui lit le contenu de ce texte.

Chapitre 6

Les liens

L'une des grandes forces des documents hypertextes, et par conséquent du langage HTML, est la possibilité de créer des références croisées. En langage HTML cela se résume à utiliser les URLs (Voir section 3.7) pour permettre de naviguer entre les différentes pages d'un ou de plusieurs sites.

6.1 Lien basique

La balise **<A>** définit un lien en HTML. Sa forme la plus simple est :

```
<A HREF=ref> Ceci est une référence. </A>
```

L'attribut **HREF** permet de spécifier l'URL correspondant au lien, par exemple «http ://www.ulb.ac.be». Lorsque l'utilisateur clique sur le texte correspondant à ce lien, le browser sera dévié vers l'URL spécifiée. Généralement, le texte représentant un lien est souligné.

Remark: *Certains browsers permettent de ne pas souligner les liens. Cela peut rendre les pages moins lisibles, mais beaucoup d'utilisateurs trouvent cela plus esthétique.*

Un exemple est montré au listing 6.1 et à la FIG. 6.1.

Listing 6.1 – Exemple de liens

```
<HTML>
  <HEAD>
    <TITLE> Links </TITLE>
  </HEAD>
  <BODY>
    Link to ULB : <A HREF="http://www.ulb.ac.be"> ULB </A><BR>
    Relative link : <A HREF="second.html"> Second page </A>
  </BODY>
</HTML>
```

L'URL peut être soit absolue comme à la ligne 6 ou relative aux répertoires du serveur web comme à la ligne 7.

Un cas intéressant est celui de l'association d'un lien à une image. Cela peut se faire en imbriquant les balises de la manière suivante :

```
<A HREF="www.ulb.ac.be"><IMG SRC="logo-ulb.gif"></A>
```

Dans ce cas, lorsque l'utilisateur clique sur l'image, le browser sera dévié vers le lien correspondant.

6.2 La balise A

Voici la définition complète de la balise **<A>** :

```
<A HREF=url TARGET=tg TITLE=t ACCESSKEY=k TABINDEX=i NAME=n REL=rl REV=rv> </A>
```

FIG. 6.1 – Exemple de liens

Où les paramètres ont la signification suivante :

HREF L'URL correspondant au lien.

TARGET Indique le nom du cadre dans laquelle le contenu sera affiché (Voir chapitre 8). Par défaut, c'est la fenêtre courante qui affichera le résultat.

TITLE Cet attribut permet l'apparition d'un texte dans une bulle d'aide lorsque l'utilisateur reste assez longtemps sur le lien. Une autre utilisation de cette information, est qu'elle sert pour associer un nom à un bookmark lorsque le document associé à celui-ci n'est pas encore visité par le browser.

ACCESSKEY Cet attribut permet d'associer un raccourci clavier avec ce lien. En spécifiant une lettre, le lien devient actif losque l'utilisateur combine la touche représentant cette lettre avec la touche ALT. Ceci dit, il faut être prudent, car il ne faudrait pas «surcharger» les raccourcis clavier du browser, par exemple ALT+F pour activer le menu «File».

TABINDEX Cet attribut permet de désigner l'ordre de tabulation des différents liens si le browser permet une telle navigation, généralement avec la touche TAB.

NAME Le nom du lien afin de pouvoir être référencé par un autre lien (Voir section 6.3).

REL Cet attribut définit une relation avec l'objet référencé par le lien. Par exemple, si la destination du lien représente le glossaire associer au document, on pourrrait écrire

```
<A HREF="words.html" REL="glossary">
```

REV Cet attribut définit une relation entre l'objet courant et l'objet référencé par le lien. Par exemple, dans le cas où on a un ensemble linéaire de documents, on pourrait mettre l'attribut **REL** à «next» et l'attribut **REV** à «prev».

Bien que les attributs **REL** et **REV** semblent très intéressants, ils ne sont supportés que par quelques browsers. Généralement on utilise la balise **<LINK>** (Voir section 6.5) qui permet d'avoir le même type de comportement.

6.3 Les liens internes

Il est également possible de naviguer à l'intérieur d'un document HTML. En effet, il arrive souvent que la longueur du document soit beaucoup plus importante que ce que le browser peut afficher à l'écran. Supposons, par exemple que nous ayons plusieurs titres dans un même document, chacun est défini par une balise **<H1>** et un attribut **NAME** appelé «titre1»,«titre2». Il est possible d'introduire en début de document une table des matières reprenant les différents titres et permettant d'aller directement à la partie concernée.

C'est l'objet des liens internes. Lorsqu'une URL contient un caractère '#', par exemple «*url#name*», cela vaut dire que l'elle fait référence à un élément dont l'attribut **NAME** vaut *name* du document défini par l'URL *url*.

Listing 6.2 – Exemple de liens internes

```
<HTML>
  <HEAD>
    <TITLE> Links </TITLE>
  </HEAD>
  <BODY>
    <H2> Contents </H2>
    <P>
      <A HREF="#a"> Go to section A </A>
      of this document.
    </P>
    <P>
      <A HREF="http://www.mysite.org/index.html#c"> Go to section C </A>
      of another site.
    </P>
    <H2><A NAME="a"> Section A </A></H2>
    <P> Sample text of section A </P>
  </BODY>
</HTML>
```

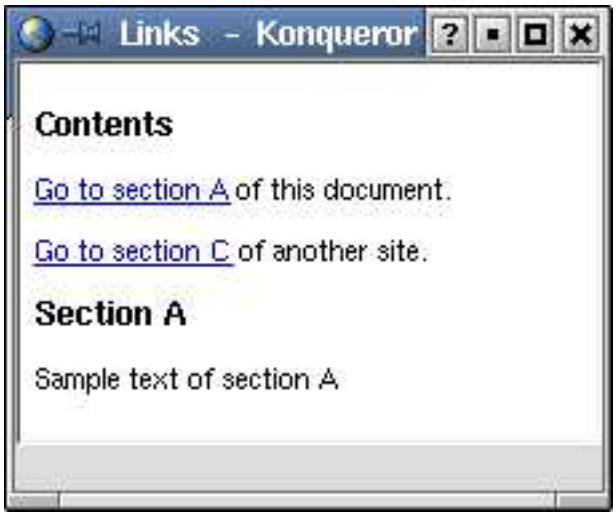


FIG. 6.2 – Exemple de liens internes

Le listing 6.2 et la FIG. 6.2 illustrent le concept de liens internes. Le lien interne de la ligne 8 fait référence à un lien dans le même document, alors que celui de la ligne 12 fait référence à un lien interne d'un autre site.

6.4 Les images actives

Le principe des images actives est d'associer plusieurs zones pouvant être sélectionnées par l'utilisateur à une grande image. Lorsqu'une zone particulière est sélectionnée, le browser est dévié vers un document HTML particulier. Il existe deux types d'images actives :

server side Lorsque l'utilisateur clique sur l'image, il faut envoyer une requête au serveur afin de décoder l'endroit et ainsi de dévier l'utilisateur vers le bon document.

client side Toute les informations sont stockées par le browser, et toutes les interactions sont gérées par celui-ci. Cette dernière solution à plusieurs avantages :

– il n'est pas nécessaire de faire appel à un service du serveur ;
– l'URL destination peut être affichée dans la ligne de statut ;

6.4.1 Les images actives coté serveur

Pour implanter une image active du coté serveur, il faut que la balise **<IMG>** représentant l'image contienne l'attribut **ISMAP**. L'exemple suivant illustre l'idée :

```
<A HREF="shapes.map"><IMG SRC="shapes.gif" ISMAP></A>
```

Les nombreux avantages des images actives coté client ont pour conséquence qu'il est généralement préconisé de les utiliser et donc de proscrire les images actives coté serveur.

6.4.2 Les images actives coté client

Pour implanter une image active, il faut rajouter l'attribut **USEMAP** dans la balise **<IMG>** et d'introduire une balise **<MAP>** et des balises **<AREA>** comme le montre le listing 5.8.

Listing 6.3 – Images actives coté client

```
<HTML>
  <HTML>
    <TITLE>  Client-Side Image Map  </TITLE>
  </HEAD>
  <BODY>
    <IMG SRC="shapes.gif" USEMAP="#shapes" BORDER="0">
    <MAP NAME="shapes">
      <AREA SHAPE="RECT" COORDS="6,50,140,143"
          HREF="r.html" ALT="Rectangle">
      <AREA SHAPE="CIRCLE" COORDS="195,100,50"
          HREF="c.html" ALT="Circle">
      <AREA SHAPE="POLY" COORDS="260,50,300,50,300,145,280,30,260,50"
          HREF="p.html" ALT="Polygon">
      <AREA SHAPE="default" HREF="default.html">
    </MAP>
  </BODY>
</HTML>
```

La balise **<AREA>** permet de décrire les différents zones dans l'image. La plupart des attributs sont identiques à ceux de la balise **<A>**, en particulier, les attributs **HREF**, **TARGET**, **TITLE** et **TABINDEX**. Mais la balise **<AREA>** définit également d'autres attributs :

SHAPE Cet attribut permet de spécifier le type de contour représentant la dite région. Les valeurs possible sont «RECT», «CIRCLE» et «POLY».

COORDS Cet attribut permet de définir les points caractéristiques du contour (Voir TAB. 6.1).

NOHREF Cet attribut permet d'indiquer que la région ne représente pas de destination.

TAB. 6.1 – Format des contours

Contour	Format des coordonnées	Example
RECT	$x_{gauche}, y_{haut}, x_{droite}, y_{bas}$	COORDS="0,0,100,50"
CIRCLE	$x_{centre}, y_{centre}, rayon$	COORDS="25,25,10"
POLY	$x_1, y_1, x_2, y_2, x_3, y_3, \ldots, x_1, y_1$	COORDS="255,122,306,53,334,62,255,122"

6.5 Les liens sémantiques

D'un point de vue sémantique, le fait de définir des liens d'un document vers un autre, par l'intermédiaire de la balise **<A>**, n'apporte aucun type d'interaction entre ces documents. C'est la raison pour laquelle la balise **<LINK>** a été introduite. La forme générale de cette balise est :

```
<LINK HREF=url REL=relation>
```

Le TAB. 6.2 reprend la liste des valeurs pour l'attribut **REL**, proposées par le langage HTML version 3.2, et permettant d'assigner une signification sémantique au document référencé par l'attribut **HREF**. Le TAB. 6.3 représente une série de valeurs proposées par les auteurs de documents HTML mais qui ne sont pas encore repris dans la norme HTML proprement dite. Les valeurs devraient pouvoir également être utilisées avec la balise **<A>**.

TAB. 6.2 – Valeurs proposées pour l'attribut **REL**

Valeur	Description du lien
ALTERNATE	Une version alternative du document courant. Cela peut être une traduction par exemple. Dans ce cas, on peut utiliser l'attribut **LANG** pour spécifier la langue.
APPENDIX	Une annexe pour un document ou un site.
BOOKMARK	Un bookmark. L'attribut **TITLE** peut être utilisé afin de spécifier le nom du bookmark.
CHAPTER	Un chapitre pour un site ou une série de documents.
CONTENTS	Une table des matières, soit pour un site entier, soit pour le document courant.
INDEX	Un index pour le document courant.
GLOSSARY	Glossaire pour les termes du document courant.
COPYRIGHT	Un document repésentant les copyrights du document courant.
NEXT	Le document suivant dans une liste linéaire de documents.
PREV	Le document précédent dans une liste linéaire de documents.
SECTION	Une section pour un site ou une série de documents.
START	Le premier document d'une série.
STYLESHEET	Une feuille de style externe (Voir partie III).
SUBSECTION	Une sous-section pour un site ou une série de documents.
HELP	Une aide sur le document courant.

TAB. 6.3 – Autres valeurs proposées

Valeur	Description du lien
NAVIGATE	Un document de navigation pour le site, comme une image active par exemple.
CHILD	La plupart des sites ont une structure hiérarchique. La relation enfant permet d'identifier des sous-documents. Les documents peuvent avoir de multiples liens **CHILD** dans la même hiérarchie.
PARENT	Relation opposée à celle de **CHILD**.
SIBLING	Une autre relation hiérarchique représentant un extrait d'un document. De nouveau, un document peut avoir plusieurs liens **SIBLING**.
BEGIN ou FIRST	Premier document d'une série linèaire de documents.
END ou LAST	Dernier document d'une série linéaire de documents.
BIBLIOENTRY	Une entrée dans la bibliographie.
BIBLIOGRAPHY	La bibliographie.
CITATION	Une citation. On l'utilise en général en conjonction avec la balise **CITE**.
DEFINITION	Une définition ou un terme. La signification du terme peut être trouvé dans le lien référencé par **GLOSSARY**.
FOOTNOTE	Représente une note de page. Cette relation est généralement utilisée avec la balise . Théoriquement, le browser devrait ouvrir une petite fenêtre avec le contenu du lien.
MADE	Identifie l'auteur du document.
AUTHOR	Des informatiions sur un des auteurs du document.
EDITOR	L'éditeur du document.
PUBLISHER	La société d'édition du document.
DISCLAIMER	Un désaveux concernant le document courant.
TRADEMARK	Un trademark concernant le document courant.
META	Document contenant des méta-informations en relation avec le document courant.
TRANSLATION	Une traduction du document dans une autre langue.

Il n'existe actuellement quasi pas de navigateurs supportant la balise **<LINK>**, à part peut-être le browser WebTV.

6.6 Les méta-informations

Un certain nombre d'informations peuvent être rajoutées dans un document HTML sans pour autant que celles-ci apparaissent à l'écran ; comme par exemple le nom de l'auteur ou encore la dernière date de

mise à jour du site. Le langage HTML connait une séquence de commentaires, mais chaque auteur ayant sa propre manière de commenter, il serait difficile d'en tirer profit, de par le manque de structuration des commentaires.

Afin de remédier à cela, le langage HTML introduit la balise **<META>**. Les balises **<META>** sont définies dans l'en-tête du document HTML, c'est-à-dire entre les balises **<HEAD>**. Le listing 6.4 montre un exemple d'une page HTML contenant des balises **<META>**.

Listing 6.4 – Utilisation de balise META

```
<HTML>
  <HEAD>
    <TITLE>  The Web Developer's Virtual Library  </TITLE>
    <META NAME = "Keywords" CONTENT = "HTML, CGI, Java, VRML,
                browsers, plugins, graphics, HTTP servers,
                JavaScript, Perl, ActiveX, Shockwave">
    <META NAME = "Description" CONTENT="Locate web authoring
                & software Internet resources at The WDVL, a
                well-organised goldmine with over 500 pages
                and thousands of links about HTML, CGI, Java,
                VRML, browsers, plugins, graphics, HTTP servers,
                JavaScript, Perl, ActiveX, Shockwave,...">
  </HEAD>
  <BODY>
      An example of HTML files containing META tags.
  </BODY>
</HTML>
```

La balise **<META>** est très ouverte. Il n'y a en effet aucune strandardisation concernant le type d'informations pouvant s'y trouver. Actuellement, le W3C travaille sur des approches plus sophistiquées pour encapsuler des méta-informations dans les documents.

6.6.1 L'attribut NAME

L'utilisation de l'attribut **NAME** est la plus facile à comprendre. La forme est :

<**META NAME**=*nature* **CONTENT**=*contenu*>

Avec l'attribut **NAME** le type de la méta-information est spécifiée, et l'attribut **CONTENT** fixe le contenu de celle-ci. Il existe des types de méta-informations qui sont comprises par les différents moteurs de recherche :

AUTHOR Cet attribut permet de spécifier le nom de l'auteur du document, éventuellement l'entreprise.

DESCRIPTION Cet attribut permet de décrire le contenu du document.

KEYWORDS Cet attribut permet d'associer des mot-clés caractérisant le document. Ceux-ci doivent être séparés par des virgules dans l'attribut **CONTENT**.

Ce type de méta-informations permet à des moteurs de recherche d'indéxer les documents HTML automatiquement. Afin d'éviter que cela se fasse de manière automatique, un fichier «robots.txt» peut être ajouté dans le répertoire racine du site web. En effet, les différents robots[1] essayent tout d'abord de lire le contenu de ce fichier afin de voir ce qu'ils peuvent indéxer. Si ce fichier n'existe pas, ils indexent automatiquement tout le site.

Le format du fichier est relativement simple, il contient un champ «User-agent» pour spécifier l'agent et un champ «Disallow» pour indiquer ce qu'il n'est pas possible de faire. Par exemple, le fichier suivant indique qu'aucun robot ne peut indexer quoique ce soit sur le site :

```
User-agent : *
Disalow : /
```

On peut également utiliser le type «ROBOT» pour l'attribut **NAME** pour spécifier la même chose, mais cela n'est pas encore supporté par tous les robots. Pour l'attribut **CONTENT** les différentes valeurs sont «ALL», «INDEX», «NOINDEX» et «NOFOLLOW».

[1] Un robot est un programme qui analyse le contenu de pages HTML et renvoie des informations à un serveur. Les moteurs de recherche s'en servent pour indéxer les nouvelles pages HTML.

6.6.2 L'attribut HTTP-EQUIV

Comme expliqué à la section 3.6, le protocole HTTP prévoit l'envoi d'une en-tête contenant des informations. A l'aide de l'attribut **HTTP-EQUIV**, il est possible de rajouter des informations dans l'en-tête HTTP lorsque le document HTML est envoyé. Le browser interprétera donc l'en-tête au moment de la lecture du document.

L'attribut **HTTP-EQUIV** permet de spécifier la nature de l'information rajoutée dans l'en-tête alors que l'attribut **CONTENT** permet de spécifier le contenu de celle-ci. Par exemple,

```
<META HTTP-EQUIV="Expires" CONTENT="Wed, 04 Jun 2000 22 :34 :07 GMT">
```

permet d'indiquer une date limite au-delà de laquelle le contenu du document ne sera plus valable.

En utilisant «**REFRESH**» comme valeur pour l'attribut **HTTP-EQUIV**, un autre document est automatiquement chargé par le browser après un délai exprimé en secondes. Par exemple, pour demander qu'après 15 secondes, le document «suivant.html» soit chargé, il faut utiliser :

```
<META HTTP-EQUIV="REFRESH" CONTENT="15,URL=suivant.html">
```

Il existe d'autres valeurs possibles pour l'attribut **HTTP-EQUIV**. Pour avoir une liste complète, il faut se reporter à un ouvrage de référence du protocole HTTP, par exemple [4].

Chapitre 7

Les tableaux

7.1 Les tableaux simples

Le langage HTML prévoit un jeu de balises permettant de représenter des informations sous forme tabulaire, comme un tableur par exemple. Le listing 7.1 et la FIG. 7.1 montrent un exemple simple de tableaux.

Listing 7.1 – Tableau (1)

```
<HTML>
  <HEAD>
    <TITLE> Table (1) </TITLE>
  </HEAD>
  <BODY>
    <TABLE BORDER="1" COLS="2">
      <CAPTION> Current Training </CAPTION>
      <TR>
        <TH> Name </TH>
        <TH> Desc </TH>
      </TR>
      <TR>
        <TD> John Smith </TD>
        <TD> Director </TD>
      </TR>
      <TR>
        <TD> Ann Brown </TD>
        <TD> Manager </TD>
      </TR>
    </TABLE>
  </BODY>
</HTML>
```

La balise principale définissant un tableau est **<TABLE>**. L'attribut **BORDER** permet de spécifier l'épaisseur des lignes séparant les lignes et les colonnes. Depuis la version 4 du langage HTML, l'attribut **COLS** a été introduit. Son but est de renseigner le browser sur le nombre de colonnes. Avec la balise **<CAPTION>**, il est possible d'associer un titre au tableau. Suivant qu'il s'agisse de la première ou de la dernière balise dans la définition du tableau, le titre apparaîtra en haut ou en bas du tableau. Chaque nouvelle ligne est insérée grâce à la balise **<TR>** (Table Row). Pour ajouter une cellule, il faut utiliser soit la balise **<TH>** (Table Header) pour les en-têtes ou la balise **<TD>** (Table Data) pour les informations.

Comme pour les listes, les tableaux peuvent être imbriqués. En particulier, ils sont souvent utilisés afin de permettre des alignements précis lorsqu'il y a de nombreux éléments dans un document HTML (Voir section 7.3).

Comme s'était le cas pour les listes, le langage HTML n'impose pas que les balises **<TR>**, **<TH>** et **<TD>** soient fermées explicitement. Mais tout le monde s'accorde à dire que fermer systématiquement les balises est une très bonne habitude, assurant ainsi une interprétation correcte de tableaux imbriqués par les browsers.

FIG. 7.1 – Tableau (1)

7.2 Les attributs COLSPAN et ROWSPAN

Les tableaux présentés à la section précédente sont de conception simple. En effet, la taille des cellules est figée. Avec les attributs **COLSPAN et ROWSPAN**, il est possible de diviser une colonne ou une ligne en plusieurs colonnes ou lignes. Le listing 7.2 et la FIG. 7.2 illustrent l'utilisation des attributs **COLSPAN** et **ROWSPAN**.

Listing 7.2 – Tableau (2)

```
<HTML>
  <HEAD>
    <TITLE> Table (2) </TITLE>
  </HEAD>
  <BODY>
    <TABLE BORDER="1" SUMMARY="This table shows details">
      <CAPTION> Current Training </CAPTION>
      <TR>
        <TH ROWSPAN=2> Name </TH>
        <TH COLSPAN=2> Module 1 </TH>
        <TH COLSPAN=2> Module 2 </TH>
      </TR>
      <TR>
        <TH> Passed </TH>
        <TH> Date </TH>
        <TH> Passed </TH>
        <TH> Date </TH>
      </TR>
      <TR>
        <TD> John Smith </TD>
        <TD> Yes </TD>
        <TD> 1.7.99 </TD>
        <TD> No </TD>
        <TD> ---- </TD>
      </TR>
      <TR>
        <TD> Ann Brown </TD>
        <TD> Yes </TD>
        <TD> 15.6.99 </TD>
        <TD> Yes </TD>
        <TD> 15.7.99 </TD>
      </TR>
    </TABLE>
  </BODY>
</HTML>
```

L'attribut **ROWSPAN** spécifie qu'une case déborde sur plusieurs lignes spécifiée par la valeur de l'attribut. Par exemple dans le cas de la FIG. 7.2, le titre «NAME» correspond à deux lignes ; pour cela, la balise correspondante (ligne 9) prend l'attribut **ROWSPAN** avec 2 pour valeur. Le browser va donc suppo-

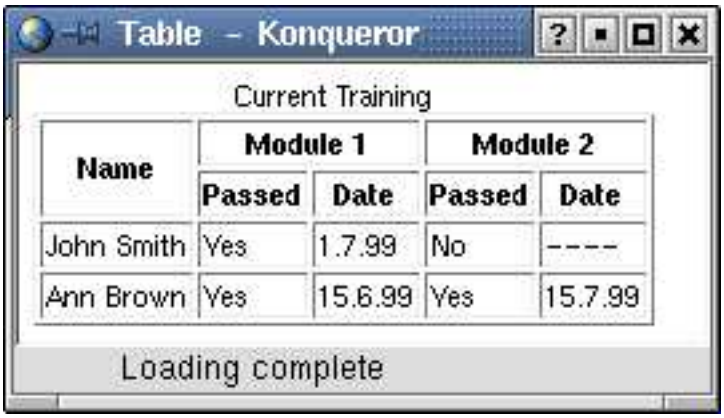


FIG. 7.2 – Tableau (2)

ser que les éléments suivants forment la premiere des deux lignes, et que la prochaine balise **<TR>** donnera les balises correspondantes à la deuxième ligne.

Les deux colonnes qui suivent sont également scindées en deux sous-colonnes. Pour faire cela, l'attribut **COLSPAN** est utilisé avec pour valeur le nombre de sous-colonnes (lignes 10 et 11). Dès lors, le browser sait que dans les prochaines balise **<TR>**, il doit trouver deux fois plus d'éléments que précédemment.

Le code HTML du document entre les lignes 14 et 17 est donc facile à comprendre. En effet, à cause de l'attribut **ROWSPAN**, il faut définir une nouvelle balise **<TR>** contenant des balises **<TH>**. De plus, à cause de l'attribut **COLSPAN**, il faut définir deux fois plus de ces balises, ce qui explique qu'il y en ait quatre.

La suite est évidente, pour chaque ligne (balises **<TR>**), il faut quatre élements pour représenter le contenu des différentes colonnes (balise **<TD>**).

7.3 Utiliser les tableaux pour la mise en page

Bien que les tableaux aient étés introduits afin de représenter des valeurs sous forme tabulée, de nombreuses extensions se sont ajoutées afin de les utiliser pour faire une mise en page plus poussée, en profitant du l'utilisation des éléments de tabulation, et donc d'alignement des cellules. En fait, le contenu de chaque cellule (balise **TD**) peut être n'importe quel élément HTML, du texte, des images ou des liens. Bien qu'il soit possible d'utiliser les balises **<CAPTION>** et **<TH>** dans ces circonstances, en pratique, ce n'est jamais le cas.

Ces extensions prennent la forme d'un certain nombre d'attributs pouvant être rajoutés avec les balises des tableaux :

WIDTH Cet attibut peut être utilisé avec les balises **<TABLE>** et **<TD>**. Il permet de fixer la longueur de la table ou de la cellule. Cette longueur peut être donnée de manière absolue (en pixels) ou relative (en spécifiant celle-ci comme un certain pourcentage de la longueur du browser (balise **<TABLE>**) ou du tableau (balise **<TD>**)).

BGCOLOR Cet attribut permet de spécifier une certaine couleur de fond. Il peut être utilisé avec les balises **<TABLE>**, **<TR>** et **<TD>**.

CELLPADDING Cet attribut permet de spécifier une certaine distance entre le bord d'une cellule et le point auquel commence le contenu. Il s'utilise avec la balise **<TABLE>**.

CELLSPACING Cet attribut permet de spécifier une distance entre deux cellules succésives. Il s'utilise avec la balise **<TABLE>**.

HSPACE Cet attribut permet de spécifier une distance horizontale entre une table et les éléments l'entourant. Il s'utilise avec la balise **<TABLE>**.

VSPACE Cet attribut permet de spécifier une distance verticale entre une table et les éléments l'entourant. Il s'utilise avec la balise **<TABLE>**.

Le listing 7.3 et la FIG. 7.3 illustrent l'utilisation des tableaux pour «diviser» le browser en deux parties : un menu à gauche et de l'information à droite.

Listing 7.3 – Tableau (3)

```
<HTML>
  <HEAD>
    <TITLE> Tableau (3) </TITLE>
  </HEAD>
  <BODY>
    <TABLE WIDTH="260">
    <TR>
      <TD WIDTH="50" BGCOLOR="Yellow">
        <A HREF="about.html"> About </A><BR>
        <A HREF="products.html"> Products </A><BR>
        <A HREF="staff.html"> Stafft </A><BR>
        </TD>
      <TD WIDTH="210">
        <H1 ALIGN="center"> Home Page </H1>
        <HR>
        <P>
          This text is the content of the home page.
        </P>
      </TD>
    </TR>
  </TABLE>
  </BODY>
</HTML>
```

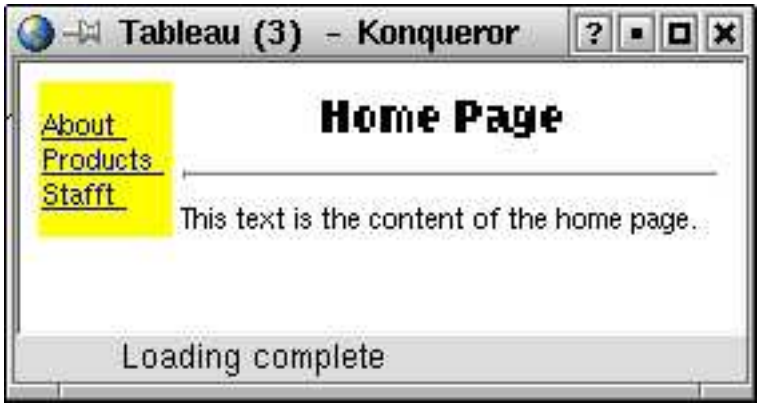


FIG. 7.3 – Tableau (3)

En combinant cela avec les attributs **ROWSPAN** et **COLSPAN**, il est possible de mettre au point de mise en page très complexe.

Remark: *La plupart des outils permettant de créer de manière «automatique» des documents HTML utilisent énormément les tableaux. Ils les utilisent systématiquement pour transformer les marges, ce qui rend le code source HTML très difficile à comprendre par la suite. C'est pourquoi, une fois le concept maîtrisé, la plupart des concepteurs préfèrent utiliser directement les balises HTML, plutôt que de passer par un de ces outils.*

Chapitre 8

Les cadres et les couches

8.1 Les cadres

L'idée des cadres («Frames» en anglais) est de pouvoir diviser un écran de browser en différentes régions indépendantes les unes des autres, mais communiquant entre elles. Chaque cadre est représenté par une URL unique. Chaque cadre peut faire apparaître des ascenseurs si le contenu du document ne tient pas dans la région attribuée.

Un bon exemple d'utilisation de cadres, comme le montre la FIG. 8.1, est celui où le site représente un livre. Dans ce cas, l'idéal est de diviser l'écran en deux parties : l'une contenant une table des matières et l'autre affichant l'information concernant les différents chapitres.

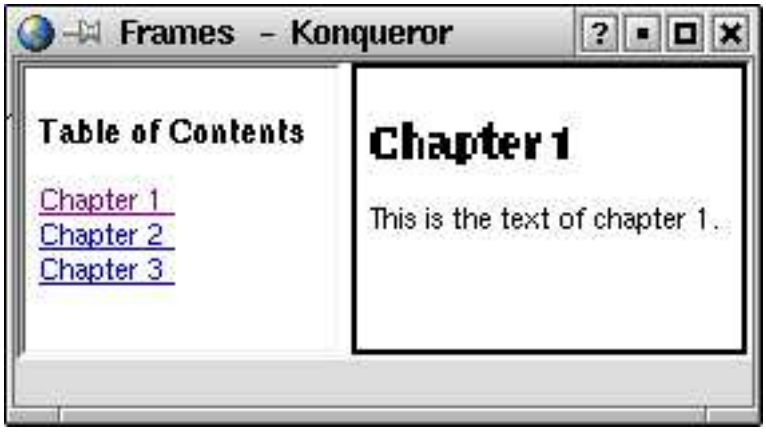


FIG. 8.1 – Un exemple de frame

8.1.1 Définir des cadres

Le listing 8.1 représente le code source du document HTML de la figure 8.1.

Pour définir un document HTML contenant des cadres, il faut remplacer la balise **<BODY>** traditionnelle par la balise **<FRAMESET>**, qui permet de diviser l'écran en plusieurs parties. Il existe de nombreux attributs accompagnant la balise **<FRAMESET>**, mais les deux plus important sont **COLS**, utilisé afin de diviser l'écran horizontalement et **ROWS**, utilisé afin de diviser l'écran verticalement. La valeur des ces attributs représente la taille des différents cadres. Ces tailles peuvent être exprimée en pixels ou en pourcentage par rapport à la taille du browser, ce qui permet aux frames de s'adapter aux dimensions du client. Un des cadres peut avoir une taille définie par «*» (Ligne 8 dans l'exemple), ce qui signifie la taille restante, c'est-à-dire la différence entre la taille du browser et la somme des tailles fixées des autres cadres. Il

Listing 8.1 – Cadres - fichier "main.html"

```
<HTML>
  <HEAD>
    <TITLE> Frames </TITLE>
  </HEAD>
  <FRAMESET COLS="120,*">
    <FRAME SRC="menu.html" NAME="menu">
    <FRAME SRC="chap1.html" NAME="main">
    <NOFRAMES>
      <P> This document needs frame support. </P>
    </NOFRAMES>
  </FRAMESET>
</HTML>
```

existe encore certains browsers ne supportant pas les cadres. La balise **<NOFRAMES>** permet de définir un contenu à afficher par le browser si celui-ci ne supporte pas les cadres.

Chaque cadre est défini par la balise **<FRAME>**. L'attribut **SRC** permet de spécifier l'URL correspondante, alors que l'attribut **NAME** permet d'identifier chaque cadre, ce qui sera nécessaire pour les faire communiquer. Il existe une relation parent-enfant entre les documents HTML utilisant les cadres. En effet, dans l'exemple donné, le fichier «main.html» (Listing 8.1) est le parent de «menu.html» et de «chap1.html» qui sont les enfants du fichier «main.html».

La balise **<FRAMESET>** permet de diviser l'écran soit horizontalement soit verticalement, mais les deux ne sont pas possible. Par contre, les balises **<FRAMESET>** peuvent s'imbriquer. Il est donc possible de diviser l'écran en deux parties verticales, et ensuite rediviser horizontalement l'une d'entre elles, comme illustré au listing 8.2.

Listing 8.2 – Imbrications de cadres

```
<HTML>
  <HEAD>
    <TITLE> Frames (2) </TITLE>
  </HEAD>
  <FRAMESET ROWS="25%,75%">
    <FRAMESET COLS="150,*">
      <FRAME SRC="logo.html" NAME="logo">
      <FRAME SRC="pub.html" NAME="pub">
    </FRAMESET>
    <FRAME SRC="main.html" NAME="main">
  </FRAMESET>
</HTML>
```

8.1.2 Communication entre cadres

Pour permettre l'interaction entre cadres, et donc permettre de changer le contenu d'un cadre à partir d'un autre, il faut utiliser l'attribut **TARGET** de la balise **<A>**, comme dans l'exemple :

```
<A HREF=ref TARGET=target> Charge la page 'ref' dans le cadre 'target' </A>
```

La valeur de l'attribut peut soit être le nom du cadre donné par l'attribut **NAME**, soit une des valeurs par défaut :

_blank Ouvre le contenu de la page dans une nouvelle fenêtre du browser.

_self Ouvre le contenu dans le cadre contenant la balise.

_parent Ouvre le contenu dans le cadre parent de celui contenant la balise.

_top Dans le cas où les cadres sont imbriqués, ouvre le contenu dans le cadre principal, c'est-à-dire celui qui n'a pas de parents.

Remark: *La spécification du langage HTML décourage l'utilisation du caractère underscore '_' pour le nom des cadres, et cela afin d'éviter toute confusion avec les valeurs par défaut possible comme **_top**.*

Le listing 8.3 définit une table des matières, et change le contenu du second cadre par l'intermédiaire de liens spécifiant pour l'attribut **TARGET** le nom de celui-ci (lignes 7-9).

Listing 8.3 – Table des matières

```
<HTML>
  <HEAD>
  </HEAD>
  <BODY>
    <H3>  Table of Contents  </H3>
    <A HREF="chap1.html" TARGET="main">  Chapter 1  </A><BR>
    <A HREF="chap2.html" TARGET="main">  Chapter 2  </A><BR>
    <A HREF="chap3.html" TARGET="main">  Chapter 3  </A>
  </BODY>
</HTML>
```

8.1.3 Les cadres flottants

Les cadres normaux ne peuvent être attachés que par rapport aux bords du browser. Microsoft a introduit l'idée de cadres flottants. Cela a été inclu dans la spécification HTML 4. Pour définir un cadre flottant, il faut utiliser la balise **<IFRAME>**. Celle-ci peut être inclue dans la balise **<BODY>** d'un document HTML normal. Les attributs principaux sont :

SRC Cet attribut permet de spécifier le document HTML qui sera affichée dans cette frame.

HEIGHT La hauteur du cadre (en pixels ou en pourcentage par rapport à la hauteur du browser).

WIDTH La largeur du cadre (en pixels ou en pourcentage par rapport à la largeur du browser).

En plus de ce la, la balise **<IFRAME>** supportent les attributs de positionnement comme **ALIGN**, **HSPACE** et **VSPACE** comme pour la balise **<IMG>**. Contrairement à la balise **<FRAME>**, la balise **<IFRAME>** doit être fermée, et contient le code HTML devant être affiché par les browsers ne supportant pas ce type de cadres. Un exemple est montré au listing 8.4 et à la FIG. 8.2.

Listing 8.4 – Cadre flottant

```
<HTML>
  <HEAD>
    <TITLE>  Frames (3)  </TITLE>
  </HEAD>
  <BODY>
    <IFRAME SRC="fig03.html" NAME="float" WIDTH="200" HEIGHT="150" ALIGN="LEFT">
      Floating frames not supported.
    </IFRAME>
    <P>
      This an example of floating frames (on the left) that can be used
      imbedded in HTML documents.
    </P>
  </BODY>
</HTML>
```

Remark: *Actuellement, seul Microsoft Explorer interprètement correctement les cadres flottants. Netscape ne les comprend pas, et Konqueror (Browser Linux/KDE) ne supportent pas encore les options d'alignement.*

8.1.4 Problèmes lors de l'utilisation de cadres

De nombreux experts sont très critiques concernant les cadres. Considérant l'implantation de la gestion des cadres par les browsers d'une part, et le manque de compréhension des designers concernant les problèmes potentiels d'autre part, ils estiment que les designers risquent de se perdre lors de la conception de sites.

Parmi les problèmes importants des cadres, il faut signaler la difficulté du design des documents en les utilisant, la confusion au niveau de la navigation à travers des documents utilisant abondamment des

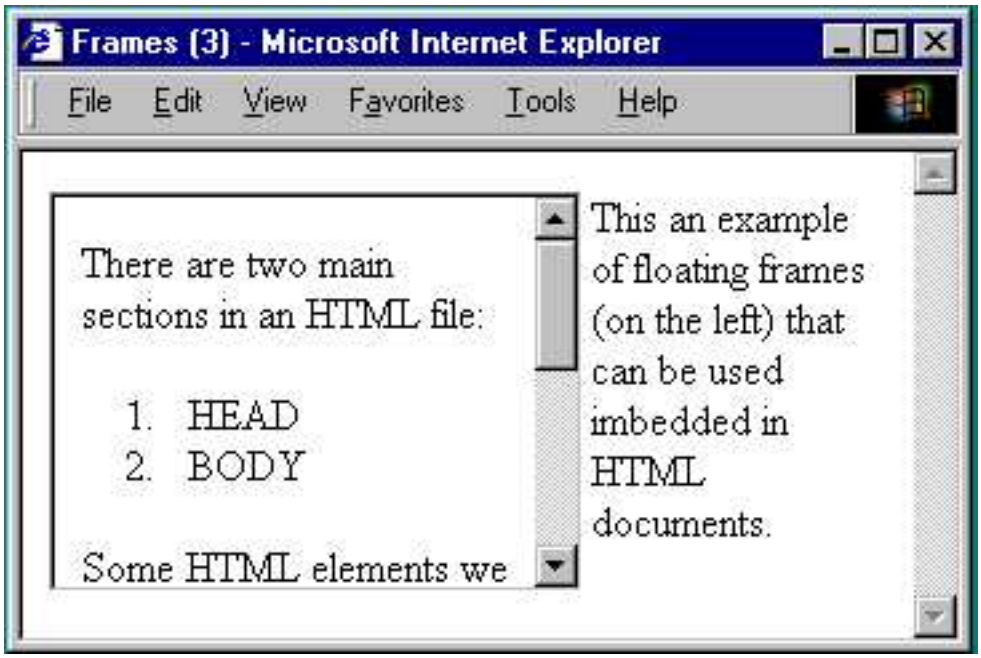


FIG. 8.2 – Cadre flottant

cadres, les problèmes avec les bookmarks, la perte du contexte des URLs ou encore des problèmes lors de l'impression.

L'utilisation de cadres peut donc être bénéfique pour la conception d'un site, mais il faut faire très attention à la manière de les utiliser.

8.2 Les couches

Les couches permettent de créer des zones de pages pouvant se superposer. En plus de la possibilité de pouvoir entasser des couches dans un document HTML, il est possible, grâce notamment à du code JavaScript Client-Side (Voir chapitre 19), de changer l'ordre des différentes couches ou de modifier le positionnement de celles-ci.

Bien qu'il soit possible d'utiliser les couches afin de contrôler le positionnement d'éléments dans un document HTML, la plupart des gens préfère utiliser le CSS (Voir chapitre III). Les couches restent pourtant une possibilité puissante, surtout si elles sont combinées avec le JavaScript Client-Side.

L'idée de base est similaire à celle des cadres, si ce n'est que les couches peuvent se superposer. Il existe deux formes de couches :

- les couches positionnées (balise **<LAYER>**) ;
- les flots de couches (balise **<ILAYER>**, «Inflow Layers» en anglais).

8.2.1 Les couches positionnées

Pour définir une couche positionée, il faut utiliser la balise **<LAYER>**, dont la forme la plus utilisée est :

```
<LAYER ID=id TOP=top LEFT=left SRC=src> Contenu </LAYER>
```

Les attributs ont la signification suivante :

ID Cet attribut permet d'assigner un nom à une couche, ce qui permettra de la manipuler par la suite.

TOP La position verticale de la couche par rapport au browser.

LEFT La position horizontale de la couche par rapport au browser.

SRC Cet attribut permet de spécifier une URL représentant le contenu de la couche.

Comme c'était le cas avec les cadres, il existe une balise **<NOLAYER>** qui contient le code HTML devant être géré par les browsers ne supportant pas les couches, dans ce cas il est nécessaire d'utiliser l'attribut **SRC**.

Le listing 8.5 et la FIG. 8.3 montrent un exemple de couches.

Listing 8.5 – Couches positionnées

```
<HTML>
  <HEAD>
    <TITLE> Couches (1) </TITLE>
  </HEAD>
  <BODY>
    <LAYER ID="11" TOP="20" LEFT="20" BGCOLOR="Yellow">
      <H3> Layer 1 </H3>
      <P> Text of layer 1 (20,20) </P>
    </LAYER>
    <LAYER ID="12" TOP="100" LEFT="50" BGCOLOR="Yellow">
      <H3> Layer 2 </H3>
      <P> Text of layer 2 (50,50) </P>
    </LAYER>
  </BODY>
</HTML>
```

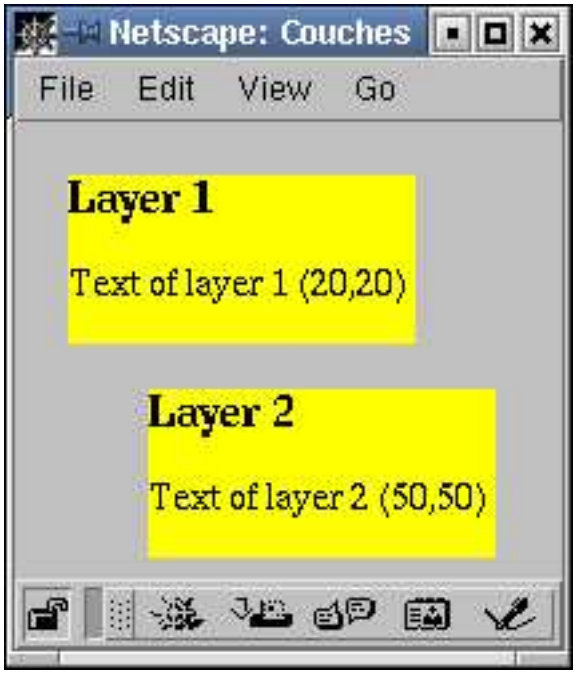


FIG. 8.3 – Couches positionnées

Remark: *Actuellement, seul Netscape gère correctement les couches. Microsoft Explorer ainsi que Konqueror (Linux/KDE) ne les gèrent pas complètement.*

8.2.2 Les flots de couches

Les flots de couches offrent un moyen de gérer des couches comme étant dans objets HTML normaux apparaissant dans le flux du document, un peu comme les images par exemple. La forme la plus utilisée se base sur la balise **<ILAYER>** et prend la forme :

```
<ILAYER ID=id TOP=top LEFT=left > Contenu </ILAYER>
```

Les attributs ont la signification suivante :

ID Cet attribut permet d'assigner un nom à une couche, ce qui permettra de la manipuler par la suite.

TOP La position verticale de la couche par rapport à son contexte actuel.

LEFT La position horizontale de la couche par rapport à son contexte actuel.

Le listing 8.6 et la FIG. 8.4 illustrent les flots de couches .

Listing 8.6 – Flot de Couches

```
<HTML>
  <HEAD>
    <TITLE> Layers (2) </TITLE>
  </HEAD>
  <BODY>
    <P>
      This
      <ILAYER ID="11" LEFT="25" BGCOLOR="Yellow">
        layer 1 is positioned 25 pixels from the left
      </ILAYER>
      of the current flow.
    </P>
    <P>
      This
      <ILAYER ID="12" TOP="15" BGCOLOR="Yellow">
        layer 2 is postioned 15 pixels below
      </LAYER>
      the current flow.
    </P>
  </BODY>
</HTML>
```

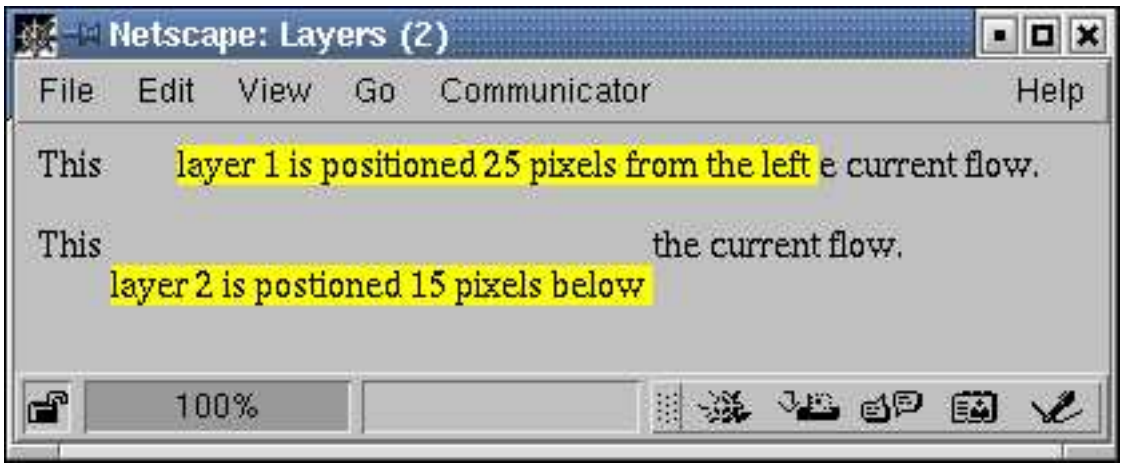


FIG. 8.4 – Couches positionnées

Le fait de décaler la première couche vers la gauche, a pour conséquence de cacher le texte normal, ce qui est logique puisque les couches peuvent se superposer, et que le browser considère que les éléments hors d'une quelconque couche appartiennent à une couche «fictive».

Remark: *Actuellement, seul Netscape gère correctement les flots de couches. Microsoft Explorer ainsi que Konqueror (Linux/KDE) ne les gèrent pas complètement.*

Chapitre 9

Formulaires

Jusqu'ici, les balises HTML présentées permettaient d'afficher des informations, mais il n'y avait aucune interaction possible avec l'utilisateur du browser ; en particulier, celui-ci ne pouvait pas renvoyer des informations vers le serveur. L'objectif des formulaires est de donner la possibilité de créer des champs et des boutons, appelés composants, dans un document HTML.

9.1 La balise FORM

La forme générale de la balise permettant de créer un formulaire est la suivante :

```
<FORM ACTION=url TARGET=target METHOD=method>
  <!-- Insèrer ici des composants -->
</FORM>
```

L'attribut **TARGET** permet d'envoyer le résultat vers un autre cadre (Voir section 8).

Les composants sont utilisés afin de permettre à l'utilisateur d'entrer les données qui seront ensuite envoyées au serveur. A chaque composant un nom peut être associé grâce à l'attribut **NAME** qui représente une variable, les données contenues dans les composants représentant la valeur de ces variables. Tous les composants peuvent avoir un attribut **READONLY**. Comme son nom l'indique, cela permet d'empêcher l'utilisateur de changer la valeur contenue dans ce composant.

9.1.1 L'attribut ACTION

L'attribut **ACTION** permet de spécifier, par l'intermédiaire d'une URL, où le formulaire devra renvoyer le contenu des composants qu'il contient. Si le serveur web supporte l'interface CGI (Voir chapitre **??**), un programme ou un script peut récupérer le contenu des composants, ce qui permet des traitements très complexes, comme par exemple écrire les informations ainsi encodées dans une base de données.

Le listing 9.1 et la FIG. 9.1 utilisent l'URL pour rediriger le résultat d'une requête dans un courrier électronique.

Listing 9.1 – Formulaire (1)

```
<HTML>
  <HEAD>
    <TITLE> Form (1) </TITLE>
  </HEAD>
  <BODY>
    <FORM METHOD="POST" ENCTYPE="text/plain"
        ACTION="mailto:admin@site.com?subjet='Coucou'">
      <INPUT TYPE="text" NAME="text">
      <INPUT TYPE="submit" NAME="submit" Value="Send to admin">
    </FORM>
  </BODY>
</HTML>
```

FIG. 9.1 – Formulaire (1)

9.1.2 L'attribut METHOD

L'attribut **METHOD** permet de spécifier la manière dont les données seront envoyées à l'entité spécifiée par l'attribut **ACTION**. Il y a deux méthodes possibles :

- la méthode GET ;
- la méthode POST.

9.1.2.1 La méthode GET

Cette méthode est celle utilisée par défaut par la plupart des browsers. Les variables ainsi que leur contenu sont renvoyés directement en ajoutant les informations dans l'URL. Par exemple, supposons que l'attribut **ACTION** spécifie l'URL «www.qql.com/cgi-bin/exec», le serveur recevra comme requête du formulaire du type :

```
www.qql.com/cgi-bin/exec?Name=ULB&Theme=Formation+Internet&NbParticipants=30
```

Après l'URL, il y a un point d'intérrogation (?). Ensuite les variables sont mises bout-à-bout et séparées par des «et commerciales» (&). Chaque variable est constituée de son nom, d'un signe égal (=) et de sa valeur. Si la valeur contient plusieurs mots, ceux-ci sont séparés par des «signes plus» (+).

Les deux inconvénients de la méthode GET, sont :

- la requête envoyée au serveur peut devenir très grande, en supposant que le browser puisse envoyer une requête aussi longue au serveur sans la tronquer d'une partie des informations ;
- toutes les informations apparaissent directement dans l'URL ce qui n'est pas très sûr.

9.1.2.2 POST

La méthode POST envoie les variables et leur contenu directement après la requête. En effet, dès que le serveur réceptionne la requête, il établit une communication avec le browser afin d'échanger les informations. Celles-ci n'apparaissent donc plus dans l'URL destinataire, et aucune restriction n'existe quant à leur taille et leur nombre.

9.2 Les composants de texte

Il s'agit de composants permettant à l'utilisateur de pouvoir entrer du texte.

9.2.1 Les champs textes

Pour créer un champ texte, la balise suivante est utilisée :

```
<INPUT TYPE="TEXT" NAME=name SIZE=size MAXLENGTH=max VALUE=défaut>
```

Les attributs ont la signification suivante :

SIZE Cet attribut permet de spécifier la taille du composant dans le document HTML.

MAXLENGTH Cet attribut permet de spécifier le nombre maximal de caractères accepté dans le champ.

VALUE Cet attribut permet de spécifier un contenu par défaut du champ.

Un exemple est donné à la FIG. 9.2 et au listing 9.2.

Listing 9.2 – Formulaire (2)

```
<HTML>
  <HEAD>
    <TITLE> Form (2) </TITLE>
  </HEAD>
  <BODY>
    <FORM METHOD="POST" ACTION="www.site.com/cgi-bin/ok">
      Last Name: <INPUT TYPE="text" NAME="LastName"><BR>
      First Name: <INPUT TYPE="text" NAME="FirstName"><BR>
      Title: <INPUT TYPE="text" NAME="title" VALUE="Ir.">
    </FORM>
  </BODY>
</HTML>
```

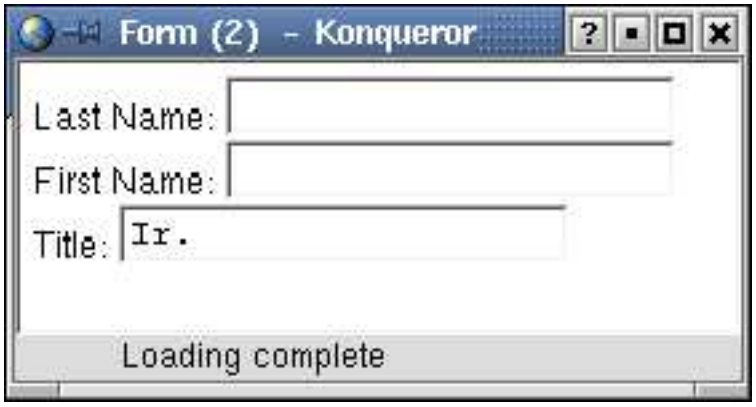


FIG. 9.2 – Un exemple de champ texte

9.2.2 Les champs de mots de passe

Lorsque l'utilisateur doit entrer un code secret, un champ texte ne peut convenir puisque le contenu de celui-ci est visible à l'écran. Pour éviter cela, il y a la possibilité de créer des champs textes dont le contenu sera remplacé à l'écran par des '*'. La syntaxe est :

```
<INPUT TYPE="PASSWORD" NAME=name SIZE=size MAXLENGTH=max VALUE=défaut>
```

La signification des paramètres est la même que pour les champs textes normaux.

9.2.3 Les champs multi-lignes

Souvent il est nécessaire de pouvoir encoder plusieurs lignes, comme dans le cas d'un commentaire. Puisque ceci n'est pas possible avec un champ texte normal, le langage HTML introduit une balise spécifique :

```
<TEXTAREA ROWS=rows COLS=cols NAME=name>
Ceci est la valeur par défaut.
</TEXTAREA>
```

La signification des attributs est la suivante :

COLS Cet attribut permet de spécifier le nombre maximal de caractères par ligne.

ROWS Cet attribut permet de spécifier le nombre maximal de lignes.

Un exemple est montré à la FIG. 9.3 et au listing 9.3.

Listing 9.3 – Formulaire (3)

```
<HTML>
  <HEAD>
    <TITLE> Form (3) </TITLE>
  </HEAD>
  <BODY>
    <FORM METHOD="POST" ACTION="www.site.com/cgi-bin/ok">
      Remarks:
      <TEXTAREA NAME="Rem" COLS="30" ROWS="5">
        By default, no remarks.
      </TEXTAREA>
    </FORM>
  </BODY>
</HTML>
```

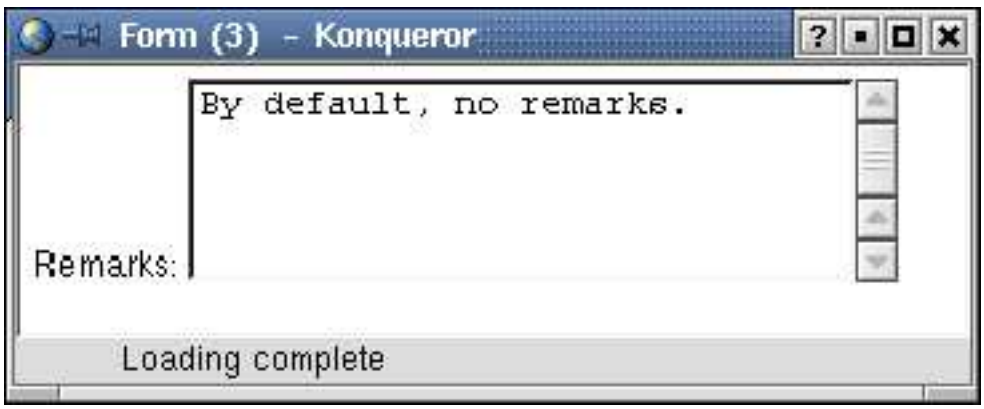


FIG. 9.3 – Formulaire (3)

9.3 Les sélecteurs d'options

Il arrive parfois que l'utilisateur ait choisir une seule valeur parmi un certain nombre de propositions. La balise **<SELECT>** permet de préciser ces choix possibles en spécifiant avec l'attribut **NAME** le nom de la variable concernée. La forme générale est :

```
<SELECT NAME=name>
  <OPTION VALUE=value1> Option 1
  <OPTION VALUE=value2> Option 2
```

```
    <OPTION VALUE=value3 SELECTED> Option 3
  </SELECT>
```

Chaque choix possible doit être encodé à l'aide de la balise **<OPTION>**. L'attribut **VALUE** permet de spécifier la valeur de la variable lorsque le choix est sélectionné. Si celui-ci n'est pas spécifié, c'est le choix lui-même qui sera renvoyé. L'un des choix peut avoir l'attribut **SELECTED** signifiant ainsi que c'est le choix par défaut.

Le listing 9.4 montre un exemple (Lignes 8-12) dont le résultat est représenté dans la FIG. 9.4 par la partie «Formation».

Listing 9.4 – Formulaire (4)

```
<HTML>
  <HEAD>
    <TITLE> Form (4) </TITLE>
  </HEAD>
  <BODY>
    <FORM METHOD="POST" ACTION="www.site.com/cgi-bin/ok">
      Formation: <BR>
      <SELECT NAME="Industry">
        <OPTION VALUE="Ch"> Chemicals
        <OPTION VALUE="Car"> Cars
        <OPTION VALUE="Bk" SELECTED > Banks
      </SELECT><BR><BR>
      Transport: <BR>
      Car: <INPUT TYPE="CHECKBOX" NAME="Car" VALUE="Yes" CHECKED>  
      Bus: <INPUT TYPE="CHECKBOX" NAME="Bus" VALUE="Yes" CHECKED>  
      Train: <INPUT TYPE="CHECKBOX" NAME="Train" VALUE="Yes"><BR><BR>
      Sex: <BR>
      Male: <INPUT TYPE="RADIO" NAME="Sex" VALUE="Male">   
      Female: <INPUT TYPE="RADIO" NAME="Sex" VALUE="Female">
    </FORM>
  </BODY>
</HTML>
```

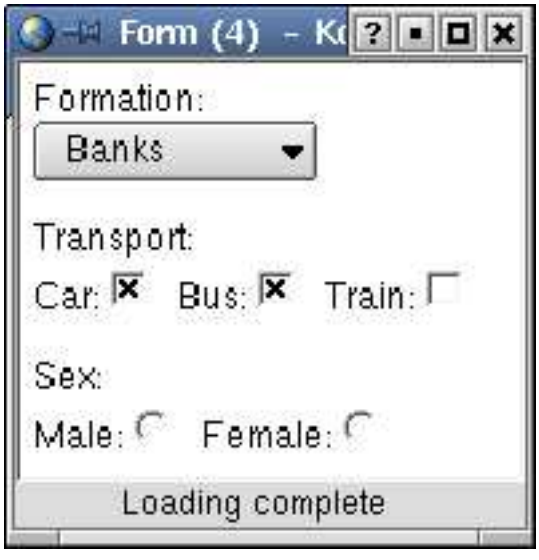


FIG. 9.4 – Formulaire (4)

9.4 Les cases à cocher

Les cases à cocher donnent la possibilité d'associer avec un choix le fait qu'il soit sélectionné ou pas. La forme générale est :

```
<INPUT TYPE="CHECKBOX" NAME=name VALUE=value CHECKED>
```

Si la case à cocher est sélectionnée, la réponse suivante sera renvoyée au serveur «*nom=valeur*», sinon rien n'est envoyé concernant cette variable. L'attribut **CHECKED** permet de spécifier que la case à cocher est sélectionnée par défaut.

Plusieurs cases à cocher peuvent avoir le même attribut **NAME**, du moment qu'elles ont des attributs **VALUE** différents. Dans ce cas, la réponse suivante sera renvoyée au serveur «*nom=val1&nom=val2*»

Le listing 9.4 (Lignes 14-16) montre un exemple dont le résultat est représenté dans la FIG. 9.4 par la partie «Transport».

9.5 Les cases à options

Les cases à options fonctionnent un peu comme les cases à cocher, si ce n'est qu'un seul choix peut être sélectionné parmi les différentes possibilités offertes pour une certaine variable. Le format est d'ailleurs identique, si ce n'est que la valeur de l'attribut **TYPE** doit avoir la valeur «RADIO».

```
<INPUT TYPE="RADIO" NAME=name VALUE=value1 CHECKED>
<INPUT TYPE="RADIO" NAME=name VALUE=value2>
```

A partir du moment où un seul choix est possible, il n'y a forcément qu'une seule balise **<INPUT>** pour un certain nom de variable qui peut avoir l'attribut **CHECKED**.

Le listing 9.4 (Lignes 18-19) montre un exemple dont le résultat est représenté dans la FIG. 9.4 par la partie «Sex».

9.6 Les boutons

L'affichage d'un bouton dans un document HTML est également possible, grâce à la syntaxe :

```
<INPUT TYPE=type VALUE=value NAME=name>
```

L'attribut **VALUE** permet d'indiquer le label du bouton, alors que l'attribut **NAME** permet de lui donner un nom. L'attribut **TYPE** ne peut prendre que deux valeurs :

RESET Lorsque l'utilisateur clique sur ce bouton, tous les champs sont vidés et reprennent leur valeur par défaut.

SUBMIT Lorsque l'utilisateur clique sur ce bouton, les données sont envoyées au serveur en utilisant la méthode spécifiée dans la balise **<FORM>**.

Il existe une alternative pour le bouton **SUBMIT** si le contenu du bouton doit être une image :

```
<INPUT TYPE="IMAGE" SRC=url NAME=name ALT=alt>
```

L'attribut **SRC** permet de spécifier l'URL contenant l'image à afficher. Enfin, l'attribut **ALT** permet de spécifier un texte qui s' affiche lorsque l'utilisateur reste quelques instants sur le bouton.

9.7 Les composants invisibles

Il y a un dernier type de composant qui est très souvent utilisé. Il s'agit des composants invisibles, c'est-à-dire des composants n'apparaissant pas à l'écran mais permettant de pouvoir associer des valeurs à des variables sans que l'utilisateur ne puisse les voir. La forme générale est :

```
<INPUT TYPE="HIDDEN" NAME=name VALUE=value>
```

Par exemple, si un utilisateur doit remplir un certain nombre d'informations réparties dans deux documents HTML afin de faciliter la lisibilité. Le premier lui demande des renseignements généraux (nom, prénom, etc...), alors que dans le deuxième document il doit entrer des informations concernant ses études. Lorsque l'utilisateur clique sur le bouton «OK» du premier document, le second document apparaît afin d'encoder les informations supplémentaires. Lorsque l'utilisateur clique sur le bouton «Enregistrer», les informations sont sensées être sauvegardées dans une base de données. Or, par défaut, dans le second document, toutes traces des informations encodées dans le premier document sont perdues. Les composants invisibless sont une solution dans ce genre de situation. Ils permettent de laisser les informations du premier document disponibles sans que l'utilisateur ne puisse les modifier.

Troisième partie

Cascading Style Sheets

Chapitre 10

Introduction

10.1 Origine des styles

L'une des principales limitations du langage HTML est qu'il mélange le contenu de l'information avec la manière de l'afficher. En effet, pour changer une information ou sa présentation, il faut manipuler le même document HTML. Cela pose quelques problèmes :

- il faut des compétences HTML pour changer le contenu, puisqu'il faut utiliser des balises HTML ;
- il faut connaître la nature de l'information pour changer sa forme ;
- alors qu'une personne peut être responsable du contenu et une autre du contenant, elles ont toutes les deux l'accès à l'ensemble, ce qui n'est pas très sûr.
- les balises sont souvent utilisées à mauvais escien (par exemple utiliser une batterie de
 pour séparer deux parties d'un document, alors que la propriété *padding* ou *margin* permet de le faire plus proprement, utiliser des tableaux pour aligner des éléments voir listing 10.1 et figure 10.1, etc.)

Listing 10.1 – Utilisation des tableaux pour la mise en page

```
<html>
  <head>
    <title>Positionning with tables example</title>example
  </head>
  <body>
    <H1 ALIGN="center">
    Positionning with tables
    </H1>
    <TABLE ALIGN="center">
      <TR>
        <TD ALIGN="center">
          <IMG SRC="ulb_ap1.gif" width="3cm" height="2cm">
        </TD>
        <TD>
        </TD>
        <TD ALIGN="center">
          This on the same line as the image
        </TD>
      <TR>
        <TD>
        </TD>
        <TD ALIGN="center">
          This is exactly placed  <BR>
          between the image and  <BR>
          the text but under both.
        </TD>
        <TD>
        </TD>
      </TR>
    </TABLE>
</body>
</html>
```

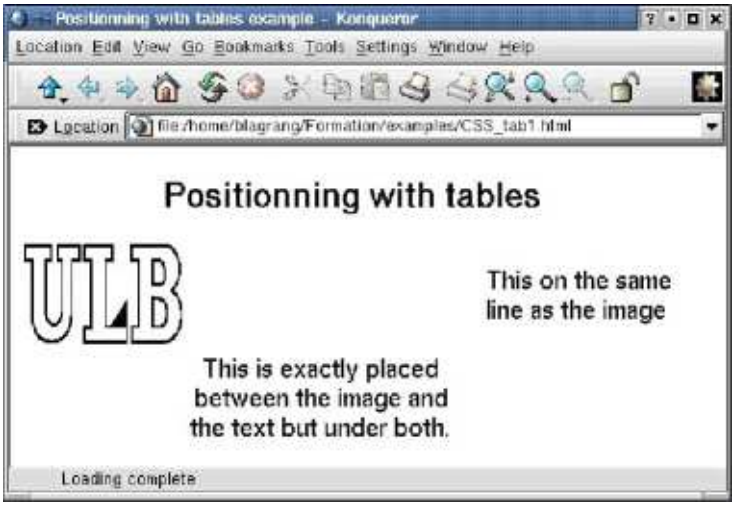


FIG. 10.1 – Exemple d'utilisation de tableaux pour la mise en page.

C'est pour répondre à cette problématique que le W3C introduit la notion de styles, appellés Cascading Style Sheets (CSS). La première mouture de cette spécification (CSS1 -Level 1) date de 1996. Une deuxième version plus élaborée (CSS2 - Level2) est sortie en 1998.

10.2 Les styles et les browsers

Il est important de signaler que ce sont les browsers qui font passer une idée de théoriciens académiques à la pratique et à l'heure actuelle, tous les browsers ne supportent pas encore l'entièreté des spécifications CSS, même si cela devrait très vite changer. En effet, le premier browser à avoir supporté partiellement le CSS est l' Internet Explorere 3.0 de Microsoft. Ensuite, ce fut le tour de Netscape 4.0 même si depuis le début, Netscape a été réticent à entrer dans cette technologie en essayant son propre standard, le JSSS. Cependant ce fut un échec et ils revinrent au CSS. Il est remarquer que l'on retrouve le JavaScript Sytle Sheet (JSSS) dans le Javascript implémenté par Netscape. Enfin, Opera 3.5 supporte également le CSS. Il faut également noter que la nouvelle version de Netscape - Netscape 6 - toujours en développement supporte presque entièrement les spécifications CSS. Cependant, son usage est encore peu répandu, notamment parce qu'il s'agit toujours d'une version beta (avec encore un grand nombre de bogues) et qu'il fonctionne très lentement par rapport aux autres browsers. Un mot sur Amaya qui est le browser supporté par le W3C. Malheureusement, ce browser est toujours en développement et ne supporte pas toutes les spécifications CSS en plus de posséder encore de nombreux bogues. Il existe des validateurs de CSS qui permettent de vérifier que les styles définis respectent bien les spécifications CSS. Pour plus d'informations à ce sujet, le lecteur pourra se référer au site suivant : http ://www.w3c.org/CSS.

Chapitre 11

Définition des styles

11.1 Forme générale

La définition d'un style revient à décrire tous les éléments qui le constituent. Les éléments non redéfinis dans le style garderont leurs valeurs par défaut déterminées par chaque browser pour autant qu'il s'agisse d'éléments correspondants à des balises HTML. Un style est constitué d'un ensemble de règles.

11.1.1 Syntaxe d'une règle :

pour définir une règle, il faut utiliser la forme générale :

```
élément { propriété1 : valeur1; propriété2 : valeur2 }
```

On remarque qu'on nomme tout d'abord l'élément (ou sélecteur), et on met les spécifications (déclarations) entre accolades. Chaque propriété est tout d'abord écrite, puis après un double-point, on écrit la valeur que l'on lui assigne. Lorsqu'il y a plusieurs propriétés, on les sépare par des points-virgules. Grâce à cette possibilité, il est donc possible de changer le format des balises définies par le langage HTML. Un exemple est donné par :

```
H1 {color : red}
```

Dans cet exemple on spécifie que les titres de niveaux 1 sont en rouge voir figure 11.1

Une fois l'élément défini, on pourra l'utiliser comme une balise normale dans un document HTML qui l'inclura dans son en-tête par l'une des méthodes expliquées à la section 11.3.

11.1.2 Grouper des sélecteurs et des règles :

il est toujours préférable de limiter la taille des fichiers transférés au browser. Il est possible de réduire la taille des styles CSS en utilisant quelques règles admises :

– Grouper les sélecteurs. Par exemple si on a :

```
H1 { font-weight : bold }
H2 { font-weight : bold }
H3 { font-weight : bold },
```

il est possible de grouper les sélecteurs dans une même règle

```
H1 H2 H3 { font-weight : bold }.
```

– Grouper les règles. Par exemple si on a pour le même sélecteur plusieurs règles s'appliquant, on groupe toutes les déclarations dans une même règle :

```
H1 { font-weight : bold }
H1 { color : blue },
devient
H1 { font-weight : bold; color : blue }
```

FIG. 11.1 – Exemple de style : H1 {color : red}

Le but des prochaines sections est de montrer quelques propriétés pouvant être utilisées dans les styles. Pour une étude approfondie, il faut se référer à [28]. Un autre bon livre pour commencer est [2].

11.2 Sélecteurs

11.2.1 Sélecteur associé à une balise HTML

Le plus simple des sélecteurs est le nom de la balise HTML pour laquelle on veut que s'applique la règle de style définie. Par exemple la règle suivante définira un titre de premier niveau (balise <H1>) dont la police utilisée sera Verdana :

```
H1 {font-family : Verdana;}
```

11.2.2 Sélecteur associé à une classe.

Comme expliqué dans la section 4.6, on peut associer via l'attribut 'CLASS' un nom de classe à différentes balises d'un document HTML. De la même manière, on peut définir des styles pour des classes. La forme est définie :

```
.classe { propriété1 : valeur1; propriété2 : valeur2 }
```

Pour définir une classe, on commence par un point, pour le reste, c'est exactement la même chose que pour les éléments. Toutes les balises de la même classe se voient donc appliquer le même style.

11.2.3 Sélecteur associé à une identité

De la même manière, il est possible d'associer un nom d'identité à différentes balises d'un document HTML via l'attribut **ID**. Par exemple une partie de document appartient à la classe 'chapitre' et s'appelle 'chapitre1'. Cette partie sera par exemple située entre les balises suivantes :

```
<DIV CLASS="chapitre" ID="chapitre1">
<P>Ceci est le premier chapitre du livre. Il parlera de blablablablablabla
blablabla blablabla blablabla ... blablabla.
</P>
</DIV>
```

Il est donc possible d'associer un style à une identité. La forme utilisée est :

```
#id {propriété1 : valeur1 ; propriété2 : valeur2}
```

Cette manière de fonctionner apporte un supplément d'information sur le contenu du document. En effet, non seulement on sait que l'on a affaire à un paragraphe mais en plus on sait que ce paragraphe appartient à un chapitre qui s'appelle chapitre1. Donc, n'importe qui lisant ce document aura ces information supplémentaire et pourra donc conclure que ce document est de type livre. L'utilisation combinée des attributs **CLASS** et **ID** avec la balise **DIV** permet d'avancer un pas supplémentaire dans la séparation du contenu et du contenant qui se rapproche grandement du eXtensible Markup Language (XML) (Voir partie IV).

11.2.4 Autres sélecteurs

Il existe un grand nombre de sélecteurs permettant d'appliquer les règles de style à des éléments bien définis. On trouvera dans la table 11.1 la liste des différents sélecteurs. Cette liste est issue des spécifications CSS de la W3C. Cependant aucun browser ne supporte l'entièreté de cette liste. Le lecteur devrait se limiter aux sélecteurs suivants : ***** ; **E** ; **F E** ; **.class** ; **#id** ; **:active** ; **:visited** ; **:link**. Attention, **F E** n'est pas supporté par Netscape 4.7. Par contre, Netscape 6 supporte d'autres sélecteurs comme **F + E** ; **F > E** ; **:before** ; **:after**.

Sélecteur	Éléments sélectionnés	Exemple
*	tous les éléments	
E	tous les éléments correspondant à la balise E	<E>
F E	tous les éléments E descendant de l'élément F	<F> <DIV> <E> </E></DIV></F> la règle s'applique seulement à ce E
F > E	tous les éléments E enfants de l'élément F	<F><E></E></f> : la règle s'applique à ce E mais pas au E du cas précédent
F + E	tous les éléments E qui suivent directement F, E et F ayant même parent	<H1><F></F><E></E></H1>
.class	tous les éléments qui appartiennent à la classe «class»	<E class="class">
#id	tous les éléments qui ont l'identité «id»	<E id="id">
:first-child	tous les éléments qui sont les premiers enfants de leur parent	<H1><E></E><F></F></H1> s'applique à E seulement
:link	tous les liens non visités	
:visited	tous les liens visités	
:active	tous les él. activés par l'utilisateur mais l'action correspondante est en cours	
:hover	id. mais la souris glisse sur l'élément	
:focus	idem mais l'élément est prêt à recevoir une entrée clavier	
[att]	tous les éléments qui ont un attribut de type "att"	<E att="val1"></E>
[att="val"]	tous les éléments qui ont un attribut "att" à la valeur "val"	<E att"=val"></E>
[att-="val"]	tous les éléments qui ont un attribut "att" qui contient le mot "val"	<E att="val 2"></E>
[attl="val"]	tous les éléments qui ont un attribut de la forme "val-..."	
E, F	tous les éléments de type E ou F	
E :first-letter	la première lettre de tous les éléments de type E	<E>Bonjour</E> : la règle s'applique à "B"
E :first-line	la première ligne de tous les éléments de type E	
E :before	le texte inséré au début du bloc de l'élément E	
E :after	le texte inséré à la fin du bloc de l'élément E	

TAB. 11.1 – Sélecteurs d'une règle définis par CSS.

11.3 Appliquer un style

Il existe trois manières d'appliquer un style à un document HTML :
- importer un style défini dans un fichier séparé ;
- inclure la définition du style directement dans la partie **<HEAD>** du document HTML ;
- utiliser une directive qui inclut le style directement à l'endroit où on l'utilise.

11.3.1 Importer un style défini dans un fichier séparé - *'<LINK>'*

La manière la plus simple pour travailler avec des styles, est de créer un fichier texte séparé contenant la définition complète du style. Par convention, on assigne à un tel fichier l'extension '.css".

Pour importer le style dans un document HTML, il faut inclure une balise **<LINK>** dans la partie **<HEAD>** qui a la forme suivante :

```
<LINK REL="STYLESHEET" HREF="ref" MEDIA="media" TYPE="text/css">
```

Par l'attribut **HREF**, on indique l'URL contenant la définition du style que l'on désire appliquer. L'attribut 'MEDIA' est défini par CSS2 et permet de prévoir plusieurs styles pour un même document suivant l'endroit où il est affiché. Par exemple, on peut spécifier un style pour l'écran (Valeur 'screen') et un autre lorsque le document est imprimé (Valeur 'printer'). Ceci dit, comme beaucoup de possibilités liées aux spécifications CSS, l'attribut 'MEDIA' n'est pas supporté par l'ensemble des browsers. Voici cependant la liste des media définis dans les spécifications CSS :

screen correspond aux écrans couleur d'ordinateur,

print pour l'impression sur papier, les propriétés définies dans print doivent être utilisées dans la vue avant impression du browser,

aural pour les synthétiseurs vocaux,

braille pour les lecteurs Braille électroniques,

embossed idem mais pour les appareils qui affichent les caractères en relief,

handheld pour les ordinateurs de poche (comme par exemple les Assistants Numériques Personnels (ANP) ou PDA) qui possèdent un écran relativement petit,

projection pour les présentations sur retro- et vidéo-projecteurs,

tty pour les terminaux qui ne supportent que des formats textes,

tv pour des écrans de type téléviseur dont la résolution est en général basse,

all pour tous les médias.

11.3.2 Inclure la définition - *'<style>'*

Une autre manière de spécifier un style, est d'écrire la définition de celui-ci directement dans le document HTML. Pour ce faire, on utilise la balise 'STYLE', qui a la forme suivante :

<STYLE TYPE="text/css" MEDIA="*media***">**
<!–
@import "*fichier*"
sélecteur { *propriété1* : *valeur1* ; *propriété2* : *valeur2* }
–>
</STYLE>

De plus, on voit que l'on insère directement la définition d'un élément dans cette balise 'STYLE'. On peut également importer le contenu d'un fichier texte dans le document HTML par l'intermédiaire de la fonction *'@import'*. On peut ainsi créer une multitude de fichiers ne définissant chacun que quelques éléments, et les importer selon les besoins. Cette approche plus flexible, nécessite par contre un temps de chargement plus long, même s'il est probable que dans l'avenir cela ne constitue plus un inconvénient.

Il peut sembler bizarre que la définition du style se trouve décrite comme un commentaire. La raison est qu'il existe encore des browsers qui ne comprennent pas les styles, donc ne connaissant pas la commande *'@import'* ni même les définitions d'éléments. Or, ceux-ci étant dans des commentaires, ces browsers peuvent traiter le document sans aucun problème. Par contre, les browsers qui comprennent la balise 'STYLE' savent qu'ils doivent interpréter le contenu de ce qui semble être un commentaire.

La valeur de l'attribut 'MEDIA' a la même signification que la balise 'LINK' définie dans la section précédente. Il existe encore une dernière façon de préciser qu'une règle ou une série de règles ne sont d'application que pour un type de média. Elle consiste à utiliser le sélecteur *@media*. Voici un exemple définissant deux règles applicables uniquement au media projection :

```
@media projection {
body {font-family : Verdana, "Sans-Serif";}
.slide {page-break-before : always;}
}
```

11.3.3 Utiliser directement des directives

Il est possible d'influencer directement le style d'une balise en ajoutant un attribut 'STYLE' contenant la définition du style valable uniquement pour cette balise. Le code HTML suivant est un exemple d'une telle pratique :

<H1 STYLE="font-size : *48pt* ; **font-family** : *Arial* ; **color** : *green*">Essai**</H1>**

Généralement, on utilise cette formule, lorsque l'on désire appliquer à un certain bloc de texte, un style qui diffère de celui adopté pour le reste du document. En effet, il vaut mieux utiliser de nombreux éléments dont le comportement ne varie jamais, plutôt que d'utiliser un petit nombre d'éléments, et de les changer régulièrement par l'intermédaire des directives.

Listing 11.1 – Utilisation directe des directives

```
<HTML>
  <HEAD>
    <TITLE>Hamlet, excerpt from act II</TITLE>II
  </HEAD>
  <BODY STYLE="color: black; background: white">
    <P STYLE="font-weight: bold">Polonius: Do you know me, my lord?
    <P STYLE="font-weight: normal">Hamlet: Excellent well, you are a fishmonger.
    <P STYLE="font-weight: bold">Polonius: Not I, my lord.
    <P STYLE="font-weight: normal">Hamlet: Then I would you were so honest a man.
  </BODY>
</HTML>
```

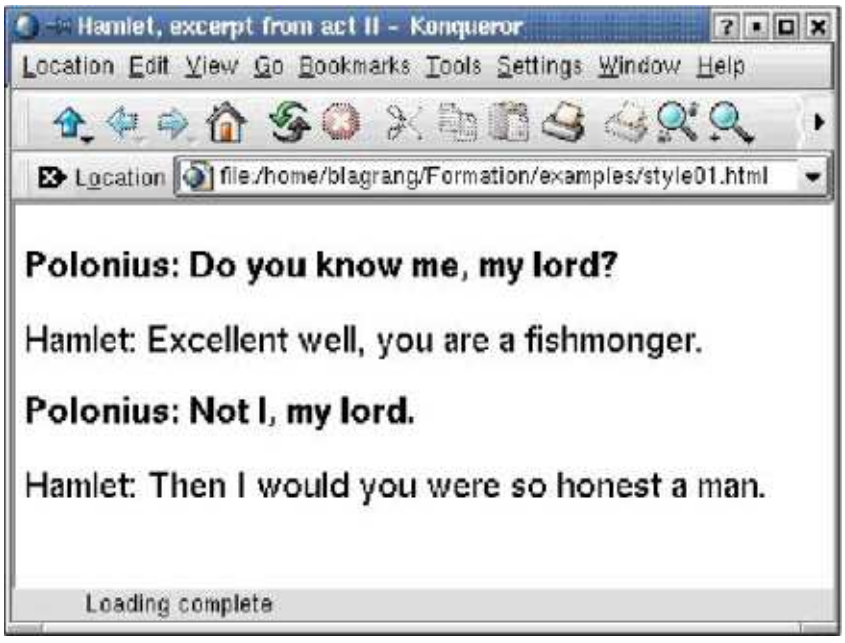


FIG. 11.2 – Utilisation directe des directives

11.3.4 Les styles et la cascade

D'où vient le nom 'Cascading Style Sheet' donné à ce système de définition de styles. En fait, il provient du système de cascades auquel il obéit. En effet, pour un document, une série de feuilles de style sont définies. Tout d'abord, les styles définis par le browser lui-même et qui constitue donc les valeurs par défaut de toutes les propriétés. Ensuite, éventuellement les feuilles de style définies par l'utilisateur lui-même ; en effet, CSS prévoit cette possibilité pour l'utilisateur du browser de définir ses feuilles de style favorites pour les documents qu'il visionne sur son browser. Enfin, les feuilles de style écrites par le concepteur de la page HTML. Il semble évident que les styles définis par le concepteur primeront sur les styles définis par l'utilisateur et sur ceux du browser. De même, les feuilles de style de l'utilisateur prendront le pas sur les styles de l'arpenteur en cas de conflit. Ce mécanisme de cascade permet de résoudre les conflits entre les règles définies par le browser, l'utilisateur et le concepteur. Cependant, les spécifications CSS donnent la possibilité à l'utilisateur de donner la primeur à ses règles en rajoutant le mot clé ' *!important*' à la fin de la déclaration. Par exemple, si l'utilisateur veut absolument avoir son texte en noir tout en n'accordant pas trop d'importance à la police, il écrira :

body {font-family : Serif ; color : black !important ;}

11.3.5 Les styles et l'héritage

Tout document HTML possède une structure arborée et les éléments de l'arbre (du document) héritent de règles tout comme les feuilles d'un arbre généalogique héritent de gênes.

Par exemple, imaginons que nous ayons le document HTML défini par le listing 11.2.

Listing 11.2 – Exemple de styles

```
<HTML>
2 <HEAD>
<TITLE>
4 Using Style Example
</TITLE>
6 <STYLE>
<!--
8   body {color : blue}
   p { color : red }
10  ul { color : green }
-->
12 </STYLE>

14 </HEAD>

16 <BODY>
<H1>
18 Big example
</H1>
20 <H2>
Title 1
22 </H2>
This line is tagged by &lt;BODY&gt; only, so it will appear in blue
24 <p>
This is an example of inheritance in the CSS specifications and it is in red.
26 </p>
<UL>
28   <li>This is the first item of the list in green
   <li>This the second, in green.
30 </UL>
<p style="color : black">
32 Here the color has been modified and it is in black.
</p>
34 </HTML>
```

Ce document possèdera la structure décrite dans la figure 11.4.

Comme on peut le voir, on définit le style pour la balise 'P' à la ligne 5, puis on le redéfinit à la ligne 12. Comme le lecteur s'y attend, les lignes 1 et 3 sont affichées en bleu et la ligne 2 en rouge. En effet, lorsque le style d'un élément est redéfini plusieurs fois, les propriétés différentes s'ajoutent. Lorsqu'une propriété est modifiée pour un bloc, comme dans l'exemple, la modification n'est valable que pour ce bloc,

FIG. 11.3 – E xemple de style : héritage.

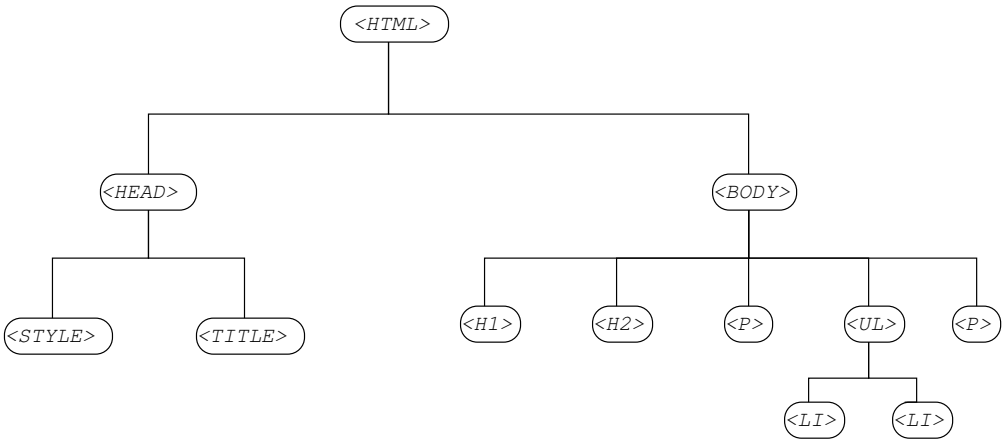


FIG. 11.4 – Structure hiérarchique d'un document HTML

et reprendra sa valeur définie avant à la sortie du bloc. Enfin, lorsque plusieurs propriétés identiques sont redéfinies au sein d'un même bloc, on applique la dernière règle définie dans le document HTML.

Certaines propriétés n'héritent pas du parent. Un exemple est la propriété ***background*** **.** En effet, la propriété *background* définit l'arrière-plan du document dans l'élément body mais, le background de tous les autres éléments sera vide (ou transparent) sauf spécification explicite contraire.

Chapitre 12

La mise en page

Le but de cet ouvrage n'est pas de vous former au graphisme. Cependant, la mise en page d'un document manipule certains éléments typographiques qu'il faut connaître. Nous commencerons donc ce chapitre par une petite introduction à quelques notions.

12.1 Boîtes et blancs

Chaque élément d'un document (à commencer par le document lui-même) est constitué d'une boîte et d'un bloc "contenu" inscrit dans la boîte. Les dimensions de la boîte, la position du contenu, la position de la boîte déterminent grandement l'aspect du document. Voyons quels paramètres définissent un élément. Le lecteur se rapportera à la figure 12.1.

Marges : tout élément possède 4 marges (haute, basse, droite et gauche). Ces marges définissent l'espace vide entre les bords du bloc "contenu" du parent et la boîte de l'élément. Ainsi, la marge du document est définie par rapport au support physique du document (fenêtre du browser, feuille de papier,...). Il faut noter qu'il est possible de donner une marge négative à un élément. Par exemple, si la marge gauche du document vaut 2cm et que l'on désire faire commencer un titre avancé de 1cm dans la marge du document, il suffira de lui assigner une marge de -1cm. En CSS, la propriété définissant la marge d'un élément est **margin**.

Marge_intérieure : ou blanc (*padding* en Anglais) définit l'espace libre entre le bloc "contenu" d'un élément et la boîte de l'élément. Comme la marge, il existe 4 marges intérieures. La propriété CSS définissant la marge intérieure est **padding**.

Bord : définit le bord de la boîte de l'élément. Ce bord peut posséder un certain style, une certaine épaisseur qu'il faut définir. Il peut être visible ou invisible. La propriété CSS définissant le bord est **border**.

Donc si l'on veut connaître la largeur du bloc "contenu" de l'élément, il suffit de retirer la marge gauche, la marge droite, la marge intérieure gauche, la marge intérieure droite et deux fois l'épaisseur du bord de la boîte à la largeur du bloc "contenu" du parent.

12.2 Unités de longueur

Une autre notion qu'il est nécessaire d'aborder sont les mesures dans la mise en page. Il y a trois moyens d'indiquer une longueur dans un CSS :

1. **unités absolues** : dans cette catégorie, on retrouve le millimètre (**mm**), le centimètre (**cm**), le pouce (**in**[1]), le point (**pt**[2]), le pica (**pc**[3]). Toutes ces unités décrivent une distance physique qui peut être mesurée directement sur le support et qui reste figée.

[1] 1pc = 12pt
[2] 72pt = 1in = 25.4 mm
[3] 1 in = 25.4 mm, cette unité est très courue chez les anglo-saxons

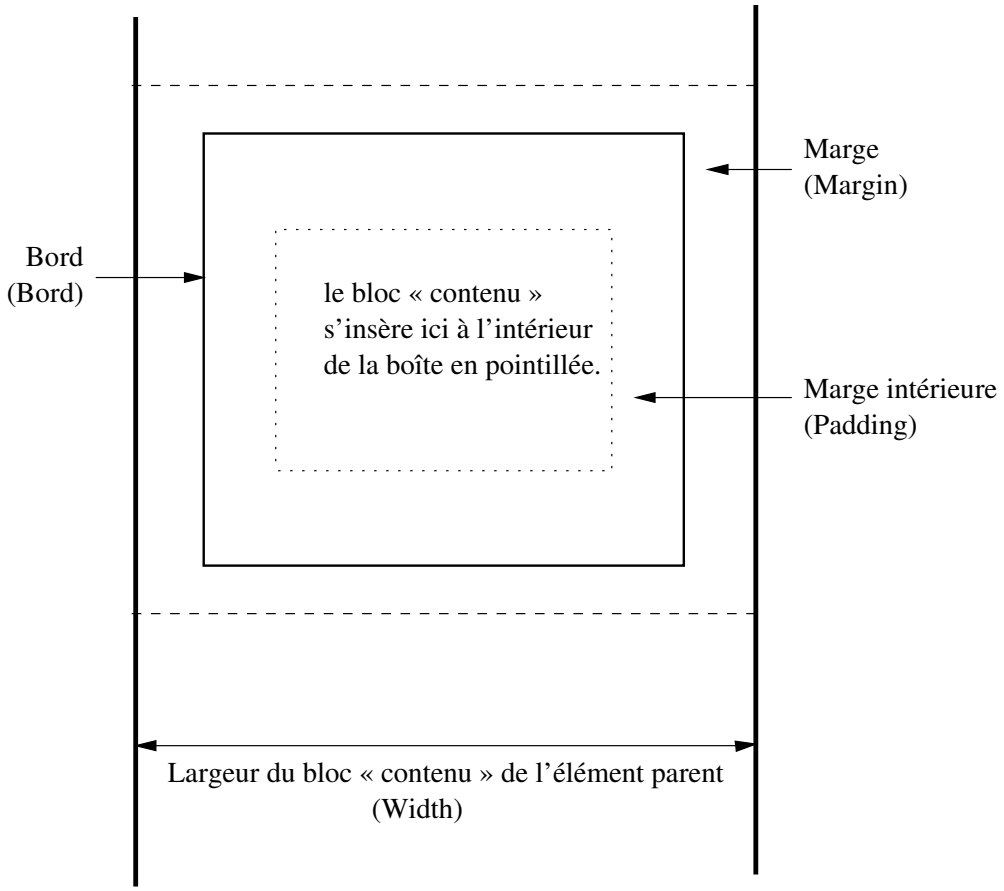


FIG. 12.1 – Paramètres typographiques d'un élément.

2. **unités relatives** : ici, la mesure est donnée relativement à une grandeur caractéristique de la fonte utilisée comme par exemple la hauteur d'un caractère (**ex** : hauteur du caractère X de la fonte. Attention, cette grandeur change en fonction du type de fonte utilisée : le X du Times Roman est plus grand que le X du Baskerville, ceci pour des tailles de fonte identiques) ou encore la taille de la fonte (**em**, à l'origine cette unité correspondait à la largeur de la lettre M. Cette fois, pour deux polices différentes mais de même taille, em définira la même longueur). Un grand avantage de ces unités relatives réside dans le fait que les longueurs se mettent à l'échelle automatiquement, les proportions restent gardées si l'on change la taille de la police. Un avantage de em sur ex est que les proportions sont gardées si l'on change la police (ce qui n'est pas le cas pour ex). Attention, il existe un cas où em est définie par rapport à la taille de la police du parent de l'élément où on l'utilise : c'est le cas de la définition de la taille de la police : **font-size**. Un autre moyen de donner une longueur est de la donner en pourcentage (**%**) par rapport à une mesure du parent.
3. **pixel** : (**px**)cette unité dépend du matériel utilisé pour la visualisation. CSS restreint cependant les variations en spécifiant que lors de l'impression, les dimensions doivent être identiques à celles de l'écran.

Dans la majorité des cas, nous recommenderons l'utilisation de em comme unité de mesure des longueurs.

Chapitre 13

Les propriétés CSS

Nous allons passer ne revue quelques propriétés CSS qui peuvent être facilement utilisées pour améliorer le layout de vos pages HTML.

13.1 Les propriétés structurelles *'display'*

En plus de donner des informations sur la manière visuelle de traiter du texte d'un style particulier, il faut pouvoir également expliciter la manière dont l'élément doit être traité dans la structure du document HTML. Pour cela, on utilise la propriété 'display'. Les valeurs reconnues par la norme CSS1 sont :

block L'entité doit être considérée comme un bloc, c'est-à-dire un paragraphe (Comme par exemple les balises 'P' et 'H1' dans le langage HTML). Par exemple nous avons le code du listing **??**, nous obtenons la figure 13.1

Listing 13.1 – Utilisation de la propriété display : block

```
<HTML>
  <HEAD>
    <TITLE> Example for propriety display:block </TITLE>
    <STYLE>
      H1,P{display : block}
    </STYLE>
  </HEAD>
  <BODY>
    <H1> Title1 </H1>
    <P> First paragraph. </P>
    <P> Second paragraph. </P>.
  </BODY>
</HTML>
```

inline L'entité doit être considérée comme du texte simple, ce qui veut dire qu'il n'y aura pas de retour à la ligne (Comme par exemple les balises 'I' et'B' dans le langage HTML). Le code du listing 13.2 nous donne la figure 13.2.

none. L'entité ne doit pas être affichée (Comme par exemple les balises 'META' dans le langage HTML).

list-item. L'entité fait partie d'une liste (Comme par exemple la balise 'LI' dans le langage HTML). Un exemple d'utilisation de cette propriété sera donné dans le chapitre sur le XML.

run-in. L'entité doit être considérée comme une entête à un paragraphe. Cette entête est précédée d'un saut de ligne mais appartient à la même ligne que l'entité suivante.

compact. L'entité est placée dans la marge de l'élément suivant si la marge est suffisament grande. Cela permat de créer des glossaires, de rajouter des petits titres dans un document.

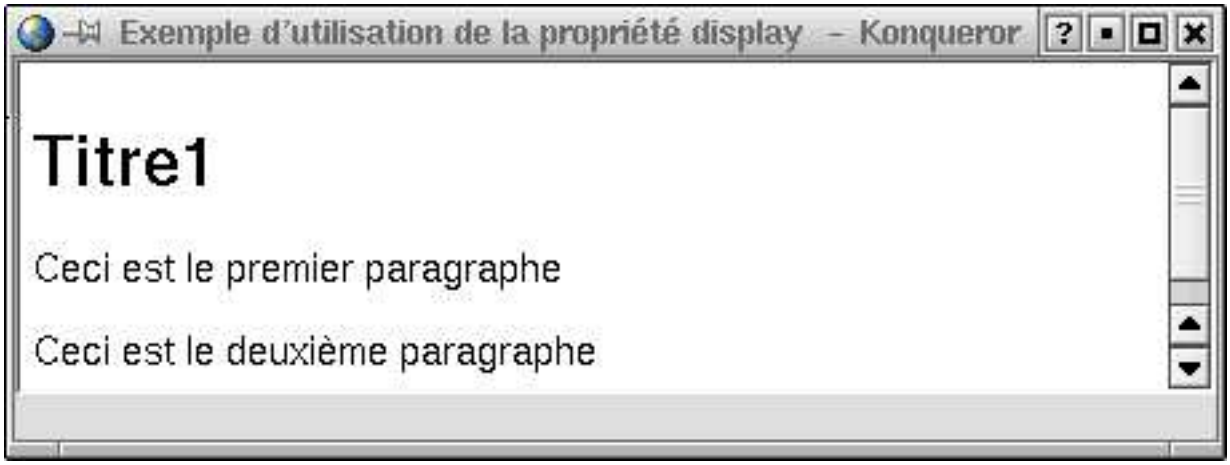


FIG. 13.1 – Utilisation de la propriété **display : block**

Listing 13.2 – Utilisation de la propriété display : inline

```
<HTML>
  <HEAD>
    <TITLE> Example for propriety display:inline </TITLE>
    <STYLE>
      H1,P{display : inline}
    </STYLE>
  </HEAD>
  <BODY>
    <H1> Title1 </H1>
    <P> First paragraph. </P>
    <P> Second paragraph. </P>
  </BODY>
</HTML>
```

La norme CSS2 introduit de nouvelles valeurs possibles, notamment pour pouvoir utiliser des entités dans des tableaux (**marker, table, inline, inline-table, table-row-group, table-header-group, table-footer-group, table-row, table-column-group, table-column, table-cell, table-caption**).

Actuellement, seules les valeurs '**block**', '**inline**' et '**none**' sont supportées par les principaux browsers. **run-in, compac**t et **list-item** sont supportés par Netscape 6.

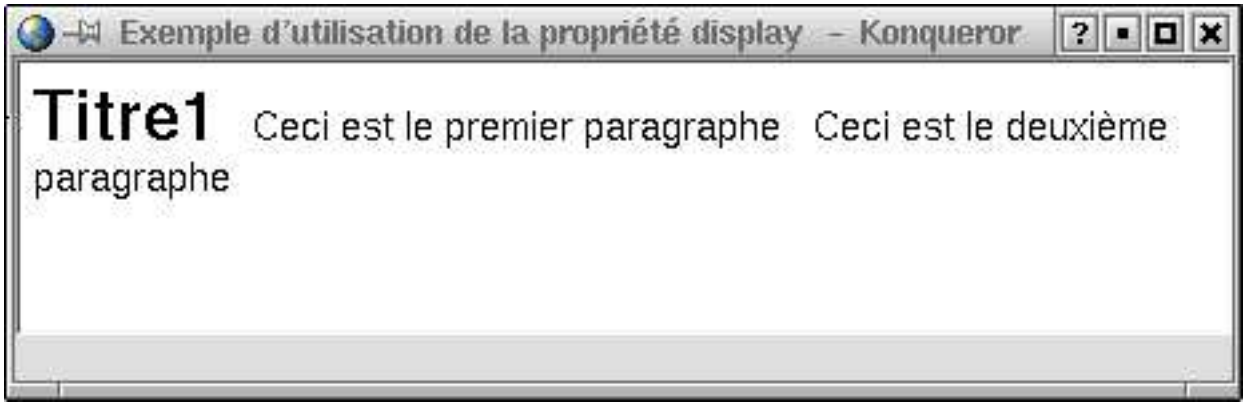


FIG. 13.2 – Utilisation de la propriété **display : Inline**

Cette propriété (ainsi que d'autres comme 'position', 'visibility',...) conjuguée avec le javascript constitue un premier élément de ce que l'on appelle le Dynamic HTML (DHTML).

13.2 Les propriétés de blancs et de boîtes

On définira dans cette section les propriétés liées aux boîtes et aux blancs (cf.12.1).

13.2.1 Propriété de marge en haut, marge en bas, marge à droite et marge à gauche - *'margin-top, margin-bottom, margin-right, margin-left'*

Ces propriétés s'utilisent de manière identique et permettent de définir les marges en haut, en bas, à droite et à gauche entre la boîte de l'élément et le bloc "contenu" du parent. Elles prennent des valeurs de longueur définie avec les unités vues ci-avant. Par exemple, si on veut définir une marge de 1cm en haut de l'élément BODY, on écrira :

body {margin-top : 1cm ;}

13.2.2 Regroupement des déclarations - *'margin'*

Cette propriété permet de regrouper les déclarations liées aux marges dans une seule déclaration. On l'utilise avec 1 seule valeur (fixe les 4 marges à la même valeur) ou avec 4 valeurs (une pour chaque marge, dans l'orde à gauche, à droite, en haut et en bas)

13.2.3 Propriétés de bordure - *'border, border-left, border-right, border-top, border-bottom'*

Ces propriétés permettent de définir la largeur (**border-width** par exemple), le style (**border-left-style**, par exemple) et la couleur (**border-top-color**, par exemple) de la boîte de l'élément, en une fois ou en séparant les côtés (**left, right, top, bottom**). Pour width, on précise l'épaisseur du trait en indiquant une longueur ; pour color, on indique la couleur à utiliser (pour la manière de définir la couleur, voir 13.5.). La sous-propriété style peut prendre les valeurs suivantes :

none il n'y a pas de trait, c'est la valeur par défaut, les bords ne sont pas affichés quelle que soit l'épaisseur qui a été définie,

hidden les bords occupent leur place mais sont invisibles,

dotted une ligne pointillée (voir figure 13.3),

FIG. 13.3 – Cadre pointillé

dashed une ligne formée de traits (voir figure 13.4),

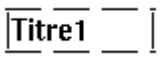


FIG. 13.4 – Cadre long pointillé

solid une ligne pleine (voir figure 13.5),

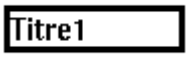


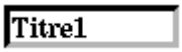
FIG. 13.5 – Cadre avec lignes pleines

double une ligne dédoublée (voir ci-dessous),

groove le bord ressemble à une tranchée creusée dans le document (voir ci-dessous),

ridge le bord possède une épaisseur qui ”sort” de l'écran (voir ci-dessous),

inset le contenu de la boîte semble être dans un renfoncement (voir ci-dessous),



outset le contenu de la boîte semble être sur un renfort (voir ci-dessous).

13.2.4 Propriétés de marges intérieures - *'padding, padding-left, padding-left, padding-top, padding-bottom'*

L'utilisation de ces propriétés est identique à celle des propriétés de marges. elles permettent de définir les marges intérieures.

13.3 Propriétés des polices - *'font'*

Les principales propriétés concernant les polices de caractères sont expliquées ici. Elles correspondent à de nombreux attributs définis dans la balise '<FONT>'.

13.3.1 La famille de la police - *'font-family'*

Avec la propriété 'font-family', il est possible de définir la famille de polices. On peut lui assigner comme valeur soit une police spéciale, comme la police Arial, soit une police générique, comme la police sans serif. Il est également possible d'assigner plusieurs polices différentes pour un même élément. Dans ce cas, l'arpenteur prendra la première qu'il possède. Par exemple si on a la règle suivante :

body {font-family : "Times New Roman", Garamond, Serif"}

L'arpenteur utilisera Times New Roman comme police s'il la possède, sinon il prendra Garamond et enfin, s'il n' a pas Garamond non plus, il choisira une police avec empattement.

Les différentes polices génériques sont les suivantes :

Serif : police avec empattement comme le TIMES NEW ROMAN, GARAMOND, TIMES ROMAN,

Sans-Serif : police sans empattement comme HELVETICA, ARIAL,

Monospace : police où chaque caractère occupe le même espace comme le COURRIER,

Cursive : comme ZAPF-CHANCERY,

Fantasy : comme la police WESTERN.

13.3.2 Le poids de la police - *'font-weight'*

Avec la propriété 'font-weight', on peut définir le poids de la police de caractères utilisée en utilisant une échelle allant de 100 (poids le plus faible) à 900 (poids le plus fort) par incrément de 100. En plus, il est possible d'utiliser les mots clés suivants dont le résultat dépend du type de la police :

normal Poids normal. Correspond à un poids de 400.

bold. Mettre le texte en gras.Correspond à une poids de 700

bolder. Mettre le texte en plus gras relativement au parent.

lighter Mettre le texte en plus léger relativement au parent.

13.3.3 La taille de la police - *'font-size'*

Avec la propriété 'font-size', on spécifie la taille de la police. La valeur peut être donnée directement en point, pica, em, ex, mm, cm, in, px soit en utilisant un des mots-clés prédéfinis ou encore un pourcentage par rapport à la taille du parent (cf. em). Les mots clés prédéfinis sont 'xx-small', 'x-small', 'small', 'medium' qui représente la valeur par défaut, 'large', x-large' et 'xx-large'. Ces mots-clés correspondent à une mesure absolue de la taille. On passe d'un niveau à un autre en multipliant la taille par 1,5.

Par exemple, si la taille par défaut est de 12 pt, la taille 'large' sera de 18 pt (12*1,5=18) et la taille 'small' sera de 8 pt (12/1,5=8).

13.3.4 Le style de la police - *'font-style'*

Avec la propriété 'font-style', on peut spécifier si la police est normale (Valeur 'normal'), en italique (Valeur 'italic') ou en oblique (Valeur 'oblique').

Il est inutile d'essayer de surcharger certains éléments HTML, comme par exemple la balise 'B', avec cette propriété, car cela n'entraineraît que des confusions.

13.3.5 La variante de la police - *'font-variant'*

Avec la propriété 'font-variant', on peut spécifier si la police doit s'écrire en petites capitales (Valeur 'small-caps') ou de manière normale (Valeur 'normal').

13.3.6 Regroupement des déclarations dans la propriété *'font'*

Il est possible de grouper toutes ces propriétés directement dans la propriété font de la manière suivante :
P {font : italic bold 2em Garamond , Serif}
correspond à la règle suivante :
P {font-family : Garamond , Serif ; font-style : italic ; font-weight : bold ; font-size : 2em}

Remarque : il existe d'autres propriétés associées à font et définies par les spécifications du CSS mais à l'heure actuelle, elles ne sont supportées par aucun browser sauf peut-être Netscape 6 (mais vu sa lenteur, nous déconseillons son utilisation pour le moment).

13.4 La couleur d'affichage - *'color'*

Avec la propriété 'color', on peut définir la couleur d'affichage du texte. Par exemple, la définition suivante définit un élément devant être considéré comme un paragraphe et dont la couleur d'affichage sera le rouge :
TITRE { color :red ; display :block}

13.5 Le fond d'écran - *'background'*

Il existe une série de propriétés permettant de définir les images utilisées comme arrière-plan ainsi que la couleur de fond pour le texte associé aux éléments contenant cette propriété dans un document HTML.

La propriété background est une des rares propriétés dont la valeur n'est pas héritée par les enfants. Par défaut le background des descendants est transparent. Ceci pour éviter d'avoir des images qui se superposent intempestivement.

De manière générale, il existe trois manières de définir une couleur :
- On utilise directement le nom des couleurs, telles qu'elles sont définies dans la norme HTML.
- On peut utiliser une notation hexadécimale de la forme *#RRGGBB* permettant d'associer avec chaque couleur fondamentale (Rouge (R), verte (G) et bleu (B)), un chiffre hexadécimal. Ceci dit, il s'agit d'une forme peu conseillée, car difficilement lisible.
- On utilise une valeur de la forme *rgb(R,G,B)* où on associe à chaque couleur fondamentale (Rouge (R), verte (G) et bleu (B)) un nombre compris entre 0 et 255.

13.5.1 La couleur du fond d'écran - *'background-color'*

Avec la propriété 'background-color', on spécifie la couleur de fond d'un élément. Elle correspond à l'attribut 'BGCOLOR' de la balise 'BODY'.

13.5.2 Une image en fond d'écran - *'background-image'*

Avec la propriété 'background-image', il est possible de spécifier une image qui sera affichée en fond d'écran du document. Elle correspond à l'attribut 'BACKGROUND' de la balise 'BODY'. La valeur de cette propriété est de la forme *url(nom)*, où *nom* correspond au nom de l'URL qui contient l'image. Un exemple d'utilisationde cette règle est donnée ci-dessous :

```
BODY {background-image : url("../pics/bcgd1.gif");}
```

13.5.3 La répétition d'une image en fond d'écran - *'background-repeat'*

Avec la propriété 'background-repeat', on peut déterminer, lorsqu'une image utilisée est plus petite que la surface visible à l'écran, la manière de gérer la situation. Les valeurs possibles sont :

repeat L'image est répétée horizontalement et verticalement. Il s'agit de la valeur par défaut. (voir figure 13.6)

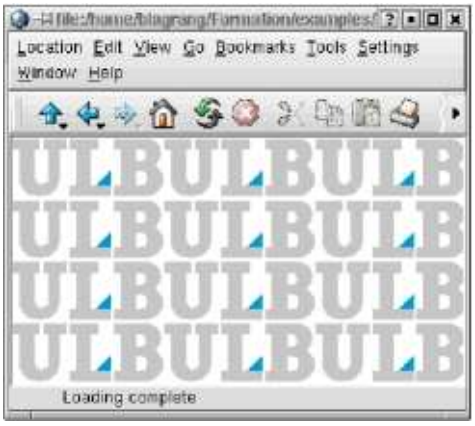


FIG. 13.6 – Répétition de l'image de background par la propriété **background-image : repeat**

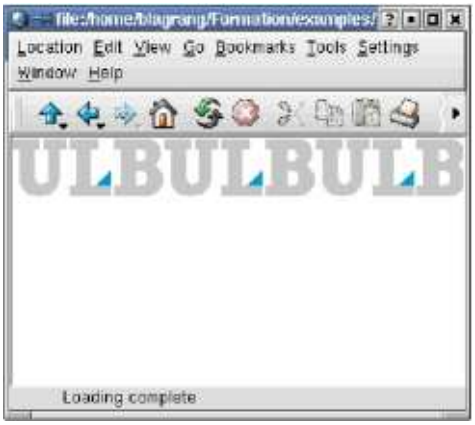


FIG. 13.7 – Répétition de l'image de background par la propriété **background-image : repeat-x**

repeat-x L'image est répétée horizontalement seulement. (voir figure 13.7)

repeat-y L'image est répétée verticalement seulement. (voir figure 13.8)

no-repeat L'image n'est pas répétée. (voir figure 13.9)

13.5.4 Fixation de l'image de fond - *'backgroung-attachment'*

Cette propriété permet de définir si l'image de fond est liée au document et donc bouge avec le document lorsque l'on déroule (*scroll*)ce-dernier ou si elle est fixée à la fenêtre du browser. Dans ce dernier cas, l'image restera immobile par rapport au browser lorsque l'on déroule le document. Les valeurs possibles sont :

scroll valeur par défaut : l'image est attachée au document et bouge avec lui,

fixed l'image est fixée dans le browser.

13.5.5 Position de l'image de fond - *'background-position'*

Cette propriété définit la position de l'image de fond (ce n'est pas valable pour une couleur de fond qui par définition occupe l'entièreté de la boîte de l'élément pour laquelle elle est spécifiée) dans la boîte occupée par l'élément. il existe trois manières de préciser cette position :

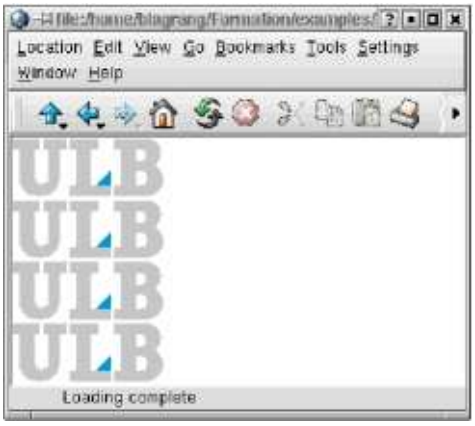


FIG. 13.8 – Répétition de l'image de background par la propriété **background-image : repeat-y**

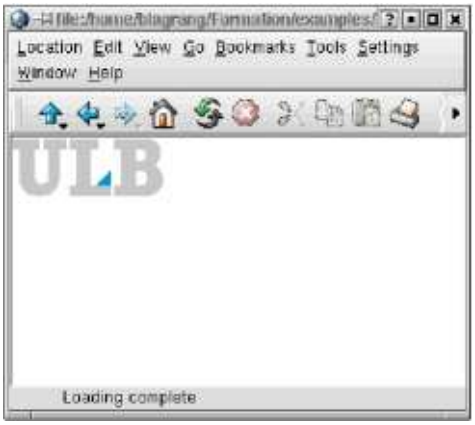


FIG. 13.9 – Répétition de l'image de background par la propriété **background-image : no-repeat**

- distance absolue du coin en haut à gauche de l'image dans la boîte,
- pourcentage relativement à la taille horizontale et/ou verticale de la boîte,
- mots-clés décrivant la position horizontale et verticale de l'image dans la boîte. Pour la position horizontale, on utilise les mots-clés *left* (alignée à gauche), *center* (centrée dans la boîte) et *right* (alignée à gauche). Les mots-clés que l'on utilise pour le positionnemenr vertical sont *top* (l'image est alignée en haut de la boîte), *center* (l'image est centrée) et *bottom* (alignée en bas).

13.5.6 Regroupement des déclarations dans la propriété *'background'*

Comme pour les propriétés liées à la police, il est possible de regrouper les déclarations liées au fond en une seule déclaration grâce à la propriété background. Un exemple est donnée ci-dessous :

BODY {background : url(backimg.jpg) top center fixed}

13.6 Les propriétés pour le texte

Il existe également un grand nombre de propriétés permettant de manipuler du texte, dont quelques unes sont décrites dans cette section. Il est important de signaler que tous les browsers ne supportent pas encore l'entièreté des propriétés.

13.6.1 L'espacement entre mots - *'word-spacing'*

Avec la propriété '*word-spacing*', on peut spécifier l'espacement entre les mots. La valeur par défaut est '*normal*', mais l'utilisateur peut spécifier une distance en utilisant l'une des unités proposées.

13.6.2 L'espacement entre lettres - *'letter-spacing'*

Avec la propriété '*letter-spacing*', on peut spécifier l'espacement entre les lettres. La valeur par défaut est '*normal*', mais l'utilisateur peut spécifier une distance en utilisant l'une des unités proposées.

13.6.3 L'espacement vertical - *'vertical-align'*

La propriété '*vertical-align*' permet de contrôler la position verticale de textes et d'images en respectant la ligne de base en vigueur. On peut utiliser certaines valeurs prédéfinies ou bien encore des pourcentages.

13.6.4 L'alignement du texte - *'text-align'*

Avec la propriété '*text-align*', on peut spécifier l'alignement du texte. Les différentes valeurs sont :

left aligne le texte à gauche,

right aligne le texte à droite,

center aligne le texte au centre,

justify justification du texte.

13.6.5 L'indentation du texte - *'text-indent'*

Avec la propriété *'text-indent'*, il est possible de spécifier une indentation pour la première ligne d'un bloc de texte. On peut utiliser des valeurs en choisissant les unités, ou en spécifiant le pourcentage de la largeur du bloc de texte auquel le style sera appliqué. On peut également utiliser des valeurs négatives.

13.6.6 L'interligne - *'line-height'*

La propriété '*line-height*' permet de spécifier l'interligne. On peut utiliser les unités, un pourcentage par rapport à la taille de la police de caractères utilisée, ou bien donner un facteur multiplicatif par rapport à cette même taille.

13.6.7 La décoration du texte - *'text-decoration'*

La propriété '*text-decoration*' permet de définir quelques effets sur le texte. Cette propriété fût introduite par Internet Explorer puis introduite dans les spécifications CSS. Elle n'est pas entièrement supportée par tous les browsers. Les différentes valeurs possibles sont définies ci-dessous. La propriété sera mise à **none** ou à toute combinaison des autres valeurs.

blink affiche le texte en clignotant,

line-through barre le texte (voir ci-dessous),

~~texte barré~~

overline dessine une ligne au-dessus du texte (voir ci-dessous),

underline dessine une ligne en-dessous du texte (voir ci-dessous),

texte sousligné

none ne fait rien.

13.6.8 Transformation du texte - *'text-transform'*

Avec la propriété '*text-transform*', on peut forcer certaines transformations du texte. Les valeurs possibles sont :

capitalize Le premier caractère de chaque mot est mis en majuscule.

uppercase Tout le texte est mis en majuscule.

lowercase Tout le texte est mis en minuscule.

none Ne fait rien et annule une valeur héritée d'un parent.

13.6.9 Ombrage du texte - *'text-shadow'*

Cette propriété ajoute une ombre au texte pour lequel la propriété est définie. Une ombre est définie par un décalage (de l'ombre par rapport au texte.Deux longueurs définissent le décalage vers la droite et vers le bas. Les valeurs négatives sont acceptées et définissent dès lors un décalage vers la gauche et vers le haut) et éventuellement une couleur et un facteur de flou (qui rend l'ombre moins nette). Par exemple, le code ci-dessous définit un ombre décalée de 0.3 em vers la droite et vers le bas, un flou de 0.1em et de couleur grise :

```
H1 {text-shadow : 0.3em 0.3em 0.1em #333}
```

Malheureusement, aucun browser ne supporte cette propriété aujourd'hui.

13.7 Propriétés de listes - *'list-style'*

Nous avons vu comment préciser qu'un élément est de type liste par la propriété display. Ici, nous verrons différentes propriétés associées aux listes. Ces propriétés sont évidemment aussi valable pour les éléments HTML qui sont par définition des éléments de type liste (comme par exemple **LI**)

13.7.1 Style du symbole de la liste - *'list-style-type'*

Il est possible de préciser le style de la numérotation d'une liste énumérée et le type de symbole d'une liste non-énumérée par cette propriété. Il est ainsi possible de définir les symboles suivants pour les listes non-énumérées :

disc c'est un disque noir plein et c'est la valeur par défaut,

circle c'est un cercel noir vide,

square c'est un carré noir vide.

Pour les listes énumérées, il est possible de définir des numératations très diverses comme :

decimal c'est à dire : 1, 2, 3, 4, ...

decimal-leading-zero 01, 02, 03, 04, ...

lower-roman i, ii, iii, iv, ...

upper-roman I, II, III, IV, ...

lower-alpha/lower-latin a, b, c, d, ...

upper-alpha/upper-latin A, B, C, D, ...

hiragana/katakana/hiragana-iroha/katakana/iroha ce sont quatre styles de numérotation japonais différents

cjk-ideographic numérotation chinoise

lower-greek numérotation avec les lettres grecques

armenian/georgian numérotation avec l'alphabet Armeniens et Géorgiens.

Enfin, le dernier style existant est **none** qui supprime tout symbole devant l'élément de la liste sans toutefois interrompre la numérotation.

13.7.2 Image pour le symbole de la liste - *'list-style-image'*

Nous venons de voir que pour les liste non-énémérées, le nombre de symbole à disposition est assez limité (3 symboles). Cependant, il est possible de remédier à cela par cette propriété. En effet celle-ci permet de donner une url au browser spécifiant l'image à utiliser comme symbole de la liste.

Elle s'utilise de la manière suivante :

```
UL {list-style-image : url(«img/pic01.gif»);}
```

Cette propriété a la priorité sur **list-style-type** si elle est présente.

13.7.3 Position du symbole de la liste - *'list-style-position'*

Le symbole peut soit être contenu dans le bloc-contenu de l'élément par la valeur **inside** et dans ce cas, le symbole fait partie intégrante du contenu de la balise soit en dehors par la valeur **ouside.**

Par exemple, si l'on applique une bordure à l'élément HTML, dans le cas de la valeur **inside**, le symbole sera à l'intérieur de la bordure et dans le cas de la valeur **outside**, le symbole sera à l'extérieur dans la marge.

13.8 Propriétés liées au media 'print'

Dans la section 11.3 nous avons vu qu'il est possible de définir des styles différents en fonction du type de media pour lequel le document est destiné : écran (screen), papier (print), baffles (aural), ... Dans cette partie, nous verrons quelques propriétés liées au media papier. Nous n'aborderons que celles qui sont partiellement supportées par les browsers (les autres leur étant inconnues, le lecteur se réferrera aux références pour plus d'informations).

13.8.1 Saut de page avant un élément - *'page-break-before'*

Par cette propriété, on définit si on doit insérer un saut de page ou non avant l'élément. Par exemple dans un livre, on désire souvent commencer un nouveau chapitre sur une nouvelle page voire même sur une page de droite. Les valeurs prises sont les suivantes :

auto valeur par défaut, le browser insère un saut de page s'il n'y a plus de place sur la page

always on insère toujours un saut de page avant l'élément

avoid le browser doit éviter d'insérer un saut de page avant l'élément (on remarquera le côté prudent de la spécification CSS qui laisse une "chance au browser" de faire de son mieux)

left le browser doit faire en sorte que l'élément commence sur une page de gauche

right idem mais l'élément doit se trouver sur une page de gauche

inherit la valeur de la propriété est héritée du parent

Par exemple si on veut qu'un chapitre commence toujours sur une page de droite, on écrira :

.chapter {page-break-before : right}

13.8.2 Saut de page après un élément - *'page-break-after'*

Cette propriété définit si on doit insérer un saut de page ou non après l'élément. Les valeurs possibles sont identiques à celles de *page-break-before*. Par exemple si on veut éviter qu'un titre de premier niveau ne se situe pas tout seul en bas de page, on écrira :

H1 {page-break-after : avoid}

13.8.3 Le sélecteur *'@page'*

Ce sélecteur permet de définir les propriétés du support papier comme les marges (*margin*, cf. 13.2), l'orientation (*size* qui peut prendre entre autre les valeurs "**portrait**" et "**landscape**"), la taille (*size* qui peut prendre également des valeurs de longueur pour définir la hauteur et la largeur du papier).

13.9 Propriétés liées aux sélecteurs *:before, :after*

Dans ce chapitre, nous verrons quels sont les outils mis à disposition par la norme CSS2 pour générer des entêtes automatiquement, du texte répétitif, etc.

13.9.1 La propriété *'content'*

Cette propriété est utilisée avec les pseudo-éléments **:after** et **:before.** Elle permet de définir le contenu de ce pseudo-élément, puisqu'il est impossible avec le langage HTML d'avoir accès à cet élément. On trouvera un exemple d'utilisation de cette propriété avec le listing et la figure Voici un exemple d'utilisation de :before. Si dans un document, on a un grand nombre de remarques a effectuer, toutes ces remarques commençant par le mot "Remarque : ", il suffit d'utiliser le listing 13.3.

Listing 13.3 – Utilisation du sélecteur :before

```
<html>
  <head>
    <title>
    Example of the :before selector
    </title>
    <style>
    <!--
    .remarque:before   {content : "Remarque"; font-weight : bold;}
    -->
    </style>
  </head>
  <body>
  <H1>
  Chapitre 1
  </H1>
  <P>
  Ceci est le premier paragraphe du chapitre qui parle de blalblabla
  blalblabla blalblabla blalblabla blalblabla blalblabla blalblabla
  blalblablablalblablablalblabla blalblabla blalblabla blalblabla blalblabla
  blalblabla blalblabla blalblabla blalblabla blalblabla blalblabla blalblabla
  blalblabla blalblabla blalblabla blalblabla blalblabla blalblabla blalblabla
   ... blalblabla blalblabla. <BR>
   Ceci termine notre chapitre sur blalblabla.
  </P>
  <P class="remarque">
  ce qui éèprcde est sans éêintrt, vraiment!
  </P>
  </body>
</html>
```

Le résultat peut être observé dans Netscape 6 (le seul pour l'instant à supporter ce sélecteur) voir figure 13.10.

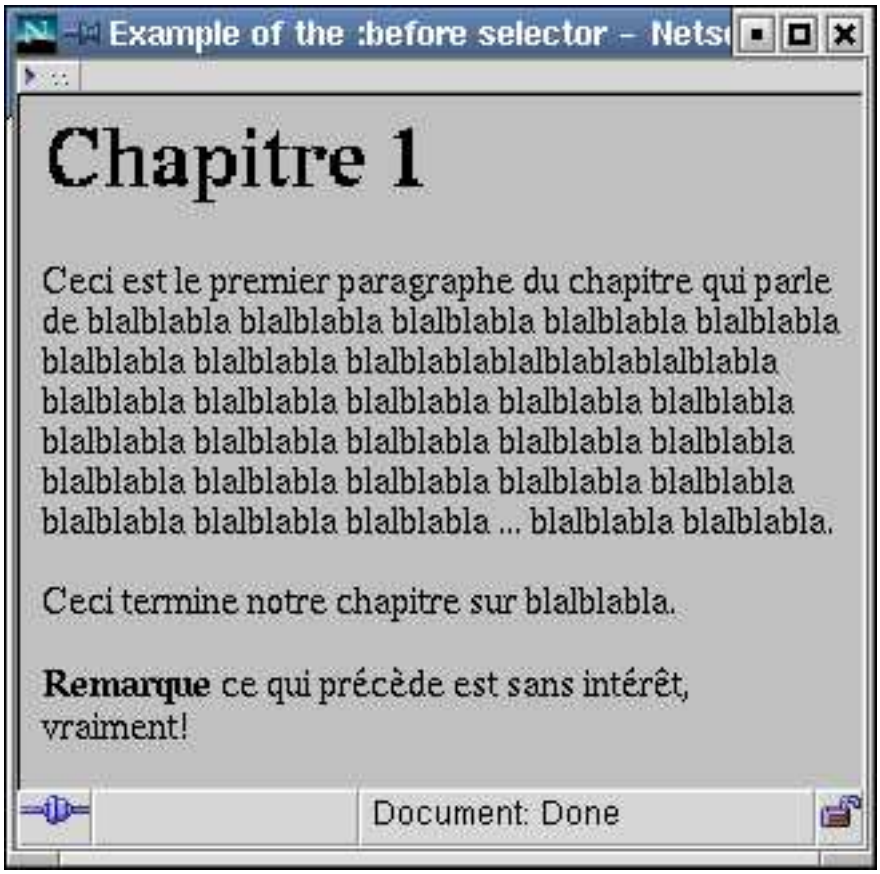


FIG. 13.10 – Utilisation du sélecteur **:before**

13.10 Les compteurs

Dans le HTML, seuls les listes possèdent un compteur d'élément. Cependant, il est possible que l'on veuille compter d'autres éléments comme des chapitres, des sections. C'est pourquoi la norme CSS2 a introduit la notion de compteur qu'il est maintenant possible d'associer à n'importe quel élément HTML. Ceci peut être très utile si l'on veut donner une entête hiérarchique aux différents titres de différent niveau.

13.10.1 Propriété de remise à zéro du compteur - *'counter-reset'*

Cette propriété premet de remettre à zéro un compteur à chaque nouvel apparition d'un élément dont la règle contient le counter-reset. Ainsi, par exemple, la ligne suivante :

```
H1 {counter-reset : section}
```

remet le compteur de section à zéro à chaque nouveau titre de niveau 1.

13.10.2 Propriété d'incrémentation du compteur - *'counter-increment'*

Cette fois, à chaque apparition d'un élément dont la règle contient la propriété, le compteur sera incrémenté d'une unité. Ainsi, par exemple dans la ligne suivante :

```
H2 {counter-increment : section}
```

à chaque titre de niveau 2 on incrémente le compteur section d'une unité.

13.10.3 L'élément *counter()*

Il est possible d'accéder au compteur grâce à l'élément **counter(nom)** où nom est le nom du compteur. Par défaut, le compteur est de type décimal. Il est possible de préciser le style du compteur. Par exemple si l'on veut faire précéder les titres de niveau 2 par leur numéro en chiffre romain, on utilisera les lignes suivantes :

```
H2 :before {content : counter(section, upper-roman) ". ";
            counter-increment section;}
```

13.10.4 L'élément *counters()*

Ce qui précède nous permet de construire des schémas de numérotation hiérarchique comme 1, 1.1, 1.2, 1.2.1,... assez facilement en associant des compteurs différents à chaque niveau de titre (6 maximum de H1 à H6). Toutefois, il est possible que des éléments soient nichés les uns dans les autres jusqu'à une profondeur arbitraire et que l'on désire également avoir une numérotation hiérarchisée. C'est le cas par exemple des listes imbriquées. Pour chacune des listes, il existe un compteur qui porte le même nom que son parent mais pas la même valeur. Il y a en fait plusieurs instances d'un même compteur. Pour avoir avoir accès à toutes les instances quelque soit le niveau auquel on se trouve, on utilise l'élément counters(itemnr) (attention au '***s***'). Voici un exemple d'utilisation de cet élément :

```
LI :before {content : counters(itemnr, ". ");
            counter-increment : itemnr;}
OL {counter-reset : itemnr;}
```

Remarque : un exemple plus détaillé des CSS sera proposé dans la section suivante sur le XML car le CSS peut également être utilisé en relation avec le XML.

13.11 Conclusion

Pour terminer, on insistera sur le fait que ce chapitre ne constitue qu'une introduction au CSS et que le lecteur se tournera vers les références (livres et électroniques : http ://www.w3c.org/css) pour un complément d'information. Le CSS constitue une première étape vers la séparation du contenu et du contenant. Nous avons vu qu'il était possible d'insérer les données d'un document entre des balises. Ces balises n'indiquant que la fonction du contenu dans le texte indépendemment de la façon dont cela va être affiché à l'écran, sur le papier, sur un PDA, etc. : par exemple la balise <H1> indiquant que son contenu est un titre de niveau 1, la balise <H1 class="chapter"> indiquant que son contenu est un titre de niveau 1 et de type 'chapter'. Les règles de style intervenant alors pour préciser la manière dont cela va être affiché à l'écran, sur le papier, sur un PDA,etc.

Cette manière de procéder est beaucoup plus propre, rend un document HTML beaucoup plus clair et constitue une première approche d'une séparation contenant-contenu plus poussée que l'on retrouve avec le XML (Voir chapitre IV).

Malgré tous les avantages qu'il présente et probablement à cause de la lenteur des butineurs à supporter toutes les spécifications CSS, le CSS n'a pas encore tout à fait percé. Il continue pourtant son petit bonhomme de chemin, ses adeptes se faisant de plus en plus nombreux et cela malgré l'essor du XML (sûrement parce que le CSS est plus facile d'approche que le XML). Nul à l'heure actuelle ne saurait dire ce qu'il adviendra du CSS, cela dépend d'un grand nombre de facteur. L'avenir en décidera.

Quatrième partie

eXtensible Markup Language

Chapitre 14

Introduction

14.1 Origine du XML

Avec l'introduction des styles, il est devenu possible de séparer le contenu de l'information avec la manière dont elle est affichée à l'écran. Le problème qu'il reste avec le langage HTML, est que l'information n'est pas structurée. En effet, supposons que nous voulions écrire du code HTML pour afficher des chansons (Voir listing 14.1 et FIG. 14.1).

Listing 14.1 – Liste de chansons en HTML

```
<H1> Deep Purple </H1>
<H2> In Rock </H2>
<UL>
  <LI> An. : 1969
  <LI> Maison de disques: EMI
  <LI> Titres:
    <UL>
      <LI> Speed King
      <LI> Child in Time
    </UL>
</UL>
<H2> Machine Head </H2>
<UL>
  <LI> An. : 1971
  <LI> Maison de disques: EMI
  <LI> Titres:
    <UL>
      <LI> Highway Star
      <LI> Picture of Home
      <LI> Smoke on the Water
    </UL>
</UL>
```

Quelqu'un qui ne connaît pas le groupe ne saurait que difficilement reconnaître la nature de l'information contenue dans ce document HTML. C'est pour cette raison, que le W3C a introduit le XML. Il s'agit d'un méta-langage à balises, c'est-à-dire que contrairement au langage HTML, le XML ne définit aucune balise précise, laissant à l'utilisateur toute la liberté d'en définir de nouvelles. Il s'agit en quelque sorte d'une généralisation du langage HTML ou d'un langage de niveau supérieur. Le listing 14.2 illustre un exemple de la même information, mais structurée en utilisant XML.

En comparant les deux listings, l'intérêt du XML apparaît clairement : en utilisant des balises spécifiques au domaine, le document est naturellement plus structuré. Il est donc plus aisé de comprendre l'information, que ce soit pour un être humain ou pour un programme de traitement.

Le XML est donc un méta-langage permettant de définir des langages spécifiques à différents domaines, appellées «applications XML». Chaque application possède sa propre syntaxe et son propre vocabulaire. Parmi les applications XML connues, citons le Chemical Markup Language (CML) qui permet le traitement des molécules en chimie, ou encore le Mathematical Markup Language (MathML) qui permet l'affichage de formules mathématiques. Le HTML n'est pas tout à fait une application XML car il ne repecte

Listing 14.2 – Liste de chansons en XML

```
<?xml version="1.0" standalone="no"?>
<?xml-stylesheet type="text/css" href="group.css"?>
<DISCOGRAPHY YEAR="2000">
<GROUP>
  <GROUP-NAME>
    Deep Purple
  </GROUP-NAME>
  <ALBUM>
    <ALBUM-NAME>
      In Rock
    </ALBUM-NAME>
    <YEAR>1969</YEAR>1969
    <PUBLISHER>EMI</PUBLISHER>EMI
    <SONG> Speed King </SONG>
    <SONG> Child in Time </SONG>
  </ALBUM>
  <ALBUM>
    <ALBUM-NAME>
      Machine Head
    </ALBUM-NAME>
    <YEAR>1971</YEAR>1971
    <PUBLISHER>EMI</PUBLISHER>EMI
    <SONG> Highway Star </SONG>
    <SONG> Picture of Home </SONG>
    <SONG> Smoke on the Water </SONG>
  </ALBUM>
</GROUP>
</DISCOGRAPHY>
```

FIG. 14.1 – Liste de chanson en HTML

pas complètement les spécifications XML. en fait seul ce qu'on appelle le eXtended HyperText Markup Language (XHTML) est une application XML. Il s'agit d'une version légèrement modifiée du HTML de manière à ce qu'il respecte les spécifications XML. Actuellement, la plupart des navigateurs ne supportent pas ou peu le XML, mais cela devrait changer dans le futur.

14.2 Technologie XML

La technologie XML s'appuie généralement sur pusieurs fichiers distincts qui ont chacun leur rôle à savoir :

1. Un fichier «.xml» : ce fichier contient toutes les *informations structurées* par les balises, définies dans le fichier «.dtd». Un exemple est donné par le listing 14.3
2. Un fichier «.dtd» : ce fichier contient la *description du langage* utilisé pour structurer l'information, c'est à dire la définition des balises, des attributs, de la valeur des attributs, du type de l'information contenue entre les balises <BALISE> et </BALISE>. Cette technologie s'appelle le Document Type Declaration (DTD). Un exemple de fichier DTD est donné avec le listing 14.4.
3. Un fichier de *style* : ce fichier peut être soit un fichier «.css» (Voir partie III) soit un fichier «.xsl» qui utilise la nouvelle technologie eXtensible Style Language (XSL). Ce fichier apporte une information sur la manière dont le contenu du document XML sera affiché à l'écran, sur le layout.

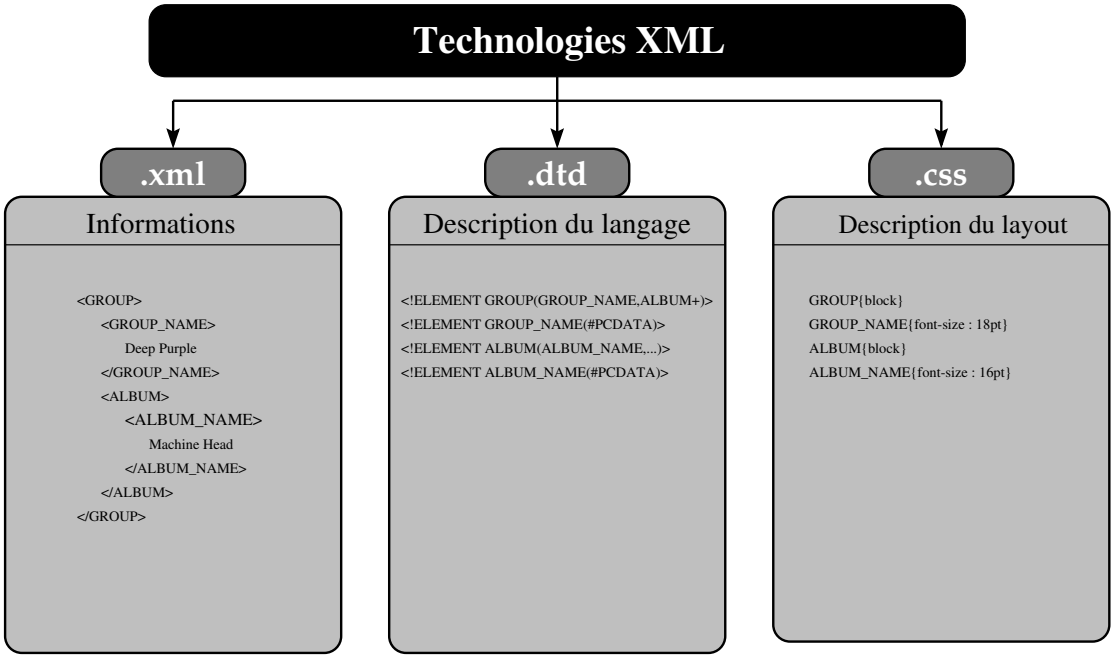


FIG. 14.2 – Structure de la technologie XML

Listing 14.3 – Exemple de fichier XML

```
<?xml version="1.0" standalone="yes"?>
<?xml-stylesheet type="text/css" href="greeting.css"?>
<GREETING>
  Hello XML!
</GREETING>
```

Listing 14.4 – Exemple de fichier DTD

```
<!ELEMENT GREETING (#PCDATA)>
```

Il faut également remarquer que ces trois fichiers peuvent être contenus dans un fichier XML unique, les informations DTD et XSL étant encapsulées dans les balises adéquates. Ces méthodes seront décrites

dans les chapitres correspondants.

Cependant, un fichier .xml peut très bien exister tout seul à l'heure actuelle car les navigateurs ne vérifient pas si un document .xml respecte la définition du fichier «.dtd». Le fichier de style est lui aussi optionnel car le layout par défaut des navigateurs permet un affichage structuré du fichier xml. Cependant, ce layout par défaut est en général assez limité. Ainsi par exemple Internet Explorer 5.5 affiche le document en gardant les balises (voir FIG. 14.3).

FIG. 14.3 – Comparaison du layout rendu par IE5.5 pour le même fichier XML sans et avec lien vers un fichier CSS

14.3 Le XML et les navigateurs

Comme pour toute nouvelle technologie, les navigateurs ont du retard sur l'implémentation de ce standard. Cependant, la plupart des navigateurs semblent lui emboîter le pas.

- *Internet Explorer 5.5 :* il comprend les fichiers XML et si il ne fait pas encore de vérifications avec le fichier DTD qui lui est joint, il vérifie quand même que le fichier existe s'il est spécifié. De plus il effectue le lien avec le fichier CSS spécifié mais avec le même support que pour le HTML, c'est-à-dire incomplet. Cependant, il supporte à moitié la technologie XLS (Voir chapitre 17).
- *Netscape* 4.7 : le cas est assez ambigu. En effet, Netscape 4.7 ne supporte pas le XML mais il l'utilise en interne (invisible pour l'utilisateur).
- *Netscape 6* : la nouvelle mouture de Netscape, Netscape 6 supporte très bien le XML et le CSS. c'est à l'heure actuelle le navigateur qui le supporte le mieux. Donc, même si nous déconseillons son utilisation pour des raisons de lenteur dues à son concept de développement (complètement développé en Java) et pour les bogues qui accompagnent ce navigateur toujours en version beta, il apparaît bien utile pour le développement de nos documents XML.

14.4 Références

Vous trouverez dans cette partie sur le XML un description plus ou moins détaillée des technologies associées au XML. Cependant pour celles qui le sont moins, nous vous donnons quelques références :

- www.w3c.org/XML/ vous y trouverez toutes les informations nécessaires sur le XML et à chaque fois ce seront des nouvelles fraiches.
- www.w3c.org/XSL/ c'est le site du développement du XSL.

Ainsi que l'ouvrage [10].

Chapitre 15

Le format XML

15.1 Structure générale d'un fichier XML

On peut voir au listing 15.1 la forme générale d'un fichier au format XML.

Listing 15.1 – Fichier XML

```
<?xml version="1.0" standalone="yes"?>
<?xml-stylesheet type="text/css" href="greeting.css"?>
<GREETING>
   Hello XML!
</GREETING>
```

Tout fichier XML doit toujours commencer par une ligne comportant la déclaration de celui-ci. L'attribut **version** permet de spécifier le numéro de version XML utilisé. L'attribut **standalone** permet de spécifier si le document est entièrement défini par lui-même (Valeur **yes**) ou s'il fait appel à d'autres documents (Valeur **no**).

Par défaut, un navigateur affiche le contenu d'un fichier XML de manière «brute» ; un affichage plus élaboré sera réaliser avec des styles. Ceci peut se faire comme indiqué à la ligne 2. Dans le fichier «greetings.css», il faut donc définir l'élément «GREETING» afin de permettre aux navigateurs d'afficher l'information correctement.

Il faut toujours une balise contenant toutes les autres, comme la balise **<HTML>** en langage HTML. Lorsqu'une balise en contient d'autres, elle devient la parente des autres balises qui sont les enfants de cette dernière. Comme les balises peuvent s'imbriquer, des enfants peuvent devenir des parents pour d'autres balises.

15.2 Utilisation de balises en XML

L'utilisation des balises en langage XML est semblable à celui du langage HTML décrit dans les chapitres précédents. Il y a cependant plusieurs différences notables qu'il est bon de signaler.

Contrairement au langage HTML, dans le langage XML, le nom des balises et de ses attributs sont sensibles à la casse des caractères. Pour le langage XML, les balises «GREETINGS» et «Greetings» sont deux balises différentes. Ceci dit, une bonne pratique consiste à ne pas utiliser des balises ne pouvant être différenciées que par leur cassure et à respecter une convention unique pour les noms.

Dans le langage HTML, certaines balises, comme **
**, sont auto-suffisantes, ce qui n'est pas le cas dans le langage XML, où chaque balise ouverte doit être fermée, même s'il n'y a aucune information entre l'ouverture et la fermeture de celle-ci, comme dans le code :

```
<Exemple attr="attr1"></Exemple>
```

Ceci dit, dans ce cas particulier, il existe une forme légèrement différente qui permet d'éviter la lourdeur de fermeture, et qui consiste à terminer la balise par '/>' :

```
<Exemple attr="attr1"/>
```

Remark: *La spécification XML précise qu'aucune tentative ne doit être faite pour essayer d'interpréter du code XML qui ne serait pas parfaitement valide. En particulier, les navigateurs acceptent parfois des imperfections dans du code HTML, en essayant de fermer les balises lorsque le designer les oublie, mis ceci est formellement interdit par XML, assurant ainsi une structuration parfaite de l'information contenue.*

15.3 Attributs ou enfants

Une question qui est souvent posée est de savoir s'il vaut mieux définir un attribut ou un enfant pour représenter une certaine partie de l'information. En effet, il est possible de définir un fichier XML 15.2 qui représenterait la même chose que 14.2, mais qui utiliserait des attributs plutôt que des enfants.

Listing 15.2 – Liste de chansons en XML

```
<GROUP NAME="Deep Purple">
  <ALBUM NAME="In Rock" YEAR="1969" PUBLISHER="EMI">
    <SONG NAME="Speed King" />
    <SONG NAME="Child in Time" />
  </ALBUM>
  <ALBUM NAME="Machine Head" YEAR="1971" PUBLISHER="EMI">
    <SONG NAME="Highway Star" />
    <SONG NAME="Picture of Home" />
    <SONG NAME="Smoke on the Water" />
  </ALBUM>
</GROUP>
```

En fait, les deux approches semblent identiques. D'une manière générale, lorsque, pour ajouter une information à une balise, un seul enfant est nécessaire, par exemple pour le nom du groupe ou de l'album, cela vaut la peine d'utiliser éventuellement des attributs. Par contre, il serait insensé d'aller ajouter un attribut «ALBUM» dans la balise «GROUP», puisqu'un groupe peut avoir fait plusieurs albums.

De plus, il faut savoir qu'avec les CSS, on ne peut afficher que le contenu des balises, et non pas leurs attributs. C'est la raison pour laquelle, un nouveau langage a été introduit : le XSL. Il permet d'afficher les attributs des différentes balises d'un document XML. L'étude de cette extension, par ailleurs encore peu supportée, est traitée au chapitre 17. Pour une étude complète, il vaut mieux se reporter à [30] où à [10].

Chapitre 16

Document Type Declaration

16.1 Utilité des DTD

Comme expliqué au chapitre précédent, le méta-langage XML permet de créer n'importe quel type de langage. Supposons qu'une médiathèque souhaite utiliser le langage XML afin de stocker l'ensemble des disques et des CDs qu'elle possède. Elle définit donc un langage XML du type de celui donné dans les exemples, en demandant à ses employés d'écrire les fichiers XML. Le problème qui va se poser est d'assurer la validité de ces fichiers. En effet, un grand nombre de personnes seront susceptible de créer ou modifier des fichiers de données en format XML. Or comme il est possible de créer n'importe quelle balise, il y a de fortes chances pour que quelqu'un fasse une faute dans son document en utilisant une balise qui ne signifie rien dans cette application.

C'est pour cette raison que les DTD ont été introduits. Il s'agit d'un moyen de décrire un langage XML donné, en particulier de spécifier les balises valides, c'est-à-dire spécifier pour chaque balise, les enfants qu'elle peut ou qu'elle doit avoir ou bien encore les attributs dont elle est composée. Il existe des programmes capables de vérifier la validité de chaque document XML en comparant celui-ci avec le DTD correspondant.

16.2 Format d'un DTD

Le code DTD peut soit être inclus directement dans le fichier DTD après les directives XML et avant le code du document XML lui-même, ou dans un fichier séparé qui sera référencé dans le document XML. On peut voir dans le listing 16.1, l'exemple du listing 15.1 avec du code DTD.

Listing 16.1 – Fichier XML avec DTD

```
<?xml version="1.0" standalone="yes"?>
<?xml-stylesheet type="text/css" href="greeting.css"?>
<!DOCTYPE GREETING [
  <!ELEMENT GREETING (#PCDATA)>
]>
<GREETING>
  Hello XML!
</GREETING>
```

On peut voir le code DTD entre la ligne 3 et ligne 5. La forme générale du code DTD est :

```
<!DOCTYPE docname [
  <!- Declaration des éléments valides ->
]>
```

Pour inclure une référence à un fichier contenant le code DTD, il faut inclure dans le fichier XML une ligne telle que :

```
<!DOCTYPE docname SYSTEM "url">
```

Une dernière forme valide permet de spécifier que n'importe quel élément inséré est valide :

```
<!DOCTYPE docname ANY>
```

16.3 Déclaration d'un élément DTD

La forme générale d'un élément DTD est :

```
<!ELEMENT nom (éléments)>
```

Les éléments pouvant être inclus dans la balise *nom* sont spécifiés entre les parenthèses . Une spécification spéciale est **#PCDATA** : cela signifie que la balise ne peut recevoir que du texte ne contenant pas de code XML, à part, bien sûr, les séquences d'échappement héritées du langage HTML. Une autre spécification **EMPTY** permet de créer des balises vides. Les autres possibilités sont le nom des éléments pouvant être contenus dans la balise définie.

Pour reprendre l'exemple des chansons du listing 14.2, on aurait :

```
<!ELEMENT GROUP (GROUP_NAME, ALBUM+)>
```

La forme «*nom*+» permet de dire qu'il faut au moins un élément, mais qu'il peut y en avoir plusieurs. Pour spécifier qu'il peut y avoir zéro ou plusieurs éléments, on utilise la forme «*nom**». Pour spécifier qu'il ne faut que deux éléments «*nom*», il suffit de mettre deux fois le nom dans la déclaration. Ainsi, l'exemple suivant :

```
<!ELEMENT ESSAI (nom1, nom2, nom2, (nom3 | nom4), nom5+, nom6*,nom7?)
```

Cette déclaration signifie que la balise «*ESSAI*», doit posséder une balise «*nom1*», deux balises *«nom2*», une balise «*nom3*» ou une balise «*nom4*», au moins une balise «*nom5*», éventuellement plusieures balises «*nom6*», éventuellement une balise *«nom7*» (Voir TAB. 16.1).

TAB. 16.1 – Symboles dans une liste d'enfants directs

Symboles	Signification
none	un et un seul
|	ou
+	un ou plusieurs
*	zéro ou plusieurs
?	zéro ou un

16.4 Déclaration d'attributs d'un élément DTD

Une balise XML peut contenir plusieurs attributs. Le langage DTD permet également de spécifier la manière de traiter les attributs dans un langage XML. Pour ce faire, la déclaration suivante est utilisée :

```
<!ATTLIST élément attribut type spécification>
```

Le nom de l'élément est premièrement indiqué, puis celui de l'attribut et enfin son type (voir TAB.16.2).

Pour la spécification, il y a plusieures possibilités :

"value" La valeur «*value*» est celle par défaut.

#REQUIRED L'attribut doit être spécifié.

#IMPLIED L'attribut ne doit pas être spécifié et n'a pas de valeur par défaut.

#FIXED *"value"* L'attribut a une valeur *«value*» fixée .

Pour les balises permettant l'utilisation de plusieurs attributs, il faut déclarer autant de commandes DTD **<!ATTLIST>** qu'il y a d'attributs.

TAB. 16.2 – Les types d'attribut

Type	Signification
CDATA	Texte n'étant pas du code XML outre les séquences d'échappement
Enuméré	Un choix parmi plusieurs proposés : (*choix1* | *choix 2* | ... | *choix n*)
ID	Un nom unique dans tout le document XML
IDREF	Une référence à un attribut 'ID' existant.
IDREFS	Liste de références séparées par des espaces d'attributs 'ID' existants
ENTITY	Le nom d'un élément DTD
ENTITIES	Liste de références séparées par des espaces de noms d'éléments DTD
NMTOKEN	Un nom valide pour le langage XML
NMTOKENS	Liste de noms valides pour le langage XML séparés
NOTATION	Le nom d'une notation définie par le DTD.

16.5 Déclaration d'entités DTD

16.5.1 Entités internes

Une entité interne DTD permet d'associer un nom avec une chaîne de caractères quelconque. La déclaration d'une entité a la forme générale suivante :

```
<!ENTITY % nom "texte de remplacement">
```

Le texte de remplacement doit toujours être compris entre deux guillemets afin de permettre le fait qu'il contienne du code XML valide. Pour référencer une entité, il faut utiliser le caractère '%' suivi du nom de l'entité et enfin du caractère ' ;'. Voici un exemple très simple d'utilisation d'une entité :

```
<!ENTITY % PCD "(#PCDATA)">
<!ELEMENT ANIMAL %PCD;>
```

Grâce à l'utilisation d'entités, le code DTD devient plus facile à lire. En effet, examinons le code du listing 16.2 :

Listing 16.2 – Exemple d'entités

```
<!ELEMENT NOM (#PCDATA)>
<!ELEMENT STUDIO (#PCDATA)>
<!ELEMENT LIVE (#PCDATA)>
<!ELEMENT CD (#PCDATA)>
<!ELEMENT MD (#PCDATA)>
<!ELEMENT LP (#PCDATA)>
<!ELEMENT AT (#PCDATA)>

<!ELEMENT ALBUM1 (NOM,(CD | MD | LP | AT),(LIVE | STUDIO),ANNEE)>

<!ENTITY % SUPPORT "(CD | MD | LP | AT)">
<!ENTITY % TYPE "(LIVE | STUDIO)">
<!ELEMENT ALBUM2 (NOM,%SUPPORT;,%TYPE;,ANNEE)>
```

Deux éléments représentant l'album d'un groupe sont définis : l'un, appellé «ALBUM1» (Ligne 9), en utilisant directement tous les éléments, et l'autre, appellé «ALBUM2» (Ligne 13), utilisant des entités. Outre le fait que cela soit plus facile à lire, cela permet également de pouvoir réutiliser les entités dans la déclaration d'autres éléments.

Remark: *Il est particulièrement intéressant de remarquer que la définition d'une entité peut contenir une référence à une autre entité.*

16.5.2 Entités externes

Une entité externe permet d'associer un nom avec le contenu d'un document référencé par une URL. La déclaration d'une entité externe a la forme suivante :

```
<!ENTITY % nom SYSTEM "url">
```

Ce type d'entité est spécialement intéressante pour découper un grand document en une série de petits documents comme illustré dans le listing 16.3.

Listing 16.3 – Exemple d'entités externes

```
<?xml version="1.0" standalone="yes"?>
<?xml-stylesheet type="text/css" href="musique.css"?>
<!DOCTYPE MUSIQUE SYSTEM "musique.dtd" [
  <!ENTITY % deeppurple "deeppurple.xml">
  <!ENTITY % ledzeppelin "ledzeppelin.xml">
]>
<MUSIQUE>
  <GROUP NAME="Deep Purple">
    %deeppurple;
  </GROUP>
  <GROUP NAME="Led Zeppelin">
    %ledzeppelin;
  </GROUP>
</MUSIQUE>
```

Chapitre 17

eXtensible Style Language (XSL)

17.1 Introduction

Le XSL permet de préciser la manière dont le document XML doit être affiché à l'écran ou sur tout autre media. Cependant, le XSL est plus complet que cela car il permet également de traduire un fichier XML en un autre fichier XML (voir même un fichier HTML ou SGML).

Le XSL est donc un double langage.

1. Transformation : encore appelé l'eXtensible Style Language Transformation (XSLT) pemet de traduire un document XML en un autre document XML, ou en un document HTML ou SGML (l'inverse n'étant pas possible vu la manière donc fonctionne le XSL). Ceci peut avoir de nombreuses applications dans l'e-commerce, l'échange de données électroniques, le traitement de données électroniques ou de meta-informations.
2. Formatage : à savoir une description du layout de l'informaiton contenue dans un document XML. Cette partie du XSL est aussi appelée l'eXtensible Style Language Formatting Object (XSLFO).

Le XSL s'utilise donc de plusieurs manières :

1. Le XSLT peut être utilisé seul, pour transformer un document XML en un autre document XML ayant une structure différente.
2. Le XSLFO peut également être utilisé seul pour formater directement du texte, ce qui n'a aucun intérêt puisque il faudrait idéalement pouvoir appliquer un même style à plusieurs éléments, voire à plusieurs documents différents mais utilisant la même structure XML. C'est donc dans cette optique qu'il faut voir le XSLFO.
3. La dernière approche consiste donc à utiliser le XSLT et le XSLFO de manière combinée.

17.2 eXtensible Style Language Transformation

17.2.1 Principe

Le XSLT travaille sur la structure arborée que possède tout document XML. En effet, un document XML définit une structure hiérarchique, représentable sous forme d'arbre. Les noeuds de l'arbres sont de 7 sortes possibles :

1. la racine, qui est l'élément parent général du document,
2. les éléments, c'est-à-dire les balises,
3. le texte, situé entre deux balises,
4. les attributs, à l'intérieur d'une balise,
5. les espaces noms (namespace), c'est à dire les séparateurs d'un document XML (qui séparent les balises ayant le même nom mais appartenant l'une à l'input et l'autre à l'output),

6. les instructions de traitement des données,
7. les commentaires.

Par exemple pour le listing 14.2, la structure en arbre est donnée à la FIG. 17.1. Chaque noeud possède 0, 1 ou plusieurs noeuds enfants qui sont eux-même des arbres.

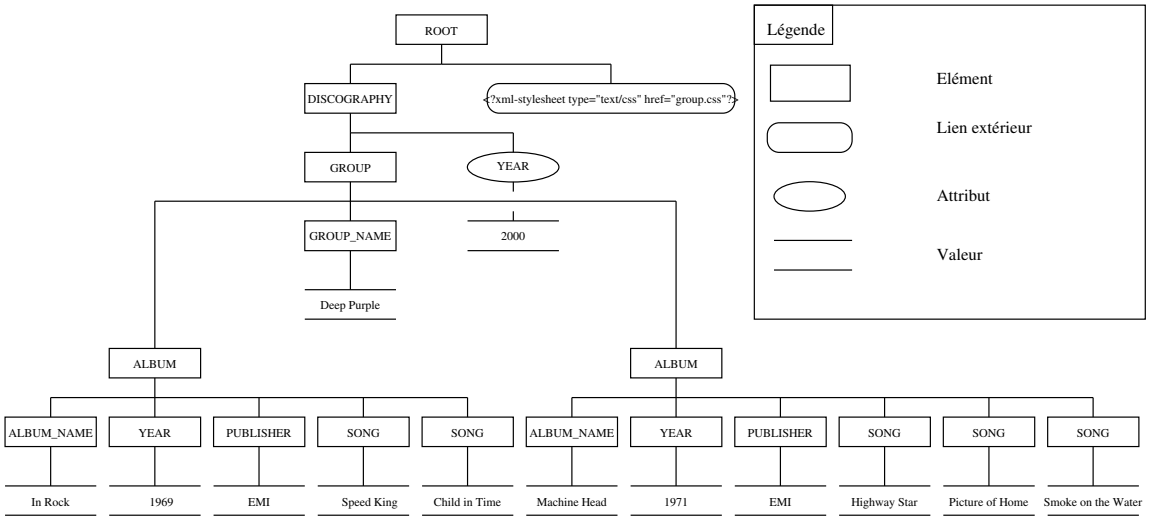


FIG. 17.1 – Structure hiérarchique arborée du listing 14.2

Le XSLT transforme un arbre XML en un autre arbre XML.

17.2.1.1 Comment cela fonctionne-t-il réellement ?

En fait, un fichier XSL possède un ensemble de règles dites de «template» ainsi que d'autres règles de style. Les règles de template possèdent tout d'abord un pattern décrivant l'arbre auquel elle s'applique et un template qui décrit l'arbre en sortie de transformation. Donc un processeur XSL parcoure tout le document XML à la recherche d'un arbre correspondant à un pattern décrit dans une de ses règle de template et lui applique la transformation correspondante.

Un template possède en général des balises, de nouvelles données, des données récupérées de l'original.

Il faut également remarquer que étant donné que le XML est un meta-langage, le XSL décrit ses templates en utilisant le XML.

17.2.2 Préambule

Donc le XSL est lui-même un élément du document XML auquel il s'applique. Le noeud parent des informations XSL est le noeud

```
<xsl :stylesheet>
```

si sa description est dans le même fichier que le document XML et le noeud

```
<?xsl-stylesheet type="text/xsl" href="..."?>
```

si il y est inclus sous la forme d'un lien externe. Le template décrivant l'output peut évidemment contenir des éléments de formatage issus du XSLFO.

Il faut également préciser le namespace des élément XSL. Le fichier XSL présentera donc le préambule suivant :

```
<?xml version="1.0"?>
<xsl :stylesheet
xmlns :xsl="http ://www.w3c.org/XSL/Transform/1.0">
```

17.2.3 Les règles de template

Les règles de template sont décrites dans des éléments du type

```
<xsl :template> ... </xsl :template>
```

Voyons comment déclarer les deux éléments qui constituent une règle de template à savoir le pattern et le template.

17.2.3.1 Les patterns

Le pattern est très simplement décrit dans l'élément **<xsl :template>** via l'attribut **match** selon une commande du type :

```
<xsl :template match="TREE">
```

où **TREE** est le pattern. Ainsi, la commande suivante :

```
<xsl :template match="/DISCOGRAPHY/GROUP">
```

indiquera que le template s'appliquera aux groupes de notre discographie.

17.2.3.2 Les templates

Les templates sont décrit à l'intérieur des noeuds **<xsl :template>**.

Les noeuds commençant par le préfixe *xsl :* indique une sélection d'une partie de l'arbre d'entrée qui sera inclue dans l'arbre de sortie. Les autres seront introduits tels quels dans l'arbre de sortie.

Par exemple le code suivant :

```
<xsl :template match="/DISCOGRAPHY/GROUP">
 <HTML>
  <BODY>
   <H1>
    <xsl :value-of select="GROUP-NAME"/>
   </H1>
  </BODY>
 </HTML>
</xsl :template>
```

appliqué au listing 18.2 à la page 121 donnera un output du style :

```
<HTML>
<BODY>
<H1> Deep Purple </H1>
<H1> Dire Straits </H1>
<H1> Radiohead </H1>
</BODY>
</HTML>
```

Les balises **<HTML>**, **<BODY>** et **<H1>** ont été restituée telle quelle dans l'output alors que la balise **<xsl :value-of select**="GROUP-NAME"**/>** a utilisé les données du document XML.

17.2.4 Les éléments de transformation

Nous verrons dans cette section quelques élément de transformation utilisés dans le XSL. Cependant, nous ne détaillerons pas toutes les spécifications étant donné qu'elles ne sont supportées que par peu de navigateur et encore partiellement.

17.2.4.1 L'élément xsl :apply-templates

Cette instruction indique que les règles de template doivent être appliquées aux enfants directs ayant pour nom la valeur de l'attribut **select**. Ainsi par exemple :

```
<xsl :template match"/">
<xsl :apply-templates select="DISCOGRAPHY"/>
</xsl :template>
```

Cette règle de template indique que tous les éléments DISCOGRAPHY de l'arbre doivent subir les règles de template.

17.2.4.2 L'élément xsl :value-of

Cett instruction indique qu'il faut introduire dans l'output la donnée de l'enfant direct ayant pour nom la valeur de l'attribut **select**.

Par exemple cette règle de template :

```
<xsl :template match="/DISCOGRAPHY/GROUP">
<xsl :value-of select="GROUP-NAME"/>
</xsl :template>
```

indique qu'il faut introduire la valeur du noeud «GROUP-NAME» enfant de l'arbre /DISCOGRAPHY/GROUP.

Il est également possible de sélectionner la valeur d'un attribut d'un élément en utilisant le symbole @.

```
<xsl :template match="/DISCOGRPAHY">
<xsl :value-of select="@YEAR"/>
</xsl :template>
```

17.3 eXtensible Style Language Formatting Object

Le XSLFO est la deuxième partie du XSL, à savoir la partie de description du layout des éléments contenus dans le fichier XML. C'est donc l'équivalent du CSS. La seule différence, mais assez majeure, est que le XSL est écrit dans un format XML alors que le CSS utilise son propre format. Rien n'empêche donc les informations de formatage d'être traitées de la même manière que les autres noeuds du document XML par un processeur XSLT. Cependant comme à l'heure actuelle, il n'existe aucun outil software permettant de traiter les objets de formatage XSLFO, nous ne détaillerons pas cette section. Mais voici quelques exemples d'instructions XSLFO.

17.3.1 Préambule

Une feuille de style XSL comporte tout d'abord une déclaration précisant les noms d'espace :

```
<xsl :stylesheet
  xmlns :xsl="http ://www.w3c.org/TR/WD-xsl"
  xmlns :fo="http ://www.w3c.org/XSL/Format/1.0"
  result-ns="fo">
```

La plupart des noeuds de formatage commenceront donc par le préfixe **fo :**, comme par exemple **fo :block**, **fo :list-item**, ...

17.3.2 Comparaison XSLFO-CSS

Il existe des similitudes entre des ressemblances XSLFO et CSS. Ainsi la ligne XSLXSL suivante :

```
<fo :block font-family="Times New Roman, Times, Serif">
```

pourrait être traduite par la règle CSS suivante :

```
fo :block {font-family : Times New Roman, Times, Serif ;}
```

A une propriété CSS **font-family** du sélecteur **fo :block** corresonpdra donc un attribut XSL de l'élément .

Remark: *Une différence importante avec le CSS réside dans les sélecteurs. En effet, alors que dans le CSS, les sélecteurs sont utilisés pour appliquer des règles de style, dans le XSL, ce sont des instructions XSLT qui permettent de sélectionner un élément auquel on va appliquer un style.*

17.3.3 Les objets XSLFO

Les objets XSLFO décrivent chacun un objet formaté, c'est-à-dire le comportement structurel. Ainsi, le contenu d'un objet **<fo :block>** sera considéré comme un bloc (cf. la propriété **display : block ;** du CSS).

Il existe un grand nombre d'objets XMLFO (environ 50) qui s'appliquent soit à des données du document comme par exemple :

```
fo :block,
fo :float,
fo :list-item,
fo :table-caption,
fo :table-cell,
```

soit au support du formatage comme le papier avec par exemple :

```
fo :region-before,
```

indiquant la marge supérieure de la page,

```
fo :region-body,
```

indiquant le contenu de la page,

```
fo :region-after,
```

indiquant la marge inférieure de la page,

```
fo :region-start,
```

indiquant la marge gauche de la page,

```
fo :region-end,
```

indiquant la marge droite de la page.

17.3.4 Les propriétés de formatage

Ces propriétés sont en fait des attributs des objets XSLFO. Il en existe un très grand nombre (environ 210) qui reprennent la plupart des propriétés CSS telles quelles comme par exemple :

```
font-size,
border,
margin,
background,
etc.
```

mais aussi de toutes nouvelles propriétés (absentes du CSS mais qui sont présentes dans la plupart des langages de formatage tel que LaTex) comme par exemple :

```
hyphenate,
page-master-even,
scale,
rule-orientation,
etc.
```

Remark: *Nous n'énumérerons pas toutes les propriétés étant donné qu'aucun navigateur ne les supporte à l'heure actuelle et que le XSL est toujours en développement à la W3C (la première recommendation officielle est sortie il y a quelques mois seulement : XSLT 1.0 ; les autres recommendations n'étant pour le moment que des premiers jets). Le lecteur consultera les références pour de plus amples informations.*

17.4 Utilisation combinée du XSLT et du XSLFO

Dans cette approche, le fichier XML est transfomré par une série d'instructions XSLT en un fichier formaté avec le langage XSLFO.

Le fichier de sortie après le traitement par le fichier XSLT de formatage (c'est-à-dire contenant des éléments du XSLFO) est donc un fichier XML ne contenant plus que des informations de formatage. Ainsi, dans notre exemple de discographie, pour que les noms de groupe soient considéré comme des blocs, le fichier de sortie contiendra des lignes du type :

```
<fo :block> Deep Purple </fo :block>.
```

Toute information sur le contenu a donc disparue. Cependant, tout ceci est invisble à l'utilisateur qui n'aura accès qu'au fichier XML de départ et aux instructions XSLT mais pas au fichier de sortie (ceci est valable dans le cas des navigateurs mais on pourrait imaginer de développer une application qui traite un document XML avec un code XSL et crée un nouveau fichier XML contenant l'output).

Voici un exemple d'utilisation combinée d'instructions de transformation et d'instructions de formatage :

```
<xsl :template match="/DISCOGRAPHY/GROUP">
  <fo :block font-size="16pt" font-family="Times, Serif" color="red">
    <xsl :value-of select="GROUP-NAME"/>
  </fo :block>
</xsl :template>
```

Chapitre 18

Exemple détaillé de la technologie XML

Nous verrons dans cette partie un exemple un peu plus complet de l'utilisation des technologies XML pour la structuration des informations. Ainsi, le lecteur pourra se faire une idée plus précise de ce que peuvent apporter ces concepts.

Nous travaillerons sur l'exemple de la discographie. Les différents listing sont situés en pages 120, 121 et 122.

18.1 Construction du DTD

Il est évident que l'élément racine de notre structure XML sera un élément contenant toute la discographie. Nous l'appellerons donc «DISCOGRAPHY». Une discographie est toujours définie à un moment particulier. en effet, la discographie de chacun évolue au cours du temps, s'enrichit d'albums, de groupes, de chansons. Nous rajouterons donc un attribut indiquant la date de validité de la discographie :

```
<DISCOGRAPHY YEAR="2000">.
```

Une discographie est constituée d'un certain nombre de groupes qui ont produit des albums, mais au moins toujours d'un groupe ('+').

```
<GROUP>
```

Un groupe possède un et un seul nom ('none'), un ou plusieurs albums ('+') :

```
<GROUP-NAME>
<ALBUM>
```

Un album possède un et un seul nom ('none'), zéro (si elle est inconnue) ou une seule année de sortie (' ?'),zéro (s'il est inconnu) ou un seul éditeur (' ?') et un ou plusieurs morceaux ('+')

```
<ALBUM-NAME>
<YEAR>
<PUBLISHER>
<SONG>
```

Une chanson peut en plus posséder une durée que nous incluerons dans un attribut LENGTH :

```
<SONG LENGTH=" ">
```

Nous réduisons volontairement les informations à celles nécessaires à la compréhension de l'exercice. Le fichier DTD correspondant peut être observé au listing 18.1 se trouvant à la fin du chapitre.

18.2 Construction du document XML

Dans cet exemple nous avons encodé 3 groupes à savoir Deep Purple, Radiohead et Dire Straits avec respectivement deux, un et un albums.

Il existe deux liens externes pour ce document car nous avons opté pour une description DTD et une feuille de style extérieure de manière à améliorer la lisibilité du document XML.

La première ligne du fichier XML sera la rituelle

```
<?xml version="1.0" standalone="no"?>
```

avec l'attribut **standalone** mis à **no** de manière à indiquer que les liens avec un DTD et une feuille de style sont externes.

Les deux lignes suivantes précisent ces liens :

```
<!DOCTYPE Discography SYTEM "xml_disco.dtd">
<?xml-stylesheet type="text/css href="xml_disco.css"?>
```

Le document complet peut être observé au listing 18.2 se trouvant à la fin du chapitre.

18.3 Construction de la feuille de style

Nous allons détailler les différentes règles qui composent cette feuille de style.

Tout d'abord, nous désirons avoir un background de l'ensemble du document blanc :

```
DISCOGRAPHY {background-color : white;}
```

Ensuite, nous voulons séparer les différents groupes qui composent la discographie et les insérer dans des cadres. Nous allons donc préciser que nous désirons une bordure autour de l'élément «GROUP» qui sera éloigné de 1cm de sa bordure :

```
GROUP {
     border-style : solid;
     border-width : 2pt;
     padding : 1cm;
}
```

Les groupes auront un fond gris clair (#c0c0c0) et la couleur par défaut du texte sera bleue et de taille 10pt :

```
GROUP {
     background-color : #c0c0c0;
     color : blue;
     font-size : 10pt;
}
```

Les groupes auront une marge de 1 cm les séparant du reste et seront structurellement considérés comme des blocs :

```
GROUP {
     margin : 1cm;
     display : block;
}
```

De la même manière, nous considérerons les entités «GROUP-NAME», «ALBUM», «ALBUM-NAME», «YEAR», «PUBLISHER» comme des blocs :

```
GROUP-NAME, ALBUM, ALBUM-NAME, YEAR, PUBLISHER {
    display : block;
}
```

Le nom de groupe sera en rouge, en petites capitales, de taille 16pt, souligné et aligné à gauche :

```
GROUP-NAME {
    text-align : left;
    color : red;
    font-variant : small-caps;
    font-size : 16pt;
    text-decoration : underline;
}
```

Un album sera séparé de l'élément qui le précède par une marge de taille égale à la taille de la police du parent à savoir 16pt :

```
ALBUM {
    margin-top : 1em;
}
```

Le nom de l'album sera écrit en vert avec une taille de 14pt et indenté de 1cm par rapport à son parent :

```
ALBUM-NAME {
    text-indent : 1cm;
    font-size : 14pt;
    color : green;
}
```

L'année de l'album, l'éditeur, et les chansons seront indentées de seulement 0.5cm, et écrit en Verdana (ou en utilisant une police sans empattement) :

```
YEAR, PUBLISHER, SONG {
    text-indent : 0.5cm;
    font-family : Verdana, Sans Serif;
}
```

Nous voulons que les chansons soient bordées à gauche d'une ligne continue (**margin-top** *:* 0pt), décalées à droite de 1.5cm par rapport aux autres éléments de l'album («YEAR», «PUBLISHER») et qu'elles soient séparées de 2pt de la ligne à gauche (**padding**) :

```
SONG {
    border-left : 1pt solid;
    margin-top : 0pt;
    margin-left : 2cm;
    padding-left : 2pt;
}
```

Ensuite, nous allons utiliser les sélecteurs **:before** pour faire précéder les années et les éditions de chacun des album d'une entête correspondante :

```
YEAR :before {
    content : "Année : ";
}
PUBLISHER :before {
    content : "Edition : ";
}
```

Puis, nous voulons que les chansons soit séparée de l'élément qui la précède d'une distance égale à la taille de la police du parent, et cela quelque soit l'élément qui se trouve au dessus. Pour cela nous allons utiliser le sélecteur + qui permet de sélectionner un élément qui suit directement un autre de même niveau hiérarchique.

```
PUBLISHER+SONG, YEAR+SONG, ALBUM-NAME+SONG {
    margin-top : 1em;
}
```

Venons en maintenant à un outil très intéressant des feuilles de style : les listes numérotées. Nous allons faire en sorte de numéroter chacune des chansons en utilisant une numérotation décimale recommençant à zéro à chaque album. Pour cela, nous allons utiliser un compteur que nous appellerons «songno» que nous remettrons à zéro à chaque album, que nous incrémenterons à chaque chanson et que nous afficherons devant celles-ci :

```
ALBUM {
    counter-reset : songno;
)
SONG :before {
    content : counter(songno);
    counter-increment : songno;
}
```

Enfin, nous devons préciser que chaque chanson doit être considérée comme un élément de liste et que le numéro de la liste fait partie de l'élément (et donc se trouve à droite de la bordure gauche des chansons) :

```
SONG {
    display : list-item;
    list-style-type : decimal;
    list-style-position : inside;
}
```

18.4 Les listing

18.4.1 Le fichier DTD

Listing 18.1 – Description du langage utilisé pour structurer les données de la discographie

```
<!ELEMENT SONG (#PCDATA)>
<!ELEMENT PUBLISHER (#PCDATA)>
<!ELEMENT YEAR (#PCDATA)>
<!ELEMENT ALBUM-NAME (#PCDATA)>
<!ELEMENT GROUP-NAME (#PCDATA)>

<!ELEMENT ALBUM (ALBUM-NAME, YEAR?, PUBLISHER?, SONG+)>
<!ELEMENT GROUP (GROUP-NAME, ALBUM+)>
<!ELEMENT DISCOGRAPHY (GROUP+)>

<!ATTLIST SONG LENGTH #CDATA #IMPLIED>
<!ATTLIST DICOGRAPHY YEAR #CDATA 'TODAY'>
```

18.4.2 Le fichier XML

Listing 18.2 – Listing des données de la discographie

```
<?xml version="1.0" standalone="no"?>
<?xml-stylesheet type="text/css" href="group.css"?>
<DISCOGRAPHY>

<GROUP>
  <GROUP-NAME> Deep Purple </GROUP-NAME>
  <ALBUM>
    <ALBUM-NAME> In Rock </ALBUM-NAME>
    <YEAR> 1969 </YEAR> 1969
    <PUBLISHER> EMI </PUBLISHER>
    <SONG> Speed King </SONG>
    <SONG> Child in Time </SONG>
  </ALBUM>
  <ALBUM>
    <ALBUM-NAME> Machine Head </ALBUM-NAME>
    <YEAR> 1971 </YEAR> 1971
    <SONG> Highway Star </SONG>
    <SONG> Picture of Home </SONG>
    <SONG> Smoke on the Water </SONG>
  </ALBUM>
</GROUP>

<GROUP>
  <GROUP-NAME> Dire Straits </GROUP-NAME>
  <ALBUM>
    <ALBUM-NAME> Love Over Gold </ALBUM-NAME>
    <SONG LENGTH="14:15"> Telegraph Road </SONG>
    <SONG LENGTH="6:45"> Private Investigation </SONG>
    <SONG LENGTH="5:49"> Industrial Disease </SONG>
    <SONG LENGTH="6:16"> Love Over Gold </SONG>
    <SONG LENGTH="7:54"> It Never Rains </SONG>
  </ALBUM>
</GROUP>
<GROUP>
  <GROUP-NAME> Radiohead </GROUP-NAME>
  <ALBUM>
    <ALBUM-NAME> OK Computer </ALBUM-NAME>
    <YEAR> 1997 </YEAR> 1997
    <PUBLISHER> Parlophone </PUBLISHER>
    <SONG LENGTH=""> Airbag </SONG>
    <SONG LENGTH=""> Paranoid Android </SONG>
    <SONG LENGTH=""> Subterranean Homesick Alien </SONG>
    <SONG LENGTH=""> Exit Music (For a Film) </SONG>)
    <SONG LENGTH=""> Let Down </SONG>
    <SONG LENGTH=""> Karma Police </SONG>
    <SONG LENGTH=""> Electioneering </SONG>
    <SONG LENGTH=""> Climbing Up The Walls </SONG>
    <SONG LENGTH=""> No Surprises </SONG>
    <SONG LENGTH=""> Lucky </SONG>
    <SONG LENGTH=""> The Tourist </SONG>
  </ALBUM>
</GROUP>

</DISCOGRAPHY>
```

18.4.3 Le fichier CSS

Listing 18.3 – Règles CSS pour la discographie

```
DISCOGRAPHY {
        background-color : white;}
GROUP {
        display : block;
        width : 7cm;
        margin : 1cm;
        border-style : solid;
        border-width : 2pt;
        padding : 1cm;
        background-color : #C0C0C0;
        color : blue;
        font-size : 10pt;}
GROUP-NAME {
        display : block;
        text-align : left;
        color : red;
        font-variant : small-caps;
        font-size : 16pt;
        text-decoration : underline;}
ALBUM {
        display : block;
        counter-reset : songno;
        margin-top : 1em;}
ALBUM-NAME {
        display : block;
        color : green;
        font-size : 14pt;
        text-indent : 1cm;}
YEAR  {
        display : block;
        text-indent : 0.5cm;
        font-family : Verdana, Sans Serif;}
YEAR:before {
        content : "Annee : ";
        font-size : 75%;}
PUBLISHER {
        display : block;
        text-indent : 0.5cm;
        font-family : Verdana, Sans Serif;}
PUBLISHER:before {
        content : "Edition : ";
        font-size : 75%;}
PUBLISHER+SONG, YEAR+SONG, ALBUM-NAME+SONG  {
        margin-top : 1em;}
SONG:before {
        content : counter(songno);
        counter-increment : songno;}
SONG  {
        display : list-item;
        list-style-type : decimal;
        list-style-position : inside;
        border-left : 1pt solid;
        margin-top : 0pt;
        margin-left : 2cm;
        padding-left : 2pt;
        font-family : Verdana, Sans Serif;}
```

18.5 Le résultat graphique

Les FIG 18.1 et 18.2 nous permettent de voir ce que l'on obtient dans un navigateur comme Netscape 6.

FIG. 18.1 – Visualisation du listing 18.2 dans Netscape 6 (1)

FIG. 18.2 – Visualisation du listing 18.2 dans Netscape 6 (2)

Cinquième partie

Programmation coté client

Chapitre 19

JavaScript Client-Side

Ce chapitre introduit le langage de programmation JavaScript, en particulier la version « Client-Side » destinée à être exécutée par les browsers. Il s'agit d'un langage orienté objets. Pour une introduction générale à la programmation orientée objets, le lecteur est invité à consulter l'annexe A.

19.1 Les élements de JavaScript

19.1.1 Historique de JavaScript

JavaScript est un langage de programmation orienté objets développé par Netscape. Microsoft supporte JavaScript par l'intermédiaire de JScript qui est un clône du premier cité. C'est finalement en 1997 qu'une spécification pour JavaScript, appellée ECMA-262 ou ECMAScript, a été définie comme un moyen d'écrire des applications portables pour Internet. Aujourd'hui les browsers les plus utilisés reconnaissent JavaScript.

JavaScript est en fait un langage interprété, c'est-à-dire que le code est traduit en code machine chaque fois que celui-ci doit être exécuté[1]. L'objectif est de permettre la réalisation de petits programmes par des non-professionnels, pour vérifier par exemple les valeurs des données dans les composants d'un formulaire ou encore pour pouvoir rajouter des portions de code HTML lors de l'analyse d'un document. Il existe deux types d'implémentations : « JavaScript Client-Side » (JavaScript du coté client) et « JavaScript Server-Side » (JavaScript du coté serveur). Elles sont toutes deux basées sur la même syntaxe et sur une série de fonctions et d'objets communs, mais définissent chacune des extensions propres à leur emploi respectif.

Dans ce chapitre nous présenterons quelques-unes des caractéristiques du JavaScript sans entrer dans les détails. Pour une étude appronfondie, nous vous conseillons les références suivantes : [14], [15], [16], [6], [3]. Le JavaScript est en fait un subtil mélange des langages C/C++ et Perl.

19.1.2 Les variables

19.1.2.1 Les types de variables

JavaScript reconnaît plusieurs types de variables :

- les nombres, comme 42 ou 12,3 ;
- les booléens, prenant comme valeur soit **true** (vrai) soit **false** (faux) ;
- les chaînes de caractères comme « essai ! ».

En plus de ces types de variables, JavaScript propose des valeurs non typées pouvant être utilisées avec tous les types de variables :

- la valeur **null** spécifiant une valeur nulle ;
- la valeur **undefined** spécifiant que la variable est indéterminée.

Il est à remarquer que pour les chaînes de caractères, il existe une série de séquences d'échappement, largement inspirées du C, permettant d'introduire des caractères spéciaux. Par exemple, la séquence « \n » force un passage à la ligne. Le TAB. 19.1 reprend une liste des principales séquences d'échappement.

[1] Les langages compilés par contre sont traduits en code machine avant l'exécution lors d'une phase de compilation.

TAB. 19.1 – Les principales séquences d'échappement

Séquence	Signification
\n	Nouvelle ligne
\r	Retour de chariot
\t	Tabulation
\'	Apostrophe
\"	Guillemet
\\	Le caractère '\'
\XXX	Le caractère représenté par la valeur octale XXX.
\xXX	Le caractère représenté par la valeur hexadécimale XX.
\uXXXX	Le caractère repésenté par le code hexadécimale Unicode XXXX.

19.1.2.2 Conversion de données

La conversion des variables est dynamique, c'est-à-dire qu'une même variable peut changer de type tout au long du programme. Par exemple :

```
var answer = 42;
answer = "Coucou";
answer = "La valeur est " + 42; // retournera "La valeur est 42"
"37" + 7; // retournera "377"
"37" - 7; // retournera 30
```

Les deux dernières lignes s'expliquent de la manière suivante : il s'agit de faire des opérations entre une chaîne de caractères et un nombre. Par défaut, JavaScript tente de transformer l'opérant à droite en celui à gauche. Donc, dans notre cas, il va tout d'abord transformer le chiffre en une chaîne de caractère « 37 ». L'opérateur d'addition existant pour deux chaînes de caractères et représentant la concétation, la *somme* de la chaîne de caractère «37» et «7» donnera «377». Par contre, l'opérateur de soustraction n'existe pas pour les chaînes de caractères. JavaScript va alors essayer de transformer la chaîne de caractères « 37 » en nombre car la soustraction entre nombres existe. Comme cette conversion est possible, il est dés lors normal que le résultat soit 30.

19.1.2.3 Déclarations de variables

Pour déclarer une variable, il suffit de lui assigner une valeur, de la déclarer via l'instruction **var** ou de la déclarer de manière dynamique via l'instruction **new** bien connue du C++. Par exemple :

```
x=12;
var x=12;
var x;      // Dans ce cas, x prend la valeur undefined
```

Il existe une différence essentielle entre les déclarations dites *statiques* et celles dites *dynamiques*. L'existence d'une variable statique dépend de l'endroit où elle est déclarée, ce que les programmeurs appellent sa visibilité. C'est au programmeur de doit décider quand une variable dynamique peut ou ne peut plus être utilisée. Une variable est dite *globale* lorsqu'elle est visible dans tout le programme. par contre une variable déclarée dans une fonction, appelée variable locale, n'est visible qu'à l'intérieur de cette fonction.

L'utilisation de **var** pour les variables globales est optionnelle ; par contre pour les variables locales, elle est obligatoire.

19.1.2.4 Les tableaux

Un tableau est un ensemble d'éléments, chacun étant représenté par des crochets (**[]**). Par exemple, la déclaration suivante :

```
voitures = ["honda","toyota","vw"];
```

Ce tableau définit 5 éléments : voitures[0] valant « honda », voitures[2] valant «toyota», voitures[4] valant «vw» et les éléments voitures[1] et voitures [3] ayant la valeur **undefined**.

19.1.3 Les expressions et les opérateurs

Une expression est un ensemble valide de variables, de valeurs, d'opérateurs et d'autres expressions pouvant être évaluées sous la forme d'une valeur.

La plupart des opérateurs en JavaScript sont les mêmes que pour le langage C/C++. On peut les diviser en plusieurs catégories :

- les opérateurs d'assignation ;
- les opérateurs de comparaison ;
- les opérateurs arithmétiques ;
- les opérateurs binaires ;
- les opérateurs logiques ;
- les opérateurs sur les chaînes de caractères ;
- les opérateurs spéciaux.

De plus, JavaScript prévoit les opérateurs spéciaux suivants :

- opérateur conditionnel ;
- opérateur virgule ;
- **delete** ;
- **new** ;
- **this** ;
- **typeof** ;
- **void** ;

19.1.3.1 Les opérateurs d'assignation

L'opérateur d'assignation de base est le signe égal (**=**). En écrivant *x*=*y*, on assigne la valeur de *y* à *x*.

Il existe une liste d'autres opérateurs d'assignation inspirés du C/C++ repris dans le TAB. 19.2.

TAB. 19.2 – Opérateurs d'assignation

Opérateur	Signification
$x+ = y \ \& \ x = x + y$ \\	
$x- = y \ \& \ x = x - y$ \\	
$x* = y \ \& \ x = x * y$\\	
$x/ = y\& \ x = x/y$\\	
$x\% = y \ \& \ x = x\%y$\\	
$x <<= y\& \ x = x << y$\\	
$x >>= y\& \ x = x >> y$\\	
$x >>>= y \ \& \ x = x >>> y$ \\	
$x\& = y\& \ x = x\&y$ \\	
$x\wedge = y\& \ x = x \wedge y$ \\	
$x\| = y\& \ x = x\|y$ \\ \hline	

19.1.3.2 Les opérateurs de comparaison

Les opérateurs de comparaison comparent les deux opérandes d'une expression et retournent une valeur booléenne suivant que la comparaison est **true** ou **false**. Le TAB. 19.3 reprend les différents opérateurs.

TAB. 19.3 – Les opérateurs de comparaison

Opérateur	Description
Egalité (==)	Retourne **true** si les deux opérandes sont égales. Si elles ne sont pas du même type, JavaScript fait les conversions appropriées avant la comparaison.
Non-égalité (!=)	Retourne **true** si les deux opérandes ne sont pas égales. Si elles ne sont pas du même type, JavaScript fait les conversions appropriées avant la comparaison.
Egalité stricte (===)	Retourne **true** si les deux opérandes sont égales et du même type.
Non-égalité stricte (!==)	Retourne **true** si les deux opérants opérandes ne sont pas égales et/ou du même type.
Plus grand que (>)	Retourne **true** si l'opérande à gauche de l'opérateur est plus grande que celle à droite.
Plus grand ou égal (>=)	Retourne **true** si l'opérande à gauche de l'opérateur est plus grande ou égale à celle à droite.
Plus petit que (<)	Retourne **true** si l'opérande à gauche de l'opérateur est plus petite que celle à droite.
Plus petit ou égal (<=)	Retourne **true** si l'opérande à gauche de l'opérateur est plus petite ou égal à celle à droite.

L'existence d'opérateurs de comparaison ne tenant pas compte des conversions est logique à partir du moment où JavaScript propose une converstion dynamique. L'exemple suivant illustre le propos :

```
if("37"==37)
  document.writeln("== OK");  // Sera affiché car après conversion, il y a égalité
if("37"===37)
  document.writeln("=== OK"); // Ne sera pas affiché car les chaînes ne sont pas des nomb
```

19.1.3.3 Les opérateurs arithmétiques

Les opérateurs arithmétiques ne fonctionnent qu'avec des opérandes numériques. Les opérateurs standards sont l'addition (**+**), la soustraction (−), la multiplication (*****) et la division (**/**). La division renvoie toujours un nombre en virgule flottante même si les deux opérateurs sont des entiers.

```
document.writeln("Le résultat de 1/2 est égal à ");
document.writeln(1/2);       // Résulat :0,5. Pour les autres langages cela serait 0.
```

En plus de ces opérateurs, JavaScript propose les opérateurs présentés dans le TAB. 19.4.

TAB. 19.4 – Les opérateurs arithmétiques

Opérateur	Description	Exemple
Modulo (%)	Retourne le reste d'une division.	12 % 5 retoune 2.
Incrémentation (++)	Ajoute un à un opérant. Il peut être utilisé en préfixe (++x), auquel cas il retourne la valeur de x après incrémentation, ou en postfixe (x++), dans quel cas il retourne la valeur de x avant incrémentation.	Si x vaut 3, ++x mettra x à 4 et retournera 4, alors que x++ mettra x à 4 et retournera 3.
Décrémentation (−−)	Enlève un à un opérant. Il peut être utilisé en préfixe (−−x), auquel cas il retourne la valeur de x après décrémentation, ou en postfixe (x−−), dans quel cas il retourne la valeur de x avant décrémentation.	Si x vaut 3, −−x mettra x à 2 et retournera 2, alors que x−− mettra x à 2 et retournera 3.
Négation (-)	Retoune la négation de l'opérant.	Si x vaut 3, -x retournera -3.

19.1.3.4 Les opérateurs binaires

Le but de ces opérateurs est de pouvoir travailler sur des entiers codés sur 32 bits. Ces opérateurs sont exactement les mêmes que dans le langage C/C++. Comme ils ne sont pour ainsi dire plus utilisés aujourd'hui, il n'est pas esentiel de les étudier ici.

TAB. 19.5 – Les opérateurs logiques

Opérateur	Utilisation	Description
« Et » logique (&&)	expr1 && expr2	Retourne **true** si les deux expressions sont évaluées à vraies.
« Ou » logique (‖)	expr1 ‖ expr2	Retourne **true** si une des deux expressions est évaluée à **true**.
« Non » logique (!)	!expr	Retourne **false** si l'expression est évaluée à **true**, sinon retourne **true**.

19.1.3.5 Les opérateurs logiques

Les opérateurs logiques permettent de combiner plusieurs expressions, en utilisant les différents opérateurs de comparaison, repris au TAB. 19.5.

Comme dans tous les autres langages de programmation, il est possible d'utiliser les parenthèses pour regrouper des opérateurs. L'exemple suivant sera **true** si a ou b est **true**, et si c ou d est **true**.

```
((a||b)&&(c||d))
```

19.1.3.6 Les opérateurs sur les chaînes de caractères

En plus des opérations de comparaison, il existe également un opérateur d'addition qui définit la concaténation des chaînes de caractères (**+** ou **+=**) :

```
var str = "my " + "string";      // str vaudra "my string"
```

19.1.3.7 L'opérateur conditionnel

L'opérateur conditionnel utilise trois opérants et fonctionne de la manière suivante :

```
condition? val1 : val2
```

Si la condition est vrai, c'est la valeur *val1* qui est renvoyée, sinon c'est la variable *val2* qui l'est. Voici un exemple :

```
status = (age>=18)? "adult" : "minor";
```

19.1.3.8 L'opérateur virgule

L'opérateur virgule évalue simplement les deux opérateurs et retourne la valeur de la seconde opérande. Cet opérateur est notamment utilisé dans les boucles **for** qui sera étudié plus tard.

19.1.3.9 delete

L'opérateur **delete** permet de détruire un objet, une propriété d'un objet ou encore un élément d'un tableau à un certain index.

```
delete nomObjet;
delete nomObjet.propriété;
delete nomObjet[indice];
```

19.1.3.10 new

L'opérateur **new** permet de créer une instance d'une classe définie par un utilisateur ou d'une des classes prédéfinies comme **Array**, **Boolean**, **Date**, **Function**, **Image**, **Number**, **Object**, **Option**, **RegExp** ou **String**.

```
nomobjet = new typeobjet ( param1 [,param2] ...[,paramN] );
```

19.1.3.11 this

Grâce à ce mot-clé, l'utilisateur possède toujours une référence sur l'objet courant. Cet opérateur permet à un objet d'appeller des fonctions nécessitant une référence.

Par exemple, supposons qu'il existe une fonction JavaScript :

```
function affiche(obj)
{
  alert("La valeur est " + obj.value);
}
```

Pour appeler cette fonction pour un certain contrôle d'un formulaire, on pourrait écrire :

```
<INPUT TYPE="text" NAME=''age" onChange="affiche(this)">
```

Le paramètre *obj* de la fonction *affiche()* sera alors remplacer par une référence sur l'objet représentant le composant «*age*».

19.1.3.12 typeof

L'opérateur **typeof** retourne une chaîne de caractères contenant le type de l'opérande de l'opérateur. Par exemple,

```
var str="coucou";
var a= ""
var i=10;
a= typeof str; // a="string"
a= typeof i; // a="number"
```

19.1.3.13 void

L'opérateur **void** permet d'évaluer une expression sans retourner de valeur. Ses formes sont :

```
void (expression);
void expression;
```

Supposons qu'une fonction affiche la valeur d'une varaible, puis incrémente celle-ci de un et renvoie le résultat :

```
function test(i) {
  alert(i);
  return(i+1);
}
```

Pour pouvoir appeller cette fonctio, il faudrait normalement l'assignée à une variable où l'utiliser dans une expression puisqu'une valeur est retournée. L'instruction **void** permet de ne pas tenir compte de la valeur de retour.

```
void (test(5));
```

19.1.4 Expressions régulières

Une expression régulière est un masque utilisé pour permettre la mise en correspondance de caractères dans des chaînes de caractères. En d'autres termes, il s'agit d'un ensemble de conventions permettant de tester l'équivalence de deux chaînes de caractères ou l'existence d'une série de sous-chaînes dans une chaîne donnée. Par exemple, l'expression régulière « /abc/ » sera équivalente à toute chaîne de caractères contenant les trois lettres « abc » ensemble et dans cet ordre. Il existe deux manières différentes de créer une expression régulière :

```
reg = /ab+c/;
reg = new RegExp("ab+c");
```

JavaScript a hérité du langage Perl les expressions régulières. Sa complexité est telle qu'une étude de l'utilisation de ces expressions régulières sortirait largement du cadre de ce cours. Une brève introduction sur ce sujet figure à l'annexe B.

La méthode **exec()** d'un objet du type **RegExp** permet de mettre en correspondance l'expression régulière et une chaîne de caractères. Si aucune mise en correspondance n'est possible, la méthode **exec()** renvoie la valeur **null**, dans le cas contraire elle renvoie la partie qui aura été reconnue.

Par exemple, l'exemple suivant crée une expression régulière capable de « reconnaitre » un numéro de téléphone valide, c'est-à-dire ayant éventuellement un préfixe de deux à quatre chiffres avec un tiret, puis une trois parties séparées par des points, dont la première peut être composée de deux ou trois chiffres, et les autres de deux.

```
re = /[\d{2,4}-]?\d{2,3}.\d{2}.\d{2}/
OK = re.exec("02-705.13.32");     // Ok vaudra "02-705.13.32,02"
OK = re.exec("0475-70.36.12");    // Ok vaudra "0475-70.36.12,0475"
OK = re.exec("705.13.23");        // Ok vaudra "705.13.23"
OK = re.exec("7.32.12");          // Ok vaudra null
```

Dans les deux premiers résultats, on remarque que les extensions sont réécrires à la fin avec une virgule. Cela provient du fait que, dans les expression régulières, les parenthèses sont non seulement des moyens de regroupement mais également un moyen de déclarer des variables. Dès lors, outre la chaîne de caractères mise en correspondance, JavaScript renvoie à la suite de l'évaluation, la variable locale ainsi créée.

19.1.5 Les instructions

JavaScript propose une série d'instructions pour la plupart directement héritées du langage C/C++ :
- des instructions conditionnelles : **if...else** et **switch** ;
- des boucles : **for**, **while**, **do while**, **label**, **break**, **continue** ;
- des manipulations d'objets : **for...in** and **with** ;
- des commentaires.

Des instructions différentes sont séparées par un « ; ».

19.1.5.1 Les instructions if...else

Ces instructions permettent d'exécuter une partie de code si une condition est **true**, ou une autre partie de code, optionnelle, si elle est **false**.

```
if (condition) {
  instructions1
}
[else {
  instructions2
}]
```

19.1.5.2 L'instruction switch

L'instruction **switch** permet d'évaluer une expression et d'exécuter différentes parties de code en fonction du résultat de cette évaluation.

```
switch (expression) {
  case label :
    instruction;
    break;
  case label :
    instruction;
    break;
  ...
  default :
    instruction;
}
```

L'instruction break permet de s'assurer que le programme n'exécute que la partie de code correspondante à la valeur évaluée. Si cette instruction est omise, le programme continue d'exécuter les instructions correspondantes à la valeur suivante, ce qui peut parfois être la volonté du programmeur.

19.1.5.3 L'instruction for

La boucle **for** se répète jusqu'à ce qu'une condition soit évaluée comme fausse. La forme générale est :

```
for([expression-initial];[condition];[expression-incrémentation]) {
  instructions;
}
```

Le déroulement d'une boucle **for** est le suivant :

1. L'expression *expression-initial*, si elle existe, est exécutée. Généralement, on initialise des compteurs de boucles.
2. La *condition* est évaluée. Si le résultat est **true** la boucle continue ; sinon elle s'arrête.
3. Les instructions dans la boucle sont exécutées.
4. L'expression *expression-incrémentation* est exécutée, et la boucle retourne à l'étape 2.

L'exemple suivant montre une boucle affichant les dix premiers nombres :

```
for(i=0;i<=10;i++) {
  document.writeln(i);
}
```

19.1.5.4 Les instruction do...while

La boucle **do...while** se répète jusqu'à ce qu'une condition soit évaluée à **false**. La forme est la suivante :

```
do {
  instructions;
} while (condition);
```

Il est évident que les instructions contenues dans la boucle seront exécutées au moins une fois.

19.1.5.5 L'instruction while

La boucle **while** se répète tant qu'une condition est évaluée à vrai. La forme est la suivante :

```
while (condition) {
  instructions;
}
```

Il est évident que les instructions dans la boucle peuvent très bien ne jamais s'exécuter si la condition est déjà fausse lors de la première évaluation.

19.1.5.6 L'instruction label

L'instruction **label** permet d'assigner un identifiant à une instruction. La forme est :

```
label :
  instruction;
```

Généralement on l'utilise pour identifier des boucles et en combinaison avec les instructions **break** et **continue**.

19.1.5.7 L'instruction break

En plus de son utilisation dans l'instruction **switch**, **break** peut-être utilisé dans les boucles sous la forme :

```
break;
break [label];
```

Cette instruction permet de forcer la sortie d'une boucle donnée. Par exemple :

```
for(i=0;i<10;i++)
  for(j=0;j<10;j++) {
    if(i==5) break suite; // Dès que i vaut 5, on sort des deux boucles
    document.writeln(i+","+j);
  }
suite :
writeln("Suite");
```

L'extrait de code montre une double boucle imbriquée dans laquelle on utilise une instruction **break** pour sortir de celle-ci dès qu'une condition est remplie.

19.1.5.8 L'instruction continue

L'instruction **continue** permet de retourner à l'endroit d'une boucle où la condition est évaluée. Sa forme est

```
continue;
continue [label];
```

L'exemple suivant montre un extrait de code utilisant l'instruction **continue** pour afficher toutes les paires comprises entre «0,0» et «9,9», sauf celles comprises entre «5,6» et «5,9».

```
boucle1 :
for(i=0;i<10;i++)
  for(j=0;j<10;j++) {
   // Dès que i vaut 5 et j vaut 6, on retourne à la première boucle
    if((i==5)&&(j==6) continue suite;
    document.writeln(i+","+j);
  }
```

19.1.5.9 Les instructions for...in

Les instructions **for...in** permettent d'itérer sur une variable spécifique toutes les propriétés d'un objet. La forme est la suivante :

```
for (variable in objet) {
  instructions}
```

L'exemple suivant prend comme argument un objet et son nom. Il itère sur les propriétés de l'objet en affichant leur nom et leur valeur.

```
function affiche(obj,nom_obj) {
  var result = "";
  for (var i in obj) {
    result += nom_obj + "." + i + " = " + obj[i] + "<BR>";
  }
  result += "<HR>";
  return result;
}
```

Grâce aux instructions **for..in**, la variable *i* contient intérativement le nom de toutes les propriétés de l'objet *obj* passé en paramètre. En utilisant la référence de l'objet *obj* comme un tableau et avec la variable *i* contenant le nom d'une propriété comme paramètre, JavaScript renvoit la valeur de cette propriété.

19.1.5.10 L'instruction with

L'instruction **with** permet de spécifier un objet par défaut pour les instructions. La forme est :

```
with (objet) {
  instructions;
}
```

Supposons qu'un objet *obj* possède deux propriétés prop1 et prop2, le code suivant :

```
obj.prop1 = 1;
obj.prop2 = 2;
```

est équivalent à :

```
with (obj) {
  prop1 = 1;
  prop2 = 2;
}
```

19.1.5.11 Les commentaires

Pour commenter le code, les deux versions de commentaire du C/C++ peuvent être utilisées :
- les commentaires commencent par « // » et vont jusqu'à la fin de la ligne ;
- les commentaires sont compris, éventuellement sur plusieures lignes, entre « /* » et « */ ».

19.1.6 Les fonctions

19.1.6.1 Définition

Le langage JavaScript permet de définir des fonctions par :
- un nom de fonction ;
- une série de paramètres entre parenthèses et séparés par des virgules ;
- éventuellement une valeur de retour ;
- des instructions JavaScript comprises entre des accolades.

Par exemple, le code suivant définit une fonction renvoyant le carré d'un nombre passé en paramètre :

```
function carre(nb)
{
  return(nb*nb);
}
```

Remark: *Tous les paramètres ayant un des types fondamentaux sont passés par valeur, c'est-à-dire que tout changement de la valeur d'un paramètre dans une fonction n'a aucune répercussion sur la valeur de la variable passée en paramètre. Par contre, si on passe un objet en paramètre, toute modications faite sur l'une des propriétés est visible en dehors de la fonction.*

19.1.6.2 L'appel à une fonction

L'appel s'effectue tout simplement en écrivant le nom de la fonction et en spécifiant les paramètres. Par exemple :

```
a = carre(2);  // a vaudra 4.
b = carre(3);  // a vaudra 9.
```

Le JavaScript, autorise les appels récursifs aux fonctions. Une fonction peut donc s'appeler elle-même.

L'exemple classique est la fonction factorielle :

```
function factorielle(n)
{
  if((n==0) || (n==1))
    return 1;
```

```
    else {
      result=(n * factorielle(n-1) );
    }
    return result;
  }
```

19.1.6.3 L'utilisation du tableau arguments

Les paramètres d'une fonction sont maintenus dans une tableau, le premier paramètre correspondant à l'indice 0. Cela permet de définir des fonctions acceptant un nombre variable de paramètres.

Supposons que l'on veuille écrire une fonction qui concactène plusieurs chaînes de caractères en les séparant par un séparateur donné comme premier paramètre :

```
function concat(separateur) {
  result = "";
  for (var i=1; i<arguments.length; i++) { //length est le nombre d'arguments
    result += arguments[i] + separateur;
  }
  return result;
}
document.writeln(concat(";","str1","str2","str3"));
document.writeln(concat(";","str1","str2"));
```

Remark: *Dans la boucle on commence à l'indice 1 puisque l'indice 0 correspond au séparateur.*

19.1.6.4 La fonction eval

La fonction **eval()** permet d'évaluer une chaîne de caractères comme s'il s'agissait d'une série d'instructions JavaScript. La forme est :

```
eval(expr)
```

Par exemple, le code suivant affichera une boite de dialogue d'alerte en affichant « coucou » :

```
var str="Coucou";
var expr="alert(str);";
eval(expr);
```

Le listing 19.1 montre un exemple plus compliqué de l'utilisation de cette fonction dans le formulaire. L'utilisateur peut entrer une expression dans le champ texte correspondant, puis lorsqu'il clique sur un bouton, JavaScript tente d'exécuter l'expression. Le résultat est représenté à la FIG. 19.1.

Listing 19.1 – Un exemple de la fonction **eval()**

```
<HTML>
  <HEAD>
    <SCRIPT LANGUAGE="Javascript">
    <!--
      function calcul(f) {
      eval("f.result.value="+f.expr.value+";");
       }
     -->
    </SCRIPT>
  </HEAD>
  <BODY>
    <FORM>
      Entrez une expression:
       <INPUT TYPE="text" NAME="expr" SIZE=15>
       <INPUT TYPE="button" VALUE="Calculer" onClick="calcul(this.form)">
    <BR>
       Resultat:
       <INPUT TYPE="text" NAME="result" SIZE=15>
    </FORM>
  </BODY>
</HTML>
```

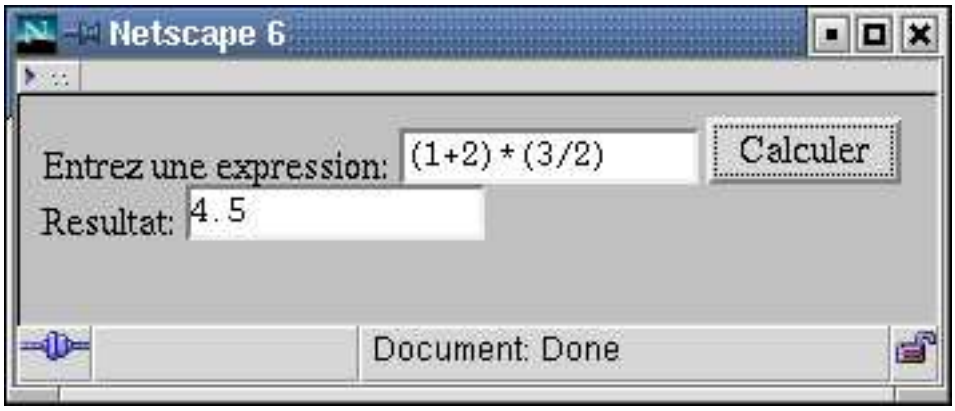


FIG. 19.1 – Un exemple de la fonction **eval()**

19.1.6.5 La fonction isFinite

La fonction **isFinite()** pemet de déterminer si l'argument correspond à un nombre fini. Si c'est le cas, elle renvoie **true**, sinon **false**. La syntaxe est :

```
isFinite(nb);
```

Ainsi, l'expression **isFinite(3/0)** renvoie **false**.

19.1.6.6 La fonction isNaN

La fonction **isNaN()** renvoie **true** si l'argument ne représente pas un nombre, sinon **false**. La forme est :

```
isNaN(param)
```

Si le paramètre est une chaîne de caractères, il regarde si celle-ci représente un nombre valide ou pas, et renvoie respectivement **false** et **true**. Par exemple :

```
a = isNaN("35");     // a=true
a = isNaN("Coucou");     // a=false
```

19.1.6.7 Les fonctions parseInt et parseFloat

Ces fonctions permettent de transformer une chaîne de caractères en un nombre réel (**parseFloat()**) ou entier (**parseInt()**).

La syntaxe de **parseFloat()** est :

```
parseFloat(str);
```

La syntaxe pour **parseInt()** est :

```
parseInt(str [, base]);
```

L'argument *base* permet de spécifier la base utilisée pour la représentation du nombre. Par exemple, en spécifiant 16 pour *base*, la chaîne de caractères peut contenir un nombre sous une représentation héxadécimale.

19.1.6.8 Les fonctions Number et String

Ces fonctions permettent de convertir un objet en un nombre ou une chaîne de caractères.

```
Number(objref);
String(objref);
```

Par exemple, l'exemple suivant met dans la chaîne de caractères *str* "Thu Aug 18 04 :37 :43 GMT-0700 (Pacific Daylight Time) 1983" :

```
D = new Date(430054663215); // nombre de secondes écoulées depuis le 01/01/1970
str = String(D);
```

19.1.6.9 Les fonctions escape and unescape

La fonction **escape()** retourne le code héxadécimal de l'argument pour la table de caractères ISO Latin. La fonction **unescape()** propose l'inverse.

```
escape(string);
unescape(string);
```

19.1.7 Les objets

19.1.7.1 Objets et propriétés

Un objet JavaScript possède plusieurs propriétés qui lui sont associées. Pour accéder aux propriétés d'un objet, on utilise l'une des deux notations :

```
nomObjet.nomPropriété;
nomObjet["nomPropriété"];
```

Il est important de noter que le nom des objets et des propriétés sont sensibles à la casse des caractères.

En plus d'utiliser les objets prédéfinis, il est également possible de créer ses propres objets. Pour instancier un objet, on utilise l'opérateur **new**.

19.1.7.2 Utilisation d'initialisateurs d'objets

Pour créer un objet en utilisant un initialisateur on écrit :

```
nomObjet = {propriété1 :valeur1, propriété2 :valeur2, ..., propriétéN :valeurN};
```

Par exemple, l'instruction suivante crée un objet et l'assigne à la variable voiture si une condition est évaluée à vrai :

```
if (condition)
  voiture = {couleur :"rouge", roues :4, moteur :{cylindres : 4, taille : 2.2}};
```

19.1.7.3 Utilisation d'un constructeur

Pour créer un constructeur, il suffit de déclarer une fonction ayant le même nom que l'objet. Par exemple pour créer un constructeur pour un objet voiture :

```
function voiture(marque, modele, annee) {
  this.marque = marque;
  this.modele = modele;
  this.annee = annee;
}
```

Dès lors, pour créer un objet de ce type, il suffit d'utiliser la fonction **new** :

```
maVoiture = new voiture("VW","Polo","1985");
```

19.1.7.4 Définir des méthodes

Une méthode est une fonction associée à un objet. Pour assigner une méthode à un objet, on commence par écrire une fonction, puis on crée une propriété associée à l'objet et on lui assigne la fonction. Voici un exemple d'une telle fonction :

```
function affichevoiture(){
  document.writeln(this.modele + " est de la marque " + this.marque);
}
```

Rappelons que l'opérateur **this** représente une référence sur l'objet courant. En d'autres termes, lorsque cette fonction sera associée à un objet, il sera possible grâce à cet opérateur d'accéder aux autres propriétés de l'objet. Ensuite, on réécrit le constructeur de l'objet dans lequel on définit une propriété qui fera référence à la fonction :

```
function voiture(marque, modele, annee) {
  this.marque = marque;
  this.modele = modele;
  this.annee = annee;
  this.affiche = affichevoiture;
}
```

Dès lors, on peut appeler la méthode de l'objet par :

```
maVoiture.affiche();
```

19.1.7.5 Détruire des objets

Pour détruire des objets, on utilise l'opérateur delete.

```
delete maVoiture;
```

19.2 Inclusion de code JavaScript dans HTML

Par JavaScript Client-Side, on entend du code JavaScript qui est exécuté par la partie client, c'est-à-dire par le browser de l'utilisateur. Le browser reçoit donc un document HTML dans lequel du code JavaScript a été intégré. Il existe une série de fonctions et d'objets JavaScript qui sont spécifiques à une utilisation du côté client.

Il faut noter qu'il existe différentes évolutions de JavaScript. Celles-ci sont d'ailleurs liées à l'évolution du browser de Netscape, comme le montre le TAB. 19.6.

TAB. 19.6 – Evolution de JavaScript avec Netscape

Version de JavaScript	Version de Netscape
JavaScript 1.0	Navigator 2.0
JavaScript 1.1	Navigator 3.0
JavaScript 1.2	Navigator 4.0–4.05
JavaScript 1.3	Navigator 4.06–4.5

La forme générale est :

```
<SCRIPT LANGUAGE="JavaScript">
  <!---
  Instructions JavaScript
  -->
</SCRIPT>
```

Pour définir du code JavaScript Client-Side, on voit que l'on utilise la balise **<SCRIPT>**. L'attribut **LANGUAGE** doit avoir la valeur «JavaScript» afin de spécifier que le langage utilisé est le JavaScript[2]. On remarque que le code JavaScript lui-même est encapsulé dans des commentaires HTML. Ceci permet à des browsers ne supportant pas le JavaScript de pouvoir traiter le document HTML puique le code JavaScript leur sera caché.

Une autre manière d'utiliser la balise **<SCRIPT>**, mais qui n'est supportée que par certains browsers, est de spécifier via une URL et l'attribut **SRC** le nom d'un fichier contenant le code JavaScript :

```
<SCRIPT LANGUAGE="JavaScript">
   SRC="http ://www.essai.com/scripts/monscript.js">
</SCRIPT>
```

Avec la balise **<NOSCRIPT>**, il y a moyen de spécifier du code HTML qui s'affiche dans le cas où le browser de l'utilisateur n'est pas capable de comprendre le JavaScript.

Remark: *La balise **<SCRIPT>** peut apparaître n'importe où dans le document HTML et il peut y avoir plusieurs balises. Généralement, on définit toutes les fonctions JavaScript qui seront nécessaires dans le document dans une seule balise **<SCRIPT>** se trouvant nichée dans la balise **<HEAD>**.*

On utilise le JavaScript le plus souvent dans deux situations.

1. On veut que le document HTML modifie quelque chose dans son contenu suivant le comportement de l'utilisateur. Par exemple, changer une image lorsque la souris se trouve dessus.
2. On veut vérifier que des données encodées dans un formulaire soient correctement formatées. En effet, cela limite les aller-retours avec le serveur dans le cas de données érronées.

19.3 Gestionnaires d'événements

Un événement correspond la plupart du temps à une action de l'utilisateur. par exemple, cliquer sur un boutton, changer le contenu d'un champ texte ou déplacer la souris sur un lien provoque la création d'un événement.

Pour permettre au script de réagir à ces événements, on définit des gestionnaires d'événement, comme **onChange()** et **onClick()** par exemple. Le TAB. 19.7 reprend la liste des gestionnaires d'événements disponibles dans JavaScript ainsi que les éléments sur lesquels ils agissent.

19.3.1 Définir un gestionnaire d'événements

Pour assigner un gestionnaire d'événements à un élément HTML, il suffit de signaler que chaque gestionnaire d'événements est considéré dans la balise HTML comme un paramètre auquel il suffit d'assigner le code correspondant. La forme générale est donc :

```
<BALISE gestionnaireEvenement="Instruction(s) JavaScript">
```

Par exemple, il existe une fonction *essai* qui ne prend aucun paramètre qui se contente d'afficher une fenêtre avec un petit message. Lorsque l'utilisateur clique sur le bouton, on voudrait que cette fonction soit appellée. On peut voir au listing 19.2 le document HTML correspondant.

19.3.2 Réinitialiser les gestionnaires d'événements

Depuis la version 1.1 de JavaScript, il est possible de réinitialiser les gestionnaires d'événements comme le montre le listing 19.3.

Remark: *Dans JavaScript 1.1 et les versions précédentes, le nom des gestionnaires doit être entièrement en minuscules, par exemple monbouton.**onclick()** ou monbouton.**onsubmit()**.*

[2]Microsoft a en effet introduit un autre langage, VBScript, directement dérivé du Visual Basic. Ceci dit, actuellement, à part Microsoft Internet Explorer, aucun autre browser ne supporte ce langage.

Evénement	Eléments consernés	Situation	Gestionnaire
Abort	Les images.	L'utilisateur interrompt le chargement d'une image.	onAbort
Blur	Les éléments des fenêtres et des formulaires.	L'utilisateur enlève le focus d'un élément.	onBlur
Change	Les zone de saisie de textes et les listes de choix.	L'utilisateur change la valeur d'un élément.	onChange
Click	Les boutons, boutons radio, les cases à cocher, les images et les liens.	L'utilisateur clique sur un élément.	onClick
DragDrop	Les fenêtres.	L'utilisateur glisse un objet sur le browser.	onDragDrop
Error	Les images et les fenêtres.	Une erreur est intervenue lors du chargement.	onError
Focus	Les éléments des fenêtres et des formulaires.	L'utilisateur donne le focus à un élément.	onFocus
KeyDown	Les documents, les images, les liens et les zones de saisie de texte.	L'utilisateur enfonce une touche.	onKeyDown
KeyPress	Les documents, les images, les liens et les zones de saisie de texte.	L'utilisateur tient une touche enfoncée.	onKeyPress
KeyUp	Les documents, les images, les liens et les zones de saisie de texte.	L'utilisateur relâche une touche.	onKeyUp
Load	Le corps du document.	L'utilisateur charge une page dans le navigateur.	onLoad
MouseDown	Les documents, les boutons et les liens.	L'utilisateur enfonce l'un des boutons de la souris.	onMouseDown
MouseMove	Rien par défaut.	L'utilisateur déplace la souris.	onMouseMove
MouseOut	Les aires et les liens.	L'utilisateur quitte une zone avec la souris.	onMouseOut
MouseOver	Les liens.	L'utilisateur se trouve avec la souris sur un lien.	onMouseOver
MouseUp	Les documents, les boutons et les liens.	L'utilisateur relâche l'un des boutons de la souris.	onMouseUp
Move	Les fenêtres.	L'utilisateur ou un script bouge une fenêtre.	onMove
Reset	Les formulaires	L'utilisateur clique sur le bouton « reset ».	onReset
Resize	Les fenêtres	L'utilisateur ou un script change la taille d'une fenêtre.	onResize
Select	Les zones de saisie de texte.	L'utilisateur sélectionne une portion de texte.	onSelect
Submit	Les formulaires.	L'utilisateur clique sur le bouton « submit ».	onSubmit
Unload	Le corps du document.	L'utilisateur quite la page.	onUnload

TAB. 19.7 – Les gestionnaires d'événements

19.3.3 Capture d'événements

Depuis JavaScript 1.2, il est possible de capturer les événements. Pour cela, JavaScript propose les méthodes suivantes aux objets **window**, **document** et **layer** :

captureEvents() Capture certains types d'événements.

releaseEvents() Ignore la capture de certains événements.

routeEvent() Assigne l'événement capturé à l'objet correspondant.

handleEvent() Traite les événements capturés (il ne s'agit pas pas d'une méthode de **layer**).

Supposons que l'on veuille capturer tous les clicks d'une fenêtre. Pour cela, il faut suivre les étapes suivantes :

1. Activation de la capture d'événements.
2. Définir un gestionnaire d'événements.
3. Enregistrer le gestionnaire.

Les étapes vont être étudiées en détail.

19.3.3.1 Activation de la capture d'événements

Pour capturer tous les événements clicks, il faut écrire :

Listing 19.2 – Un exemple de gestionnaire d'événements

```
<!DOCTYPE HTML PUBLIC "~//W3C//DTD HTML 4.01//en"
  "http://www.w3c.org/TR/html4/strict.dtd">
<HTML>
  <HEADER>
    <TITLE> Exemple de JavaScript </TITLE>
    <SCRIPT LANGUAGE="JavaScript">
    <!--
      function Coucou()
      {
        alert("Premier programme en JavaScript");
      }
    // -->
    </SCRIPT>
  </HEADER>
  <BODY>
    <H1> Exemple de JavaScript </H1>
    <FORM>
      <INPUT TYPE="button" VALUE="Appuyer" onclick="Coucou()">
    </FORM>
  </BODY>
</HTML>
```

Listing 19.3 – Réinitialiser un gestionnaire d'événements

```
<SCRIPT LANGUAGE="JavaScript">
<!--
  function essai1() {
    ...
  }
  function essai2() {
    ...
  }
-->

<FORM NAME="monFormulaire">
  <INPUT TYPE="button" VALUE="Calculer" NAME="monBouton" onClick="essai1();">
</FORM>

<SCRIPT>
  document.monFormulaire.monBouton.onClick=essai2();
</SCRIPT>
```

```
window.captureEvents(Event.CLICK);
```

Si on veut capturer plusieurs événements, on les sépare par un trait vertical (|), comme par exemple :

```
window.captureEvents(Event.CLICK | Event.MOUSEDOWN | Event.MOUSEUP);
```

La liste de tous les événements est reprise dans [6].

19.3.3.2 Définir un gestionnaire d'événements

Pour définir un gestionnaire d'événements, il suffit de définir une fonction qui prend un seul argument du type **event**. Par exemple :

```
function gestionClick(e) {
  // Le code ici.
}
```

Plusieurs possibilités se présentent :

- La fonction retourne **true**. Dans ce cas, l'événement est entièrement traité par le document. Aucun objet, tel un bouton, ne traitera l'événement.
- La fonction retourne **false**. L'événement est complètement ignoré. Cela permet par exemple de rendre le bouton droit de la souris inactif dans un document HTML.
- La fonction appelle **routeEvent**(). JavaScript tente d'envoyer l'événement à un autre gestionnaire d'événements défini par l'utilisateur, par exemple celui du document. Si il ne trouve pas un tel gestionnaire, il appelle celui de l'objet concerné par l'événement, comme un bouton.

– La fonction appelle **handleEvent()** d'un objet particulier du document, qui le traitera avec son gestionnaire d'événements.

19.3.3.3 Enregistrer le gestionnaire

Pour enregistrer le gestionnaire, il suffit de l'assigner comme un gestionnaire d'événement normal :

```
window.onClick = gestionClick;
```

19.4 Utiliser les objets de navigation

Lorsque l'utilisateur charge un document dans un navigateur, un certain nombre d'objets sont créés. En fait, chacun des objets correspond à des entités reprises dans le document HTML. L'objectif étant d'avoir une structure d'objets correspondant à la hiérarchie du document. La FIG. 19.2 reprend la hiérarchie des objets définis par JavaScript.

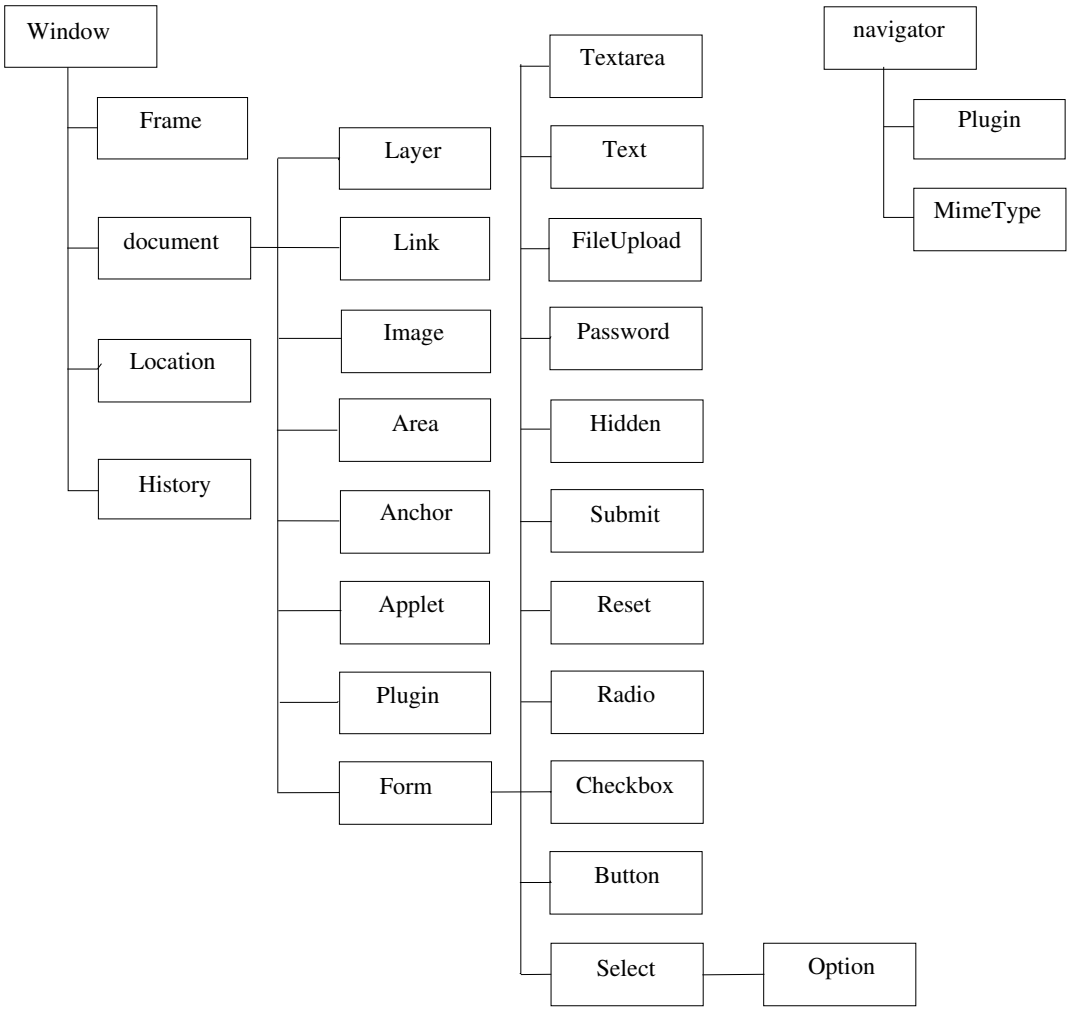


FIG. 19.2 – Hiérarchie d'objets du navigateur

JavaScript utilise comme convention que le noms des classes commencent par une majuscule, alors que le nom des variables commencent pas une minuscule. Ainsi, **Window** représente la classe des fenêtres avec ses propriétés et ses méthodes, alors que **window** représente une variable contenant une référence sur

la fenêtre courante du browser. Il est évident, même si cela peut entraîner des confusions, que la variable **window** est de la classe **Window**.

19.4.1 Interaction entre JavaScript et HTML

Chaque entité HTML est automatiquement transformée en un objet du même type et considéré comme une propriété de l'objet encapsulant le précédent. Par exemple, supposons que nous ayons le code HTML suivant :

```
<FORM NAME="monFormulaire">
  <INPUT TYPE="text" NAME="nomUser" SIZE=20>
  <INPUT TYPE="text" NAME="ageUser" SIZE=3>
</FORM>
```

Dès lors, pour accéder aux différents éléments, on écrit :

```
<SCRIPT>
  document.write(document.monFormulaire.nomUser.value);
  document.write(document.monFormulaire.ageUser.value);
</SCRIPT>
```

19.4.2 Les objets importants

19.4.2.1 Les objets window et Frame

L'objet **window** est le parent de tous les autres objets du navigateur. L'objet **Frame** quant à lui représente un élément défini par la balise **<FRAME>** du HTML. Parmi les nombreuses méthodes proposées par JavaScript pour l'objet **window**, signalons les suivantes :

open() et **close()** Ouvre ou ferme une fenêtre de browser.

alert() Affiche une boite de dialogue avec un message.

confirm() Affiche une boite de dialogue de confirmation avec un bouton «OK» et «Cancel».

prompt() Affiche une boite de dialogue permettant d'introduire une valeur dans un champ texte.

blur() et **focus()** Enlève ou donne le focus à la fenêtre.

scrollTo() Déroule le contenu de la fenêtre jusqu'à une certaine position.

setInterval() Evalue une expression ou appelle une fonction à des périodes spécifiques.

setTimeout() Evalue une expression ou appelle une fonction dès que le temps indiqué est écoulé.

Parmi les propriétés de l'objet **window**, signalons **location** qui contient l'URL actuellement affichée dans la fenêtre. Par exemple, pour rediriger le browser vers la page d'accueil de l'ULB, on écrit le code suivant :

```
location = "http ://www.ulb.ac.be"
```

19.4.2.2 L'objet document

Chaque page possède un objet du type **document**. Les deux méthodes les plus utilisées sont **write()** et **writeln()** qui permettent de générer du code HTML.

De plus, l'objet **document** possède une série de propriétés représentant les différentes couleurs. De même, la propriété **cookie** permet de travailler avec les cookies.

19.4.2.3 L'objet Form

Chaque formulaire d'un document crée un objet du type **Form**. Tous les objets de ce type sont stockés dans un tableau appellé **forms**. Le premier formulaire correspond à l'indice à 0, le second à l'indice 1 et ainsi de suite. De même, les différents éléments d'un formulaire (les boutons, les zones de saisie de texte, etc.) sont stockés dans un tableau du formulaire appellé **elements**. Ainsi, le premier élément du premier formulaire peut être accéder par :

```
document.forms[0].elements[0]
```

De plus, cet objet définit deux méthodes **submit()** et **reset()** qui permettent respectivement de soumettre le formulaire ou de réinitialiser ses éléments avec leur valeur par défaut.

Chaque élément du formulaire comprend une propriété **form** qui est une référence au document auquel il appartient.

19.4.2.4 L'objet location

L'objet **location** possède des propriétés qui sont basées sur l'URL courante. Par exemple, la propriété **hostname** contient des informations sur le serveur web. L'objet **location** possède également deux méthodes :

reload() Force le document a être rechargé.

replace() Remplace l'entrée dans l'historique courante par l'URL spécifiée.

19.4.2.5 L'objet history

L'objet **history** contient une liste de chaînes de caractères représentant les URLs visitées par le client. En utilisant les méthodes **current**(), **next()** et **previous()** il est possible de naviguer dans l'historique. De plus, la méthode **go()** permet de se déplacer par saut par rapport à l'entrée courante. Ainsi, pour revenir deux pages avant, il faut utiliser :

```
history.go(-2)
```

De même, le code suivant recharge la page courante :

```
history.go(0)
```

19.4.2.6 L'objet navigator

L'objet **navigator** contient des informations sur la version du browser utilisé, comme son nom et sa version. De plus, il possède trois méthodes :

javaEnabled() Spécifie si le Java est disponible ou pas.

preference() Permet de travailler avec des préférences d'utilisateurs.

taintEnabled() Spécifie si les données « tainted » sont disponibles ou pas.

19.4.3 Les tableaux d'objets

La plupart des objets possèdent des propriétés qui sont en fait des tableaux d'autres objets. Le TAB. 19.8 reprend l'ensemble de ces propriétés.

19.5 Utiliser les fenêtres et les cadres

JavaScript permet de manipuler les fenêtres, avec l'objet **window**, et les cadres, avec l'objet **Frame,** résultant du code HTML. Cette section va étudier les deux objets correspondants.

19.5.1 Ouvrir et fermer des fenêtres

Une fenêtre est automatiquement créée lorsque le navigateur est ouvert. En plus des menus permettant d'ouvrir et de fermer des fenêtres, on peut également utiliser des fonctions JavaScript.

TAB. 19.8 – Les tableaux d'objets

Objet	Propriété	Description
document	**anchors**	Représente les balises **<A>** qui contiennent un attribut **NAME**.
	applets	Représente les balises **<APPLET>**.
	embeds	Représente les balises **<EMBED>**.
	forms	Représente les balises **<FORM>**.
	images	Représente les balises **<IMG>**. Les images créées avec le constructeur **Image**() ne sont pas inclus dans ce tableau.
	layers	Représente les balises **<LAYER>** et **<ILAYER>**.
	links	Représente les balises **<AREA HREF=**"..."**>** ainsi que les objets **Link** créé avec la méthode **link**.
Form	**elements**	Représente les différents éléments du formulaire.
Function	**arguments**	Représente les arguments de la fonction.
navigator	**mimeTypes**	Représente tous les type MIME supportés par le client.
	plugins	Représente tous les plugins installés sur le client.
select	**options**	Représente les options d'un objet **Select** (Balises **<OPTION>**).
window	**frames**	Représente toutes les balises **<FRAME>** et **<FRAMESET>**
	history	Représente l'historique de la fenêtre.

19.5.1.1 Ouvrir une fenêtre

Pour ouvrir une fenêtre, on utilise la méthode **open()**. La forme de cette méthode est :

```
window.open(URL [, target [, attributes]]);
```

Les différents paramètres ont la signification suivante :

URL Permet de spécifier quelle URL sera chargée dans la nouvelle fenêtre.

target Permet d'associer un nom qui pourra être utilisé avec l'attribut **TARGET**.

attributes Permet de spécifier des attributs, comme les dimensions de la fenêtre, la visibilité des menus, la visibilité des barres de défilement, etc.

Par exemple :

```
maFenetre = window.open("mapage.html","infoFenetre","toolbar=no,scrollbars=yes")
```

La méthode renvoie une référence à un objet du type **window**. Il n'est pas obligatoire de récupérer ce paramètre, même si cela est nécessaire pour ensuite y accéder, par exemple pour interagir avec son contenu.

19.5.1.2 Fermer une fenêtre

Pour fermer une fenêtre, on utilise la méthode **close()**. Pour fermer la fenêtre créée dans l'exemple précédent, on utilise :

```
meFenetre.close()
```

19.5.2 Utiliser les cadres

L'objet **frameset** est un type spécial de fenêtre, car il peut posséder plusieurs zones indépendantes à l'écran, les cadres, stockées dans le tableau **frames**.

19.5.2.1 Créer des cadres

L'exemple suivant crée un ensemble de trois cadres :

```
<FRAMESET ROWS="90%, 10%">
  <FRAMESET COLS="30%, 70%">
    <FRAME SRC="categorie.html" NAME="listeCadre">
    <FRAME SRC="titre.html" NAME="contenuCadre">
  </FRAMESET>
  <FRAME SRC="navigation.html" NAME="navigationCadre">
</FRAMESET>
```

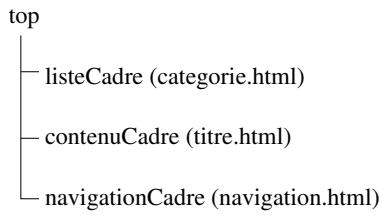


FIG. 19.3 – Exemple de cadres

On peut voir à la FIG. 19.3, la hiérarchie correspondante. Pour passer d'un cadre à l'autre, on peut utiliser le tableau **frames** :
- *listeCadre* est **top**.**frames**[0] ;
- *contenuCadre* est **top**.**frames**[1] ;
- *navigationCadre* est **top**.**frames**[2].

19.5.2.2 Mise à jour des frames

Pour changer le contenu d'un cadre, il suffit de changer sa propriété **location**. Pour reprendre l'exemple introduit précédemment, supposons que le cadre *contenuCadre* est un bouton qui, lors d'un clic met à jour le contenu du cadre *listeCadre*. On pourrait le définir par :

```
<INPUT TYPE="button" VALUE="Changer"
   onClick="top.frames[0].location='nouveau.html' ">
```

19.5.3 Références aux fenêtres et aux cadres

19.5.3.1 Référence pour accéder aux propriétés et aux méthodes

Pour utiliser une référence à une fenêtre ou à un cadre, afin de manipuler des propriétés ou d'appeller une méthode trois techniques sont disponibles :
- **self** ou **window** sont des synonymes pour la fenêtre courante. Par exemple, pour fermer la fenêtre courante, on peut utiliser la forme **self**.**close**() ou **window**.**close**().
- **top** ou **parent** sont des synonymes pour des noms de fenêtres. **top** est une référence pour la fenêtre de navigation la plus haute dans la hiérarchie. **parent** peut être utilisé avec les cadres et est une référence de l'objet **frameset** qui contient le cadre.
- Le nom de la fenêtre est également une variable. Par exemple, *maFenetre*.**close**() ferme la fenêtre dont la balise **NAME** était «*maFenetre*».
- Lorsque le nom de la fenêtre est omise, les propriétés référées ou les méthodes appellées sont celles de la fenêtre courante. Ainsi, l'appel **close**() est identique à **window**.**close**().

19.5.3.2 Référence pour accéder un lien hypertexte

Pour accéder à une fenêtre depuis un lien, on utilise le paramètre **TARGET** en lui associant le nom de la fenêtre. Par exemple :

```
<A HREF="nouveau.html" TARGET="maFenetre"> Afficher </A>
```

19.6 Fonctionnalités additionnelles

Le JavaScript introduit quelques fonctionnalités additionnelles comme :
- les URLs JavaScript ;
- L'utilisation de la barre de statut ;
- L'utilisation des cookies.

19.6.1 Les URLs JavaScript

Parmi les différents types d'URLs souvent rencontrés, on retrouve **http :**, **ftp :**, **file :** ou encore **mailto :**. Avec les navigateurs supportant le JavaScript, il est également possible d'utiliser le type d'URL **javascript :**. Par exemple, le lien suivant permet de recharger la page courante :

```
<A HREF="javascript :history.go(0)"> Recharger ! </A>
```

En fait, derrière le type **javascript :**, vous pouvez utiliser n'importe quelle instruction JavaScript, en particulier un appelle à une fonction qui aurait été définie dans le document.

19.6.2 Utiliser la barre de statut

L'objet **window** possède deux propriétés importantes :

defaultStatus Permet de spécifier le message par défaut qui doit être affiché dans la barre de statut.

status Permet de spécifier un message transitoire dans la barre de statut, par exemple lorsqu'un utilisateur se trouve sur un lien.

Par exemple, l'exemple suivant affiche un message « Afficher le contenu » lorsque l'utilisateur se trouve sur le lien :

```
<A HREF="contenu.html"
  onMouseOver=" window.status='Afficher le contenu'; return true ">
  Contenu.
</A>
```

19.6.3 Les cookies

Les cookies sont un mécanisme permettant de stocker sur la machine client des informations permanentes, comme par exemple la langue que l'utilisateur utilise. Chaque cookie est un ensemble d'informations, associé éventuellement avec une date d'expiration, ayant la forme suivante :

```
nom=valeur;expires=expDate;
```

nom est le nom de l'information, alors que *valeur* est la valeur de celle-ci. La seconde partie est optionnelle et permet d'ajouter une date de validation. La forme de *expDate* doit être la suivante :

```
Wdy, DD-Mon-YY HH :MM :SS GMT
```

Lorsqu'il n'y pas de date d'expiration, le cookie est perdu dès que l'utilisateur quitte son navigateur. Dans le cas contraire, le navigateur stocke la valeur du cookie sur la machine client uniquement si la date d'expiration n'est pas antérieure à la date actuelle sur la machine client.

19.6.3.1 Limitations

Les cookies ont des limitations :
- au plus 300 cookies peuvent être stockés sur la machine client ;
- la longueur maximale du cookie, son nom ainsi que sa valeur, est de 4 Koctets ;
- au plus 20 cookies peuvent être stockés sur un domaine.

19.6.3.2 utilisation des cookies avec JavaScript

L'objet **document** possède une propriété **cookie** : une chaîne de caractères contenant la liste de tous les cookies, noms et valeurs, du navigateur. Deux opérations peuvent être effectuées à l'aide des cookies :
- associer une valeur à un nom de cookie, en ajoutant éventuellement une date d'expiration ;
- récupérer la valeur d'un cookie en connaissant son nom.

La fonction suivante permet par exemple d'ajouter un cookie :

```
function associeCookie(nom, valeur, expiration) {
  document.cookie = nom + "=" + escape(valeur) +
    ((expiration==NULL)? "" : ("; expires=" + expiration.toGMTString()))
}
```

On remarque l'utilisation de la function **escape()** afin de permettre à la valeur de pouvoir contenir des caractères spéciaux, comme par exemple des points ou des points virgule. La fonction **toGMTString()**, comme son nom l'indique, permet de transformer une chaine de caractères représentant une date au format obligatoire pour les cookies.

La fonction suivante retourne la valeur d'un cookie en utilisant le nom de ce dernier :

```
function valeurCookie(nom) {
  var Recherche = nom + "=";
  if(document.cookie.length > 0) { // Y a-t-il des cookies?
    offset = document.cookie.indexOf(Recherche); // On cherche la chaine "nom="
    if(offset!=-1) { // Le cookie existe?
      // Position de début du cookie
      offset += Recherche.length;
      // Position de fin du cookie
      end = document.cookie.indexOf(";", offset);
      if(end==-1)
        end = document.cookie.length;
      return unescape(document.cookie.substring(offset,end));
    }
  }
}
```

19.7 JavaScript Server-Side

Par JavaScript Server-Side, on entend du code JavaScript qui sera exécuté par le serveur web, pour autant que celui-ci soit capable de l'exécuter. Comme on peut le voir au listing 19.4, du code JavaScript Server-Side est défini par la balise **<SERVER>**.

Listing 19.4 – Un exemple de JavaScript Server-Side

```
<!DOCTYPE HTML PUBLIC "~//W3C//DTD HTML 4.01//en"
2   "http://www.w3c.org/TR/html4/strict.dtd">
<HTML>
4   <HEAD>
      <TITLE> Exemple de JavaScript 2 </TITLE> 2
6   </HEAD>
    <BODY>
8     <H1> Exemple de JavaScript 2 </H1> 2
      <SERVER>
10      document.write("Last Updated on: "+document.lastModified);
      </SERVER>
12  </BODY>
</HTML>
```

Lorsque le browser demande le document HTML, le serveur web détecte qu'il contient du code JavaScript Server-Side, l'exécute de sorte que le document qui est envoyé au browser ne contienne plus le code JavaScript mais le résultat de celui-ci.

JavaScript Server-Side offre une série de fonctions et d'objets qui n'existent pas pour le JavaScript Client-Side, notamment des méthodes pour pouvoir facilement manipuler des bases de données, ce qui est nécessaire pour la gestion de sites dynamiques. En effet, le rôle du JavaScript Server-Side est de donner un moyen d'agir sur le contenu qui est envoyé aux différents clients, alors que le rôle du JavaScript Client-Side est de fournir des moyens d'agir au niveau du browser.

L'avantage de l'utilisation de JavaScript Server-Side par rapport à l'utilisation de l'interface CGI (présentée au chapitre **??**), est qu'il est plus facile à utiliser et entièrement portable d'un système d'exploitation

à un autre pour autant que le serveur web puisse l'interpréter. L'inconvénient principal est que ses possibilités sont limitées.

Un concepteur qui connnaîtrait déjà certains langages de programmation, comme le C/C++ ou Perl, aurait donc intérêt à les utiliser en combinaison avec l'interface CGI. Par contre, un concepteur avec très peu d'expérience en matière de programmation pourrait adopter le JavaScript Server-Side.

Contrairement à JavaScript Client-Side, le JavaScript Server-Side ne fait pas l'objet d'une normalisation. En fait, très peu de serveurs web supportent cette extension. Depuis, le langage PHP (référez-vous au chapitre 22) s'est imposé dans cette catégorie et est disponible sur tous les types de serveurs. Pour plus d'informations concernant le JavaScript Server-Side, le lecteur est invité à consulter [17].

Chapitre 20

Java

Ce chapitre introduit le langage de programmation orienté objet Java. Pour une introduction générale à la programmation orientée objets, le lecteur se réfèrera à l'annexe A.

20.1 Introduction à Java

20.1.1 Qu'est que le Java ?

Le Java est un langage orienté objet portable développé initialement par James Gosling de la société Sun. Il voulait utiliser un langage de programmation orienté objet mais il trouvait que le C++ était beaucoup trop compliqué à maîtriser. En fait, le langage Java dans sa version actuelle a plusieurs influences :

- la concurence comme dans Mesa (un langage de recherche développé par Xerox) ;
- les exceptions comme dans Modula-3 ;
- l'édition dynamique comme dans Lisp ;
- les définitions d'interfaces comme dans Objective C ;
- la gestion mémoire automatique comme dans Lisp ;
- les instructions ordinaires du C/C++.

De plus, des librairies d'objets orientées vers l'application au domaine d'Internet sont incluses dans la définition du langage. D'autre part, utilise le codage de caractères Unicode sur 16 bits plutôt que sur 8 bits comme ASCII.

20.1.2 Portabilité

Par « portabilité », on entend que le langage Java a été développé avec Internet en toile de fond. Il s'agissait donc de concevoir un langage pour une utilisation dans des systèmes hétérogènes dont les librairies graphiques et les caractéristiques des réseaux étaient différentes. Java propose des possibilités identiques pour l'ensemble des systèmes. En fait, Java inclut dans sa spécification des caractéristiques qui sont généralement laissées au soin des compilateurs, comme par exemple la taille des entiers.

Java propose la même Application Programmer Interface (API), c'est-à-dire le même ensemble de librairies, de fonctions et d'objets, quel que soit le système, ce qui veut dire que le code source d'une application Java est identique pour l'ensemble des systèmes, même au niveau graphique, ce qu'aucun autre langage ne propose en standard.

Pour des raisons d'efficacité, les environnements Java supportent des appels natifs, c'est-à-dire l'exécution d'une portion de code écrite dans d'autres langages comme le C/C++. Par exemple, les opérations d'entrée-sortie Java font appel aux fonctions équivalentes de chaque plate-forme. Les appels natifs permettent d'utiliser du code existant ou d'effectuer certaines tâches plus rapidement. Il est évident que l'utilisation d'appels natifs sort du cadre du langage Java proprement dit et nuit à la portabilité de l'application Java finale.

20.1.3 Indépendance vis-à-vis de l'architecture

Si le langage Java était uniquement portable, cela signifirait que pour créer un programme pour Macintosh à partir d'un programme conçu pour Windows, il suffirait de reprendre le code source pour le second système d'exploitation et de le recompiler pour le premier. Par « indépendance vis-à-vis de l'architecture », on entend que le code binaire d'un programme Java est identique quel que soit le système sur lequel il est exécuté.

En fait, le compilateur Java ne génère pas directement du code machine pouvant être exécuté par un processeur particulier, mais génère plutôt un code binaire intermédiaire entre sa forme « source » et sa forme « code ». Un autre programme va ensuite lire le code binaire Java et le transformer en code machine pour le système qui exécute le programme compilé. Pour porter le système Java sur une nouvelle plateforme informatique, il suffit d'écrire un interpréteur qui supporte le «Java Virtual Machine», c'est-à-dire un programme capable de transformer le code binaire Java en code machine. Pour une description complète de la « Java Virtual Machine », le lecteur est invité à consulter [5].

Le langage Java permet donc d'écrire des programmes complexes sans se soucier du système qui va l'exécuter. Par rapport à des langages compilés, tels le C/C++ ou le Pascal, un programme écrit en Java est portable. Il est cependant plus lent à exécuter à cause de l'interpréteur Java. Toutefois, le programme sera exécuté plus rapidement qu'avec des langages de scripts tels que Perl ou Python car une phase de compilation précède l'interprétation. Notons également que comme le programme est distribué sous forme binaire, il est quasiment impossible d'en déduire les algorithmes utilisés.

20.1.4 Les applications et les applets

Il existe deux manières différentes d'exécuter une application Java :
- sous la forme d'un programme indépendant exécuté à partir de la ligne de commande, on parle d'une « application » ;
- sous la forme d'un programme inclus dans une page web qui sera exécuté lorsque celle-ci sera traitée par les browsers, on parle alors d'une « applet ».

L'idée originale était d'utiliser le Java uniquement pour des petites applets, mais de plus en plus de programmes importants sont développés en Java[1]. En effet, rien dans le langage Java ne limite la taille des programmes.

Les applets et les programmes diffèrent par les privilèges d'exécution qu'ils possèdent, ainsi que par la manière dont ils spécifient l'endroit où l'application démarre :
- les applications déclarent une fonction **main**, comme dans le langage C/C++, qui définit le début du programme ;
- les applets font appel à un objet qui dérive directement de l'objet **applet**, qui définit le début du programme.

20.1.5 La sécurité et les applets

L'objectif des applets étant de permettre l'exécution d'une application Java en utilisant Internet, l'utilisateur doit être certain que son interpréteur fournisse un minimum de protections, puisqu'il lui est impossible de savoir si le programme binaire Java fait bien ce qu'il prétend faire.

Donc, si aucune précaution n'existait, il serait possible pour une applet de détruire des fichiers importants ou encore de lire des messages confidentiels. Afijn d'assurer cette sécurité, les applets s'exécutent dans un environnement ne possédant que des droits restreints :
- il est pas possible de lire ou d'écrire dans les fichiers de l'utilisateur (par défaut) ;
- aucune connexion vers un serveur autre que celui dont vient l'applet ne peut être ouverte ;
- il est impossible d'exécuter une applet sur un système en utilisant l'équivalent de la fonction **system()** en C ;
- il n'est pas possible de charger une librairie.

Certains navigateurs proposent d'autres types de sécurités supplémentaires en sus de celles évoquées.

[1] La société Corel avait développé une série d'outils bureautique entièrement en Java.

20.2 Éléments de Java

Dans cette partie, nous allons étudier les principaux éléments de Java. Leur description complète, est disponible dans [7], [5] et [19].

20.2.1 Les identificateurs

Un identificateur est un nom utilisé par le programmeur pour référencer une entité dans Java telle qu'une variable, un objet ou une fonction. Contrairement à d'autres langages de programmation, il n'y a aucune restriction quant à la longueur des identificateurs dans Java. Un identificateur doit commencer par une lettre, un underscore (_) ou le signe dollar ($), et peut également contenir des chiffres.

Java étant développé avec l'intention de pouvoir l'utiliser dans un cadre international, il se base sur le codage Unicode permettant donc d'utiliser une grande gamme de lettres, en particulier les lettres accentuées. Ainsi, les identificateurs suivants sont valides en Java :

```
i
essai
$_99
vélos
```

20.2.2 Les types de base

Java propose des types de base dont l'ensemble des caractéristiques est indépendant du système :

boolean Type booléen codé sur un 1 bit et pouvant prendre les valeurs **false** et **true**.

char Un caractère codé sur 16 bits et contenant un code Unicode.

byte Un entier signé sur 8 bits allant de -128 à 127.

short Un entier signé sur 16 bits allant de $-32\,768$ à 32767.

int Un entier signé sur 32 bits allant de $-2\,147\,483\,648$ à $2\,147\,483\,647$.

long Un entier signé sur 64 bits allant de $-9\,223\,372\,036\,854\,775\,808$ à $9\,223\,372\,036\,854\,775\,807$

float Un nombre en virgule flottante utilisant le codage IEEE 754 sur 32 bits allant de -3,4E38 à +3.4E38.

double Un nombre en virgule flottante utilisant le codage IEEE 754 sur 64 bits allant de -1.7E308 à +1.7E308.

void Pas de type.

Pour utiliser une variable du type de base, il suffit de la déclarer :

```
int i;
```

Pour déclarer une fonction ne renvoyant aucune valeur, ce que certains autres langages comme le Pascal appellent une procédure, on déclare :

```
void RenvoiRien(int i) {}
```

Remark: *Le langage C++ dont dérive Java permet d'utiliser le type **void** pour expliciter qu'une fonction n'utilise pas de paramètres. En Java, cette spécification est interdite.*

20.2.3 Les expression et les opérateurs

Une expression est un ensemble valide de variables, de valeurs d'opérateurs et d'autres expressions pouvant être évaluées sous la forme d'une valeur. La plupart des opérateurs en Java sont les mêmes que pour le langage C/C++. On peut les diviser en plusieurs catégories :

- les opérateurs d'assignation ;
- les opérateurs de comparaison ;
- Les opérateurs arithmétiques ;

- les opérateurs binaires ;
- les opérateurs logiques ;
- les opérateurs sur les chaînes de caractères ;
- les opérateurs spéciaux.

20.2.3.1 Les opérateurs d'assignation

L'opérateur d'assignation de base est le signe égal (=). En écrivant $x = y$, on assigne la valeur de y à x. Il existe une liste d'autres opérateurs d'assignation inspirée du C qui sont repris dans le Tab. 20.1.

TAB. 20.1 – Opérateurs d'assignation

Opérateur	Signification
$x+ = y$ & $x = x + y$ \\ $x- = y$ & $x = x - y$ \\ $x* = y$ & $x = x * y$ \\ $x/ = y$ & $x = x/y$ \\ $x\% = y$ & $x = x\%y$ \\ $x <<= y$ & $x = x << y$\\ $x >>= y$ & $x = x >> y$ \\ $x\& = y$ & $x = x\&y$ \\ $x\wedge = y$ & $x = x \wedge y$ \\ $x\| = y$ & $x = x\|y$ \\ \hline	

20.2.3.2 Les opérateurs de comparaison

Les opérateurs de comparaison compare les deux opérandes et retourne une valeur booléenne suivant que la comparaison est **true** ou non. Le TAB. 20.2 reprend les différents opérateurs.

TAB. 20.2 – Les opérateurs de comparaison

Opérateur	Description
Egalité (==)	Retourne **true** si les deux opérandes sont égales.
Non-égalité (!=)	Retourne **true** si les deux opérandes ne sont pas égales.
Plus grand que (>)	Retourne **true** si l'opérande à gauche de l'opérateur est plus grande que celle à droite.
Plus grand ou égal (>=)	Retourne **true** si l'opérande à gauche de l'opérateur est plus grande que ou égal à celle à droite.
Plus petit que (<)	Retourne **true** si l'opérande à gauche de l'opérateur est plus petite que celle à droite.
Plus petit ou égal (<=)	Retourne **true** si l'opérande à gauche de l'opérateur est plus petite que ou égale à celle à droite.
Comparaison de type (instanceof)	Retourne **true** si l'objet est l'instance du type de classe passé en paramètre.

20.2.3.3 Les opérateurs arithmétiques

Les opérateurs arithmétiques ne fonctionnent qu'avec des opérandes numériques. Les opérateurs standards sont l'addition (**+**), la soustraction (−), la multiplication (*****) et la division (**/**). La division renvoie toujours un nombre en virgule flottante même si les deux opérandes sont des entiers.

En plus de ces opérateurs, Java proposent les opérateurs présentés au TAB. 20.3.

TAB. 20.3 – Les opérateurs arithmétiques

Opérateur	Description	Exemple
Modulo (%)	Retourne le reste d'une division.	12 % 5 retoune 2.
Incrémentation (++)	Ajoute un à une opérande. Il peut être utilisé en préfixe (++x), auquel cas il retourne la valeur de x après incrémentation, ou en postfixe (x++), auquel cas il retourne la valeur de x avant incrémentation.	Si x vaut 3, ++x mettra x à 4 et retournera 4, alors que x++ mettra x à 4 et retournera 3.
Décrémentation (−−)	Enlève un à une opérande. Il peut être utilisé en préfixe (−−x), auquel cas il retourne la valeur de x après décrémentation, ou en postfixe (x−−), auquel cas il retourne la valeur de x avant décrémentation.	Si x vaut 3, −−x mettra x à 2 et retournera 2, alors que x−− mettra x à 2 et retournera 3.
Négation (−)	Retourne la négation de l'opérant.	Si x vaut 3, −x retournera −3.

20.2.3.4 Les opérateurs binaires

Le but de ces opérateurs est de pouvoir travailler sur des entiers codés sur 32 bits. Ces opérateurs sont exactement les mêmes que dans le langage C. Comme ils ne sont pour ainsi dire plus utilisés aujourd'hui, il ne seront pas étudiés ici.

20.2.3.5 Les opérateurs logiques

Les opérateurs logiques permettent de combiner plusieurs expressions utilisant les différents opérateurs de comparaison. Ils sont repris au TAB. 20.4.

TAB. 20.4 – Les opérateurs logiques.

Opérateur	Utilisation	Description
« Et » logique (&&)	expr1 && expr2	Retourne **true** si les deux expressions sont évaluées **true**.
« Ou » Logique (‖)	expr1 ‖ expr2	Retourne **true** si une des deux expressions est évaluée à **true**.
« Non » Logique (!)	!expr	Retourne faux si l'expression est évaluée à **true**, sinon elle retourne **true**.

20.2.3.6 Les opérateurs sur les chaînes de caractères

En plus des opérations de comparaison, il existe également un opérateur de concaténation (**+** ou **+=**). Ainsi, l'expression "my " **+** "string" retourne "my string".

20.2.3.7 Les opérateurs spéciaux

Java prévoit les opérateurs spéciaux suivants :

- opérateur conditionnel ;
- opérateur virgule ;
- **delete** ;
- **new** ;
- **this**.

20.2.3.7.1 L'opérateur conditionnel L'opérateur conditionnel utilise trois opérandes et fonctionne de la manière suivante :

```
condition? val1 : val2;
```

Si la condition est vrai, c'est la valeur *val1* qui est renvoyé, sinon c'est la variable *val2* qui l'est. Voici un exemple :

```
status = (age>=18)? "adult" : "minor";
```

20.2.3.7.2 L'opérateur virgule L'opérateur virgule évalue simplement les deux opérateurs et retourne la valeur de la seconde opérande. Cet opérateur est notamment utilisé dans les boucles **for** qui seront étudiées plus tard.

20.2.3.7.3 delete L'opérateur **delete** permet de détruire un objet, une propriété d'un objet ou encore un élément d'un tableau à un certain index.

```
delete Objet;
```

20.2.3.7.4 new L'opérateur **new** permet de créer une instance d'une classe définit par un utilisateur ou d'une des classe prédéfinies par Java.

```
nomobjet = new typeobjet ( param1 [,param2] ...[,paramN] );
```

20.2.3.7.5 this Grâce à ce mot-clé, l'utilisateur possède toujours une référence sur l'objet courant. Cet opérateur permet à un objet d'appeller des fonctions nécessitant une référence.

20.2.3.8 super

Grâce à ce mot-clé, l'utilisateur possède toujours une référence sur la classe mère de l'objet courant. Cet opérateur permet d'appeler des membres d'une classe mère qui auraient été éventuellement cachées lors de l'héritage.

20.2.4 Les instructions

Java propose une série d'instructions directement héritées du langage C/C++ :
- les instructions conditionnelles : **if...else** et **switch** ;
- les boucles **for**, **while**, **do while**, **break**, **continue** ;
- les commentaires.

Des instructions différentes sont séparées par « ; », et on regroupe les instructions entre une accolade ouverte ({) et une accolade fermée (}).

20.2.4.1 Les instructions conditionnelles

20.2.4.1.1 Les instructions if...else Les instructions **if ... else** permettent d'exécuter une partie de code si une condition est **true**, ou une autre partie de code, optionnelle, si elle est **false**.

```
if (condition) {
  instructions1
}
[else {
  instructions2
}]
```

20.2.4.1.2 L'instruction switch L'instruction switch permet d'évaluer une expression et d'exécuter différentes parties de code en fonction du résultat de cette évaluation.

```
switch (expression) {
  case label :
    instruction;
    break;
  case label :
    instruction;
    break;
  ...
  default :
    instruction;
}
```

L'instruction **break** permet de s'assurer que le programme n'exécute que la partie de code correspondante à la valeur évaluée. Si cette instruction est omise, le programme continue d'exécuter les instructions correspondantes à la valeur suivante.

20.2.4.2 Les instructions de boucles

20.2.4.2.1 L'instruction for Une boucle **for** se répète jusqu'à ce qu'une condition soit évaluée à **false**. La forme générale est :

```
for([expression-initial];[condition];[expression-incrémentation]) {
  instructions;
}
```

Le déroulement d'une boucle **for** est le suivant :

1. L'expression *expression-initial*, si elle existe, est exécutée. Généralement, on initialise des compteurs de boucles.
2. La *condition* est évaluée. Si le résultat est **true**, la boucle continue, sinon elle s'arrête.
3. Les instructions dans la boucle sont exécutées.
4. L'expression *expression-incrémentation* est exécutée, et la boucle retourne à l'étape 2.

L'exemple suivant montre une boucle affichant les dix premiers nombres :

```
for(i=0;i<=10;i++) {
  System.out.println(i);
}
```

20.2.4.2.2 Les instruction do...while La boucle **do...while** se répète jusqu'à ce qu'une condition soit évaluée à **false**. La forme est la suivante :

```
do {
  instructions;
} while (condition)
```

Il est évident que les instructions contenues dans la boucle seront exécutées au moins une fois.

20.2.4.2.3 L'instruction while La boucle **while** se répète tant qu'une condition est évaluée à vrai. La forme est la suivante :

```
while (condition) {
  instructions;
}
```

Il est évident que les instructions dans la boucle ne s'exécutent jamais si la condition est déjà **false** lors de la première évaluation.

20.2.4.2.4 L'instruction break En plus de son utilisation dans l'instruction **switch**, **break** peut-être utilisé dans les boucles sous la forme :

```
break;
```

Cette instruction permet de forcer la sortie d'une boucle donnée.

20.2.4.2.5 L'instruction continue L'instruction **continue** permet de retourner à l'endroit d'une boucle où la condition est évaluée. Sa forme est

```
continue;
```

20.2.4.3 Les commentaires

Le code peut être commenté comme en C++ :
– les commentaires commençant par « // » vont jusqu'à la fin de la ligne ;
– les commentaires compris entre « /* » et « */ » peuvent s'étendre sur plusieures lignes.

20.3 Les objets dans Java

En plus des types de base, le programmeur Java peut construire des classes, c'est-à-dire des objets. Cette possibilité est d'autant plus importante que Java est un langage résolument orienté objets. Contrairement à d'autres langages de programmation orientés objet, Java nécéssite qu'une classe et le code associé figurent dans le même fichier. Lorsque l'utilisateur désire qu'une classe puisse être utilisée en dehors du fichier dans laquelle elle est déclarée, il est nécessaire qu'elle soit la première classe définie dans ledit fichier, et que le nom de celui-ci corresponde au nom de la classe.

En fait, dans Java, les classes sont le seul moyen de regrouper des entités afin de créer de nouvelles structures. Dans Java, tout est un objet, ainsi même la fonction **main** est une fonction statique d'une classe implicite. Toutes les classes dans Java doivent hériter d'une manière ou d'une autre de la super-classe **Object** qui fournit certaines opérations de base (comparaison, notification, etc.) pour n'importe quel objet du système.

Ainsi pour la plupart des types de base, il existe une classe Java correspondante comme indiqué dans le TAB. 20.5.

TAB. 20.5 – Classe Java correspondant aux types de base

Type de base	Classe Java correspondante
boolean	Boolean
char	Character
int	Integer
long	Long
float	Float
double	Double

20.3.1 Les classes

Une classe se déclare à l'aide du mot-clé **class**. Ensuite, entre les accolades qui suivent, on déclare les variables et les fonctions membres. On peut voir au listing 20.1 un exemple d'une classe représentant une fraction.

Pour déclarer une nouvelle instance d'une classe, on déclare une variable et on utilise ensuite l'opérateur **new** pour créer cette instance :

```
Fraction f;
f=new Fraction(2.0,3.0);
```

Listing 20.1 – Exemple de classe

```
class Fraction{
2   float num;
    float den;
4   Fraction() {
      num=1;
6     den=1;
    }
8   Fraction(float n,float d) {
      num=n;
10    den=d;
    }
12  float evalFraction() {
      return(num/den);
14  }
}
```

20.3.1.1 L'accès aux membres

L'accès aux membres d'une classe se fait à l'aide de l'opérateur « . ». Par exemple :

```
f.num=3.0;
System.out.println(f.evalFraction());
```

Comme illustré au listing 20.1 à la ligne 13, on accède aux membres d'une classe en mentionnant simplement leur nom.

Par défaut, l'accès aux membres d'une classe, variables ou fonctions, est public, ce qui peut être précisé en mettant le mot-clé **public** devant la déclaration. Il existe néanmoins d'autres types d'accès :

- l'accès protégé (mot-clé **protected**) permettant de restreindre l'accès aux méthodes (définies à la section 20.3.2) et variables membres à la classe et aux classes directement dérivées ;
- l'accès privé (mot-clé **private**) permettant de restreindre l'accès aux méthodes et variables membres uniquement à la classe elle-même.

20.3.1.2 Les membres statiques

Normalement les membres ne sont accessible qu'au travers d'une instance de la classe. Par exemple, les variables « num » et « den » existent pour toutes instance de la classe « Fraction ». En utilisant le modificateur **static**, on rend un membre *unique* pour toutes les instances de la classe. Ces membres statiques sont utilisés par exemple pour définir des constantes :

```
class Pendule {
  ...
  static float gravitation=10.0;
  ...
}
Pendule.gravitation=9.80;
```

Toutes les instances de la classe *Pendule* ont une variable identique *gravitation*. Pour modifier sa valeur, on accède à cette variable en utilisant le nom de la classe, ce qui est logique puisque cette variable « appartient » à l'ensemble de la classe et non pas à ses instances particulières.

20.3.2 Les méthodes

Une méthode est une fonction d'une classe. On peut voir au listing 20.1, un exemple de fonction (lignes 12-14). Dans cet exemple précis, la fonction de prend aucun paramètre. Pour passer des paramètres à une fonction, il suffit de les déclarer dans les paranthèses suivant le nom de cette dernière et précédant le code lui-même :

```
maFonction(int i,monTypeObjet obj) {
}
maFonction(3,4);    // Appel à la fonction "MaFonction"
```

La règle pour le passage des paramètres est très simple en Java :

- si le paramètre passé est d'un des types fondamentaux, il est passé par valeur, c'est-à-dire que toute modification de la variable à l'intérieur de la fonction n'affecte pas la variable utilisée lors de l'appel ;
- si le paramètre passé est une classe, le paramètre est passé par référence, c'est-à-dire que toute modification de l'une des variables de l'objet à l'intérieur de la fonction affecte la variable de l'objet utilisé lors de l'appel.

20.3.2.1 Surcharge des méthodes

Le Java autorise la surcharge des méthodes, c'est-à-dire de donner le même nom à plusieurs fonctions, en les distinguant par le nombre de paramètres nécessaires à leur appel ou leur type. Par exemple,

```
void Print(int i,int j) {...};
void Print(int i)       {...};
void Print(Object obj)  {...};
```

Lors de l'appel à la fonction, Java décide, en fonction du nombre de paramètres et de la nature de ceux-ci, quelle fonction est réellement appellée.

20.3.3 Création d'objets

Les objets dans Java sont alloués de manière dynamique par l'intermédiaire de l'opérateur **new**. Java se charge d'allouer correctement la quantité de mémoire nécessaire à l'objet. De plus, les objets sont détruits par le « ramasse-miettes » lorsqu'ils ne sont plus référencés, c'est-à-dire lorsqu'ils ne sont plus nécessaires au programme.

Lors de la création d'un objet, Java fait appel à un constructeur, une méthode qui porte le même nom que la classe. Dans l'exemple du listing 20.1, la classe « Fraction » définit deux constructeurs, l'un sans paramètres (lignes 4-7), l'autre avec deux paramètres (lignes 8-11). Ceci permet de créer une instance du type « Fraction » de deux manières différentes :

```
Fraction f1=new Fraction();
Fraction f2=new Fraction(2.0,3.0);
```

Pour signifier à Java qu'un objet n'est plus nécessaire, il faut lui assigner la valeur **null** comme dans l'exemple suivant :

```
f1=null;
```

L'objet est alors géré par le « ramasse-miettes ».

Remark: *Lorsque qu'une fonction se termine, Java sait automatiquement que ses variables locales ne référencent plus d'objet. Il n'est pas nécessaire de leur assigner la valeur* ***null****.*

20.3.4 Destruction d'objets

Comme expliqué précédemment, Java s'occupe de la destruction des objets. Le programmeur ne doit donc pas s'en soucier.

20.3.4.1 Le ramasse-miettes

Java utilise la technique du « ramasse-miette » (« garbage collector » en anglais) pour supprimer les objets qui ne sont plus utiles. Il s'agit d'un algorithme qui s'exécute en tâche de fond avec une faible priorité, et qui vérifie s'il existe des objets qui ne sont plus référencés. Si c'est le cas, Java les détruit et libère la mémoire allouée.

Notons que l'appel à la méthode **System.gc()** dans une application, force de manière explicite le ramasse-miettes à mettre de l'ordre dans la mémoire.

20.3.4.2 Terminaison

Chaque classe peut déclarer une méthode **finalize()**, systématiquement appellé pour chaque objet juste avant sa destruction. Elle permet notamment de « nettoyer » l'application. Pa exemple, Java désalloue correctement la mémoire, mais ne ferme pas forcément les fichiers qui auraient étés ouverts.

Dans les autres langages orientés objets, comme le C++ ou le Pascal, cette fonction est appellée un destructeur.

20.3.5 Dérivation et héritage

Pour dériver une classe à partir d'une autre, on utilise le mot-clé **extends**. Un exemple est présenté au listing 20.2.

Listing 20.2 – Dérivation de classes

```
class Animal{
  float weight;
  ...
  Animal() {
    ...
  }
  void eat(){
    ...
  }
  ...
}

class Mammal extends Animal{
  int heartRythm;
  // Has also weight
  ...
  void Mammal() {
    super();
    ...
  }
  void breathe() {
    ...
  }
  // Has also eat
  ...
}
```

Dans cet exemple, un objet du type « Mammal » possède à la fois la variable d'instance « weight » et la méthode « eat ». Elles sont héritées de la classe « Animal ».

Contrairement à d'autres langages, comme le C++, une classe ne peut dériver que d'une seule classe ; Java ne permet pas d'héritage multiple.

Il est possible de déclarer dans une classe fille une variable portant le même nom qu'une variable de la classe mère. On parle alors de variables *masquées*, puisque la variable de la classe mère n'est plus accessible dans la classe fille, même si évidemment elle existe encore à la suite de l'héritage. L'opérateur **super** permet d'accéder à cette variable de la classe mère. Il représente une référence de l'objet courant mais du type de la classe mère. On se sert également de **super** pour appeller le constructeur de la classe mère à partir de celui de la classe fille (ligne 18).

De même les méthodes peuvent être redéfinie dans la classe fille. Java implante ainsi le polymorphisme (référez-vous à l'annexe A.4) : l'appel n'est pas défini de manière statique lors de la compilation, mais dynamiquement, c'est-à-dire lors de l'exécution du programme. Il est toutefois possible, grâce au modificateur **final**, d'empêcher qu'une méthode puisse être redéfinie par l'une des classes dérivées.

Comme une classe fille hérite de toutes les caractéristiques de la classe mère, même si certaines sont redéfinies à l'aide de méthodes dynamiques, il est possible d'utiliser une instance d'une classe fille partout où une instance d'une classe dont elle dérive directement ou indirectement pourrait être utilisée. Par exemple, le code suivant est correct :

```
void faireQql(Animal obj) {
  obj.eat();
```

```
}
Mammal m=new Mammal();
faireQql(m);    // Ok, car m est aussi du type Animal
```

20.3.6 Méthodes et classes abstraites

Une méthode peut être déclarée abstraite avec le modificateur **abstract** pour spécifier qu'il s'agit uniquement d'un prototype ou d'une définition qu'il faut redéfinir dans le classe dérivant de celle où ces méthodes sont implantées. Dès qu'une classe contient au moins une méthode abstraite, la classe est dite *abstraite* et doit être déclarée comme telle avec le modificateur **abstract**. Il est impossible d'instantier une classe abstraite.

```
abstract class classeAbstraite {
  ...
  abstract méthodeAbstraite(String nom);
  ...
}
classeAbstraite essai=new classAbstraite();   // Erreur car classe abstraite
```

Les méthodes abstraites fournissent un puissant moyen de conception orientée objets, car elles permettent d'établir le fonctionnement général d'une classe sans se soucier de la manière dont les fonctionnalités seront effectivement implantées.

20.3.7 La classe Object

La classe « java.lang.Object » est la mère de tous les objets en Java. Il s'agit d'une classe primordiale dont toutes les autres classes doivent être dérivées. Elle implante des méthodes fondamentales, comme celles permettant de gérer la synchronisation d'accès et de modification des objets dans une utilisation multi-thread. Parmi les méthodes les plus souvent utilisées, on retrouve :

- la méthode **toString()** appelée lorsque l'objet doit être représenté sous la forme d'une chaîne de caractères ;
- la méthode **equals()** déterminant si deux objets sont équivalents, et qui peut se voir surchargée pour les besoins d'une classe particulière ;
- la méthode **hashCode()** qui retourne un entier représentant une signature unique pour une instance d'une classe donnée, ce nombre étant utilisé par exemple lors du processus d'insertion d'objets dans une table de hachage (**HashTable**).

20.4 Les paquetages

Les paquetages sont un moyen de regrouper plusieurs classes présentant certains points communs. Un paquetage peut être compéré à une section d'une bibliothèque : tous les livres parlant de biologie se trouvent au même endroit afin de faciliter leur consultation. Les paquetages proposent une série de classes couvrant des fonctionnalités d'un domaine et qui pourront être réutilisées dans un programme.

20.4.1 Unité de compilation

En Java, on appelle le code source d'une classe une unité de compilation. Elle ne contient qu'une seule définition de classe et porte son nom. Si on veut qu'une unité de compilation appartienne à un certain package, on utilise l'instruction **package**. Par exemple on pourrait avoir une première unité de compilation :

```
package mesoutils.texte;
class ComposantTexte {
  ...
}
```

et une deuxième

```
package mesoutils.texte;
class CorrectionTexte {
  ...
}
```

Les deux classes appartiennent au paquetage « mesoutils.texte ».

20.4.2 Les noms de paquetages

Les noms des paquetages sont construits de manière hiérarchqiue en utilisant le point (.) comme séparateur. En fait, le nom complet d'un paquetage donne une indication de son emplacement sur le système qui fait appel à lui.

Notons qu'il n'existe pas de notion de sous-paquetage. Par exemple, si un autre paquetage « mesoutils.texte.poesie » est construit, les classes définies ne doivent être considérées comme membres du paquetage « mesoutils.texte ».

20.4.3 Visibilité d'une classe

Par défaut, une classe n'est visible, et donc utilisable, que par les autres classes du même paquetage qu'elle. Afin de la déclarer publique, le modificateur **public** doit être utilisé, comme dans l'exemple :

```
package mesoutils.texte;
public class EditeurTexte{
  ...
}
```

La classe « EditeurTexte » peut alors être utilisée par n'importe quelle autre classe. Par contre, les classes « ComposantTexte » et « CorrectionTexte » ne pourront pas être utilisées en dehors du paquetage « mesoutils.texte ».

20.4.4 Importation de classes

Pour permettre à une classe de pouvoir utiliser une classe publique, il faut importer cette dernière dans l'unité de compilation. Cela se fait avec l'instruction **import**.

```
package tesoutils.general;
import mesoutils.texte.EditeurTexte;
class MaClasse {
  EditeurTexte zone;
  ...
}
```

Une fois une classe importée, on peut utiliser son nom « court » : il n'est plus obligatoire de donner le nom complet comprenant le nom du paquetage. Il est également possible d'importer toute les classes d'un certain paquetage en utilisant un astérisque (*) :

```
import mesoutils.texte.*;
```

Toutes les classes publiques de ce paquetage peuvent alors être utilisées dans l'unité de compilation où figure cette instruction.

20.4.5 Le paquetage anonyme

Une classe définie dans une unité de compilation pour laquelle aucun nom de paquetage a été spécifié est automatiquement intégrée dans le paquetage anonyme (**unnamed**). Ces classes peuvent être utilisées sans qu'il soit nécessaire d'utiliser l'instruction **import**.

20.5 Les interfaces

Une interface est une liste de méthodes définissant des comportements pour un objet. Toute classe qui veut satisfaire une interface doit déclarer chacune des méthodes énumérées dans cette dernière. Une interface ressemble purement et simplement à des classes ne possédant que des fonctions abstraites. Pour définir une interface, on utilise le mot-clé **interface**, et ensuite on énumère toutes les méthodes lui appartenant.

```
interface Conduisible {
  boolean démarrerMoteur();
  void couperMoteur();
}
```

Pour dire qu'une classe donnée implante une interface, on utilise le mot-clé **implements**.

```
class Automobile implements Conduisible {
  boolean démarrerMoteur() { ... }
  void couperMoteur(); { ... }
}
```

On peut maintenant déclarer des variables du type « Conduisible » et faire des actions dessus, comme le montre les exemples suivants, en supposant que l'on a défini un deuxième type de classe « TondeuseGazon » implantant également cette interface :

```
Automobile auto=new Automobile();
TondeuzeGazon tondeuze=new TondeuzeGazon();
Conduisible véhicule;
véhicule=auto;       // Ok -> "Automobile" implante "Conduisible"
véhicule.démarrerMoteur();
véhicule.couperMoteur');
véhicule=tondeuze;   // Ok -> "TondeuseGazon" implante "Conduisible"
véhicule.démarrerMoteur();
véhicule.couperMoteur');
```

Remark: *Une classe peut implanter une interface et hériter d'une classe.*

20.5.1 Les variables d'interface

Il est possible de déclarer des variables statiques dans une interface. Cette fonctionnalité autorise l'utilisation de paramètres prédéfinis pour les méthodes définies. Par exemple :

```
interface Redimensionner {
  static final int GRAND=0, MOYEN=1, PETIT=2;
  void changerTaille(int taille);
}
```

20.5.2 Interfaces et paquetages

Les interfaces se comportent comme des classes : elles ne sont visibles que dans le paquetage dans lequel elles sont définies. Pour les rendre visible pour les autres paquetages, il faut les déclarer publique. Il ne peut y avoir qu'une seule interface publique dans une unité de compilation.

20.5.3 Les sous-interfaces

Une interface peut étendre une autre interface, et donc hériter de toutes les méthodes définies dans celle-ci, par l'intermédiaire de l'instruction **extends**. On parle alors d'une sous-interface.

```
interface ConduisibleAccel extends Conduisible {
  float accélérer(float acc);
}
```

20.5.4 Intérêt des interfaces

Les interfaces permettent d'implanter des fonctionnalités similaires à l'héritage multiple, interdit en Java. Pour comprendre cela, il suffit de considérer l'exemple d'une classe *Container* qui implante une liste générique d'objets, chaque objet devant implanter une méthode *Compare()*. On pourrait implanter cela par :

```
class BaseObj {
  int Compare() {return(0)}
}
class Container {
  BaseObj tab[];
  ...
}
```

Donc, pour pouvoir utiliser la classe *Container* avec une classe *X*, il suffit de faire dériver la classe *X* de la classe *BaseObj*. Mais, si on veut utiliser *Container* avec une classe existante, par exemple **Boolean**, c'est impossible, car il faudrait créer une classe qui dérive de **Boolean**, pour pouvoir utiliser les variables et méthodes de celle-ci, et de *BaseObj* pour pouvoir l'utiliser dans *Container*. C'est dans ces situations que les interfaces sont utiles. En effet, si on écrit :

```
interface BaseObj{
  int Compare();
}
class Container {
  BaseObj tab[];
  ...
}
```

tout devient possible, car il est possible de créer une classe qui dérive the **Boolean** et qui implémente l'interface nécessaire pour être utiliser dans *Container*.

20.6 La librairie AWT

La librairie Abstract Windowing Toolkit (AWT) fournit un grand nombre de classes permettant de construire un Graphical User Interface (GUI) en Java. Ces classes permettent de créer des fenêtres, de dessiner, de manipuler des images et d'utiliser des composants comme des boutons, des barres de défilement ou des menus déroulants. Les classes de l'AWT appartiennent toutes au paquetage **java.awt**. La FIG. 20.1 présente l'ensemble des classes de cette librairie.

Les classes et fonctionnalités de la librairie AWT sont identiques quel que soit le systèmes d'exploitation utilisé (par exemple Linux/X Window, Macintosh ou Window NT/98). Pour complèter l'indépendance vis-à-vis de la plate-forme, l'AWT utilise des toolkits interchangeables qui appellent le système de fenêtres de la plate-forme. Les programmes utilisant l'AWT sont donc indépendants des détails de l'implémentation. Ainsi, si une applet nécessite un bouton, elle crée un toolkit « bouton » propre à l'environnement, qui donc aux autres boutons Windows sur une machine Windows ou aux boutons Macintosh sur une machine Macintosh.

Une étude détaillée de la librairie sortirait largement du cadre du présent exposé, c'est pourquoi nous nous contenterons de présenter quelques uns des composants les plus facile à utiliser. Pour une référence complète sur ce sujet, le lecteur est ivité à consuler [32].

20.6.1 Concepts de GUI en Java

Avant d'introduire quelques exemples d'utilisation des composants, il est nécessaire d'expliquer les quelques concepts qui sont à la base de l'AWT.

20.6.1.1 Composants AWT

Un composant est un objet fondamental dans l'interface utilisateur Java. Tout ce qui est vu à l'écran dans une application, comme un bouton ou un menu par exemple, est un composant AWT. Tous les composants

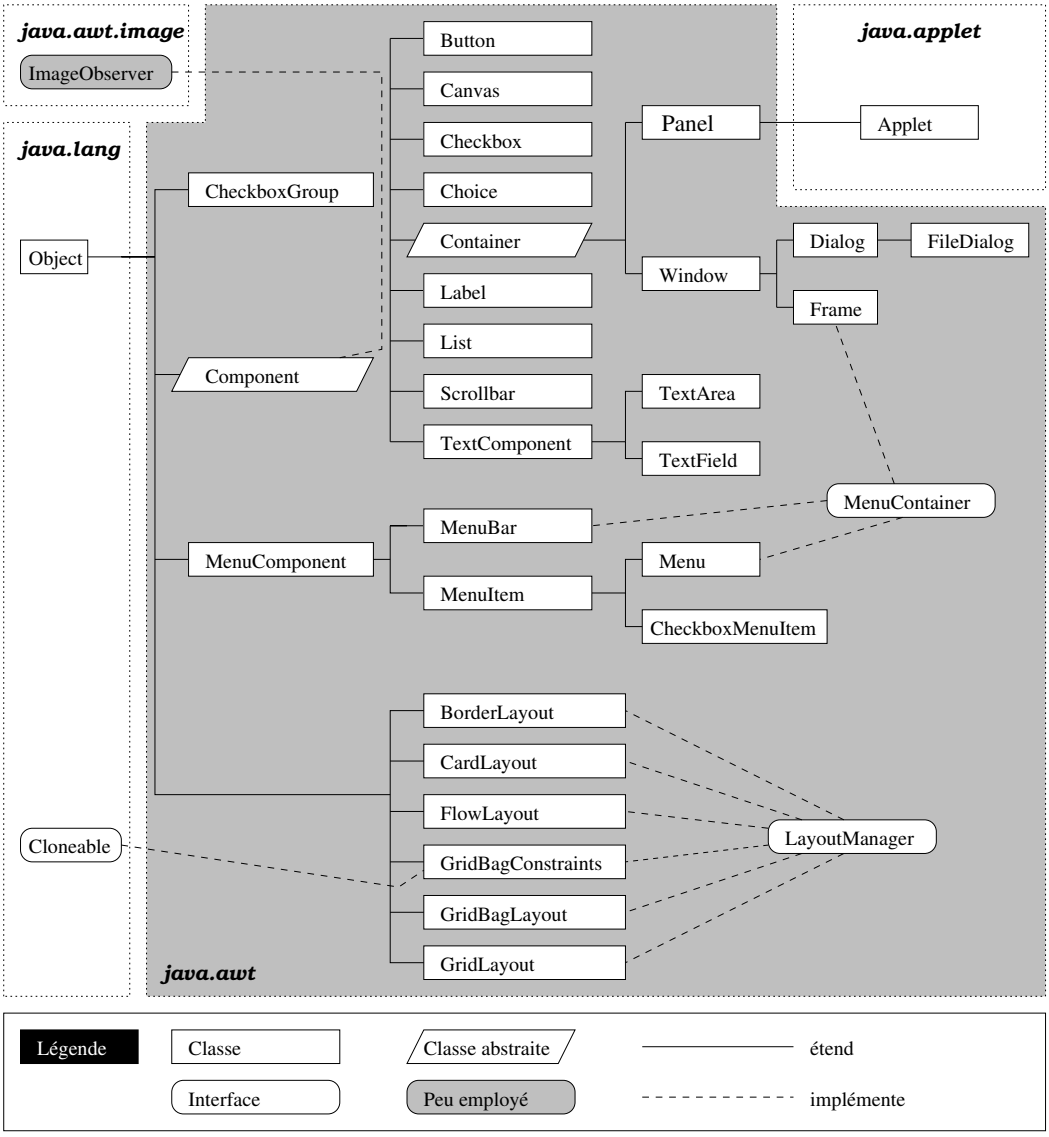


FIG. 20.1 – Classes de la librairie AWT

sont dérivés de la classe abstraite **java.awt.Component**, elle-même dérivée directement de la classe **Object** (référez-vous à la FIG. 20.1).

Les fonctionnalités d'un composant peuvent être divisées en deux catégories :

- l'apparence, c'est-à-dire l'ensemble des variables et des méthodes permettant de contrôler l'aspect général du composant ;
- le comportement, c'est-à-dire l'ensemble des variables et des méthodes permettant de faire réagir le composant aux événements.

Les composants délèguent certaines responsabilités aux objets conteneurs dans lequels ils résident, qui sont eux-mêmes des composants pouvant regrouper plusieurs compsants. Ainsi une demande à un composant, par exemple se redessiner après un déplacement, provoque automatiquement la même demande à tous les composants qu'il contient.

20.6.1.2 Peer

Afin de permettre aux composants d'interagir avec le monde réel, il faut des objets contenant du code natif afin d'interagir avec le système d'exploitation de la machine. Afin de garder l'interaction aussi propre que possible, Java utilise un ensemble d'interfaces « peer ». Une interface peer permet à un composant Java d'utiliser son composant réel, l'objet « peer », dans l'environnement utilisateur. Même si le mécanisme est encapsulé directement à l'intérieur de la classe **Component**, il est important de bien le comprendre, car il entraîne certaines limitations dans l'utilisation des composants. Par exemple, lorsqu'un composant comme l'objet **Button** est crée et affiché pour la première fois à l'écran, la classe **Component** demande à la classe **Toolkit** de l'AWT de créer un objet peer correspondant, comme le montre la FIG. 20.2. L'objet **Toolkit** contient des méthodes pour créer des instances de chaque type de composant peer. L'avantage d'une telle approche est que les objets **Toolkit** peuvent être remplacés pour fournir de nouvelles implentations des composants sans affecter le reste du fonctionnement de Java, en particulier sans devoir recompiler les programmes.

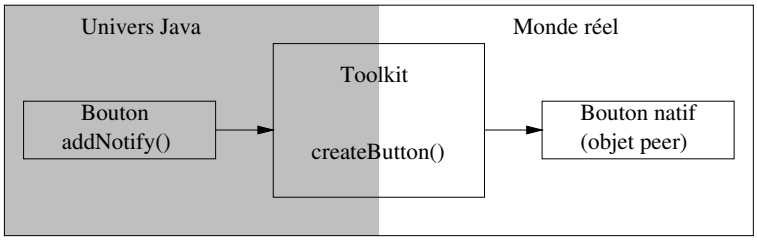


FIG. 20.2 – Un toolkit pour créer un objet peer

20.6.1.3 Affichage et rafraîchissement

Les composants doivent pouvoir se redessiner eux-mêmes à tout moment. Par exemple, lorsque un composant est masqué par une fenêtre et qu'il doit réapparaître, un thread AWT lui demande de se redessiner en appelant sa méthode **paint()**. Comme la méthode **paint()** ne peut faire aucune supposition quant au motif de l'appel, elle doit redessiner l'ensemble du composant, même si certaines zones ne nécessitent pas cette mise à jour. On préfère généralement spécifier la partie qui a changé et ne redessiner que cette zone. C'est la raison pour laquelle il existe une méthode **update()** qui définit la région à redessiner. Un composant n'appelle jamais directement la méthode **update()**, mais il appelle plutôt la méthode **repaint()**.

Les deux méthodes **paint()** et **update()** ne prennent qu'un seul argument : un objet **Graphics**, qui représente le contexte graphique du composant.

20.6.1.4 Événements

Les événements sont créés par l'environnement de la machine et sont expédiés à un composant via des méthodes de gestion d'événements. Lorsqu'un gestionnaire d'événements est appelé, il reçoit au moins un objet du type **Event** en paramètre. Un composant peut gérer des événements à deux endroits :

- dans la méthode **handleEvent()** de la classe **Component**, qui gére les événements à un niveau inférieur ;
- par l'intermédiaire de méthodes réservées certains types d'événements, par exemple **mouseDown()**, **mouseUp()**, **KeyDown()** et **keyUp()**.

Un composant n'est pas obligé de traiter l'événement qu'il reçoit. Le gestionnaire d'événements du composant renvoie la valeur **true**, si l'événement a été traité. Par contre, si le gestionnaire renvoie la valeur **false**, l'événement est passé au conteneur. Enfin, si aucun des composants n'est intéressé par l'événement, il est passé à l'objet composant peer. Bon nombre d'événements sont traités par ces objets. De même, si l'objet peer décide de ne pas traiter l'événement, c'est l'objet peer du conteneur qui le reçoit.

En fait, lors d'un événement, la première méthode appelé est **postEvent()** de la classe **Component**. Généralement, elle se contente d'implanter le schéma décrit précédemment, mais elle peut aussi être utilisée afin qu'un composant puisse générer ses propres événements.

Remark: *Pour recevoir des événements, un composant doit avoir le focus, c'est-à-dire qu'il doit être sélectionné. Pour obtenir le focus, un composant peut appeller la méthode **getFocus()** de la classe **Component**. La classe **Component** implante également deux méthodes **gotFocus()** et **lostFocus()** qui sont appelées lorsque celui-ci reçoit ou perd le focus.*

20.6.1.5 Conteneurs

Un conteneur est un composant qui en contient et en gère d'autres . Dans AWT, tous les conteneurs dérivent de **Container** (référez-vous à la FIG. 20.1). Un objet **Frame** représente une fenêtre de haut niveau, pouvant par exemple avoir une barre de menu grâce à la méthode **setMenuBar()**. L'objet **Frame** dérive de l'objet **Window**. L'objet **Panel** est un conteneur générique utilisé pour grouper des composants.

Parmi les méthodes spécifiques de la classe **Container**, la méthode **add()** permet d'ajouter un composant « fils » au conteneur.

Un gestionnaire de placement est associé à chaque conteneur. C'est un objet qui contrôle le placement et la taille des composants à l'intérieur de la zone d'affichage du conteneur. Grâce à la méthode **setLayout()** il est possible d'en installer un autre. L'AWT définit une série de gestionnaires de placement qui implémentent certains schémas de mise en page assez répandus. Le gestionnaire de placement par défaut pour l'objet **Panel** est la classe **FlowLayout**, alors que pour l'objet **Frame**, il s'agit de la classe **BorderLayout** qui permet de placer un nombre limité d'objets aux endroits indiqués par « North », « South » et « Center ». Par exemple, pour ajouter un nouveau composant dans la partie supérieure du conteneur, on utilise :

```
add("North",nouveauComposant);
```

Lorsque le conteneur a subi des modifications nécessitant un réagencement des composants qu'il contient, par exemple après la modification de sa taille, on dit que le conteneur est invalide. Pour le rendre valide, il faut appeller la méthode **validate()** de la classe **Container**. Celle-ci appelle la méthode **layout()** du gestionnaire qui va modifier la taille et la position des différents composants pour tenir compte des changements du conteneur.

20.6.2 Applets

Un objet **Applet** s'attend à être inclus dans un document et à être utilisé dans un environnement de visualisation qui fournisse ses propres ressources. Comme l'objet **Applet** dérive de l'objet **Frame**, il peut contenir des composants.

La classe **Applet** définit quatre méthodes qu'une applet peut redéfinir, appelées par le browser pour influencer son comportement :

- la méthode **init()** est appelée une seule fois après l'instanciation de l'applet ;
- la méthode **start()** est appelée chaque fois que l'applet devient visible ;
- la méthode **stop()** est appelée chaque fois que l'applet devient invisible, par exemple lorsqu'elle disparaît de la partie visible du document qui la contient ;
- la méthode **destroy()** est appelée pour permettre à l'applet de faire le ménage avant sa destruction.

Une applet peut demander des renseignements concernant l'environnement HTML dans lequel elle évolue, par exemple pour récupérer des images se trouvant dans ce document HTML.

Les méthodes **getDocumentBase()** et **getCodeBase()** permettent de déterminer l'URL du document où l'applet se trouve et l'URL du fichier contenant l'applet elle-même. Grâce à ces deux méthodes, elle peut construire des URL relatives à partir desquels elle peut charger des images ou ouvrir des fichiers contenant des données.

Comme l'explique la section 20.7, il est possible de passer des paramètres à l'applet Java à partir de la balise **APPLET** du langage HTML. La méthode **getParameter()** permet de récupérer la valeur du paramètre dont le nom est passé à la méthode.

```
String nomImage = getParameter("nomImage");
```

Avec la méthode **showStatus()**, une applet peut demander au browser d'insérer du texte dans la barre de statut. Avec la méthode **getAppletContext()**, une applet peut récupérer un objet **AppletContext**, afin d'utiliser certaines méthodes de celui-ci, dont **showDocument()** qui permet de changer le contenu du browser avec un nouveau document :

```
getAppletContext().showDocument(url);
```

Remark: *Signalons que les browsers ne sont pas obligés de tenir compte des méthodes **showStatus()** et **showDocument()**. Le comportement des browsers par rapport à ces méthodes est généralement la conséquence d'options de l'utilisateur.*

20.6.3 Utiliser des composants et des conteneurs

Les composants peuvent être séparés en deux catégories :
- les composants fonctionnels qui ont un comportement interactif et réagissent à l'utilisateur lorsqu'il clique avec la souris par exemple (boutons, barres de défilement, etc.) ;
- les composants « briques de base » qui ne font pas grand chose mais permettent de connaître très facilement le nombre de composants fonctionnels présents à un moment donné.

Lorsque l'on travaille avec des composants fonctionnels, on s'intéresse plus au résultat de leurs actions qu'aux détails de leur fonctionnement. Par exemple, lors de la manipulation d'un objet **Button**, on voudrait savoir quand il a été préssé, sans pour autant connaître la durée de la pression. Lorsqu'ils sont utilisés, la plupart des composants fonctionnels envoient un événement générique **ACTION_EVENT** qui est passée à la méthode **action()**. Cette méthode patu alors être détaillée pour tenir compte de détails d'événement associé à un composant.

20.6.3.1 Composants intelligents

Un événement peut être géré par un composant. Il s'agit alors d'un composant intelligent. Dans l'exemple du listing 20.3, on définit un composant « AppuieBouton » intelligent qui gère ses propres événements.

Listing 20.3 – Composant intelligent

```
import java.awt.*;
public class Appuie extends java.applet.Applet {
  public void init() {
    add(new AppuieBouton());
  }
}
class AppuieBouton extends Button {
  AppuieBouton() {
    setLabel("Cliquez ici!");
  }
  public boolean action(Event e,Object arg) {
    System.out.println("Merci.");
    return(true);
  }
}
```

L'objet « AppuieBouton » hérite de l'objet **Button** qui implémente un bouton. La partie intéressante est la redéfinition de la méthode **action()** (lignes 11-14). Lorsque l'on appuie sur le bouton, cette méthode est appelée est affiche un message sur la console Java.

20.6.3.2 Conteneurs intelligents

Si une interface utilisateur comporte beaucoup de composants simples, il est généralement plus aisé de faire gérer les événements par le conteneur. En effet, les conteneurs dérivent indirectement de la classe **Component** et peuvent donc également redéfinir les gestionnaires d'événements.

Dans l'exemple montré au listing 20.4, on crée une espèce de **Panel** qui s'appelle « ButtonsGrid » et qui gère un nombre configurable de composants **Button**. On peut voir le résultat à la FIG. 20.3.

Listing 20.4 – L'applet Touches

```
import java.awt.*;
public class Touches extends java.applet.Applet {
  public void init() {
    add(new ButtonsGrid(16));
    resize(preferredSize());
  }
}
class ButtonsGrid extends Panel {
  ButtonsGrid(int n) {
    int s = (int)Math.sqrt(n);
    setLayout(new GridLayout(s,s));
    for(int i=0;i<n;i++)
      add(new Button(Integer.toString(i)));
  }
  public boolean action(Event e,Object arg) {
    System.out.println("Button "+arg+" pressed");
    return(true);
  }
}
```

FIG. 20.3 – L'applet Touches

Afin de gérer le placement des boutons, on utilise un gestionnaire de placement **GridLayout** (ligne 11), qui comme son nom l'indique permet de gérer une grille de composants. Ensuite, on crée autant de composants **Bouton** que le nombre spécifié dans le constructeur en les labélisant par leur numéro.

La partie intéressante est la redéfinition de la méthode **action()** (lignes 15-18). Celle-ci prend deux paramètres, un objet du type **Event** représentant l'événement et un objet générique du type **Object** qui l'a provoqué. Dans l'exemple, on se contente d'afficher un message contenant le nom de l'objet, mais des traitements plus compliqués peuvent être implantés, par exemple mettre les objets en « grisé » :

```
public boolean action(Event e,Object arg) {
  ((Component)e.target).disable();
  return(true);
}
```

On recupère de l'objet **Event** la cible de l'événement. On fait un transtypage[2] afin de le transformer en un objet du type **Component**, que l'on grise en utilisant la méthode **disable()**.

20.6.3.3 Qui envoie un événement ?

La première chose à faire lors de la gestion d'un événement est d'identifier son émetteur. Dans l'exemple 20.3, cela ne posait pas de problème car il n'y avait qu'un seul élément. Mais dans l'exemple 20.4, il a fallut

[2]Un transtypage consiste à transformer un type de variable en un autre. Ces transtypages doivent être effectués prudemment car le résultat peut être « bizarre », surtout si les objets ne sont pas compatibles. Ceci dit, Java effectue la vérification avant de faire un transtypage et émet éventuellement un message d'erreur.

s'aider du paramètre « arg ».

On peut toujours récupérer la variable **target** de l'objet **Event** afin de voir qui a envoyé l'événement. Afin d'avoir plus d'informations sur cet objet, on peut utiliser les opérateurs **instanceof** et d'égalité (==). Par exemple, si la programme ne contient qu'une seule barre de défilement, il suffit de s'assurer que l'objet est bien du type **Scrollbar** :

```
handleEvent(Event e) {
  if(e.target instanceof Scrollbar) {
    ...
  }
  ...
}
```

Par contre, si le programme contient plusieurs barres de défilement, il faut tester l'identité de l'objet :

```
action(Event e, Object arg) {
  if(e.target == maBarreDéfilementGauche) {
    ...
  }
  ...
}
```

Remark: *Il ne faut jamais oublier de renvoyer **false** pour tous les événements qui ne sont pas traités par le gestionnaire, afin de rendre le programme beaucoup plus résistant lorsque des nouvelles fonctionnalités seront ajoutées à l'interface.*

20.6.4 Composants texte

La librairie AWT propose principalement trois composants pour traiter du texte :
- **Label** qui permet de créer un label contenant du texte ne pouvant pas être modifié ;
- **TextField** qui est une zone éditable d'une seule ligne ;
- **TextArea** qui est un éditeur de texte multiligne avec barres de défilement verticale et horizontale.

Un exemple de l'utilisation de ces trois composants est présenté dans l'applet reprise au listing 20.5 et dont le résultat est présenté à la FIG. 20.4. Lorsque l'utilisateur entre quelque chose dans la zone de texte simple et appuie sur la touche « Return », son contenu est automatiquement ajouté dans l'éditeur de texte.

FIG. 20.4 – Applet TextExample

Remark: *Des objets **Panel** ont été utilisés pour contenir les composants texte dans l'applet. Ceci permet de garder des tailles « normales » pour les composants texte quelque soit la taille de l'applet elle-*

Listing 20.5 – Applet TextExample

```
import java.awt.*;
public class TextExample extends java.applet.Applet {
  TextArea zone;
  TextField field;
  Label label;
  int nbLines=10,nbCols=20;
  public void init() {
    setLayout(new BorderLayout());
    Panel p=new Panel();
    p.add(zone=new TextArea(nbLines,nbCols));
    zone.setFont(new Font("TimesRoman",Font.BOLD,12));
    zone.setText("Default Text \n");
    add("North",p);
    p=new Panel();
    p.add(field=new TextField(nbCols));
    add("Center",p);
    p=new Panel();
    p.add(label=new Label("Label"));
    add("South",p);
  }
  public boolean action(Event e,Object arg) {
    if(e.target instanceof TextField) {
      String s=zone.getText()+arg+"\n";
      zone.setText(s);
      field.setText("");
      return(true);
    }
    return(false);
  }
}
```

même. De plus, on a également utilisé un gestionnaire de placement ***BorderLayout*** *pour une question de commodité.*

Les deux méthodes les plus importantes des composants texte sont :

- **getText()** qui renvoie le contenu du composant ;
- **setText()** qui permet de changer le contenu du composant.

Regardons de plus près ce que fait la méthode **action()** (lignes 21-29). Tout d'abord, on regarde si l'objet qui a envoyé l'événement est du type **TextField**. Si c'est le cas, on crée une chaîne de caractères (ligne 23) en récupérant le texte contenu dans l'éditeur et en y ajoutant la représentation sous forme d'une chaîne de caractères de l'objet qui a envoyé l'événement (dans le cas de la classe **TextField**, il s'agit du contenu de celui-ci). On assigne ensuite le résultat à la zone d'édition et on vide le champ texte simple. On renvoie ensuite **true** afin de signaler que l'événement a été traité correctement. Dans tous les autres cas de figure, on renvoit **false** puisque l'événement n'est pas traité.

En utilisant la méthode **setEditable()** il est possible de mettre les composants en mode « lecture » uniquement. De plus, les composants gèrent des selections, c'est-à-dire des portions de texte sélectionné à la souris ou à l'iade du clavier. On peut récupérer le texte sélectionné en utilisant **getSelectedText()**.

20.6.5 Menus et sélecteur d'options

Un objet **Menu** est un menu déroulant standard portant un nom propre. Les menus peuvent gérer des sous-menus et ainsi permettre la construction de menus complexes. Notons que les menus doivent être attachés à une barre de menu (**MenuBar**) qui elle-même ne peut être attachée qu'à un objet **Frame**. Il n'est donc pas possible d'insérer un objet **Menu** ou **MenuBar** dans n'importe quel conteneur.

Un objet **Choice** est un sélecteur d'options, c'est-à-dire un composant permettant à un utilisateur de sélectionner un item parmi une série proposée sous forme d'un menu déroulant. Bien qu'il s'inspire d'un menu normal, un objet **Choice** peut-être ajouté à n'importe quel conteneur.

Comme les menus ne peuvent être insérés que dans un objet du type **Frame**, il doit être crée dans toute applet que désire utiliser des menus. L'applet « MenuExample » (listing 20.5) ouvre une fenêtre séparée du browser. Le résultat est présenté à la FIG. 20.5.

Remark: *La fenêtre contient un bande affichant le message « Warning : Applet Window ». Tous les*

systèmes d'exploitations proposent, si l'utilisateur le souhaite, des messages similaires lorsqu'une applet demande l'ouverture d'une fenêtre. Ceci permet à l'utilisateur de faire la différence entre une fenêtre système venant de sa propre machine, et une fenêtre venant de l'extérieur. Une applet pourrait en effet demander le mot de passe de l'utilisateur, et le renvoyer au site auquel elle appartient.

Listing 20.6 – Applet MenuExample

```
import java.awt.*;
public class MenuExample extends java.applet.Applet {
  public void init() {
    Frame f=new EatFrame();
    f.show();
  }
}

class EatFrame extends Frame {
  EatFrame() {
    setTitle("Example of Menu");
    setLayout(new FlowLayout());
    add(new Label("Type"));
    Choice c=new Choice();
    c.addItem("Chinese");
    c.addItem("French");
    c.addItem("Italian");
    add(c);
    Menu m=new Menu("Who");
    m.add(new MenuItem("Smith"));
    m.add(new MenuItem("Durand"));
    Menu subMenu=new Menu("Company");
    subMenu.add(new MenuItem("PIZZA'60"));
    subMenu.add(new MenuItem("FAST CHINESE"));
    m.add(subMenu);
    m.add("Quit");
    MenuBar mb=new MenuBar();
    mb.add(m);
    setMenuBar(mb);
  }

  public boolean action(Event e, Object arg) {
    if(arg.equals("Quit")) {
      dispose();
      return(true);
    }
    if(e.target instanceof Choice) {
      System.out.println("Type : "+arg);
      return(true);
    }
    if(e.target instanceof Menu) {
      System.out.prinln("Who : "+arg);
      return(true);
    }
    return(false);
  }
}
```

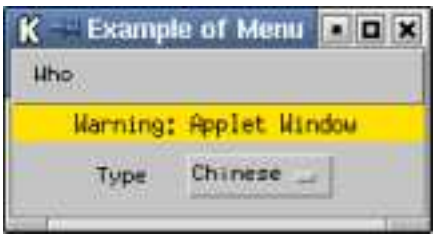


FIG. 20.5 – Applet MenuExample

L'essentiel se passe dans le constructeur de « EatFrame » (lignes 10-30). Tout d'abord la méthode **setTitle()** fixe le nom de la fenêtre. Ensuite, on crée un instance de la classe **Choice** (ligne 14). En utilisant sa

méthode **addItem()**, on ajoute les éléments qui définiront le sélecteur d'options (lignes 15-17). La méthode **getSelectedItem()** permet de récupérer l'élément actuellement sélectionné dans le sélecteur.

Le menu est ensuite créé. Il peut contenir d'autres menus ainsi que des élements de menus (Objets **MenuItem**). Un **MenuItem** est caractérisé par une chaîne de caractères qui définit son titre. Un sous-menu contenant lui-même des éléments a également été attaché. L'entrée «Quit» permet de quitter l'application.

La méthode **action**() (lignes 32-46), traite les trois possibilités suivantes :

1. L'utilisateur a choisi l'élément «Quit» (lignes 33-36). On demande de quitter l'application via la méthode **dispose()**, puis on retourne **true**.
2. L'utilisateur a changé le choix dans le sélecteur d'options (lignes 37-40), on se contente d'afficher à la console Java le choix fait, puis on retourne **true**.
3. L'utilisateur a choisi un élément quelconque du menu (lignes 41-44), on se contente d'afficher à la console Java le nom de l'élément du menu sélectionné, puis on retourne **true**.

Chaque fois qu'un objet du type **Frame** est utilisé, il est conseillé d'ajouter le code suivant au gestionnaire d'événements :

```
public boolean handleEvent(Event e) {
  if(e.id==Event.WINDOW_DESTROY) {
    dispose();
    return(true);
  }
  ...
}
```

Ce code gère l'événement de destruction de la fenêtre qui peut être émis par les gestionnaires de fenêtres de l'environnement, comme lorsque l'option « Fermer » du menu de la fenêtre est sélectionné.

20.6.6 Les cases à cocher

Un objet **Checkbox** est un bouton à bascule avec un nom associé. En attachant à un objet **CheckboxGroup** plusieurs boutons de ce type, on peut les rendre mutuellement exclusifs, c'est-à-dire que la sélection de l'un implique la désélection des autres.

Un tel exemple est repris dans le listing 20.7 dont le résultat est présenté à la FIG. 20.6. L'utilisateur peut sélectionner plusieurs logiciels, mais qu'un seul système d'exploitation.

FIG. 20.6 – Applet CheckExample

Le **Panel** « softPanel » contient l'ensemble des objets **Checkbox** représentant les différents logiciels pouvant être installés. Cela se fait en utilisant la méthode **add()** (lignes 8-10). Le **Panel** p contient quant à lui, l'ensemble des objets **Checkbox**, représentant les différents OS possibles, regroupés au sein d'un

Listing 20.7 – Applet CheckExample

```
import java.awt.*;
public class CheckExample extends java.applet.Applet {
  Panel softPanel=new Panel();
  CheckboxGroup osGroup=new CheckboxGroup();

  public void init() {
    setLayout(new GridLayout(3,1));
    softPanel.add(new Checkbox("Office"));
    softPanel.add(new Checkbox("Mail"));
    softPanel.add(new Checkbox("Browser"));
    add("North",softPanel);
    Panel p=new Panel();
    Checkbox c;
    p.add(c=new Checkbox("Linux"));
    c.setCheckboxGroup(osGroup);
    p.add(c=new Checkbox("Windows"));
    c.setCheckboxGroup(osGroup);
    p.add(c=new Checkbox("Macintosh"));
    c.setCheckboxGroup(osGroup);
    osGroup.setCurrent(c);
    add("Center",p);
    p=new Panel();
    p.add(new Button("Install"));
    add("South",p);
    resize(preferredSize());
  }

  public boolean action(Event e,Object arg) {
    if(!(e.target instanceof Button))
      return(false);
    Checkbox c=osGroup.getCurrent();
    System.out.println("OS : "+c.getLabel());
    Component[] softs=softPanel.getComponents();
    for(int i=0;i<softs.length;i++)
      if((c=(Checkbox)softs[i]).getState())
        System.out.println("Add Software : "+c.getLabel());
    return(true);
  }
}
```

CheckboxGroup « osGroup ». La méthode **setCheckboxGroup()** permet d'associer un objet **Checkbox** à un objet **CheckboxGroup**. Un des choix peut être sélectionné par défaut par l'intermédiare de la méthode **setCurrent()**. Enfin, on rajoute un bouton.

La méthode **action()** (lignes 28-38) ne traite que les événements générés par le bouton. L'objectif est d'écrire sur la console Java le système d'exploitation choisi ainsi que tous les logiciels sélectionnés. Avec la méthode **getCurrent()** de l'objet **CheckboxGroup**, on récupère l'élément sélectionné. On utilise ensuite la méthode **getLabel()** pour inscrire le nom associé à cet événement. On déclare ensuite un tableau de composants auquel on assigne l'ensemble des composants du **Panel** « softPanel » (ligne 33), c'est-à-dire les objets **Checkbox** qu'il contient. On les parcours (ligne 34-36) en testant, après transtypage (ligne 35), s'ils sont sélectionnés, grâce à la méthode **getState()**. Si c'est le cas, on affiche le nom du logiciel.

20.6.7 Utiliser un Canvas

L'objet **Canvas** peut être utilisé comme une zone de dessin. La principale différence entre ce composant et les autres composants présentés jusqu'ici, est qu'il ne possède aucune fonctionnalité par défaut. On peut assimiler un objet **Canvas** à un tableau blanc sur lequel on peut dessiner ce que l'on veut. Pour cela, on redéfinit la méthode **paint()** de cette classe.

Un exemple d'une applet incluant un objet **Canvas** est présenté dans le listing 20.8, dont le résultat peut être vu à la FIG. 20.7.

Dans le constructeur du **Canvas** (lignes 10-13), on précise la couleur de fond avec la méthode **setBackground**. La partie intéressante est la redéfinition de la méthode **paint()**, qui prend comme paramètre un objet du type **Graphics** représentant le contexte graphique où l'on peut dessiner. Trois méthodes de dessin ont été utilisées dans cet exemple :

Listing 20.8 – Applet CanvasExample

```
import java.awt.*;

public class CanvasExample extends java.applet.Applet {
  public void init() {
    add(new MyCanvas(140,90));
  }
}

class MyCanvas extends Canvas {
  MyCanvas(int w,int h) {
    resize(w,h);
    setBackground(Color.white);
  }
  public void paint(Graphics g) {
    g.drawOval(0,0,139,89);
    g.drawRect(60,10,20,79);
    g.drawLine(20,50,30,60);
    g.drawLine(30,60,120,80);
    g.drawLine(120,80,105,40);
    g.drawLine(105,40,20,50);
  }
}
```

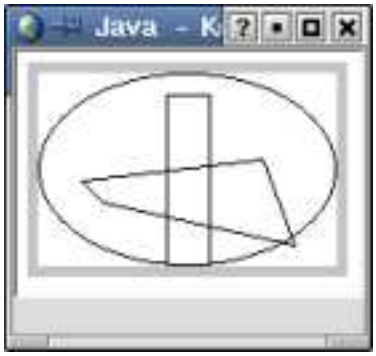


FIG. 20.7 – Applet CanvasExample

- la méthode **drawOval()** qui dessine une ellipse et qui prend comme paramètres les coordonnées représentant le rectangle circonscrit ;
- la méthode **drawRect()** qui dessine un rectangle et qui prend comme paramètres les coordonnées du coin gauche supérieure ainsi que la longueur et la largeur ;
- la méthode **drawLine()** qui permet de dessiner une ligne entre deux points dont les coordonnées sont passées en paramètre.

Remark: *En spécifiant que la taille du **Canvas** à h et w, les coordonnées varient entre [0,h-1] et [0,w-1], ce qui explique que les coordonnées du coin inférieur droit pour l'ellipse soit (139,89) alors que la taille est de (140,90).*

20.7 Les applets et HTML

Pour inclure des applets Java dans un document HTML, on utilise la balise **<APPLET>** dont la forme la plus utilisée est :

`<APPLET CODE=url WIDTH=width HEIGHT=height> ... </APPLET>`

La signification des attributs est la suivante :

CODE URL du fichier contenant le code compilé de l'applet.

WIDTH Largeur de la zone allouée à l'applet.

HEIGHT Hauteur de la zone allouée à l'applet.

Il est possible de placer entre les balises **<APPLET>** du code HTML affiché par le browser si celui-ci n'est pas capable d'exécuter l'applet spécifiée par l'attribut **CODE**.

Le langage HTML permet également à un document de passer des paramètres à l'applet, par l'intermédiaire de la balise **<PARAM>** qui doit se trouver à l'intérieur de la balise **<APPLET>** relative à l'applet à laquelle la balise fait référence. La forme de cette balise est :

```
<PARAM NAME=name VALUE=value>
```

où l'attribut **NAME** permet de définir le nom du paramètre et l'attribut **VALUE** sa valeur.

20.8 Autres caractéristiques de Java

La langage Java propose d'autres caractéristiques dont l'étude complète sortirait du cadre de cet ouvrage. Une petite description des caractéristiques principales sera faite dans les sections suivantes.

20.8.1 Threads

Java est un langage multithread : à un instant donné, il peut y avoir plusieurs threads qui s'exécutent. Un thread est un système d'exécution séparé dans un même programme. En fait, les threads partagent le même espace d'adressage et peuvent donc partager les variables et les méthodes. Les threads sont beaucoup plus économes que les procéssus, il est en effet possible pour une application de lancer plusieurs centaines de threads.

Remark: *Tous les systèmes d'exploitations ne sont pas multithreads. Dans ce cas, les threads sont implantés sous la forme de procéssus et ne sont donc plus aussi avantageux. Ceci dit, il est pourtant conseillé de les utiliser, car lorsque tous les systèmes d'explotations pourront les utiliser de manière optimale, il y aura un vrai gain sans pour autant devoir redévelopper, ou même recompiler, les applications Java.*

Le langage Java utilise déjà plusieurs threads ern interne, comme le «ramasse-miettes» ou encore dans la librairie AWT concernant la méthode **repaint()**.

Le Java propose une classe particulière pour gérer les threads, il s'agit de la classe **Thread**. Il est également possible de créer un thread en utilisant l'interface **Runnable**. Les principales méthodes sont **start()** et **stop()** qui sont appelés pour lancer et arrêter un thread. Pour plus d'information, il faut consulter [31].

20.8.2 Exceptions

Les mots-clés **try** et **catch** permettent de gérer des conditions particulières appelés exceptions. Une exception est un message qui est envoyé, normalement en réponse à une erreur, pendant l'exécution d'une instruction ou d'une méthode. Lorsqu'une condition exceptionnelle se produit, un objet la représentant est créé. Il contient des informations comme le problème ou la condition particulière qui ont fait lever l'exception. On peut comparer les exceptions à des événements, car Java arrête l'exécution là où l'exception est levée, et on dit que l'objet est envoyé («throw» en anglais) par cette partie du code. La section qui reçoit l'objet exception capture («catch» en anglais) l'exception.

La construction du type **try/catch** permet de capturer des exceptions pour une partie de code. Si une exception est levée dans le bloc se trouvant à l'intérieur de l'instruction **try**, Java regarde si l'une des sections **catch** suivantes capture l'exception. Si c'est le cas, le code correspondant à cette section est exécuté. Sinon, l'exception est répercutée via la pile d'appel à la méthode appelante. Si l'exception n'est pas capturée ici, elle est encore envoyée à un niveau supérieur, et ainsi de suite jusqu'à ce qu'elle soit capturée.

Il s'agit donc d'un mécanisme très souple de gestion des erreurs puisque des exceptions se produisant au fin fond du code peuvent être ramenées à la surface pour y être capturées.

Remark: *Les exceptions nécessitent un mécanisme plus «lourd» qu'une simple gestion d'erreurs traditionnelle. En effet, lorsqu'une exception se produit, toutes les variables crées dynamiquement sur le tas dans le bloc où celle-ci s'est produite, devraient probablement être détruites. C'est la raison pour laquelle beaucoup de programmeurs, notamment en C++, hésitent à les utiliser. Ceci dit, la souplesse de celles-ci*

vaut largement la lourdeur associée. De plus, l'implantation des exceptions sera forcément améliorée dans les prochaines années dans les différents langages.

20.8.3 Les entrées-sorties

Parmi les caractéristiques qui diffèrent entre les systèmes d'exploitations, se trouve la structure du système de fichiers, c'est-à-dire la manière dont le système d'exploitation gérent les fichiers et les répertoires. Java étant un langage de programmation complet, il doit donc permettre l'accès à des fichiers et à des répertoires. C'est la raison pour laquelle Java implémente le paquetage **java.io** dont le but est de fournir une interface indépendante permettant la manipulation de fichiers.

La classe **java.io.File** encapsule les accès aux informations concernant les fichiers ou les répertoires. Elle donne des informations sur les fichiers et effectue un certain nombre d'opérations de base comme supprimer un fichier ou crée un répertoire. En fait, l'objet **File** n'accède pas directement aux données en lecture/écritures : ces opérations sont effectuées par des streams. Un stream, dans la terminologie Java, représente un flot de données ou un canal de communications avec une autre entité au bout.

Pour des informations concernant les entrées-sorties avec Java, il faut consulter [9].

20.8.4 La programmation réseau

Le Java a été concu dès le début avec pour objectif de pouvoir être utilisé dans le monde internet. Permettre la programmation réseau de manière indépendante est donc une nécessité pour Java. C'est la raison pour laquelle Java fournit un paquetage **java.net** offrant la possibilité de faire des développements orientés réseau. Les classes de ce paquetage se décomposent en deux catégories :

1. L'ensemble des classes permettant la manipulation de sockets et donc d'utiliser le procotole réseau standard pour faire communiquer des machines. Les sockets sont une interface bas-niveau qui permettent d'envoyer des flots de données entre des applications qui ne sont pas forcément sur la même machine. C'est la base des applications client-serveur.
2. L'ensemble des classes permettant de manipuler les URLs afin d'accéder à certaines ressources réseau bien connues comme des documents ou des applications qui se trouvent sur des serveurs. Avec les URL, une application peut récupérer un fichier complet à partir d'un serveur connecté au réseau en implantant que quelques lignes de code.

Pour trouver des informations sur la programmation réseau, il faut consulter [11].

20.8.5 Java et Java DataBase Connectivity (JDBC)

La plupart des entreprises stockent leur données dans des bases de données telles que Oracle, Sybase SQL Serveur, MySQL ou encore Microsoft SQL Server. L'intérêt d'utiliser les bases de données conjointement avec les technologies liées à internet a vite été clair pour la majorité des gens. Il était donc normal que Java offre une possibilité de travailler avec les bases de données.

En fait, Java profite non seulement aux programmeurs souhaitant interagir avec des bases de données, mais aussi aux créateurs de ces dernières, qui disposent maintenant de fonctionnalités comme :

- la traduction d'objets en unités relationnelles facilitée ;
- l'indépendance vis-à-vis des bases de données ;
- l'informatique distribuée.

Afin de permettre de contrôler l'accès aux bases de données, Java fournit l'interface JDBC. Celle-ci se base sur le Stuctured Query Language (SQL). Le résultat des requêtes est transformé en variables Java qui peuvent ensuite être traitées par le programme sans aucune difficulté. Une étude complète de JDBC peut être trouvée dans [21].

Sixième partie

Programmation coté serveur

Chapitre 21

Programmation CGI

Le Common Gateway Interface (CGI) offre un moyen de construire des sites dynamiques, en fournissant deux interfaces :

- l'une permettant de récupérer des paramètres encodés au niveau du client, c'est-à-dire du browser ;
- l'autre permettant à une application de générer du code HTML qui sera renvoyé au browser comme s'il s'agissait du contenu d'un document.

Il est nécessaire que le serveur web supportent cette interface afin de pouvoir l'utiliser. En pratique, tous les serveurs actuels la fournissent en standard. Pour une description complète, il est conseillé de consulter [8].

21.1 Appel CGI

On peut voir à la figure 21.1 un schéma d'un appel CGI.

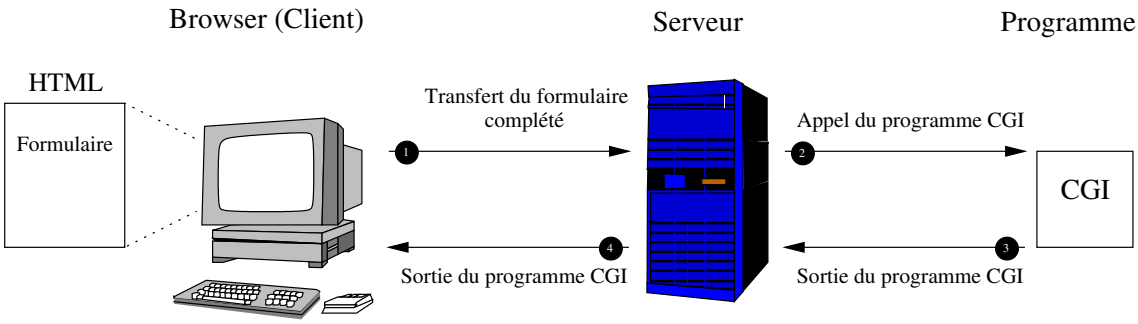


FIG. 21.1 – Schéma d'un appel CGI

On peut décomposer ce schéma en quatre parties :

1. L'utilisateur remplit, via son browser, un formulaire contenant un certain nombre de champs à remplir. Suivant les attributs de la balise **<FORM>** (Voir chapitre 9), le contenu des différents champs sont envoyés au serveur via une URL spécifiée par l'attribut **ACTION**.
2. Si cette URL fait référence à un programme ou à un script CGI, le serveur web effectue l'appel à celui-ci en lui communiquant la valeur des variables.
3. L'application s'exécute, en faisant éventuellement appel à des informations se trouvant dans une base de données située sur le serveur. La sortie de l'application est capturée par le serveur («STDOUT» pour la plupart des langages de programmation).
4. Le serveur renvoit la sortie au client qui va essayer de traiter celle-ci. En particulier, si la sortie est au format HTML, le browser affichera le résultat directement.

21.2 Applications CGI

Concernant les applications CGI, il y a immédiatement deux questions qui se posent :

1. Comment le server web reconnaît-il qu'une URL fait référence à une application CGI ?
2. Quel langage de programmation choisir pour développer des applications CGI ?

21.2.1 Configuration du serveur

Bien que la configuration des différents serveurs ne sont pas les mêmes, il existe une constante concernant la configuration de l'interface CGI : le serveur réserve une série de répertoires (généralement tous dans un répertoire racine «cgi-bin») pour les applications CGI. Le noms des fichiers pouvant prendre des extensions si souhaité (Comme «.exe» sous Windows).

Remark: *Certains systèmes d'exploitations, comme UNIX/Linux, nécessitent de plus que les applications puissent être exécuté par n'importe quel utilisateur du système, puisque l'accès via le protocole HTTP est anonyme.*

21.2.2 Choix du langage de programmation

En fait, le CGI est une interface et non un langage ou en environnement particulier. Dés lors, l'utilisateur peut choisir le langage qu'il lui convient. Généralement, pour une utilisation avec CGI, le programmeur choisit un langage possédant les caractéristiques suivantes :

- puissante manipulation de texte possible ;
- création de liens avec les bibliothèques et utilitaires d'autres logiciels comme des bases de données ;
- accès aux variables d'environnement (sous UNIX).

De manière générale, le programmeur peut utiliser deux approches possibles pour chaque application CGI :

- un programme, c'est-à-dire un code compilé, créé en utilisant des langages comme le Pascal, le C++ ou encore Visual Basic ;
- un script, c'est-à-dire un script qui sera interprété au moment de son exécution, créé en utilisant des langages comme le Perl, AppleScript, Tcl ou encore le «C Shell».

Les programmes s'exécutent généralement plus rapidement que les scripts, mais ils ne sont pas portables, au mieux, il faut chaque fois les recompiler pour les différentes plate-formes sur lesquels tournent les serveurs web. Ceci dit, la majorité des applications utilisant l'interface CGI sont développées en utilisant le langage à script Perl. En effet, outre des fonctions complexes permettant de manipuler du texte, le serveur Apache (Utilisé dans 80% des sites professionels) le comprend en «natif», c'est-à-dire qu'il ne nécessite pas d'interpréteur externe. Pour plus de renseignements sur Perl, il faut consulter [26] et [27].

21.3 Gestion des entrées

Lorsqu'une application CGI est exécuté, le serveur lui communique un ensemble de renseignements tels que :

- des informations concernant le client, le serveur et l'utilisateur ;
- les données introduites par l'utilisateur dans un formulaire ;
- spécification du chemin d'accès.

21.3.1 Manipulation des variables d'environnement

Les variables d'environnement d'UNIX permettent de récupérer des renseignements essentiels dont un programme CGI a généralement besoin. Le tableau 21.1 présente une liste exhaustive de toutes les variables actuellement disponibles.

Afin d'illustrer l'utilisation de ces varaibles, on introduit un petit script Perl qui produit une page HTML contenant quelques informations sur le serveur web utilisé. Il est montré au listing 21.1. Pour comprendre l'application, il suffit de savoir que la commande **print** permet de produire une chaîne de caractères vers

Variable d'environnement	Description
GATEWAY_INTERFACE	Version de CGI mise en oeuvre par le serveur.
SERVER_NAME	Nom ou adresse IP du serveur.
SERVER_SOFTWARE	Nom et version du serveur qui a appelé le programme.
SERVER_PROTOCOL	Nom et version du protocole utilisé lors de la requête.
SERVER_PORT	Numéro du port de la machine cliente.
REQUEST_METHOD	Méthode employée pour la requête.
PATH_INFO	Spécification du chemin d'accés.
PATH_TRANSLATED	Version décodée du contenu de PATH_INFO.
SCRIPT_NAME	Chemin virtuel du script excécuté (Par exemple "/cgi-bin/program.pl").
DOCUMENT_ROOT	Répertoire d'accueil des documents Web.
QUERY_STRING	Contenu de la requête, passé au programme à la suite de l'URL et précédée du signe " ?".
REMOTE_HOST	Nom de la machine de l'utilisateur.
REMOTE_ADDR	Adresse IP de l'utilisateur.
AUTH_TYPE	Méthode d'authentification de l'utilisateur.
REMOTE_USER	Nom authentifié de l'utilisateur.
REMOTE_IDENT	Identification de l'utilisateur d'utilisation de l améthode d'identification RFC 931.
CONTENT_TYPE	Type MIME des entrées (Par exemple "text/html")
CONTENT_LENGTH	Taille des données en octets.
HTTP_FROM	Adresse électronique de l'utilisateur qui est fournie par certains browsers.
HTTP_ACCEPT	Liste des types MIME reconnus par le client.
HTTP_USER_AGENT	Browser utilisé par le client.
HTTP_REFERER	Adresse internet du document précédemment consulté avant l'appel du programme.

TAB. 21.1 – Liste des variables d'environnement CGI

la sortie et que la commande **$ENV** renvoit une chaîne de caractères contenant la valeur de la variable d'environnement passée en paramètre.

Listing 21.1 – Utilisation des varaibles d'environnement

```
#!/usr/local/bin/perl;
2
print "Content-type: text/html\n\n";
4 print "<HTML>\n";
print "<HEAD><TITLE>  Example  CGI  </HEAD></TITLE>";
6 print "<BODY>";
print "Serveur Name: ",ENV$('SERVER_NAME'}, "<BR>\n";
8 print "Serveur Software: ",ENV$('SERVER_SOFTWARE'}, "<BR>\n";
print "</BODY></HTML>"
10 exit(0);
```

On peut facilement déduire le fonctionnement de l'application :
- on imprime d'abord l'en-tête HTTP afin de prévenir le client que le résultat est au format HTML (Ligne 3) ;
- on imprime les balises nécessaire à un document HTML de base (Lignes 4-6 et 9) ;
- on imprime directement du texte en utilisant notamment **$ENV** pour récupérer les informations sur le serveur (Lignes 7-8).

21.3.2 Acquisition des données du formulaire

Suivant la méthode utilisée dans le formulaire, grâce à l'attribut **METHOD** de la balise **<FORM>**, les données encodées dans les différents champs sont acheminées de manière différente vers le serveur :

GET Le nom des variables ainsi que leurs valeurs sont ajoutées derrière l'URL en étant précédés d'un signe « ?».

POST Le serveur reçoit les données sous la forme d'un flux d'entrée.

Dans le cas de la méthode GET, le serveur stocke dans la variable d'environnement **QUERY_STRING** l'ensemble des informations concernant les variables et leurs valeurs. L'application doit donc récupérer le contenu de celle-ci.

Dans le cas de la méthode POST, le serveur stocke dans la variable d'environnement **CONTENT_LENGTH** la taille totale des informations qu'il a reçut. Puis, il les envoit via l'entrée standard («STDIN» dans la plupart des langages) à l'application. L'application doit donc elle-même lire de l'entrée standard les informations.

Remark: *Il est donc généralement nécessaire que l'application décode le contenu afin de retrouver toutes les informations dont il a besoin. Ceci dit, il existe pour la plupart des langages des bibliothèques gratuitement disponibles, fournissant ce genre de service, évitant au programmeur la tâche rébarbative de décoder ces informations.*

21.3.3 Spécification d'un chemin d'accès

En plus des données introduites par l'utilisateur, il est possible de rajouter dans l'URL la spécification d'un chemin d'accès, en l'introduisant à la suite du nom de l'application. Par exemple,

```
http ://site.com/cgi-bin/affiche.pl/cgi/cgidoc.txt
```

Dès que le serveur reconnait le nom d'un programme CGI, ici «affiche.pl», la chaîne correspondante au chemin d'accès, ici «/cgi/cgidoc.txt», est stockée dans la variable d'environnement **PATH_INFO**. Dans le même temps, le serveur met dans la variable d'environnement **PATH_TRANSLATED** le chemin d'accès complet du document, dans le cas présent «/home/httpd/public/cgi/cgidoc.txt».

21.4 Gestion des sorties

L'utilisateur accède à une application CGI comme s'il s'agissait d'un document statique, si ce n'est qu'il s'agisse en réalité d'un document dynamique puisque le même document, c'est-à-dire la même application CGI, peut produire des sorties différentes suivants les circonstances. Tant que le résultat est sensé être du texte simple ou un document HTML, le résultat ressemble à quelque chose de standard, c'est-à-dire que le browser affiche simplement le résultat. Ceci dit, il est possible de faire des choses plus complexes avec le CGI :

- insérer graphiques et autres données binaires ;
- réclamer que le browser place le document dans un cache mémoire ;
- requérir du serveur l'expédition d'un document existant.

21.4.1 En-têtes générés par une application CGI

Le plus simple pour une application CGI est de retourner de l'information sous la forme d'un document HTML. Dans ce cas, la première chose qu'elle doit envoyer est un en-tête de la forme :

```
Content-type : text/html
```

Remark: *Il est nécessaire de laisser un ligne vide, c'est-à-dire de rajouter deux retours de chariot, entre l'en-tête et la suite des informations.*

En fait, l'intitulé **Content-type** référence le type MIME des données qui seront envoyées par la suite. Cependant, il ne s'agit là que d'un en-tête HTTP parmi d'autres, on peut également indiquer :

- la taille des données ;
- le nom d'un document existant ;
- les codes d'état HTTP.

Le tableau 21.2 présente les intitulés HTTP les plus utiles.

Le tableau 21.3 présente les intitulés HTTP introduits par Netscape et actuellement supportés par la plupart des browsers, mais s'ils ne font pas partie de la spécification HTTP officielle.

L'orde dans lequel sont placés les éléments n'a aucune importance.

Intitulé	Description
Content-length	Taille du flux de sortie (en octets), considéré comme étant un binaire.
Content-type	Type MIME du flux de sortie.
Expires	Date et heure de fin de validité du document, lequel devra alors être rechargé à partir du serveur.
Location	Redirection du serveur (ne doit en aucun cas figuré dans un intitulé complet).
Pragma	Document caché / non-caché.
Status	Etat de la requête (ne doit en aucun cas figuré dans un intitulé complet).

TAB. 21.2 – Intitulés HTTP usuels

Intitulé	Description
Refresh	Le client charge à nouveau le document mentionné.
Set-Cookie	Le client stocke les données spécifiées sous forme d'un cookie sur sa machine.

TAB. 21.3 – Intitulés HTTP reconnus par la plupart des browsers

21.4.2 Redirection du serveur

Une application CGI peut très bien provoquer l'accès à un document existant en redirigeant le serveur. Pour cela, on utilise l'en-tête **Location** afin de transmettre au serveur le nom du document à afficher. La figure 21.2 illustre le mécanisme.

Il est important de signaler que l'intitulé **Content-type** ne doit pas apparaître dans l'en-tête pusiqu'elle sera introduite par le serveur vers qui on aura détourner le serveur courant.

21.4.3 Intitulés «Expires» et «pragma»

La plupart des browsers placent automatiquement les documents consultés dans un «cache» afin d'éviter de devoir le recharger systèmatiquement du serveur lorsque l'utilisateur en a besoin. Ceci permet aux documents les plus souvent consultés d'être chargés beaucoup plus rapidement par le browser puisqu'il va chercher ceux-ci sur son propre disque dur et non sur internet. Il est évident que cette technique est très performante concernant les documents statiques, par contre pour les applications CGI, ce n'est peut-être pas une bonne idée de conserver «une fois pour toute» le résultat de celles-ci.

Les intitulés **Expires et pragma** permettent d'éviter ce genre de problème. En rajoutant de l'en-tête HTTP :

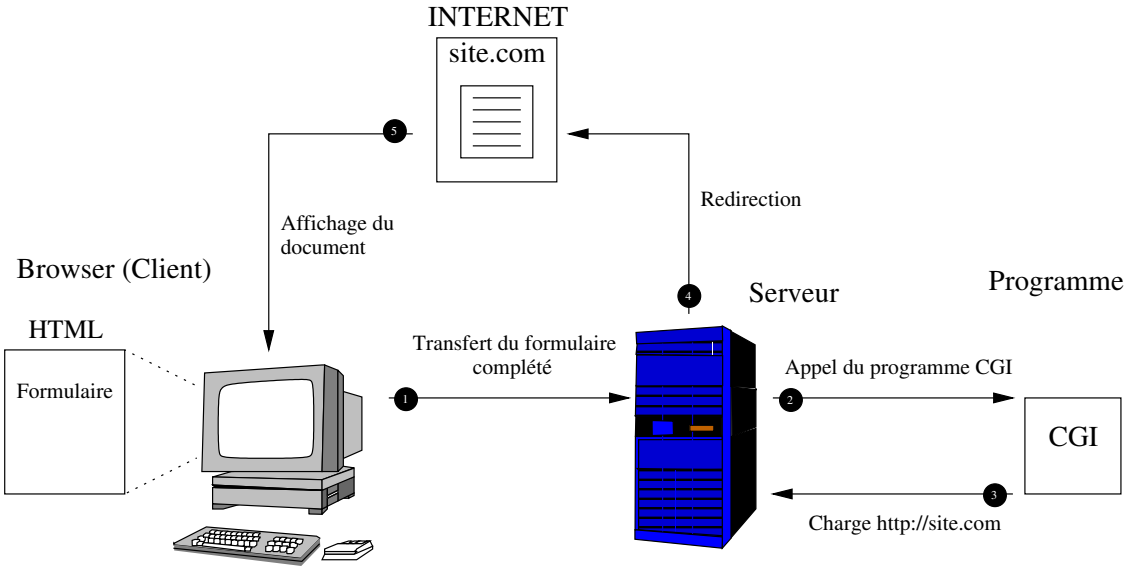


FIG. 21.2 – Redirection du serveur

```
Pragma : no cache
```

On évite que le browser mette le document dans son cache, c'est-à-dire que celui-ci sera systématiquement rechargé à partir du serveur chaque fois que l'utilisateur désirera le consulter.

Par contre, en rajoutant dans l'en-tête HTTP :

```
Expires : Friday 27-Dec-01 05 :29 :10 GMT
```

On spécifie une date à partir de laquelle le document ne sera plus valable. Tant que cette date n'est pas atteinte, le browser pourra utiliser le document se trouvant dans son cache, mais dès que la date est atteinte, il sera forcé de la recharger à partir du serveur.

21.4.4 Codes d'état

Un certain nombre de codes sont utilisés par HTTP afin de représenter l'état de la requête. La sortie d'une application CGI contient parfois ce type de renseignement. L'intitulé **Status** se compose de trois chiffres formant un code d'état, lequel est ensuite décrit dans une chaîne de caractères. Les codes les plus usuels se trouvent repris dans le tableau 21.4.

Code	Message
200	Réuisste
204	Aucune réponse
301	Document déplacé
401	Autorisation non valable
403	Accès interdit
404	Introuvable
500	Erreur système
501	Non implénté

TAB. 21.4 – Codes d'état HTTP

21.4.5 Intitulés complets (Non-Parsed)

Lorsque l'intitulé est imcomplet et comprend uniquement l'indispensable **Content-type**, le serveur mémorise la sortie du programme puis ajoute dans l'intitulé certains renseignements, comme la date et l'heure ainsi que le code d'état «200 OK».

Ceci dit, tout application CGI est en mesure de générer un intitulé exhaustif contenant tous les renseignements nécessaires. Un temps appréciable peut ainsi être gagné car la sortie est directement envoyé au client sans que le serveur ne doive modifier le contenu (Voir figure

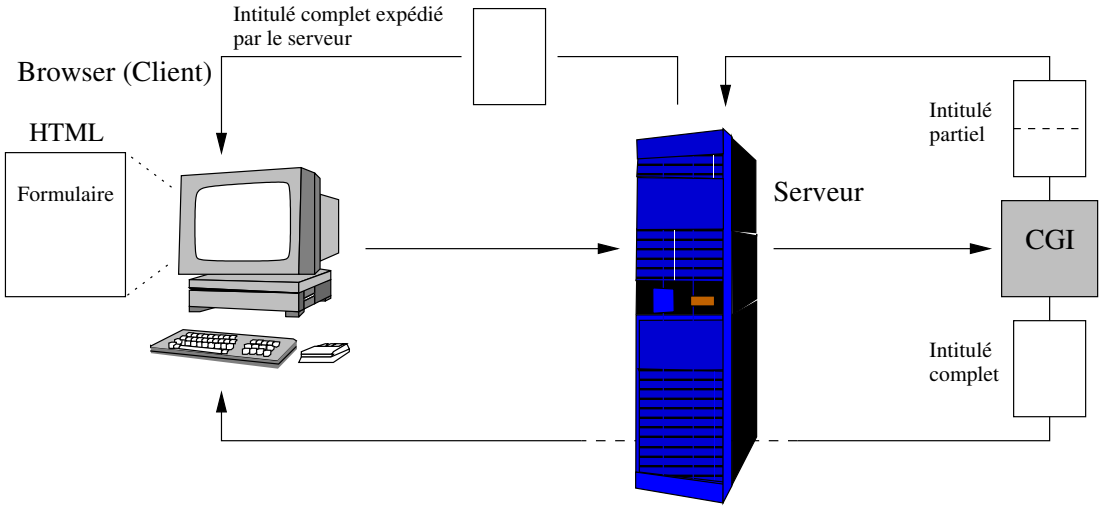


FIG. 21.3 – Intitulé complet / partiel

Chapitre 22

PHP Hypertext Processor

22.1 Introduction à PHP

Le langage PHP Hypertext Processor (PHP) est un langage interprété côté serveur datant de 1994. Il s'agissait à l'époque d'un outil de création de pages personnelles, d'où la première appellation PHP : Personal Home Page. PHP fonctionne à l'intérieur d'un document HTML et lui permet notamment de générer du contenu dynamique répondant aux requêtes de l'utilisateur. Il offre de nombreuses possibilités, en particulier celle de pouvoir se connecter à des bases de données. Il s'agit d'un produit «Open Source», donc gratuit et ouvert à toute modification utile. Il est compatible avec différents systèmes d'exploitation comme Linux ou Windows, avec différents serveurs web (Apache ou IIS) et accède à la plupart des bases de données typiques en natif, comme MySQL par exemple, ou par l'intermédiaire d'Open DataBase Connectivity (ODBC).

Il existe de nombreux sites dédiés à PHP, dont le plus complet est http ://www.php.net. Pour plus de détails concernant PHP, on peut consulter [1] et [12].

22.2 Programmer dans un environnement web

22.2.1 Traitement des documents PHP

On peut voir à la FIG. 22.1 le schema de traitement d'un document PHP.

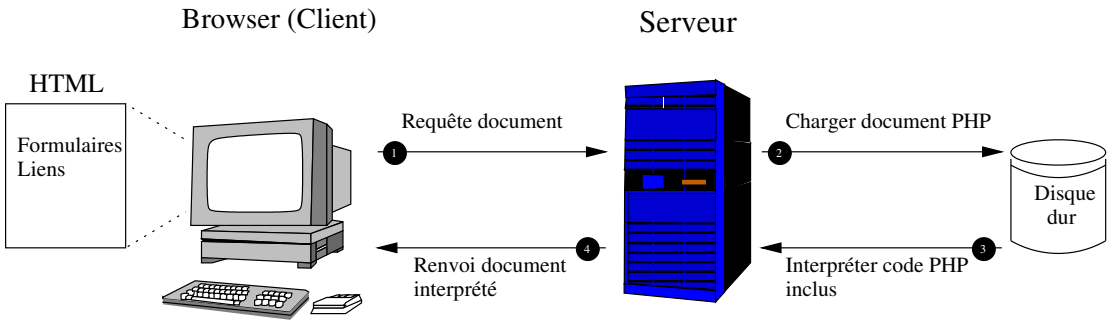


FIG. 22.1 – Schéma de traitement d'un document PHP

Quand un utilisateur désire visualiser un document HTML contenant du code PHP, le browser émet une requête vers le serveur web. Le serveur web procède en quatre étapes :

1. Lecture de la requète du browser.
2. Identification de la page requise.

3. Exécution des instructions PHP pour modifier la page.
4. Envoi de la page au browser via Internet.

Le code HTML pur est donc uniquement interprêté au niveau du browser et pas exécuté sur le serveur. L'utilisation de PHP ouvre de nombreuses portes en matière de sites dynamiques :
- modification facile du contenu d'une page web en intervenant sur des données et pas dans le code HTML ;
- création de pages personnalisées n'intéressant qu'un utilisateur particulier ;
- affichage et actualisation d'une base de données inclue dans une page web ;
- obtenir une réponse de l'utilisateur et renvoyer de l'information en fonction de la réponse.

22.2.2 Insertion de code PHP dans une page HTML

Le code PHP est un script incorporé à une page HTML, exécuté sur le serveur avant d'être envoyé au navigateur. Il existe quatre façons de préciser au serveur qu'il s'agit de code PHP :
- instruction XML :

```
<?php echo ("place du code PHP");?>
```

- instruction SGML :

```
<?echo ("place du code PHP");?>
```

- pour les habitués de JavaScript :

```
<SCRIPT LANGUAGE='php'>
  echo ("place du code PHP");
</SCRIPT>
```

- type ASP :

```
<% echo ("place du code PHP"); %>
```

On peut voir au listing 22.1 un premier exemple de code PHP.

Listing 22.1 – Exemple de code PHP (1)

```
<HTML>
  <?php
    echo("Hello World!!!");
  ?>
</HTML>
```

L'instruction **echo()** produit une sortie vers le navigateur. Il faut aussi noter les points-virgules qui terminent chaque instruction PHP. Si on examine au niveau du navigateur le code source de la page, on obtient ceci :

```
<HTML>
Hello World!!!
</HTML>
```

Le client ne reçoit bel et bien qu'une simple page HTML.

22.2.3 PHP et code côté client

L'instruction **echo()** permet également d'envoyer du code côté client au navigateur comme l'illustre le listing 22.2.

Le navigateur exécute le code JavaScript et affiche donc une boîte d'alerte.

Listing 22.2 – Exemple de code PHP (2)

```
<HTML>
  <?php
    echo("<SCRIPT Language='JavaScript'> alert('Hello World!!!'); </SCRIPT>");
  ?>
</HTML>
```

22.2.4 Variables PHP et formulaires

PHP est un langage faiblement typé. Il n'est donc pas nécessaire de déclarer les variables, ni de définir le type de données qu'elles contiennent avant de les utiliser. Le type de données peut donc changer si le contenu en est modifié. Les variables sont toujours précédées du caractère $, ce qui permet à PHP de les différencier des autres éléments du langage. L'exemple du listing 22.3 donnera le même résultat que celui du listing 22.1.

Listing 22.3 – Exemple de code PHP (3)

```
<HTML>
  <?php
    $world="Hello World!!!";
    echo($world);
  ?>
</HTML>
```

Dans les premières versions, PHP était essentiellement un langage de manipulation des données provenant des formulaires. Afin de faciliter l'accès au contenu des champs des formulaires, PHP déclare pour chacun d'eux une variable dont le nom est celui défini par l'attribut **NAME** de l'élément auquel vient s'ajouter un '$' comme toutes les variables PHP.

Prenons comme exemple le formulaire représenté à la FIG. 22.2 et dont le code est présenté au listing 22.4. On voit que lorsque l'utilisateur appuye sur le bouton «OK», le formulaire est envoyé à un document PHP[1].

Listing 22.4 – Exemple de formulaire

```
<HTML>
  <BODY>
    <FORM ACTION="res.php3" METHOD="POST">
        Introduire un message :
        <INPUT TYPE="TEXT" NAME="msg">
        <INPUT TYPE="SUBMIT" VALUE="Ok">
    </FORM>
  </BODY>
</HTML>
```

Il est possible de récupérer le contenu du champ simplement en utilisant la variable créée par PHP comme le montre le listing 22.5. Le résultat peut être visualisé à la FIG. 22.3.

22.2.5 Commentaires et caractères d'échappement

Commenter son code est une règle de bonne pratique qui permet d'aider au débogage et à éclaircir le fonctionnement d'un programme. L'ordinateur ignore totalement les commentaires.

- Une ligne ou portion de ligne de commentaire sera définie comme commentaire à la rencontre d'une double barre oblique (//) ou d'un dièse (#).
- Un commentaire multiligne est ouvert par /* et terminé par */ comme en C/C++.

Les séquences d'échappement permettent d'afficher des symboles spéciaux pour certaines tâches particulières, comme par exemple le signe '$' qui est utilisé pour les noms de variables. A l'intérieur des guillemets

[1] Les documents contenant du code PHP ont généralement le suffixe «.php3» pour la version 3 ou «php» pour la version 4.

FIG. 22.2 – Exemple de formulaire

Listing 22.5 – Code PHP traitant les formulaires

```
<HTML>
  <BODY>
      <?php
      echo("Message: ");
      echo($msg);
    ?>
  </BODY>
</HTML>
```

(comme dans la fonction **echo**()), il s'agit de placer des apostrophes par exemple pour écrire une chaîne de caractères. Quand on veut écrire des apostrophes ou des guillemets à l'intérieur d'un texte, il faut utiliser une barre oblique inverse pour qu'ils ne soient pas compris comme le début d'une chaîne : \' et \". Les différentes séquences d'échappement et leur signification sont reprises dans le TAB. 22.1.

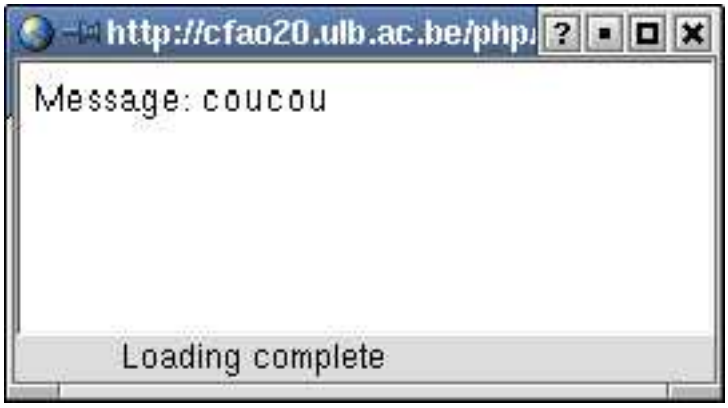


FIG. 22.3 – Code PHP traitant les formulaires

TAB. 22.1 – Séquences d'échappement

Séquence	Description
\"	Guillemet
\'	Apostrophe
\\	Barre oblique inverse
\$	Signe dollar
\n	Nouvelle ligne
\r	Retour chariot
\t	Tabulation

22.3 Les élements de PHP

22.3.1 Variables, constantes et types de données

22.3.1.1 Types fondamentaux

PHP possède trois types de données fondamentaux : entier, double et chaîne de caractères.

1. L'entier utilise quatre octets de mémoire et sert à représenter un chiffre ordinaire dépourvu de décimale dans la plage de -2^{31} à 2^{31}. Exemple : 2.
2. Le double sert à représenter une valeur possédant des décimales et un exposant. Exemple : 2.0
3. La chaîne de caractère représente du texte, y compris les caractères de ponctuation et numériques. Exemple : "2" ou "2 heures"

PHP, au contraire de nombreux autres langages, ne possède pas de type booléen (VRAI ou FAUX)[2]. Les trois autres types se chargent de le suppléer :

1. Pour les entiers, 0 représente FAUX alors que toute autre valeur représente VRAI.
2. Il en va de même pour les doubles, où 0.0 et ses équivalents (0.000,...) représentent FAUX.
3. La chaîne vide ("") correspond aussi à FAUX.

22.3.1.2 Identificateurs de variables

Ils sont sensibles à la casse : $world et $WORLD sont deux variables différentes. Par contre, les fonctions et structures internes du langage ne le sont pas, on peut donc indifféremment utiliser **echo()** ou **ECHO()**. Les identificateurs peuvent être composés de n'importe quelle suite de lettres, de chiffres, de traits de soulignement ou signes dollar, mais ne peuvent pas débuter par un chiffre.

22.3.1.3 Constantes

La fonction **define()** permet de créer une constante :

```
ex : define("SOCIETE", "Microsoft"); define("VERSION", 3);
```

La fonction **defined()** permet de vérifier si une constante existe ou non. Le TAB. 22.2 donne la liste des constantes définies par défaut dans PHP.

Remark: *Ces informations sont également disponibles en exécutant la fonction **phpinfo()**.*

22.3.1.4 Déclaration et initialisation de variables

Une variable est directement déclarée en PHP dès qu'une valeur lui est affectée (avec le signe =, à noter que l'opérateur d'égalité est ==).

[2] En tout cas jusqu'à la version 3 de PHP. En effet, dans la version 4, un type bouléen a été introduit.

TAB. 22.2 – Constantes internes à PHP

Constante	Signification
TRUE	Vrai
FALSE	Faux
PHP_VERSION	Donne le numéro de version de l'analyseur PHP en cours d'exécution
PHP_OS	Indique le système d'exploitation côté serveur
FILE	Nom du script en cours
LINE	Numéro de ligne courant à l'intérieur du script

22.3.1.5 Conversion de types et transtypage

Le type d'une variable est déterminé en fonction de la valeur affectée.

```
$a = 1; // $a est un entier
$a = 1.2; // $a est devenu un double
$a = "A"; // $a est une chaîne
```

22.3.1.6 Conversions de chaînes de caractères

Quand on effectue une opération numérique sur une chaîne, PHP évalue la chaîne en tant que nombre comme le montre l'exemple suivant :

```
$adr = "50 avenue F.D.Roosevelt";
$x = $adr + 10; // $x prend la valeur 60, $adr est évalué comme 50 (car début de chaîne)
```

Il est également possible de modifier volontairement le type d'une variable (transtypage) :

```
$a = 11.2; // $a est un double
$a = (int) $a; // $a est un entier = 11
$a = (double) $a; // $a redevient un double (11.0)
$a = (string) $a; // $a devient une chaîne ("11")
```

Les principales fonctions utiles pour les variables sont énumérées au TAB. 22.3.

TAB. 22.3 – Quelques fonctions utiles

Fonction	Description
gettype()	Détermine le type d'une variable
settype()	Définit de façon explicite le type d'une variable
isset()	Détermine si une variable possède une valeur
unset()	Détruit une variable pour libérer la mémoire qui lui est associée
empty()	Renvoie true si la variable n'est pas définie (opposé de isset())
is_int(), is_double(), is_string()	Déterminent si la variable est du type indiqué
intval(), doubleval, strval()	Renvoie une valeur forcée entière, double ou string d'une variable

22.3.2 Opérateurs

22.3.2.1 Opérateurs arithmétiques

Il s'agit des opérateurs classiques : +, -, *, /, %
Ce dernier donne le reste de la division entière d'un nombre par un autre (modulo).

22.3.2.2 Opérateurŝ unitaires

Le signe moins sert également à modifier le signe d'une variable :

```
$a= -2;
$b= -$a; // $b = 2
```

22.3.2.3 Opérateurs de comparaison

Ils servent à tester une condition dont le résultat sera **TRUE** ou **FALSE**. Ils sont repris dans le TAB. 22.4.

TAB. 22.4 – Opérateurs de comparaison

Opérateur	Définition
==	Egalité
>	Supérieur à
>=	Supérieur ou égal à
<	Inférieur à
<=	Inférieur ou égal à
!=	Différent de
<>	Différent de

Remark: *Il ne faut pas confondre opérateur d'affectation = et de comparaison == (égalité).*

22.3.2.4 Opérateurs logiques

Le TAB. 22.5 montre les principaux opérateurs logiques.

TAB. 22.5 – Opérateurs logiques

Opérateur	Définition
&&	Et
and	Et
||	Ou
or	Ou
xor	Ou exclusif
!	Négation

22.3.2.5 Opérateur de concaténation de chaînes

Il s'agit du point. Cet opérateur peut être très utile pour assembler de longues chaînes, comme des requêtes de bases de données, morceau par morceau.

```
$nom = "John";
$prenom = "Smith";
$nomcomplet = $nom . " " . $prenom;
```

Remark: *Il est possible d'isoler un nom de variable en utilisant les accolades {} (ex : ${nom}).*

22.3.2.6 Opérateur ternaire ou conditionnel

Ternaire signifie que l'opérateur exerce son action sur trois opérandes. Il n'existe qu'un seul opérateur ternaire dans PHP inspiré du C/C++. Il se construit comme suit

```
a ? b : c
```

- a est une condition booléenne placée avant le point d'interrogation ;
- b est la valeur renvoyée si la condition est vérifiée (**TRUE**) ;
- c est la valeur renvoyée dans le cas contraire (**FALSE**) ;

Exemple :

```
$a == 0 ? "variable a est nulle" : "variable a non nulle"
```

22.3.2.7 Opérateur binaires

Les opérateurs binaires sont listés dans le TAB. 22.6. Ils ne sont pas très souvent utilisés en pratique.

TAB. 22.6 – Opérateurs binaires

Opérateur	Définition
&	Et
|	Ou
^	Ou exclusif
~	Non
>> x	Décale les bits de x positions vers la droite
<< x	Décale les bits de x positions vers la gauche

Exemple :

```
$a >> $b // décale les bits de $a de $b places vers la droite
```

22.3.3 Instructions

Comme dans tous les langages, le PHP propose un jeux d'instructions de base :
- des instructions conditionnels : **if…else**, **swith** ;
- des instructions de boucles : **for**, **while**, **do…while**.

22.3.3.1 Les instructions if…else

Cette instruction **if** est bien connue, puisque c'est quasi la même dans la plupart des langages de programmation. La syntaxe est :

```
if($pays=="be")
  echo("Belgique");
```

Si plusieurs instructions doivent être exécutées après le **if**, on les place entre accolades.

```
if($pays=="be")
{
  echo("Pays : ");
  echo("Belgique");
}
```

Le branchement conditionnel existe également. Il s'agit du **else** qui, placé après le **if** permet d'exécuter du code quand la condition n'est pas remplie.

```
if($a<0)
  echo("Négatif")
else
  echo("Positif ou nul");
```

Le mot-clé **elseif** permet de tester des conditions alternatives si celle du **if** n'est pas remplie.

```
if($a<0)
  echo(" Négatif")
elseif($a==0)
  echo("Nul")
else
  echo("Positif");
```

PHP propose aussi la structure suivante sans accolades pour faciliter l'imbrication de code HTML :

```
if(cond1) :
  opération1;
elseif(cond2) :
  opération2;
elseif(cond3) :
  opération3;
else :
  opération4;
endif;
```

22.3.3.2 L'instruction swith

L'instruction **switch** permet d'évaluer une expression et d'exécuter différentes parties de code en fonction du résultat de cette évaluation.

```
switch($pays)
{
  case "be" :
    echo("Belgique");
    break;
  case "ca" :
    echo("Canada");
    break;
  case "de" :
    echo("Allemagne");
    break;
  default :
    echo("Autre pays");
}
```

Les instructions comprises entre **case** et **break** sont exécutées quand la condition correspondante sur «$pays» est vérifiée. L'instruction **break** arrête donc l'exécution du code sans quoi, dès que la condition est vérifiée, le reste du code compris entre les accolades sera exécuté. L'instruction **default** est facultative. Elle s'exécute si aucune correspondance n'a été trouvée au-dessus.

22.3.3.3 L'instruction while

Dans la boucle **while**, le code est exécuté tant qu'une condition est vérifiée :

```
while(condition)
{
  // instructions;
}
```

Il est possible d'arrêter la boucle de façon précoce avec l'instruction **break**. On peut aussi mettre fin à une itération précise mais sans arrêter la boucle avec l'instruction **continue**. Comme dans le cas de l'instruction **if**, il existe une syntaxe adaptée à l'imbrication de blocs HTML :

```
while(condition) :
  instructions;
endwhile;
```

22.3.3.4 Les instructions do…while

Ces instructions sont similaires aux boucles **while** si ce n'est que la condition est vérifiée à la fin de chaque itération et pas au début.

```
do {
  instructions;
} while(condition);
```

22.3.3.5 L'instruction for

La boucle **for** se répète jusqu'à ce qu'une condition soit évaluée comme fausse. La forme générale est :

```
for([expression-initial];[condition];[expression-incrémentation]) {
  instructions;
}
```

Le déroulement d'une boucle **for** est le suivant :

1. L'expression *expression-initial*, si elle existe, est exécutée. Généralement, on initialise des compteurs de boucles.
2. La *condition* est évaluée. Si le résultat est vrai, alors la boucle continue, sinon elle s'arrête.
3. Les instructions dans la boucle sont exécutées.
4. L'expression *expression-incrémentation* est exécutée, et la boucle retourne à l'étape 2.

L'exemple suivant montre une boucle affichant les dix premiers nombres :

```
for($i=0;$i<10;++$i) {
  echo($i);
}
```

La syntaxe adaptée à HTML existe également :

```
for([expression-initial];[condition];[expression-incrémentation]) :
  instructions;
endfor;
```

22.3.4 Incorporer un fichier dans du code PHP

Ceci se fait grâce aux instructions **require()** et **include()**.

– l'instruction **require()** est remplacée par le contenu du fichier spécifié. Le seul inconvénient de cette instruction, est qu'elle ne peut être utilisée à l'intérieur d'une boucle afin d'appeler un fichier différent lors de chaque itération. Ce problème est résolu par l'instruction **include()** ;
– l'instruction **include()** accède à un fichier externe mais elle évalue et exécute le code à chaque rencontre de celle-ci plutôt que de remplacer l'instruction par le code contenu dans le fichier externe lors de la première exécution.

Ces fichiers externes sont recherchés dans le répertoire mentionné dans la directive **include_path** du fichier «php.ini». Deux autres directives sont également en rapport avec ce sujet : **auto_prepend_file** et **auto_append_file**. Elles permettent de faire des "require" automatiquement à un fichier externe au début ou à la fin de tout fichier PHP.

22.3.5 Quitter un document PHP

Si une erreur sérieuse se produit, il peut être utile de quitter le document. Ceci se fait à l'aide de l'instruction **exit()**.

22.3.6 Fonctions

22.3.6.1 Structure des fonctions

Une fonction est un bloc de code défini une fois pour toutes et qui peut être invoqué à d'autres endroits du programme. En général, elle reçoit un ou plusieurs arguments qui vont subir des opérations et renvoie une valeur en résultat. PHP dispose comme nous l'avons vu plus haut de fonctions pré-définies. Nous allons ici apprendre à créer de nouvelles.

Prenons l'exemple d'une déclaration d'une fonction recevant un nombre et renvoie le cube de ce nombre :

```
function cube($nb)
{
  return($nb*$nb*$nb);
}
```

L'appel à la fonction se fait tout simplement en invoquant son nom avec l'argument souhaité :

```
echo(cube(6)); // affiche 216
```

La structure d'une fonction est donc la suivante :

```
function nom_fonction(paramètres)
{
  opérations effectuées avec les paramètres;
  return valeur_renvoyée;
}
```

Lors de l'exécution de **return**, la fonction s'arrête et l'exécution revient à la ligne l'ayant invoquée (elle substitue à l'appel la valeur renvoyée par **return** s'il y en a une). Les arguments constituent le moyen d'envoyer des données que la fonction doit traiter, comme par exemple 6 dans l'appel «cube(6)». *$nb* est remplacé par 6 dans le corps de la fonction.

Par défaut, les arguments sont transmis par valeur, c'est-à-dire que la variable paramètre interne à la fonction contient une copie de la valeur qui a été transmise. Il est également possible de passer un élément par référence. Ceci est effectué en plaçant une esperluette (&) avant le nom du paramètre (ex : **function** cube (&$nb)). Toute modification apportée à la variable paramètre modifie la variable de l'instruction d'appel.

On peut encore affecter une valeur par défaut à un paramètre. Ceci permet de rendre l'argument facultatif, comme dans l'exemple :

```
function cube($nb=2)
{
  return($nb*$nb*$nb);
}
echo(cube()); // affiche 8
```

Les paramètres possédant des valeurs par défaut doivent être placés à droite des autres.

22.3.6.2 Portée des variables

A l'intérieur d'une fonction, une variable possède une portée locale. Les variables extérieures à une fonction possèdent une portée globale : elles peuvent être atteintes et modifiées par tout fragment du code du script hors fonction. Pour accéder à une variable globale à l'intérieur d'une fonction, on se sert de l'instruction **global**.

```
function nom_fonction()
{
  global $var;
  echo($var);
}
```

Il est également possible d'avoir recours au tableau intégré **$GLOBALS** qui contient toutes les variables globales du script.

```
function nom_fonction()
{
  echo($GLOBALS["var"]);
}
```

Remark: *Il est important de remarquer qu'il ne faut pas rajouter un signe '$' pour accéder à une variable globale via le tableau **$GLOBALS**.*

La notion de durée de vie est également importante dans l'utilisation des variables. Par défaut les variables créées à l'intérieur d'une fonction sont détruites, et perdent donc leur valeur, lorsque la fonction est quittée. Il peut être souhaitable qu'une variable garde sa valeur d'un appel à l'autre de la fonction dans laquelle elle est définie. Il faut pour se faire définir une variable statique.

```
function nom_fonction()
{
  static $nb = 0;
  ++$nb;
}
```

La valeur de *$nb* sera dès lors conservée entre deux invocations. Lors du premier appel, la variable statique va être créée et se voir affecter la valeur spécifiée par défaut (0 dans le cas ci-dessus). A chaque appel suivant, elle récupérera la valeur sans être réinitialisée.

22.3.6.3 Fonctions imbriquées et récursivité

Même quand des fonctions sont imbriquées, elles sont globales au script. On peut donc faire appel indépendemment à des fonctions faisant partie d'autres fonctions à n'importe quel endroit du script.

La récursivité est le résultat de l'invocation d'une fonction par elle-même. Elle sert à décomposer un problème (souvent mathématique) en opérations élémentaires. Au niveau de la vitesse d'exécution, il est plus efficace d'avoir recours aux boucles qu'à la récursivité.

22.3.6.4 Affectation d'une fonction à une variable

PHP permet à une variable de faire référence à une fonction. Dans l'exemple suivant cela peut être très pratique, soit l'URL d'une page à charger contenue dans la variable *$URL* :

```
switch($type_navigateur)
{
  case "NN" : // Netscape Navigator
    $fonction_charge = "load_nn";
    break;
  case "IE" : // Internet Explorer
    $fonction_charge = "load_ie";
    break;
  default :
    $fonction_charge = "load_generic";
}
// Appel de la fonction
$fonction_charge($URL);
```

22.3.7 Tableaux

Un tableau est composé d'éléments contenant chacun une valeur et un indice. En PHP, les éléments d'un tableau ne doivent pas tous être du même type. L'initialisation d'un tableau peut se faire à l'aide des indices (séquentiels ou pas) ou sans indices auquel cas ceux-ci sont attribués à partir de 0. Pour dénombrer les éléments d'un tableau, on utilise la fonction **count()**.

– avec indices :

```
$element[0] = "bonjour";
$element[1] = "salut";
```

– sans indices :

```
$element[] = "bonjour";   // indice = 0
$element[] = "salut";    // indice = 1
```

– indices non séquentiels :

```
$element[47] = "bonjour";
$element[21] = "salut";
```

TAB. 22.7 – Fonctions de manipulation des tableaux

Fonction	Description
current()	Retrouve la valeur de l'élément actif d'un tableau
key()	Retrouve la clé de l'élément actif d'un tableau
each(), list()	Permettent de parcourir un tableau
next(), prev()	Servent à se déplacer dans un tableau
reset()	Ramène le pointeur interne sur le premier élément
array_walk()	Appliquer une fonction donnée à chaque membre d'un tableau

Un tableau peut également s'initialiser avec le constructeur **array()** :

```
$element = array("bonjour", "salut");
```

Dans le TAB. 22.7, on peut retrouver quelques unes des fonctions les plus employées avec les tableaux.

Il est possible d'indexer des tableaux à l'aide de chaînes de caractères utilisées comme clés de tri :

```
$element["no1"] = "salut"
```

PHP permet encore de créer des tableaux multi-dimensionnels et contient plusieurs fonctions de tri affectant les tableaux (**sort()**, **asort()**, **rsort()** , **arsort()**, **ksort()**, **krsort()**, **usort()**, **uasort()**, **uksort()**, **?**). Toutes ces fonctions sont détaillées dans les références disponibles dans la bibliographie ainsi que sur les sites traitant du PHP.

On peut enfin ajouter que les tableaux sont particulièrement intéressants pour récupérer des données provenant de formulaires HTML où le nombre de données est susceptible de varier. Au lieu d'introduire comme nom du champ d'un formulaire un nom de variable comme nom1, on utilisera nom[]. Ceci peut s'avérer très intéressant pour introduire des données dans une table par exemple.

22.3.8 Programmation orientée objet

Les trois concepts de la programmation orientée objet (abstraction, encapsulation, héritage) sont expliqués dans l'annexe A. Les classes renferment des propriétés et des méthodes relatives à un objet. En PHP, une classe est définie par l'instruction **class**. Les propriétés sont définies avec le mot-clé **var** et elles sont initialisées ou non. Il faut remarquer que PHP ne permet pas les constructeurs multiples.

```
class nom_classe
{
  // propriétés
  var $hote= "localhost";
  var $erreur;
  // méthodes
  function nom_classe{}// constructeur
}
```

Chaque instance d'une classe possède un nom et le pointeur d'accès aux membres est -> comme en C++ et à l'inverse de VB ou Java où le point est utilisé.

On écrit donc :

```
$nom_objet->nom_propriété
```

La création d'une instance se fait grâce au mot clé **new**. En PHP, toutes les méthodes et les objets sont publics. PHP ne possède pas de destructeurs en tant que tels mais il est toujours possible d'utiliser la fonction **unset()** dont nous avons déjà parlé.

L'héritage permet de créer une nouvelle classe à partir d'une classe existante. L'instruction **extends** permet de définir une classe enfant.

```
class nom_classe_enfant extends nom_classe
{
  // propriétés complémentaires
  // méthodes complémentaires
}
```

22.3.9 Manipulation de chaînes de caractères et expressions régulières

Nous n'allons pas rentrer dans le détail des fonctions de manipulation de chaînes de caractères car elle sont extrêmement bien détaillées sur les sites et dans les ouvrages de référence. On peut voir au TAB. 22.8, les principales fonctions. Il faut noter qu'en PHP, le premier caractère est le numéro 0. Donc, pour extraire les deux premiers caractères d'une chaîne de caractères, il faut utiliser :

```
$day=substr($date,0,2);
```

TAB. 22.8 – Fonctions de chaînes simples

Fonction	Description
substr()	Renvoie un fragment de chaîne
trim()	Elimine tout caractère vide excédentaire au début ou à la fin d'une chaîne
chr()	Renvoie le caractère correspondant à un code ASCII
ord()	Contraire de chr()
strlen()	Renvoie le nombre de caractères d'une chaîne
printf()	Produit une chaîne mise en forme vers le navigateur
sprintf()	Produit une chaîne mise en forme sans envoi vers le navigateur
number_format()	Produit une mise en forme de nombres

Les expressions régulières quant à elles, constituent une méthode évoluée de manipulation et de comparaison de chaînes mais sont souvent assez complexes. L'annexe B propose une brève introduction aux expressions régulières. Les fonctions de PHP qui s'y rapportent sont reprises succinctement dans le TAB. 22.9.

TAB. 22.9 – Fonctions pour expressions régulières

Fonction	Description
ereg(), eregi()	Recherches d'identification
ereg_replace(), eregi_replace()	Recherche et remplacement
split()	Sert à fractionner des expressions régulières
sql_regcase()	Sert à créer des expressions régulières insensibles à la casse

22.4 Manipulation de fichiers et stockage de données

22.4.1 Gestion de fichiers

Il est évidemment possible d'accéder à des fichiers grâce à PHP. Le TAB. 22.10 présente les principales fonctions susceptibles d'aider l'utilisateur à traiter des fichiers.

Il y a bien entendu également bon nombre de fonctions qui permettent la gestion des répertoires, comme montre le TAB. 22.11.

Une syntaxe orientée objet est disponible pour manipuler des répertoires. L'objet correspondant appartient à la classe **dir**. Les trois méthodes associées sont : **read()**, **rewind()**, **close()**. Les propriétés sont **handle** et **path**.

22.4.2 Bases de données non relationnelles

Ce type de bases de données, ne connaissant pas les relations entre tables est géré par une interface appelée DBA (database abstraction) et capable de travailler avec des formats de bases de données différents. Les données sont stockées sous forme de paires clé/valeur. Le gros avantage de ces bases de données est qu'elles se composent de fichiers et ne nécessitent pas de logiciel supplémentaire pour être exploitées.

TAB. 22.10 – Fonctions de gestion des fichiers

Fonction	Description
fopen()	Ouverture de fichiers sur le serveur local ou via HTTP ou FTP sur Internet
fclose()	Fermeture de fichier
fpassthru()	Envoie le contenu du fichier vers le flux de sortie
fread()	Permet de lire une chaîne dans un fichier ouvert
fgetc()	Permet de lire un seul carcatère précis dans un fichier
fgets()	Permet de lire une chaîne de longueur spécifique
fgetss()	Même effet que fgets() mais retire les balises HTML
file()	Lit le contenu d'un fichier et le place dans un tableau
fputs() ou fwrite()	Ecrivent une chaîne dans un fichier
rewind()	Remet le pointeur au début du fichier
fseek()	Déplace le pointeur vers une position spécifique
ftell()	Donne la position de l'indicateur dans un fichier
feof()	Indique si le pointeur a atteint la fin du fichier
copy()	Permet de copier un fichier
unlink()	Supprime un fichier
rename()	Renomme un fichier
file_exists()	Renvoie true si le fichier existe

TAB. 22.11 – Fonctions de gestion des répertoires

Fonction	Description
chdir()	Définit le répertoire courant
opendir()	Ouvre un répertoire
readdir()	Retourne le nom de la prochaine entrée du répertoire
rewinddir()	Revient au premier élément du répertoire
closedir()	Ferme le répertoire
mkdir()	Permet de définir un nouveau répertoire
rmdir()	Supprime un répertoire (si celui-ci est vide)

Ceci dit, elles sont rarement utilisées, car les sociétés et les organismes utilisent des bases de données relationnelles capables de gérer de manière efficace des grands volumes de données.

22.5 PHP et les bases de données relationnelles

Pour une introduction sur les bases de données relationnelles et le Stuctured Query Language (SQL), il faut consulter l'annexe C.

Bien que PHP prenne en charge des APIs donnant accès à un grand nombre de bases de données comme Sybase, Oracle et permette également de passer par ODBC (Open Database Connectivity) pour créer des applications plus génériques, on va se focaliser dans le cadre de cet ouvrage sur MySQL. En effet, cette base de données est non seulement rapide et capable de gérer des grandes tailles, mais de plus celle-ci est disponible gratuitement (Voir le site http ://www.mysql.org). Actuellement, près de 80% de sites web utilisent MySQL comme base de données. Pour une référence complète concernant MySQL, il faut consulter [25].

22.5.1 Utilité des bases de données

Pour répondre au mieux aux attentes des utilisateurs d'un site, il peut être utile d'afficher dynamiquement uniquement l'information dont il a besoin ainsi que de permettre l'utilisation d'informations en constante évolution. Pour profiter au maximum des possibilités des bases de données relationnelles, on utilise le moteur mettant en oeuvre la spécification standard du langage de requête structurée : Stuctured Query Language.

Trois éléments sont dès lors indispensables pour créer un site dynamique avec lien vers une base de données :

– un serveur web configuré de manière appropriée (par exemple, Apache) ;
– un serveur de bases de données (par exemple MySQL) ;
– un language tel que PHP pour intégrer dynamiquement ces données au site Web.

PHP propose une série de fonctions permettant de manipuler les bases de données. Comme expliqué, nous allons étudier l' API de PHP pour les accès à MySQL. Les instructions les plus usitées de l'API MySQL sont reprises dans le TAB. 22.12.

TAB. 22.12 – Fonctions de l'API MySQL de PHP

Fonction	Description
mysql_connect	Crée une connection à un serveur MySQL
mysql_pconnect	Crée une connection persistante
mysql_close	Met fin à la connexion (inopérante avec mysql_pconnect)
mysql_create_db	Crée une base de données sur le serveur MySQL
mysql_drop_db	Supprime une base de données
mysql_select_db	Sélectionne une base de données pour l'activer
mysql_query	Envoie au serveur une instruction SQL qu'il exécute
mysql_db_query	Envoie au serveur une instruction SQL avec le nom de la DB active
mysql_list_dbs	Liste les base de données disponibles sur le serveur
mysql_list_tables	Liste les tables d'une base de données
mysql_num_rows	Retourne le nombre de lignes concernées par une query ou une liste
mysql_tablename	Extrait le nom de la table/base de données contenu dans une liste
mysql_list_fields	Extrait les informations relatives à une table
mysql_num_fields	Extrait le nombre de champs d'un jeu de résultats (query ou liste)
mysql_field_len	Extrait la longueur d'un champ
mysql_field_name	Extrait le nom d'un champ de la base de données
mysql_field_type	Retrourne le type de données d'un champ
mysql_field_flags	Extrait les drapeaux d'un champ
mysql_field_table	Extrait le nom de la table à laquelle apparatient un champ spécifique
mysql_affected_rows	Extrait le nombre de lignes concernées par une requête SQL
mysql_insert_id	Extrait l'identificateur de généré par le dernier INSERT
mysql_fetch_row	Extrait d'un résultat la prochaine lignes et la place dans un tableau
mysql_data_seek	Définit le pointeur de ligne interne dans une liste ou une query
mysql_fetch_field	Extrait du jeu de résultats les informations relatives aux colonnes
mysql_field_seek	Attribue des informations d'un champ à un autre
mysql_fetch_object	Retourne un objet correspondant à la ligne extraite d'une liste ou query
mysql_fetch_array	Extrait la ligne sous forme d'un tableau associatif
mysql_fetch_lengths	Extrait la longueur de champ de la dernière ligne lue
mysql_result	Extrait les données d'une liste ou d'une query
mysql_free_result	Libère la mémoire réservée à un résultat

22.5.2 Utiliser une base de données

Pour accéder à une base de données, PHP nécessite une connexion vers celle-ci. Pour se connecter à MySQL, PHP utilise la fonction **mysql_connect()** :

```
mysql_connect(host,utilisateur,pwd);
```

host Le nom de la machine contenant la base de données MySQL.

utilisateur Un utilisateur MySQL.

pwd. Le mot de passe associé.

Une fois que PHP est connecté, il faut encore lui donné le nom de la base de données avec laquelle on désire travailler. Pour cela, on utilise la fonction **mysql_select_db()** :

```
mysql_select_db(nom);
```

où, *nom* représente le nom de la base de données.

22.5.3 Exécuter une requête

Pour émettre une requête SQL vers une base de données MySQL, PHP utilise la fonction **mysql_query()** :

```
mysql_query(sql);
```

où *sql* est une chaîne de caractères représentant la requête SQL. Cette fonction retourne une valeur nulle si un problème s'est produit lors de la requête. En effet, outre une mauvaise syntaxe pour celle-ci, il se peut que l'utilisateur avec lequel PHP se connecte sur la base de données ne puisse pas effectuer la requête pour des raisons de droits.

Comme expliqué dans l'annexe C, on peut diviser les instructions SQL en deux catégories, celles qui renvoient des records, comme **SELECT**, et celles qui ne renvoient rien, comme **INSERT** ou **DELETE**.

Dans le cas où la requête émise par PHP renvoit des records, il faut que PHP passe les records en revue afin de les traiter. Pour cela, PHP dispose de deux fonctions **mysql_num_rows()** qui donne le nombre de records renvoyés par la requête, et la fonction **mysql_fetch_row** qui renvoit un tableau contenant la valeur des différents champs , le premier champ ayant l'indice 0 pour le record courant et passe ensuite au suivant.

On peut voir au listing 22.6, un programme PHP qui se connecte à une base MySQL et qui affiche une série de liens.

Listing 22.6 – Code PHP accédant à MySQL

```
<HTML>
  <HEAD>
    <TITLE> Exemple de Test MySQL </TITLE>
  </HEAD>
  <BODY>
    <?php
      mysql_connect("localhost","root","");
      mysql_select_db("hp");
      $result=mysql_query("SELECT html,title FROM htmls");
      for($i=0;$i<mysql_num_rows($result);$i++)
      {
        $tbl=mysql_fetch_row($result);
        echo("<A HREF=".$tbl[0].">".$tbl[1]."</A><BR>");
      }
    ?>
  </BODY>
</HTML>
```

Aux lignes 7 et 8, PHP se connecte sur la dite base. Puis il émet une requête de sélection (Ligne 9) dans une table «htmls» qui contient des informations sur des URLs, en demandant chaque fois le nom de l'URL et le titre de celle-ci qui correspondent respectivement aux champx «html» et «title» de cette table. Ensuite, il utilise une boucle **for** (Ligne 10-14) pour appeler la fonction **mysql_fetch_row** autant de fois qu'il y a de records renvoyés par la requête. Pour chaque record, il crée le code HTML nécessaire pour créer un lien vers une URL (Ligne 13). On peut voir le résultat à la FIG. 22.4.

22.5.4 Autres fonctions MySQL

Il existe d'autres fonctions permettant de faire des choses typiques à MySQL. Par exemple, la fonction **mysql_num_dbs()** renvoit un tableau contenant toutes les bases de données MySQL disponibles sur le serveur sur lequel PHP est connecté, dont le nombre d'éléments est donné par la fonction **mysql_num_rows()**. Ceci dit, elles sont rarement utilisées dans le cadre d'une utilisation dans un site web, mais plutôt dans des environnements intranet.

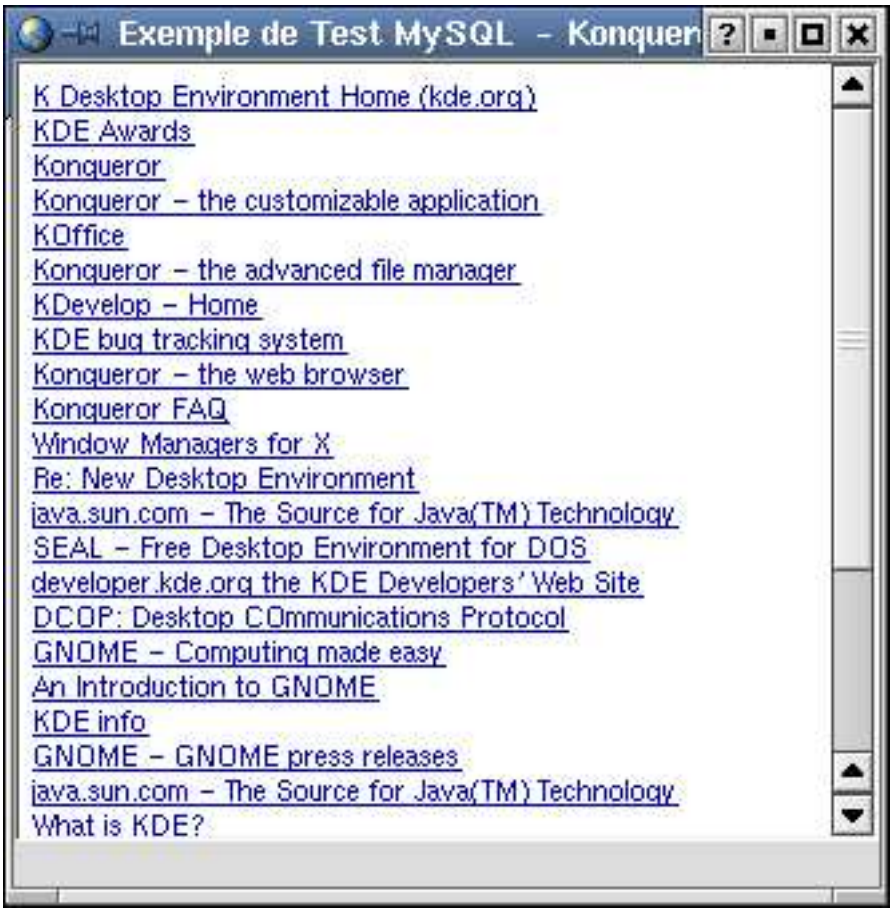


FIG. 22.4 – Code PHP accédant à MySQL

Septième partie

Sécurité sur internet

Chapitre 23

Introduction : Concepts de sécurité

23.1 Introduction

Ce chapitre est consacré à une présentation générale du problème de la sécurité sur Internet. Le WWW-Web est un système complexe, qui permet à de nombreuses personnes ou entreprises de partager une masse importante d'informations. La sensibilité de ces informations est très variable, mais de plus en plus de données stratégiques sont aujourd'hui échangées ou partagées via un réseau Intranet ou Internet. De par la nature même de ces réseaux informatiques, de nombreuses ressources sont impliquées dans ce partage ou cet échange d'informations : même si ces ressources ont des capacités d'accès parfois réduites (dû notamment aux débits des lignes), elles ont toutes accès les unes aux autres.

La sécurisation d'Internet consiste donc à la fois à protéger les données, que ce soit lors de leur transfert ou lors de leur stockage sur une machine, mais aussi l'intégrité même des machines connectées au réseau, que ce soit le serveur ou le client.

Avant d'entrer dans le détail des causes et solutions techniques envisageables pour les différents problèmes de sécurité, nous allons poser le problème en définissant les différentes formes d'attaques, ainsi que les ressources qu'elles visent.

Pour des renseignements complets concernant la sécurité sur les sites commerciaux sur Internet, il faut consulter [24].

23.2 Les virus informatique : la première forme de menace

Les virus informatiques existent depuis que la programmation informatique existe. Mais grâce à l'arrivée d'Internet et à l'interconnexion d'un nombre croissant d'ordinateurs, ils sont de plus en plus dangereux. Non seulement ils se répandent plus rapidement, mais ils ne s'attaquent plus seulement à l'intégrité de la machine : en effet, la connexion elle-même est parfois devenue un élément critique du système et bloquer celle-ci est aussi dommageable.

Un «virus» est un programme informatique falacieux, qui se recopie automatiquement et associe ses copies à d'autres programmes ou fichiers. Il n'est transmis à d'autres ordinateurs que si un fichier infecté est transmis explicitement. Un "worm" est un programme similaire à un virus, à la différence qu'il transmet automatiquement ses copies à d'autres ordinateurs, via une connexion au réseau, où il s'exécute une seule fois. Un «trojan» (de «trojan horse» = cheval de Troie) est un programme qui semble inoffensif (de par la fonction qu'il est censé exécuter), mais qui cache en réalité une fonction malicieuse ou un virus.

Tous ces termes désignent en fait de petits programmes informatiques qui sont censés altérer le fonctionnement de la ressource qu'ils infectent :

- Exécution d'une commande bénigne un très grand nombre de fois, afin de perturber le fonctionnement hardware ;
- Effacement pur et simple du disque dur ;
- Bloquage des connexions au réseau (envoi d'un nombre élévé de requêtes ou connexion par modem sur une ligne internationale) ;

– Vol de données (par envoi d'une copie du disque dur via modem ou réseau) ;
– Modification systématique d'un certain type de données (le carnet d'adresse par exemple).

Même si les virus peuvent avoir des effets catastrophiques sur l'intégrité des données mais aussi des machines, que ce soit un serveur ou un simple PC, il est pratiquement possible d'en venir à bout, que ce soit par l'utilisation d'un programme anti-virus (mis à jour régulièrement) ou par la réinitialisation systématique de toute machine infectée, ainsi que sa déconnexion temporaire du réseau. Il est très difficile de prévoir une attaque par un virus (qu'il vienne d'Internet via un e-mail ou par une disquette), mais il est possible de s'en protéger relativement efficacement, tout en conservant les avantages de la connexion.

Cependant, toutes ces attaques, basées sur un programme qui arrive à l'insu de l'utilisateur sur une machine, et qui s'exécute sur celle-ci, sont également réalisables à distance. Grâce à la connexion au réseau Internet et avec quelques connaissances en programmation, un pirate pourra faire autant de dégats qu'un virus. La difficulté réside ici dans la détection de l'attaque, qui vise parfois simplement à utiliser la machine comme passerelle, sans s'attaquer à celle-ci, mais également dans la prévention, car elle exploite généralement les failles de sécurité offertes par les logiciels.

23.3 Les pirates informatiques : une menace systématique

Le piratage informatique regroupe aujourd'hui l'ensemble des activités criminelles, liées à l'informatique. La forme la plus ancienne est sans doute le virus, mais aujourd'hui le cadre plus large du web est la cible de ce que l'on nomme des attaques. Elles sont le fait de programmeurs, généralement très compétents dans leur domaine, qui ont généralement un des trois objectifs suivants :

– pirater des informations ou des machines dans le but d'établir une fraude, un vol ou un trafic ;
– pirater dans le but de modifier ou détruire purement et simplement des données ;
– passer outre les sécurités mises en place pour le simple jeu, ou pour mettre en évidence les lacunes de ces sécurités.

Les pirates s'attaquent à des données, lors de leur transit entre le client et le serveur ou aux machines où elles sont stoquées. Il faudra donc distinguer trois cas bien précis : sécuriser le serveur web, sécuriser la transmission de données et sécuriser le client.

23.3.1 Sécuriser le serveur web

Le "serveur web" désigne indifféremment la machine ou le logiciel qui permet de gérer les requêtes des navigateurs. La sécurisation vise plutôt la machine, même si le logiciel n'est pas étranger au problème. La sécurisation du serveur web doit être axée sur 3 points :

1. Premièrement, le serveur est un PC comme un autre et il est souvent connecté à un réseau local. Il faut donc le protéger des virus, mais également appliquer des techniques de restrictions d'accès aux seules personnes qui en ont le droit.
2. Le serveur web contient généralement deux types d'informations : les pages du site (ou des sites) qu'il héberge et une base de données d'informations qu'il collecte. Les pages web doivent être protégées, car elles représentent la visibilité d'une société sur Internet : leur portée commerciale est très importante et leur contenu est aujourd'hui considéré comme critique. D'autre part les informations collectées relèvent très souvent de la vie privée, voire d'informations financières. Elles ne peuvent donc pas être mises à la disposition de tout le monde. Ceci implique l'obligation de mettre un système de sécurité en place, de limiter très fort l'accès, même en interne, voire de physiquement protéger cette machine.
3. Les logiciels qui rendent possibles les services d'Internet sont malheureusement truffés d'erreurs de programmation, qui s'apparentent souvent, pour le programmeur averti à des portes d'entrée sur le système, que ce soit aux ressources de la machine, aux données qui s'y trouvent ou aux autres machines du réseau local. Il est donc nécéssaire de vérifier constamment ces logiciels, et surtout tout ce qui se passe sur la machine.

Les bonnes pratiques de sécurisation de cette machine sont donc de l'isoler au maximum du réseau local, de limiter les accès physiques au matériel, de limiter le nombre d'utilisateur ayant le droit de se connecter

à la machine. D'un point de vue logiciel, il est important de surveiller le comportement des logiciels de service Internet et de limiter leur nombre sur une même machine, afin de ne pas multiplier les risques. Il faut également au besoin limiter les possibilités de requêtes.

23.3.2 Sécuriser les informations en transit

La sécurisation des données sur Internet rime aujourd'hui avec la protection contre l'interception d'informations au cours de leur transfert entre un client et un serveur.

Il existe de nombreuses manières d'empêcher l'interception : isoler la ligne de communication, cacher l'information parmi des données inutiles ou crypter l'information de manière à ce que seules les personnes concernées puissent la décoder. Le cryptage est aujourd'hui la seule méthode que l'on puisse réellement utiliser sur le Web.

Une autre forme d'attaque liée au transfert de donnée est le refus de service, résultant d'une interruption du réseau. Cette interruption peut provenir d'un évenement imprévu, comme une fibre cassée ou une erreur dans les tables de routage, mais peut également être le fait d'une attaque systématique contre un serveur. Il s'agit d'engorger la connexion du serveur avec des milliers de requêtes inutiles, empêchant les requêtes légitimes de passer. Il est malheureusement très difficle de se prémunir contre ce genre d'attaque.

23.3.3 Sécuriser l'ordinateur d'un utilisateur

Le troisième intervenant dans le problème de la sécurité est le client. La machine du client est un PC, connecté au réseau par un modem ou via un réseau local.

Plusieurs logiciels utilisent la connexion avec Internet pour fonctionner, que ce soit le navigateur ou le programme de messagerie. Ceux-ci peuvent offrir des portes d'entrée sur la machine (son contenu ou ses ressources) de par leurs erreurs de programmation. Correctement exploitées, ces erreurs de programmation donnent un accès illimité aux pirates, que ce soit pour accéder aux données contenues sur la machine ou aux autres machines du réseau. Le client est donc également une cible de choix pour le pirate, d'autant qu'il dispose généralement de moins de connaissances techniques et de moins de ressources pour contrer une attaque.

Chapitre 24

La sécurité du côté du client

24.1 Introduction

Le client, au sens informatique du terme est une machine connectée via un réseau à d'autres machines, lui donnant le droit de consulter et d'utiliser un certain nombre de ressources. Dans le cas d'Internet, la machine cliente sera soit connectée via un modem et une ligne téléphonique le reliant à un fournisseur d'accès (ISP : Internet Service Provider), soit via un réseau local, lui-même connecté au réseau Internet.

La machine cliente n'a donc pas pour objet de mettre des ressources à disposition ; tout au plus, dans le cas d'un réseau local, peut-elle disposer d'un répertoire partagé pour l'échange d'informations. Cependant la machine cliente sert rarement uniquement à accéder à Internet : elle contient un certain nombre de données critiques, que ce soit pour son utilisateur ou pour sa société. Elle représente donc une cible de choix pour les pirates informatiques, ainsi qu'une proie facile pour les virus.

Il faut en fait distinguer trois types de menaces qui pèsent sur une machine de type client, connectée à Internet :

1. La destruction de l'intégrité de la machine : effacer le contenu, modifier le contenu, bloquer le processeur, bloquer les accès au réseau ;
2. L'utilisation de la machine comme passerelle vers un réseau local (Intranet) ou vers d'autres machines pour des manipulations frauduleuses ;
3. L'utilisation frauduleuse des données : vol du contenu du disque dur (plans, fichiers, adresses), installation d'un programme de transmission systématique, utilisation de données pour les transactions bancaires, bloquage du modem sur une ligne téléphonique internationale...

Si une partie de ces menaces sont le fruit de virus, de nombreuses attaques sont planifiées systématiquement par des pirates informatiques, utilisant les failles des programmes installés sur la machines, comme les browsers ou les programmes de mail, voire même les systèmes d'exploitation.

24.2 Les failles des navigateurs

Les navigateurs sont des logiciels relativement complexes, et qui sont appelés à le devenir encore plus. Ceci est essentiellement dû au fait qu'ils intègrent de très nombreuses fonctionnalités et qu'ils font appel à beaucoup d'éléments. Plus le programme sera compelexe, plus la chance de voir apparaître des erreurs de programmation (et donc dans la plupart des cas, des failles de sécurité) sera grande.

Les premiers navigateurs datent du début des années 80, lors de la création du langage HTML. Leur unique utilité était à l'époque de visualiser des documents écrits dans ce langage. Depuis, de nombreux navigateurs sont apparus sur le marché et d'autres types de documents que le HTML sont apparus sur le Web.

De nombreux navigateurs ont été la cible de programmeurs avertis, qui y ont trouvés de nombreuses erreurs de programmation. Ces erreurs permettent généralement au pirate de faire exécuter une commande par la machine cliente sans que l'utilisateur n'en soit averti, et sans qu'il puisse l'en empêcher.

Cependant, le danger le plus important dans les navigateurs est qu'ils peuvent faire appel à des programmes extérieurs. En effet, le navigateur ne permet, avec ses fonctions de base, que de visualiser un nombre très limité de format de fichier. Pour pallier cet inconvénient, le navigateur peut faire appel à des applications d'aide (Helper Applications). Le principe, basé notamment sur la technologie MIME, permet d'ouvrir un programme extérieur à la session du navigateur. C'est un des moyens de visualiser un fichier audio ou vidéo (les plug-ins permettent des fonctionnalités similaires, comme nous le verrons dans les sections suivantes).

Le problème majeur c'est que l'application ainsi ouverte possède elle-aussi des failles de sécurité, mais surtout, elle ne fait pas l'objet d'une procédure de sécurisation particulière. L'exemple typique serait l'ouverture d'un fichier .doc à partir du Web, via le browser. Celui-ci reconnaît le type de fichier et ouvre une session de Microsoft Word. Si ce fichier contient une macro se lançant à l'ouverture, celle-ci sera plus que probablement exécutée sans que l'utilisateur ne puisse rien faire. Le cas peut également apparaître avec des fichiers de code source. Le navigateur reconnaît le type de fichier et lance un compilateur adapté. Celui-ci permet alors d'exécuter n'importe quel programme, donnant un accès total à un pirate.

Il est donc important de ne pas associer certaines applications avec le navigateur, notamment :

- Microsoft Word et Microsoft Excel, car il sont équipés d'une extension Visual Basic (il s'agit du même problème que celui des macros virus) ;
- Tout programme qui contient une ligne de commande Visual Basic ;
- Perl, Python, Tcl/Tk qui sont des langages de programmation par script ;
- Les commandes de base de Unix ou de DOS ;
- Des interpréteurs PostScript, car ils contiennent des commandes d'ouverture, de lecture et d'effacement de fichiers.

24.3 Java et JavaScript

La section précédente était consacrée aux problèmes liés au navigateur lui-même : ses erreurs de programmation et la liberté qu'il peut prendre d'ouvrir des programmes extérieurs. Il y a également plusieurs langages de programmation qui ont été développés plus spécifiquement pour le web. Les programmes sont intégrés dans des pages web et exécutées sur la machine du client.

Afin de limiter au maximum les risques encourus par de telles pratiques, ces programmes sont exécutés à l'intérieur d'un environnement au pouvoir limité. Ainsi l'accès aux fichiers sur la machine est bien souvent interdit et toute tentative est signalée à l'utilisateur. Nous détaillerons dans cette section les cas de Java et JavaScript. Ces deux langages de programmation, à la syntaxe proche du C++, sont destinés à rendre les sites web plus dynamiques, que ce soit du côté serveur ou du côté client.

24.3.1 Java

Bien que Java soit aujourd'hui connu comme un langage permettant d'écrire des programmes qui sont téléchargés d'Internet pour s'exécuter sur le client, il n'a pas été inventé pour être un langage sécurisé. Il était en effet destiné à une communauté fermée de programmeurs, utilisant sa portabilité pour diverses applications.

Cependant, son repositionnement stratégique sur le Web a rapidemment soulevé le problème de la sécurité. En effet, par essence, tout utilisateur peut aller chercher n'importe quelle applet, pour autant qu'elle soit mise à disposition sur un serveur. Il faut donc mettre en place un système de protection des utilisateurs contre les programmes pervers ou simplement mal écrits. Pour résoudre le problème de ces derniers, les inventeurs de Java ont préféré la simplicité : la plupart des éléments difficiles à maîtriser dans les langages de programmation évolués ont été supprimés. Ceci permet de réduire fortement les risques encourus par un programme mal écrit. Cependant, même bien écrit, un programme peut faire des dégats, surtout si c'est l'intention de son développeur. C'est pourquoi il faut limiter les capacités d'un programme venant de l'extérieur.

Java se base sur 4 technologies pour limiter ce qu'un programme téléchargé peut faire :

1. Java Sandbox. Les programmes Java n'ont pas la possibilité de manipuler directement le hardware ou de faire des appels directs aux fonctions du système d'exploitation. Le programme Java est exécuté

sur un ordinateur virtuel, à l'intérieur d'un environnement virtuel.

2. SecurityManager Class. Si tous les programmes Java ne pouvaient pas envoyer d'informations via le réseau, s'ils ne pouvaient pas accéder aux fichiers ni accéder aux ressources, ils seraient sécurisés ; ils auraient également peu d'intérêt. C'est pourquoi Java possède une série de classes de base, à l'intérieur de la sandbox, lui permettant un accès aux ressources. Afin de contrôler l'appel à ces classes, elles sont toutes précédées par une classe appellée SecurityManager, quia pour objet de déterminer si l'action doit être permise ou pas.
3. Java Class Loader. Comme les sécurités implémentées dans Java sont elles-mêmes écrites en Java, il n'est pas impensable d'envisager de remplacer celles-ci par de fausses classes moins restrictives et ainsi mener une attaque. Afin d'éviter cela, le Class Loader examine toutes les classes qui vont être utilisées par un programme et vérifie qu'elles ne vont pas à l'encontre de l'environnement d'exéctution.
4. Bytecode Verifier. Afin de protéger l'environnement d'exécution, le Bytecode Verifier assure que l'applet java a bien été écrite dans le langage Java et n'utilise donc pas de fonctionnalités autres, comme des pointeurs par exemple. En effet, une erreur de ce type pourrait provoquer une interuption du programme, lors de son exécution par la Java Virtual Machine, permettant ainsi au pirate de prendre le contrôle de la machine.

La FIG. 24.1 illustre le fonctionnement et l'interaction de ces différents systèmes de sécurité. Malgré cela, de nombreuses erreurs de programmation du langage ou de ses outils ont été découvertes par des équipes de programmeurs. La plupart ont pu être corrigées par les concepteurs de Java.

Malgré tous ces efforts, Java reste un langage peu sûr permettant théoriquement de faire tout et n'importe quoi avec le client. C'est pourquoi de nombreux navigateurs appliquent une politique de restriction importante. Ainsi dans Microsoft Internet Explorer et dans Netscape Navigator, Java peut être rendu complètement inactif, et lorsqu'il est actif, les possibilités sont très réduites pour les applets téléchargées depuis Internet. Pour les applets lancées à partir du client directement (Java permet d'écrire des programmes comme tout autre langage), les accès sont plus libres.

Pour plus de renseignements concernant la sécurité et Java, il faut consulter [18].

24.3.2 JavaScript

JavaScript est un langage développé par Netscape pour améliorer l'interactivité de son navigateur avec les utilisateurs. Les programmes écrits en JavaScript (que l'on appelle parfois abusivement des Scripts, car ils sont généralement courts), permettent de modifier l'aspect du navigateur, à partir du fichier HTML. Même si pour Netscape Navigator, écrit lui-même en JavaScript, les interactions sont plus grandes, tous les navigateurs modernes supportent ces fonctionnalités.

Par définition, les programmes écrits en JavaScript devraient être plus sûrs que d'autres, pour diverses raisons :

– Il n'y a pas de fonctions prévues en JavaScript pour accéder aux fichiers système du client ;
– Il n'y a pas de fonctions prévues pour ouvrir des connexions au réseau.

Mais l'évolution du langage a nécéssité de lui donner de plus en plus d'accès aux ressources de la machine cliente. De plus, comme nous l'avons déjà vu, les attaques ne concernent pas toujours l'intégrité de données sur une machine ; la machine elle-même peut être l'objet de l'attaque. Ainsi, on retrouve les problèmes de sécurité de JavaScript dans essentiellement deux domaines : le refus de services et les violations de vie privée.

24.3.2.1 JavaScript et la vie privée

Comme le code JavaScript réside à l'intérieur du navigateur, il a accès à une série d'informations auxquelles le navigateura lui aussi accès :

1. Un programme JavaScript est capable d'envoyer un e-mail et aussi de détecter le nom ou l'adresse e-mail de l'utilisateur. Ainsi, il est possible de faire envoyer automatiquement un message d'insulte ou de collecter le nom de l'utilisateur qui accède une page donnée.

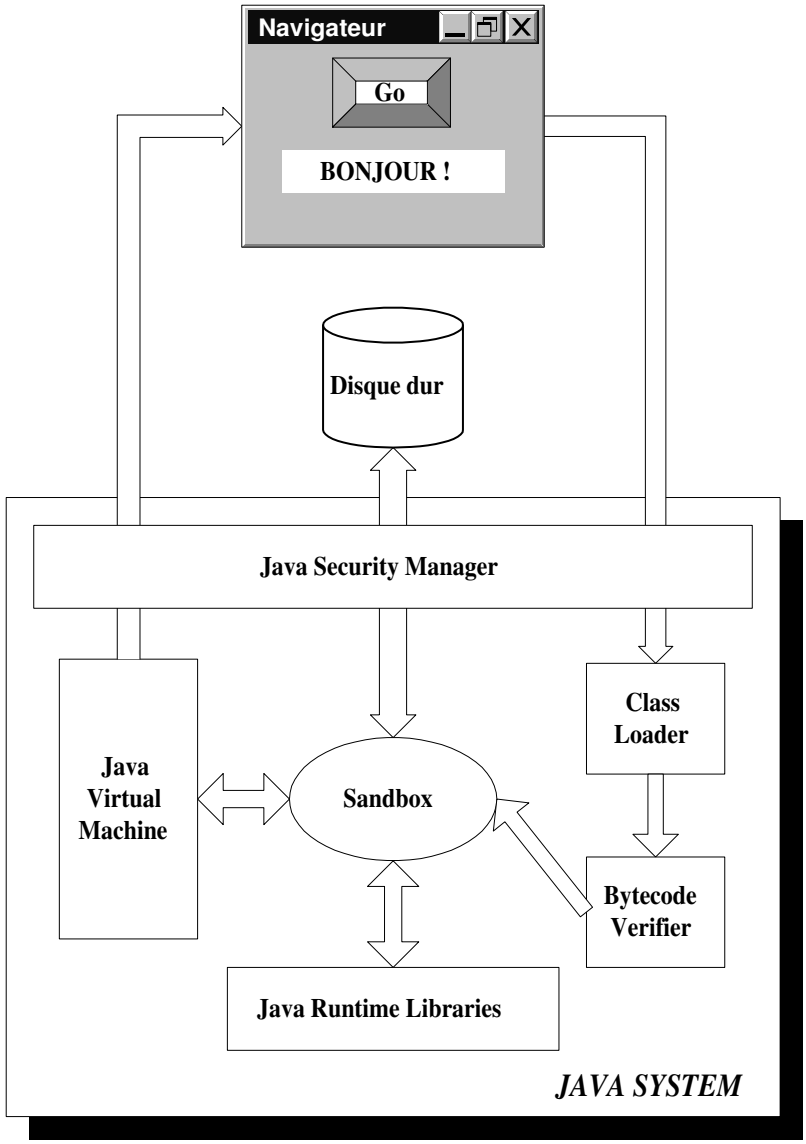


FIG. 24.1 – Sécurité et Java

2. Les programmes JavaScript ont accès à l'historique de navigation de l'utilisateur. Combiné à la précédente possibilité, un système de statistique à l'insu de l'utilisateur peut être facilement mise en place.
3. Un programme JavaScript a accès à l'URL visitée. Il est donc possible de forcer l'ouverture de certaines pages.
4. JavaScript permet d'afficher des boîtes de dialogue. Ces boîtes ont le look-and-feel du navigateur, ce qui permet aux pirates de tromper l'utilisateur. Parmi les exemples de fraudes mises en place par ce système, citons :
   – la boîte de dialogue qui apparaît pour signaler un problème de connexion réseau et demandant à l'utilisateur de réintroduire son mot de passe réseau ou son mot de passe modem ;
   – la boîte de dialogue qui demande le numéro de carte de crédit.

Nous verrons également plus tard les problème liés aux cookies.

24.3.2.2 JavaScript et le refus de service

Les attaques basées sur le refus de services consistent à bloquer les ressources d'une machine en les engorgeant. Un exemple assez simple consiste à écrire une boucle de calcul quasi infinie (ou du moins très longue), ce qui a pour effet de mobiliser les ressources du processeur et de la mémoire et d'arriver aux limites de ceux-ci. Un autre exemple consiste à écrire une boucle infinie qui ouvre à chaque fois une boîte de dialogue.

Il existe peu de moyens pour arrêter ces attaques, si ce n'est de couper le navigateur, voire d'éteindre la machine. Il arrive cependant souvent que les utilisateurs ont peu de notions hardware et software ; ces attaques, bien que stupides sur le fond peuvent s'avérer de véritables catastrophes pour les utilisateurs peu avertis.

Il est également très difficile de se prémunir contre ce genre d'attaque, le meilleur moyen étant alors d'interdire purement et simplement l'éxécution de tout programme JavaScript. Parmi les règles de bonne pratique, il faudra faire particulièrement attention aux programmes demandant un accès au réseau (blocage de la connexion) ou demandant beaucoup de ressources processeur ou mémoire.

24.4 ActiveX et Plug-Ins

Une des choses les plus dangereuses à faire avec un ordinateur connecté à Internet est de télécharger un programme et de l'exécuter, essentiellement parce que le système d'exploitation de l'ordinateur ne limite aucunement les accès que peut avoir un programme une fois lancé.

Comme nous l'avons vu précédemment, les navigateurs ont par essence des possibilités réduites. Afin d'améliorer l'interactivité avec l'utilisateur, de nombreux programmes peuvent être téléchargés pour compléter le navigateur. Nous avons déjà envisagé le cas des applications d'aide. Les mêmes fonctionnalités peuvent être réalisées à l'aide de Plug-Ins.

Les Plug-Ins sont des petits programmes qui, une fois téléchargés, s'intègrent au navigateur (essentiellement Netscape Navigator) après une installation manuelle (ce qui est un élément positif pour la sécurité). Associés à certains types de fichier via un MIME, ils sont exécutés si l'utilisateur désire visualiser un fichier de ce type. L'avantage est que le fichier à visualiser reste dans la mémoire ce qui accélère le processus.

Il faut cependant savoir que les Plug-Ins ont un accès total au contenu de la machine et qu'ils sont écrits par des tiers, dont on ne peut être sûr de l'identité ou des intentions. D'autre part, dès que le navigateur détecte un fichier nécéssitant un Plug-In il fait automatiquement appelle à celui-ci (pour autant qu'il soit présent sur la machine ; dans le cas contraire il vous propose de le télécharger). Il s'agit donc d'être certain du fonctionnement de ce programme.

Microsoft a, lui, décidé d'utiliser des éléments similaires, appelés contrôles ActiveX. Sans entrer dans les détails, ces contrôles ont des fonctionnalités similaires à certains Plug-Ins mais sont beaucoup plus puissants. Parmi les différences significatives, notons le fait qu'ils s'installent automatiquement sans intervention de l'utilisateur. Ils ont également accès à des fonctionnalités du système Windows. Ainsi il est possible d'utiliser un ActiveX pour commander un Shutdown de Windows 95 à partir d'une page HTML.

Ces différents exemples montrent le danger de faire appel à des programmes téléchargés à partir d'Internet, de par le simple fait qu'il est impossible de dire ce qu'ils vont faire à l'avance. Seule la confiance dans la personne qui a écrit le programme peut être un début d'assurance ; c'est notamment l'objet des systèmes d'authentification de code, proche des certificats digitaux comme nous le verrons au chapitre suivant.

24.5 Log Files et cookies

Les fichiers de traçabilité (Log Files) sont des fichiers résidant sur les serveurs web, dans lesquels sont répertoriés tous les accès qui ont été faits à un fichier donné :

- Le nom et l'adresse IP de la machine d'ou provient la requête ;
- Date et heure de la requête ;
- L'URL demandée ;
- Le temps nécéssaire à traiter la requête ;
- Le nom de l'utilisateur, dans le cas où il figure dans l'entête de la requête ;

- Les erreurs éventuelles ;
- L'URL de la dernière page visitée avant la requête (ce que l'on appelle le «refer link») ;
- Le type de navigateur.

Toutes ces informations sont stockées systématiquement sans que l'utilisateur puisse y faire quoi que ce soit, ni sans qu'il y ait d'assurance sur leur utilisation future. Si dans la plupart des cas, ces informations sont peu critiques, elles permettent cependant de mettre en place des systèmes de statistique et de recherche assez puissants, mettant ainsi en cause la protection de la vie privée des utilisateurs.

Une autre forme de trace existant des sites qui sont visités avec le navigateur est les cookies. Il s'agit de blocs de texte au format ASCII, qui sont envoyés par un serveur web à un navigateur lors du premier passage sur une page web. Une fois reçu, le cookie est stocké et estrenvoyé au serveur à chaque nouvelle visite. Les cookies permettent de faciliter la navigation en permettant au navigateur de stocker la langue dans laquelle le site est visité, évitant ainsi de devoir faire ce choix à chaque visite.

Le problème esentiel avec les cookies est qu'ils permettent d'identifier les personnes. Il est ainsi possible de connaître exactement les préférences d'un utilisiateur, de voir ce qu'il a visité comme site web. Tout cela à son insu. Il est cependant possible d'empêcher l'utisation des cookies, ce qui peut aussi diminuer les accès aux sites qui n'ont pas prévu de ne pas pouvoir les utiliser.

Chapitre 25

La sécurité de la transaction

25.1 Introduction

La protection des numéros de carte de crédit lors de transaction sur le web est probablement l'exemple le plus concret d'un besoin de sécurité. En effet, de par la technologie utilisée pour réaliser l'interconnexion des ordinateurs sur Internet (le protocole TCP-IP), l'information partant du client vers le serveur ou vice versa, va transiter par plusieurs machines (que ce soient des éléments de connexion comme des routeurs ou d'autres PCs). Cette information, bien que découpée en paquets de taille fixe, est complètement lisible. Rien n'empêche donc un utilisateur de surveiller le trafic qui passe sur son ordinateur et d'intercepter systématiquement le contenu de chaque paquet. Dans beaucoup de cas il n'interceptera que des bribes de texte de pages HTML, mais il pourra à l'occasion intercepter une suite de chiffre correspondant à un numéro de carte, ou une suite de caractères correspondant à un mot de passe. Il est donc nécéssaire de mettre en place un système qui permette de masquer cette information critique, afin qu'un intrus dans la communication ne puisse pas la comprendre (il est en effet impossible d'empêcher l'interception du message : ce qui est par contre nécéssaire, c'est que celui qui intercepte le message ne puisse pas le comprendre et donc l'utiliser). C'est l'objet de la cryptographie, qui est une technique mathématique pour brouiller de l'information.

L'autre grand problème lorsque l'on transmet un numéro de carte de crédit ou tout autre information par le web, est de savoir ce que le destinataire va en faire. En effet, rien n'empêche à priori un fraudeur de mettre en place un site prétendant offrir des services de vente par correspondance, mais sans avoir aucun produit à vendre. Cette nouvelle manière de pratiquer le vol de carte de crédit est malheureusement très courante, et la seule solution est d'être toujours certain de la personne avec qui l'on communique. C'est pourquoi les certificats digitaux ont été mis en place. Il s'agit d'un système d'identification de personnes physiques ou morales : un certificat digital sera l'équivalent d'une carte d'identité. Ce certificat, pour avoir une valeur quelconque, doit être délivré par une autorité reconnue, que l'on appelle autorité de certification : cette société prend la responsabilité d'assurer vis-à-vis de tiers que la personne possédant un certificat est bien celle qu'elle prétend être.

Dans ce chapitre nous allons aborder ces deux techniques dans le détail. Nous parlerons ensuite du protocole SSL, qui est une couche spécifique rajoutée au protocole TCP-IP pour implémenter techniquement ces éléments dans le navigateur. Enfin nous aborderons le cas des transactions bancaires pour illustrer ces concepts.

25.2 La cryptographie

La cryptographie regroupe un ensemble de techniques pour sécuriser de l'information : le principe consiste à transfomer les mots (ou tout autre forme de message) en une suite de caractères qui n'ont aucun sens pour les personnes non concernées. Une personne autorisée pourra cependant déchiffrer la suite de caractère et retrouver le message original.

Le concept même de la cryptographie est très ancien : il est utilisé depuis que les armées existent et ont la possibilité d'intercepter les messagers ennemis. Les techniques antiques consistaient à remplacer les

lettres par d'autres ou par des chiffres suivant une règle établie dans une table ou dans un code (d'où le nom de chiffrage ou de transposition du message). Le déchiffrage du message consistait alors à effectuer l'opération inverse, à l'aide de cette même table. L'apparition des transmissions par ondes hertziennes et par lignes télégraphiques a créé le besoin de fabriquer des appareils électromécaniques, capables de brouiller les communications.

Si le cryptage d'un message assure sa sécurité, c'est essentiellement parce qu'il est pratiquement impossible de décoder le message sans le code. Celui-ci est donc un élément stratégique hautement protégé. Il faut cependant remarquer que, théoriquement, il est toujours possible de décoder un message sans la clé. La complexité du système doit simplement assurer que cette opération prenne trop de temps ou de ressources pour en valoir la peine.

Aujourd'hui les besoins sont les mêmes, mais les techniques de communication et surtout les capacités de calcul ont évolué. Le principe est donc fondamentalement le même mais les technologies de cryptage actuelles sont simplement plus complexes.

25.2.1 Terminologie

25.2.1.1 Cryptage

Opération par laquelle le message (le *texte en clair*) est transformé en un second message (le *texte chiffré*) en utilisant une fonction mathématique complexe (l'*algorithme de cryptage*) et un code spécifique (la *clé de cryptage*).

25.2.1.2 Décryptage

Opération inverse de la première qui permet de transformer le texte chiffré en texte clair, à l'aide d'une seconde fonction mathématique (l'*algorithme de décryptage*) et un code spécifique (la *clé de décryptage*).

25.2.2 Algorithmes à clé symétrique

Dans ce cas, le code utilisé pour crypter et décrypter le message est le même. Ces algorithmes sont aujourd'hui très répandus sur Internet, car il sont très rapides et relativement faciles à implémenter. Le seul problème est que la clé unique doit être partagée par les deux parties communiquantes. Ceci implique le transfert de cet élement stratégique par une voie sécurisée, mais également de posséder une clé spécifique pour chaque personne avec qui l'on communique.

La FIG. 25.1 illustre le principe d'une communication entre deux parties à l'aide d'une clé privée. Roméo crypte le message avec la clé. Juliette elle la seule autre personne à posséder cette clé et à pouvoir décrypter le message. Si Juliette veut répondre à Roméo, elle utilisera la clé pour crypter son message et Roméo qui est l'autre unique personne à posséder cette même clé pourra à son tour le décrypter.

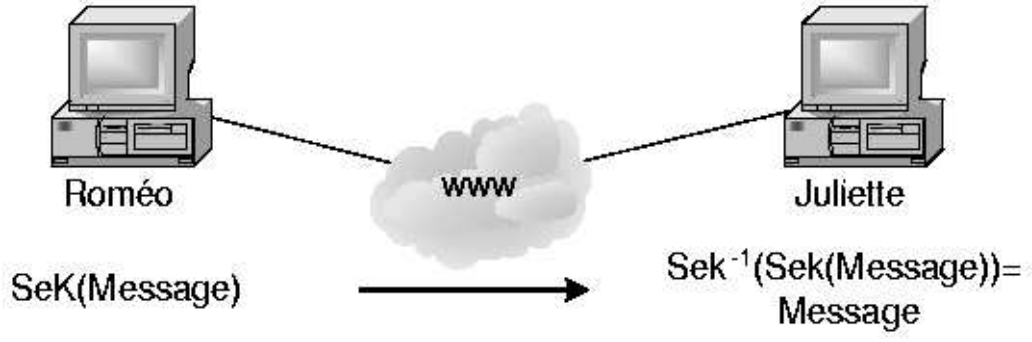


FIG. 25.1 – Algorithmes à clé symétrique

25.2.3 Algorithmes à clé asymétrique (ou publique)

Dans ce cas, il existe une paire de clés : une pour crypter le message et une autre pour le décrypter. La clé de cryptage est généralement appelée *clé publique*, car elle est disponible pour tout le monde, tandis que la clé de décryptage est appelée *clé privée*, car elle est la propriété unique de la personne. Chaque personne possède donc deux clés qui lui sont spécifiques : ceci réduit considérablement le nombre de clés nécéssaire pour établir des communications à plusieurs.

Ces algorithmes sont généralement plus lents car le cryptage doit être plus robuste. En effet, puisque tout le monde peut disposer de la clé publique, permettant de crypter le texte clair, il faut que le texte chiffré soit très difficilement décodable, ce qui rend le système d'autant plus sûr.

La FIG. 25.2 illustre le principe d'une communciation entre deux parties à l'aide d'une paire de clés. Roméo crypte son message avec la clé publique de Juliette. Celle-ci est la seule à pouvoir décoder ce message car pour cela il lui faut sa clé privée. Si Juliette veut répondre à Roméo, elle devra utiliser la clé publique de Roméo pour crypter sa réponse, tandis que Roméo utilisera sa clé privée à lui, pour décrypter la réponse de Juliette.

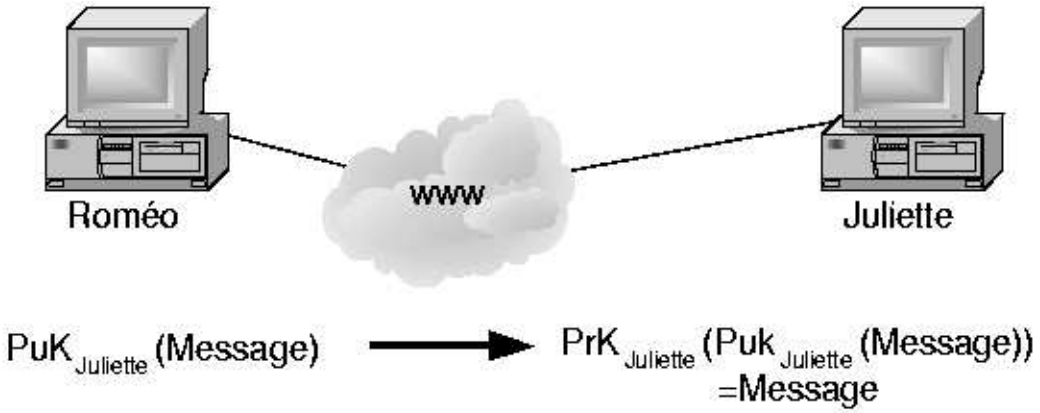


FIG. 25.2 – Algorithmes à clé asymétrique

25.2.4 Signature digitale

La signature digitale est une extension du cryptage par clé asymétrique. En effet, puisque la clé privée est à priori un identifieur unique d'une personne (car elle est censée être protégée), le cryptage à l'aide de la clé privée produit donc un texte chiffré unique, soit une signature digitale. Celle-ci peut être décryptée à l'aide de la clé publique.

Ce principe est illustré à la FIG. 25.3. Ainsi si Roméo veut envoyer un message signé à Juliette, non seulement il crypte son message avec la clé publique de Juliette, mais en plus il le signe en le cryptant avec sa clé privée. Juliette, elle, est la seule à pouvoir lire ce message avec sa propre clé privée. Elle peut de plus décoder la signature avec la clé publique de Roméo et vérifier l'authenticité du message.

L'avantage du système de cryptage asymétrique est dans ce cas-ci indéniable. En effet, non seulement la paire de clés permet permet à tout le monde d'envoyer un message crypter à Roméo qu'il sera le seul à pouvoir lire, mais en plus il permet à Roméo de signer ses réponses d'une manière univoque.

25.2.5 Algorithmes hybrides

Si les algorithmes asymétriques offrent plus de sécurité, ils sont également beaucoup plus lent (10 à 100 fois) que les algorithmes symétriques. Dans le cas d'un échange soutenu d'informations (comme les transactions bancaires), ces algorithmes sont pratiquement inutilisables. C'est pourquoi un troisième type d'algorithme existe : il s'agit d'un système hybride. Un algorithme asymétrique (plus lent) est utilisé pour

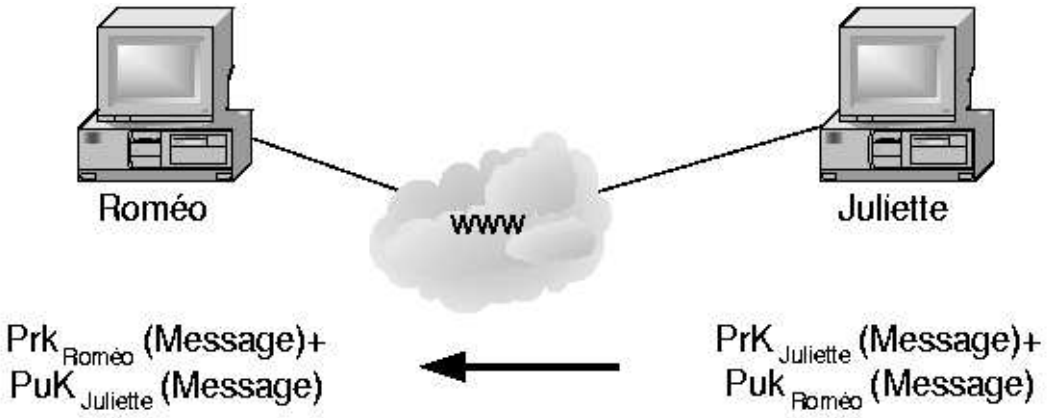


FIG. 25.3 – Signature digitale

établir une communication et échanger une *clé de session*. Cette clé sera ensuite utilisée par un algorithme symétrique (rapide) pour chaque échange au cours de la communication. Une fois celle-ci terminée, cette clé est effacée. La plupart des systèmes aujourd'hui utilisés sont basés sur ce principe.

25.2.6 Robustesse de la cryptographie

La capacité d'un système de cryptage à protéger l'information d'une attaque est appelée sa robustesse. Celle-ci dépend de nombreux facteurs, parmi lesquels :

- Le secret de la clé : dans le cas symétrique c'est essentiel, dans le cas asymétrique seule la clé privée doit être protégée ;
- La difficulté de deviner la clé ou d'essayer toutes les combinaisons possibles : les clés sont généralement de longues suites de chiffres ;
- La difficulté de pouvoir décrypter le messsage chiffré sans la clé (robustesse de l'algorithme) ;
- L'absence de contournements possibles de l'algorithme de décryptage ;
- L'abilité à déchiffrer l'entièreté du message si l'on a pu déchiffrer une partie ;
- Les propriétés du message chiffré et la connaissance que l'on peut en obtenir (l'apparition de suite qui se répètent peut être une source de déchiffrage).

Malheureusement, la robustesse ne se calcule pas et est très difficile à prouver. Il est beaucoup plus facile de prouver que le système n'est pas sûr, que le contraire.

25.3 Les certificats digitaux

Comme nous l'avons vu à la section précédente, les algorithmes de cryptage peuvent permettre une transaction sécurisée, à savoir que seules les parties concernées ont accès au texte clair. Avec la signature digitale, les correspondants ont même l'assurance que le message reçu corrrespond à quelqu'un ayant une clé privée. Ce qu'il manque à ce système c'est l'assurance que la personne qui possède la clé privée est bien celle qu'elle prétend être. C'est le but du certificat digital.

De la même manière, lorsque vous utiliser votre carte de crédit pour effectuer un paiement ou une location, ce qui garantit le paiement, c'est le fait qu'à la carte corresponde une personne physique, ce dont s'assurent la compagnie bancaire et l'Etat. Le principe du certificat digital est de fournir une carte d'identité digitale, qui assure le lien entre la paire de clés privée et publique et une personne physique. L'autre élément essentiel qui garantit la validité de l'identification, c'est qu'en cas de fraude, avec une carte de crédit par exemple, le commerçant est assuré d'être payé : en effet la compagnie bancaire possède une assurance contre la fraude. Il en va de même pour un certificat digital. Plus celui-ci sera important (délivré uniquement sur présentation d'une carte d'identité et contre paiement d'un montant annuel), plus

celui-ci sera assorti d'une garantie financière importante contre une utilisation frauduleuse, ce qui assure la protection de l'utilisateur du certificat et celle du commercant qui se fie à celui-ci.

Les organismes qui délivrent des certificats digitaux sont appellés Certification Authority (CA), ou encore *Trusted Third Party (TTP)* (tierce partie de confiance). Il en existe de plusieurs types. En effet, on peut imaginer une société qui délivre des certificats en interne à ses employés, assurant par la connaissance qu'elle a de ceux-ci, la validité de ces certificats. Des organismes extérieurs peuvent également délivrer des certificats, comme *GlobalSign*[1] ou *VeriSign*.

Les certificats ne sont pas uniquement délivrés pour assurer le lien entre une personne virtuelle (généralement une adresse e-mail) et une personne physique, pour effectuer des transactions ou envoyer des messages. Il est également possible d'obtenir un certificat pour un serveur, ce qui assure les personnes qui le visitent, qu'il appartient bien à une société identifiable. Enfin, il est également possible de délivrer des certificats pour des programmes. Le principe, appellé *Code Signing* ou signature de code source, consiste à signer des programmes exécutables avec une signature digitale. Celle-ci est obtenue grâce à une clé, délivrée par un organisme (CA) qui assure le développeur. C'est un moyen de procurer une certaine assurance sur la source de ce programme et une confiance dans ce que celui-ci va effectivement faire, une fois exécuté.

25.4 Le protocole SSL

SSL est l'acronyme de Secure Sockets Layer. Il s'agit d'un protocole générique pour envoyer de l'information cryptée sur Internet.Il s'agit d'une couche spécifique existant entre la couche TCP/IP et la couche application. En effet, la couche TCP/IP a pour unique mission de délivrer un flux d'information entre deux adresses et ce sans erreur. La couche SSL permet de spécifier un plus grand nombre de caractéristiques concernant cette transmission : authentification et non répudiation du serveur en utilisant une signature digitale ; authentification et non répudiation du client en utilisant une signature digitale ; confidentialité de l'information lors de son transfert via un système de cryptage ; intégrité de l'information en utilisant un système de code d'authentification (ce qui permet de vérifier si le message a été intercepté lors de son transfert). SSL a été développé par Netscape afin de pouvoir se positionner sur le marché comme seul vendeur de produits sécurisés. Depuis, ce protocole est intégré dans tous les browsers et les serveur webs. Une évolution de SSL vers un protocole standardisé est en cours : le protocole (**TLS!**). Le principe mis en place lors du développement du SSL est basé sur le découpage des fonctions de réseau en couche. L'idée est que l'utilisateur (qui interagit avec la couche application) ne se soucie pas de la manière dont l'information sera transmise vers un serveur donné. Tout au plus choisit-il d'utiliser un mode de transfert de type sécurisé. La couche SSL se charge d'établir la connexion avec le serveur, en vérifiant les identités respectives et d'effectuer la transmission sûre. L'utilisateur ne voit donc rien de particulier lors de l'utilisation de cette fonctionnalité de son navigateur. Les signes distinctifs du SSL sont le "s"qui se rajoute derrière http et une petite clé, indiquant l'état de la connexion sécurisée. Le gros avantage du SSL est de séparer les différentes tâches à effectuer pour établir un système de communication sécurisé. Ainsi les fonctionnalités d'authentifiaction, de cryptage et de respect de l'intégrité sont séparées, ce qui rend difficile le piratage de tout le système avec la même méthode. De plus cela permet également d'utiliser des algorithmes et des clés différentes pour chaque opération. Ainsi les opérations d'établissement de la communication (authentification) utilisent des sytèmes asymétriques (plus lent, mais à n'effectuer qu'une seule fois), tandis que les systèmes d'échange (cryptage des messages) est basé sur un algorithme symétrique et une clé secrète échangée lors de l'autenthification. Signalons encore que la couche SSL est utilisée pour différents services du protocole TCP/IP. Ainsi http, smtp, news, pop3 ou ldap ont tous leurs équivalents sécurisés, respectivement https, ssmtp, snews, spop3 et ssl-ldap. Pour plus de renseignements, il faut consulter [22].

[1] Cette société belge est l'une des plus importantes CA en Europe.

Chapitre 26

La sécurité du serveur

26.1 Introduction

La machine sur laquelle le serveur web est lancée est une machine cruciale pour l'entreprise. En effet, cette machine a accès à tous les fichiers utilisés par le serveur web, parmi lesquels, les pages HTML, mais aussi la base de données. Cette machine contrôle aussi le trafic entrant et sortant du serveur web. Il est donc essentiel de la protéger. Cette machine sera d'autant plus une ressource sensible de l'organisation qu'elle sera l'hôte d'autres services que le serveur web externe : les fichiers partagés de l'intranet, le serveur de mail, etc...

Outre les précautions habituelles pour sécuriser un ordinateur connecté à Internet, il est nécéssaire de protéger un peu plus le serveur : limiter les accès physiques à la machine, mais aussi limiter le nombre de personnes ayant le droit de se connecter sur la machine. Une bonne pratique consiste également à limiter le nombre de services que le serveur pourra rendre, de manière à limiter les risques d'intrusion via d'autres programmes.

Plus généralement, les ressources de l'entreprise, qui sont connectées à Internet, doivent faire l'objet d'une attention particulière de la part du service d'administration informatique. En effet, le nombre de machines et d'utilisateurs, mais aussi les informations disponibles rendent les réseaux locaux plus intéressants et plus faibles pour le pirate. Parmi les outils à mettre en place, notons :

- les logiciels d'audit statique, qui vont pouvoir identifier les points faibles des machines ;
- les logiciels de surveillance : ils peuvent identifier les changements survenus à une partie du système (détectant ainsi l'arrivée d'un pirate) ;
- les logiciels de surveillance du réseau, qui se chargent de tester la sécurité des différents services de réseau ;
- les logiciels d'alarme, qui opérent tout le temps, cherchant à détecter les signes d'une intrusion sur le réseau ou sur une machine particulière ;
- les backups, qui permettent de restaurer les dégats commis par un pirate et qui de manière générale sont une nécéssité pour la sauvegarde des informations.

Pour des informations complètes concernant la sécurité d'UNIX/Linux et Internet, il faut consulter [23].

26.2 Le contrôle d'accès

Le serveur web est un moyen efficace de distribuer de l'information aux utilisateurs d'Internet. Cependant, il est parfois nécéssaire de limiter l'accès à certaines informations :

- de l'information à destination des employés de la société : un intranet ;
- une publication électronique dont la table des matières est publique mais les articles payants ;
- des informations techniques à destination des clients seulement ;
- un système d'accès à distance pour des employés itinérants, comme les commandes des représentants.

Il existe en fait trois types de sécurité que l'on peut mettre en place pour arriver à restreindre l'accès à de l'information :

- l'utilisation d'URLs cachées ;
- la restriction de l'accès à un groupe particulier de machines (suivant leurs adresses IP) ;
- la restriction de l'accès à un groupe d'utilisateurs, en fonction de leur identification.

26.2.1 Les URLs cachées

Le moyen le plus simple de restreindre l'accès à certaines informations est de ne pas publier celle-ci. L'idée des URLs cachées est de créer des sous-répertoires d'un site web et de ne pas référencer de liens vers ceux-ci. Seules les personnes autorisées en connaissent l'existence. On peut comparer cette pratique à celle de mettre sa clé dans le pot de fleur. Elle est protégée, tant que personne ne sait qu'elle s'y trouve. La différence, avec Internet, c'est que les moteurs de recherche peuvent très bien tomber dessus par hasard, en lançant des robots, c'est-à-dire des procédures d'indexage automatique du contenu d'un site.

26.2.2 Les restrictions basées sur les machines

La plupart des logiciels de serveurs web permettent de restreindre l'accès à des machines particulières, sur base de leur adresse IP ou de leur nom DNS. Ce système est très efficace pour mettre en place un intranet. La seule nécessité est de posséder un réseau où les machines sont connectées via le protocole TCP/IP. La restriction sur le nom DNS est aussi efficace, surtout si les adresses IP sont très hétérogènes et nombreuses, ce qui peut arriver pour une société disséminée dans le monde. Comme on le voit l'intranet n'est pas une limitation physique en terme de distance, mais bien une restriction logique sur l'accès à l'information. L'avantage de cette technique est qu'elle est transparente pour l'utilisateur (qui a accès à l'information). Cependant, elle n'est pas complètement sécurisée, car il est possible de trafiquer les adresses IP, pour faire croire que la requête vient d'une adresse IP acceptée. Il est également possible de trafiquer le serveur DNS, de manière à ajouter des noms acceptés dans celui-ci. Ce serveur est donc un élément critique du système. Les Firewalls permettent entre autre d'implémenter ce genre de solution, commme nous le verrons au chapitre 27.

26.2.3 Les restrictions basées sur les utilisateurs

La manière la plus efficace de restreindre l'accès est l'utilisation d'un système d'identification des personnes qui veulent se connecter. Une paire login/password est créée pour chaque utilisateur et seuls ceux qui sont référencés ont accès aux répertoires ou aux fichiers d'un certain serveur web. Lorsque l'utilisateur se connecte, via le navigateur, sur une partie privée du site, le serveur demande son login et son mot de passe, ce que le browser relaye en affichant une boîte de dialogue. Le problème des mots de passe est qu'ils sont souvent oubliés, qu'ils doivent, pour rester efficaces, être changés régulièrement et finalement, qu'ils doivent être différents pour chaque système (le mail, windows, l'intranet,...). Afin de pallier cet inconvenient, il est également possible de mettre en place un système basé sur des clés publiques et privées. Les systèmes les plus avancés sont basés sur une clé privée stockée dans un support physique, comme une smart card ou une empreinte digitale. L'avantage du sytème de contrôle des utilisateurs est qu'il permet à ceux-ci de se connecter depuis n'importe quel ordinateur reconnaissant leur support physique.

Chapitre 27

Firewalls

Le but de ce chapitre est de fournir une introduction aux firewalls. En effet, traiter de la mise en place d'un firewall dépasserait largement le cadre de ce présent ouvrage. Pour trouver des informations concernant l'implantation d'un firewall, il faut consulter [33].

27.1 Principe des firewalls Internet

Les firewalls constituent un type de sécurité de réseau efficace. Un firewall agit comme une porte coupe-feu permettant à un incendie de ne pas se répandre dans le bâtiment :

- il restreint l'accès à un point précis ;
- il empêche les agresseurs de s'approcher de vos autres défenses ;
- il restreint la sortie à un point d'accès.

Un firewall est le plus souvent installé au point où votre réseau interne à protéger est connecté à l'Internet, comme le montre la FIG. 27.1.

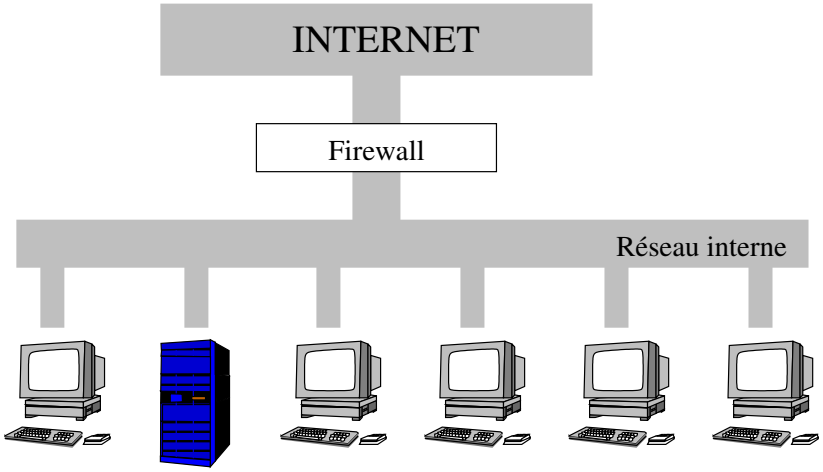


FIG. 27.1 – Un firewall séparant généralement un réseau de l'Internet

Tout le trafic provenant de l'Internet ou partant de votre réseau interne vers Internet, passe à travers le firewall. Ce dernier peut donc vérifier le trafic et éventuellement le limiter voire l'empêcher. Cela veut dire que tout ce qui transite – e-mail, transfert de fichiers, login distant – est conforme à la politique de sécurité implantée.

Un firewall est constitué d'une partie «hardware» et d'une partie «software». Les composants matériels peuvent être composés d'un routeur, d'un ordinateur hôte ou d'une combinaison d'ordinateurs, de routeurs

et de réseaux.

Bien qu'un firewall ne soit pas inviolable et que sa mise en oeuvre peut être coûteuse, aussi bien en termes financiers que techniques, il est actuellement le moyen le plus sûr pour connecter un réseau sur l'Internet.

Il existe principalement deux approches pour les firewalls :
- le filtrage de paquets ;
- le firewall purement mandataire.

Ces deux approches seront étudiées plus loin.

27.1.1 Que peut faire un firewall ?

Les firewalls peuvent faire plusieurs choses pour améliorer la sécurité :
- un firewall est un goulet d'étranglement dans lequel tout le trafic entrant et sortant doit passer et permet donc de concentrer la sécurité à l'endroit où le réseau est connecté à Internet ;
- le firewall autorise le passage uniquement aux services «approuvés» et uniquement dans le cadre de règles bien définies ;
- comme tout le trafic passe par le firewall, il est l'endroit idéal pour collecter des informations sur l'utilisation des systèmes et des réseaux ;
- le firewall peut empêcher que des problèmes touchant certaines sections du site ne se transmettent à d'autres sections.

27.1.2 Que ne peut pas faire un firewall ?

Le firewall ne peut pas résoudre tous les problèmes liés à la sécurité :
- le firewall ne peut pas éviter qu'un utilisateur copie des données sur une disquette ou un cd-rom et sort du bâtiment avec l'un ou l'autre dans sa malette ;
- s'il existe d'autres points de connection vers l'Internet, comme des modems, le trafic peut éviter de passer par le firewall et le réseau est donc menacé ;
- un firewall est destiné à protéger contre des menaces connues, il est donc impossible d'implanter un firewall une fois pour toute en espérant qu'il protégera éternellement ;
- un firewall ne peut pas protéger contre les virus.

27.2 Définitions

Nous allons commencer par donner quelques définitions de termes qui seront utilisés dans la suite de ce chapitre.

Firewall Un composant ou un ensemble de composants qui restreignent l'accès entre un réseau protégé et l'Internet ou d'autres ensembles de réseaux.

Hôte Un ordinateur rattaché au réseau.

Bastion Un ordinateur qui doit être hautement sécurisé car il est vulnérable aux attaques, généralement parce qu'il a un accès à Internet.

Hôte à double réseau (Dual homed host) Un ordinateur classique doté d'au moins deux interfaces (ou homes).

Paquet L'unité fondamentale de communication sur Internet.

Filtrage de paquets L'action qu'un dispositif entreprend pour contrôler sélectivement le flux de données vers et depuis un réseau. Les filtres de paquets autorisent ou bloquent les paquets entrant ou sortant du réseau.

Réseau périphérique (perimeter network) Un réseau ajouté entre un réseau protégé et un réseau extérieur afin de fournir une couche de sécurité supplémentaire. Une telle zone est parfois appellé De-Militarized Zone (DMZ).

Serveur mandataire ou proxy Un programme qui communique avec les serveurs externes à la demande des clients internes. Les clients mandataires parlent aux serveurs mandataires qui relaient les requêtes approuvées vers les vrais serveurs et relaient les réponses vers les clients.

27.3 Architectures des firewalls

Cette section décrit plusieurs façons d'implanter un firewall.

27.3.1 Architecture d'hôte à double réseau

Cette architecture est construite autour d'un ordinateur possédant deux interfaces réseau, l'une étant connectée à Internet et l'autre connectée au réseau protégé (Voir FIG. 27.2). Les paquets venant d'un des réseaux ne sont pas directement routés vers l'autre réseau, de sorte qu'il est possible de bloquer complètement le trafic IP.

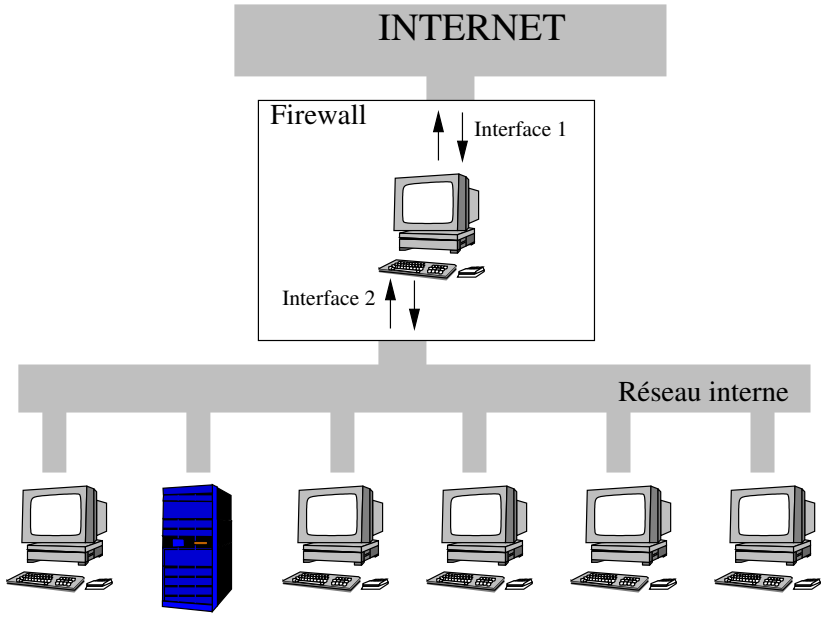


FIG. 27.2 – Architecture d'hôte à double réseau

Cette architecture peut fournir un très haut niveau de sécurité, puisqu'il est possible de bloquer tous les paquets. Cependant, il nécessite un travail considérable pour configurer ce type de firewall correctement, et donc de profiter pleinement des avantages de cette solution.

27.3.2 Architecture d'hôte à écran

Une architecture d'hôte à écran procure des services depuis un hôte qui n'est rattaché qu'au réseau Internet en utilisant un routeur séparé. Dans cette architecture, la principale sécurité est fournie par le filtrage de paquets, par exemple en empêchant les gens de contourner les serveurs mandataires pour établir des connexions directes. La FIG. 27.3 montre une version simplifiée de l'architecture d'hôte à écran.

Le bastion est situé sur le réseau intérieur. Le filtrage de paquets se trouve sur le routeur écran mis en place de telle manière que le bastion est le seul système sur le réseau interne auquel peuvent avoir accès les hôtes de l'Internet. Même dans ce cas, seules certaines connexions sont autorisées. Tout système externe essayant d'accéder aux systèmes ou aux services internes devra se connecter à cet hôte. Le bastion doit donc maintenir un haut niveau de sécurité.

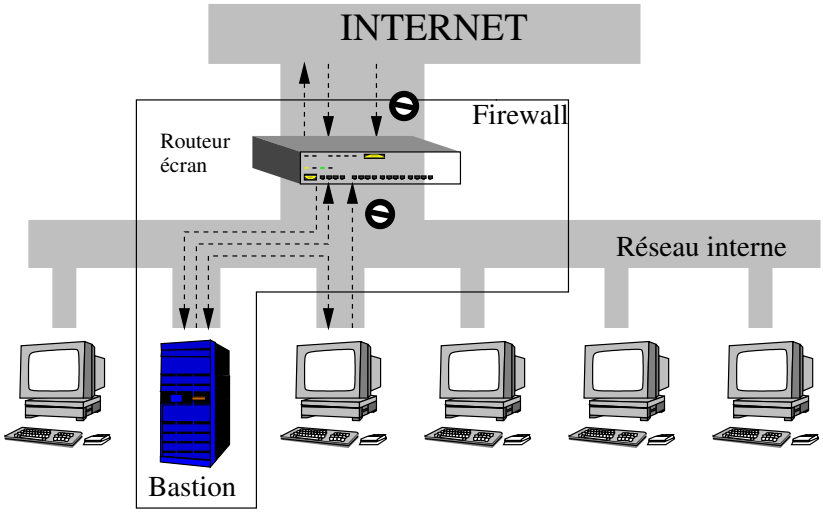


FIG. 27.3 – Architecture d'hôte à écran

La configuration du filtrage de paquets sur le routeur écran peut remplir les fonctionnalités suivantes :
- permettre à d'autres hôtes internes d'ouvrir des connexions vers des hôtes Internet pour certains types de services ;
- interdire toutes les connexions depuis des hôtes internes.

Même s'il peut sembler que cette architecture soit moins bonne que celle à double réseau, dans la pratique, cette architecture fournit dans la plupart des cas une meilleure sécurité. Ceci dit, si un agresseur arrive à s'introduire dans le bastion, plus rien ne s'interpose entre lui et les autres systèmes du réseau interne.

27.3.3 Architecture de sous-réseau à écran

L'architecture de sous-réseau à écran ajoute une couche de sécurité supplémentaire à l'architecture d'hôte à écran en ajoutant un réseau périphérique qui isole encore plus le réseau intérieur de l'Internet (Voir FIG. 27.4).

Les bastions étant par nature les machines les plus vulnérables, l'isolation d'un bastion sur un réseau périphérique permet de réduire l'impact d'une intrusion sur celui-ci, car l'agresseur ne dispose que d'un accès limité, puisqu'il n'est pas encore passer à travers le routeur interne.

On peut aller plus loin et créer plusieurs couches de réseaux périphériques entre le monde extérieur et le réseau interne. Les services les moins fiables et les plus vulnérables sont placés sur les périphériques extérieurs, loin du réseau interne.

27.3.4 Variations sur les architectures

Les architectures présentées sont les plus fréquentes. On rencontre cependant de nombreuses variations basées sur l'architecture de sous-réseau écran. En effet, une grande souplesse est possible dans la configuration et la combinaison de composants de firewalls. Voici quelques variations que l'on retrouve régulièrement :
- on peut utiliser plusieurs bastions, chacun étant responsable d'un service particulier ;
- on peut fusionner le routeur intérieur et extérieur en un seul routeur ;
- on peut fusionner le bastion et le routeur extérieur ;
- on peut utiliser plusieurs routeurs extérieurs entre le routeur intérieur et différentes portions du réseau interne à protéger.

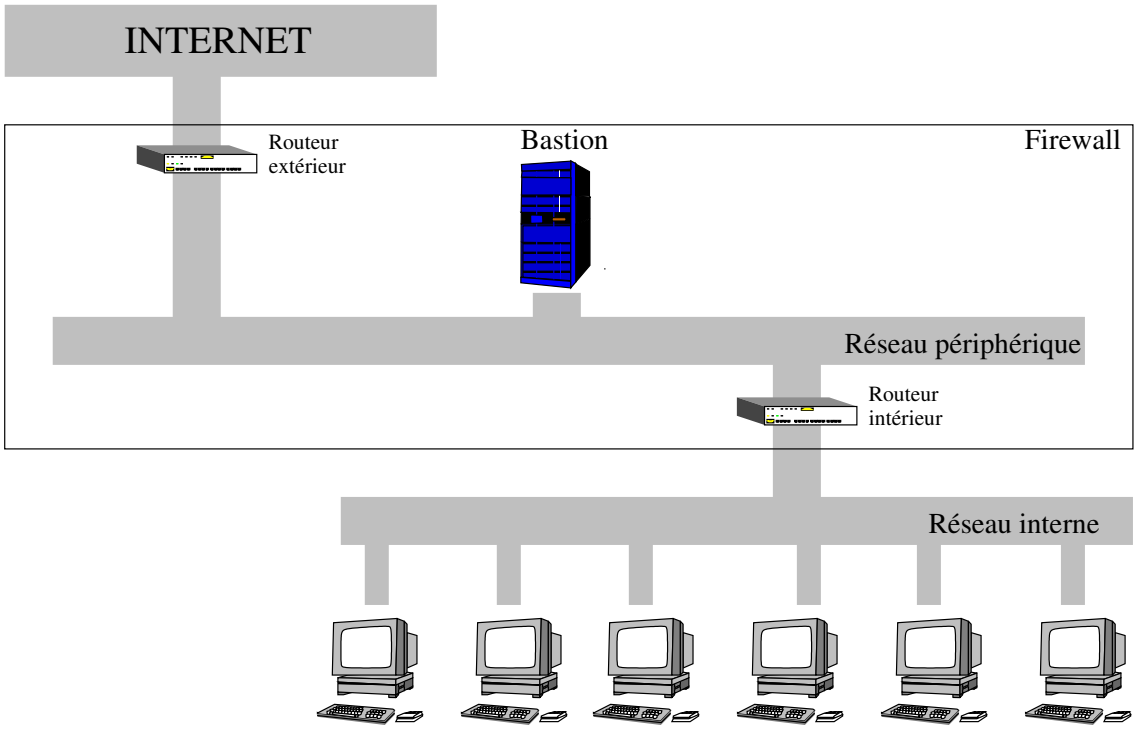


FIG. 27.4 – Architecture de sous-réseau écran

27.3.5 Firewalls internes

Bien que les firewalls soient essentiellement utilisés pour protéger un réseau interne du réseau Internet, il est également possible d'utiliser des firewalls à l'intérieur du réseau interne, afin d'empêcher certains types de services entres des portions différentes de celui-ci. Il existe de nombreuses raisons expliquant cette option :

- il existe des réseaux de test ou de recherche où tout ce qui se passe n'est pas toujours parfaitement contrôlé ;
- il existe des réseaux qui sont moins sûrs que le reste du site, par exemple des réseaux de démonstration ou de formation où des personnes extérieures sont souvent présentes ;
- il existe des réseaux qui sont plus sécurisés que le reste du site, comme des projets de développement secrets où des informations confidentielles circulent.

Dans ces situations, on réalise des firewalls internes, c'est-à-dire des firewalls qui sont situés entre deux parties de la même organisation.

27.4 Les bastions

Un bastion peut être comparé au hall d'entrée d'un immeuble : les personnes étrangères ne peuvent pas avoir accès aux étages ou emprunter l'ascenseur, mais ils peuvent se déplacer librement dans celui-ci.

Le bastion est très exposé du fait qu'il est connu de l'Internet, et qu'il peut servir de base pour une attaque vers d'autres machines du réseau interne. Il est donc important de prêter une attention particulière à sa configuration. De manière générale, il y a deux principes à suivre :

1. Tout service se trouvant sur le bastion peut comporter des trous de sécurité, il faut donc que le bastion soit le plus simple possible, c'est-à-dire qu'il fournisse le minimum de services et avec le moins de privilèges possible.

2. Il faut toujours se demander ce qu'il se passerait si le bastion tombait entre les mains d'un agresseur. En particulier, il faut s'assurer que les clients du réseau interne n'accordent que la confiance minimale au bon fonctionnement du bastion, afin d'éviter que tout le système de sécurité s'écroule en cas d'attaque réussie.

27.4.1 Choix d'une machine pour un bastion

D'une manière générale, il est conseillé d'utiliser le système d'exploitation UNIX/Linux. En effet, c'est celui qui offre le plus d'outils Internet et tous les services Internet sont disponibles sous ce système d'exploitation.

Contrairement à ce que l'on pourrait penser, il n'est pas nécessaire d'avoir une machine puissante pour un bastion. En effet, la principale limitation au niveau de la rapidité de traitement est plus la vitesse de la connexion que celle du CPU de la machine. De plus, si elle est victime d'une intrusion, une machine lente sera moins efficace pour attaquer les systèmes internes. Par contre, il faut penser à choisir une configuration avec une grande mémoire vive ainsi qu'une grande capacité de stockage. En plus de cela, il est bon également qu'il possède les fonctionnalités suivantes :

- un système de stockage interne permettant de faire un backup complet de la machine en une seule passe ;
- un lecteur de CD-ROM afin de pouvoir facilement intaller le système d'exploitation ainsi que les mises-à-jour des logiciels qui y sont installés ;
- il faut pouvoir installer facilement un autre disque à des fins de maintenance ;
- le disque de démarrage doit également pouvoir être facilement enlevé et rattaché à une autre machine, encore une fois pour des raisons de maintenance.

Il n'est pas nécessaire que le bastion possède des capacités graphiques. A partir du moment où il n'est pas utilisé pour des tâches journalières, un terminal simple pour une utilisation console est suffisante. De plus, des programmes comme X-Window sont également potentiellement attaquables.

27.4.2 Emplacemennt du bastion sur le réseau

Le bastion doit se trouver sur un réseau ne transportant pas de données confidentielles, et de préférence sur un réseau qui lui est propre.

La plupart des interfaces «Ethernet» et «Token Ring» peuvent fonctionner en «promiscuous» mode, dans lequel tous les paquets du réseau auquel est raccordé l'interface sont capturés, et non les seuls paquets destinés à la machine. Cette possibilité s'avère très utile pour l'analyse et les mises au point, mais peuvent également servir pour espionner le trafic. C'est pour cette raison qu'on essaye plutôt de mettre le bastion sur un réseau périphérique.

27.4.3 Choix des services fournis par le bastion

Le bastion fournit tous les services dont votre réseau a besoin sur Internet, ou qu'il veut fournir, et qui ne peuvent être offerts en toute sécurité sans filtrage de paquets. Aucun service n'étant pas destiné à fonctionner depuis ou vers Internet ne doit être disponible sur le bastion, comme les services d'impression. Il faut en effet supposer que le bastion sera compromis à un moment ou à un autre, et que tous les services présents seront donc diponibles depuis Internet.

On peut diviser ces services en quatre classes :

Les services sûrs. Ceux-ci peuvent être fournis via filtrage de paquets, si cette approche est utilisée (Dans un firewall purement mandataire, tout doit être fourni par le bastion, ou rien du tout).

Les services peu sûrs à l'origine mais que l'on peut sécuriser Ceux de cette catégorie peuvent être fournis par le bastion.

Les services peu sûrs, que l'on ne peut pas sécuriser Il faut les désactiver et les fournir via une machine sacrificielle, c'est-à-dire une machine n'offrant aucun autre service et qui est donc sacrifiée pour un seul.

Les services sans rapport avec Internet Il faut les désactiver.

Les services les plus communément acceptés et refusés sur les bastions sont tout les protocoles se situant juste au-dessus des couches TCP/IP :
- le courrier électronique (SMTP) ;
- le transfert de fichiers (FTP) ;
- la recherche d'informations par menus (Gopher) ;
- la recherche d'informations par mot-clés (WAIS) ;
- la recherche d'informations par hypertexte (HTTP) ;
- les news Usenet (NNTP) ;
- la recherche des noms de domaines (DNS).

27.4.4 Les comptes utilisateurs sur le bastion

Il faut limiter au maximum le nombre de comptes utilisateurs sur le bastion et si possible les supprimer. En effet, leur absence donnera une sécurité accrue, pour plusieurs raisons :
- la vulnérabilité des comptes eux-mêmes ;
- la vulnérabilité des services requis par les comptes ;
- une stabilité et une fiabilité réduite sur le bastion ;
- une corruption fortuite, ou non, de la sécurité du bastion par les utilisateurs ;
- des difficultés accrues de détection des attaques.

27.5 Le filtrage de paquets

Le filtrage de paquets est un mécanisme de sécurité réseau qui fonctionne en contrôlant les données qui arrivent du réseau, ou qui partent vers lui. Alors qu'un routeur[1] normal se pose comme unique question lorsqu'il reçoit un paquet la manière dont faire suivre celui-ci, un routeur filtreur de paquets se demande s'il doit faire suivre le paquet. Le routeur filtreur de paquets répond à cette dernière question en fonction de la politique de sécurité du réseau.

27.5.1 Utilité de filtrer les paquets

Le filtrage de paquets permet de contrôler, c'est-à-dire autoriser ou interdire, les transferts de données basés sur :
- l'adresse d'où proviennent les données (en théorie) ;
- l'adresse à laquelle elles sont destinées ;
- les protocoles de session et d'application utilisés pour transférer ces données.

La plupart des firewalls n'effectuent pas de contrôle sur le contenu des données. Certains logiciels, comme le produit Firewall-1 de CheckPoint, sont des exceptions limitées à cette règle.

Le filtrage de paquets permet entre autre :
- d'empêcher l'utilisation de Telnet afin d'éviter de se loger depuis l'extérieur sur une machine du réseau interne ;
- permettre à tout le monde d'envoyer des e-mails via SMTP ;
- une machine particulière peut envoyer des news via NNTP, les autres ne le pouvant pas.

De même, un firewall ne permet pas :
- d'autoriser un certain nombre d'utilisateursà utiliser Telnet et de l'interdire aux autres ;
- de ne pouvoir transférer que certains fichiers et pas d'autres.

Certaines protections ne peuvent être fournies que par des routeurs filtrants. Il est par exemple intelligent de rejeter tous les paquets qui possèdent une adresse source interne – c'est-à-dire des paquets prétendant provenir de machines internes alors qu'ils viennent de l'extérieur – car ce genre de paquets fait généralement partie d'attaques par falsification de paquets, où un agresseur prétend provenir d'une machine interne. Seul un routeur filtreur de paquets se trouvant sur un réseau périphérique est capable de reconnaître ce type de paquets. La FIG. 27.5 illustre ce type d'attaque.

[1] Le routeur est utilisé ici au sens large, c'est-à-dire que cela peut aussi bien être du matériel actif qu'un ordinateur dédié.

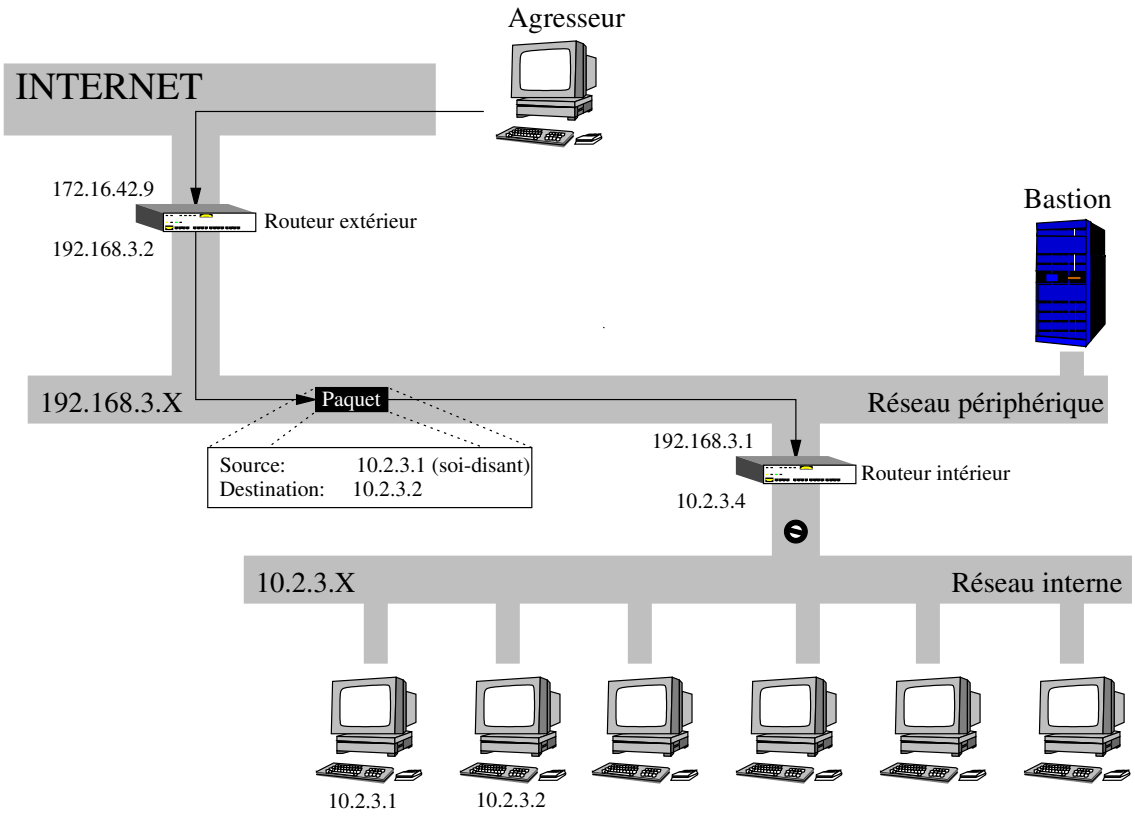


FIG. 27.5 – Falsification d'adresse source

27.5.2 Avantages du filtrage de paquets

Le filtrage de paquets comporte de nombreux avantages :
- un routeur écran peut protéger un réseau entier s'il est placé à un endroit stratégique ;
- il n'est pas nécessaire d'avoir la collaboration des utilisateurs car, contrairement aux systèmes mandataires, il n'est pas nécessaire d'installer des logiciels spécifiques sur les machines clientes ;
- de nombreux routeurs fournissent ce type de sécurité en standard.

27.5.3 Inconvénients du filtrage de paquets

Le filtrage de paquets comporte également des inconvénients :
- les outils actuels ne sont pas parfaits, en effet il existe quelques limitations communes :
  - les règles de filtrage sont difficiles à configurer ;
  - une fois configurées, les règles de filtrage sont difficiles à tester ;
  - certains outils ont des fonctionnalités incomplètes, rendant certains types de filtrage très difficiles voire impossibles ;
  - les outils contiennent des bogues qui risquent de créer plus de problèmes que les systèmes mandataires.
- certains protocoles ne conviennent pas au filtrage, en particulier les services basés sur le protocole RPC ;
- certaines règles ne peuvent pas être appliquées par des routeurs filtrant normaux, car il n'est par exemple pas possible de déterminer quel utilisateur envoie le paquet et vers quelle application ; donc seuls l'hôte d'où provient le paquet et le port auquel il est destiné sont visibles.

27.5.4 Configuration d'un routeur à filtrage de paquets

Le but n'est pas d'expliciter l'ensemble des éléments nécessaires à la configuration d'un routeur à filtrage de paquets, mais de faire quelques remarques utiles.

Tout d'abord, il faut rappeler que les protocoles sont généralement bidirectionnels, ils impliquent presque toujours qu'une partie envoie une requête ou une commande, et que l'autre renvoie une réponse quelconque. Cela ne sert à rien d'autoriser les paquets Telnet sortants, emportant donc les caractères entrés au clavier, et d'interdire les paquets entrants, ramenant les résultats des commandes lancées sur la machine distante.

De plus, bloquer la moitié d'une connexion ne sert à rien non plus, car de nombreuses attaques peuvent réussir si les agresseurs peuvent envoyer des paquets même s'ils ne recoivent pas de réponse. En effet, les réponses étant assez prévisibles, ils peuvent continuer leur conversation sans les voir.

Il ne faut pas confondre «services entrants et sortants», avec «paquets entrants et sortants». En effet, un service Telnet est sortant, mais nécessite des paquets entrants et sortants. Votre filtrage se fait sur les paquets entrants et sortants et non sur les services.

La manière la plus sûre de configurer un routeur filtreur de paquets est de partir d'une situation où aucun paquet n'est transmis. Ensuite seulement, on rajoute les règles permettant certains types de paquets entrants ou sortants d'être transmis. C'est une bien meilleur approche que celle partant d'une situation où tout est permis, et en ajoutant des règles afin d'empêcher certains paquets d'être transmis.

27.6 Systèmes mandataires

Le mandatement (proxying) donne accès à Internet à une seule machine, ou à un petit nombre, tout en paraissant y connecter tous vos hôtes. Ceux qui y ont réellement accès jouent le rôle de mandataires pour les autres machines et exécutent les requêtes de ces dernières.

Un serveur mandataire, pour un protocole ou un ensemble de protocoles, tourne sur un hôte à double réseau ou un bastion. Une machine cliente qui veut parler au monde extérieur, parle en fait avec le serveur mandataire. Celui-ci évalue les requêtes des clients, et relaie celles-ci vers le serveur réel si elles sont approuvées.

Pour l'utilisateur, il n'y a aucune différence entre communiquer avec le serveur mandataire ou avec le serveur réel. Le serveur, quant à lui, discute avec un utilisateur se trouvant sur le serveur mandataire, sans savoir qu'en réalité l'utilisateur se trouve ailleurs.

Les systèmes mandataires permettent de fournir un accès à Internet aux utilisateurs en automatisant les interactions avec le monde extérieur. L'utilisateur a l'impression de discuter directement avec le serveur réel puisque les systèmes mandataires cachent toutes les interactions. On peut voir à la FIG. 27.6 un schéma représentatif du principe.

Le logiciel mandataire tournant sans login utilisateur, l'hôte est donc à l'abris des manipulations des utilisateurs. De plus, il est impossible d'installer un logiciel non contrôlé pour atteindre Internet, puisque le serveur mandataire joue le rôle de point de contrôle.

27.6.1 Avantages du mandatement

D'une part, à partir du moment où les utilisateurs ont l'impression qu'ils ont directement accès à Internet, ils chercheront moins facilement à contourner le firewall installé. Les services mandatés permettent donc aux utilisateurs d'accéder aux services Internet depuis leurs propres systèmes.

Les systèmes mandataires comprennent les protocoles, et peuvent donc enregistrer plus facilement une trace efficace. Par exemple dans le cas du service FTP, ils peuvent se contenter d'enregistrer les commandes lancées et les résultats reçus du serveur, ce qui est beaucoup plus utile et concis qu'une trace de tous les paquets ayant transités.

27.6.2 Inconvénients du mandatement

Il existe aussi quelques inconvénients à l'utilisation des services mandatés :

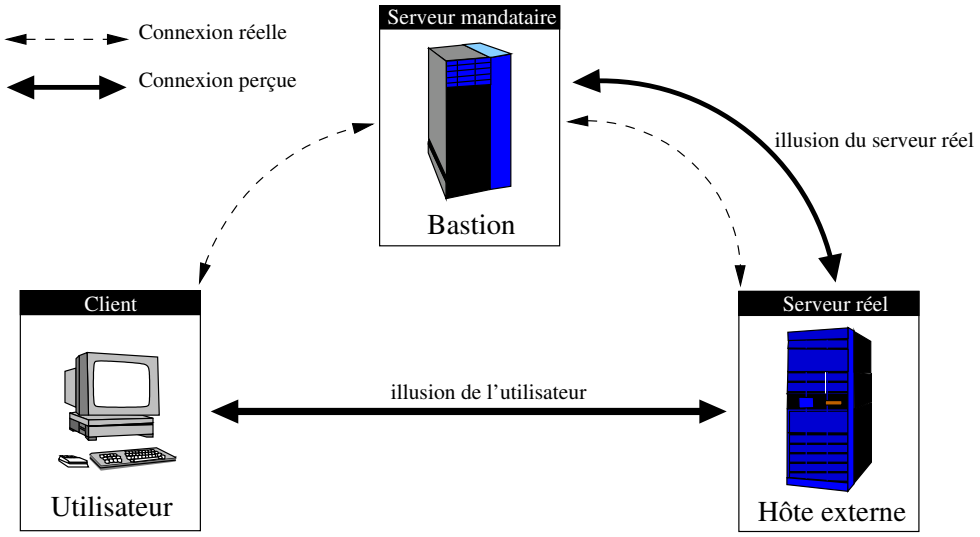


FIG. 27.6 – Mandataires – réalité et illusion

– les systèmes mandatés sont à la traîne des autres systèmes, il est plus difficile de trouver des versions éprouvées ;
– les systèmes mandatés peuvent demander des serveurs différents pour chaque service, car les solutions couvrant l'ensemble des protocoles de manière efficace sont rares ;
– il faut généralement effectuer des modifications sur les machines clients et/ou sur les procédures ;
– le mandatement n'est pas envisageable pour tous les services ;
– les systèmes mandatés ne protègent pas de toutes les faiblesses des protocoles.

27.6.3 Fonctionnement d'un système mandataire

Les détails de fonctionnement d'un système mandataire diffèrent d'un service à l'autre, et sortiraient largement du cadre du présent ouvrage. Certains services fournissent des services mandatés facilement ou automatiquement ; il suffit de changer la configuration du serveur normal. Mais pour la plupart, il faut installer un logiciel spécifique du coté serveur.

Du coté client, il faut l'une des deux choses suivantes :
– un logiciel client personnalisé connaissant la manière de contacter le serveur mandataire au lieu du serveur réel ;
– des procédures utilisateurs personnalisées dans lequel l'utilisateur utilise un logiciel standard permettant de se connecter sur le serveur mandataire et demander de se connecter sur le serveur réel.

27.6.4 Terminologie des serveurs mandataires

27.6.4.1 Mandataire au niveau application ou au niveau circuit

Un mandataire au niveau application connaît l'application précise pour laquelle il est mandaté : il comprend et interprète les commandes du protocole d'application. En général, un mandataire au niveau application utilise des procédures modifiées. La version la plus extrême d'un tel système est Sendmail qui implante un protocole «store-and-forward» (stocker et faire suivre).

Un mandataire au niveau circuit crée un circuit entre le client et le serveur sans interpréter le protocole d'application. En général, ce type de mandataire utilise des clients modifiés. L'avantage d'un mandataire au niveau circuit est qu'il fournit un support pour un grand panel de protocoles différents. Mais tous les protocoles ne peuvent pas être facilement gérés par ce système, par exemple le protocole FTP.

27.6.4.2 Mandataires générique et dédié

Un serveur mandataire dédié ne sert qu'à un seul protocole, en général il s'agit de mandataires au niveau application. Un serveur générique par contre sert pour plusieurs protocoles, et sont bien souvent des mandataires au niveau circuit.

27.6.4.3 Serveurs mandataires intelligents

Un serveur mandataire peut faire beaucoup plus que se contenter de relayer des requêtes, on parle alors de serveurs mandataires intelligents. Par exemple, le mandataire HTTP du CERN cache les données afin que plusieurs requêtes sur celles-ci n'aillent pas sur Internet.

Il est plus facile de faire évoluer des serveurs dédiés, au niveau application, en étendant les fonctionnalités au fur et à mesure de leur maturité, que ceux au niveau circuit.

Annexes

Annexe A

Les langages orientés objets

A.1 Programmation traditionnelle

Dans les langages de programmation traditionnels, les données sont séparées du code permettant de les manipuler. Certes, la plupart des langages proposent des possibilités de réunir plusieurs variables dans une même structure (**struct** enC++ ou **record** en Pascal), mais cela ne suffit pas toujours à représenter correctement la problématique.

Prenons comme exemple un point. On pourrait le représenter de la manière suivante :

```
Structure Point :
  Entier X
  Entier Y
Fin.
Fonction DessinerPoint(couleur est Entier,point est Point)
  OSDessinerPoint(point.X,point.Y,couleur)
Fin.
Fonction DeplacerPoint(point est Point,nx est Entier,ny est Entier)
  DessinerPoint(CouleurFond,point);
  point.X=nx;
  point.Y=ny;
  DessinerPoint(CouleurPlan,point)
Fin.
Principal
  Point pt;
  pt.X=10;
  pt.Y=10;
  DessinerPoint(CouleurPlan,pt);
  DeplacerPoint(pt,25,25);
Fin.
De même, on pourrait représenter un cercle par le code suivant :
Structure Cercle :
  Entier X
  Entier Y
  Entier R
Fin.
Fonction DessinerCercle(couleur est Entier,cercle est Cercle)
  OSDessinerCercle(cercle.X,cercle.Y,cercle.R,couleur)
Fin
Fonction DeplacerCercle(cercle est Cercle,nx est Entier,ny est Entier)
  DessinerCercle(CouleurFond,cercle);
  point.x=nx;
  point.y=ny;
```

```
  DessinerCercle(CouleurPlan,point)
Fin
```

A.2 Encapsulation

L'idée de l'encapsulation est de réunir au sein d'une même entité appellée «objet» ou «classe» tous les éléments, variables et fonctions, la représentant. Pour reprendre l'exemple du point, on pourrait écrire une classe :

```
Classe Point :
  Entier X
  Entier Y
  Fonction Dessiner(couleur est Entier)
  Fonction Deplacer(nx est Entier,ny est Entier)
Fin.
Fonction Dessiner de Point(couleur est Entier)
  OSDessinerPoint(X,Y,couleur)
Fin.
Fonction Deplacer de Point(nx est Entier,ny est Entier)
  Dessiner(CouleurFond) // La fonction de Point
  X=nx;
  Y=ny;
  Dessiner(CouleurPlan) // La fonction de Point
Fin.
Principal
  Point pt;
  pt.X=10;
  pt.Y=10;
  pt.Dessiner(CouleurPlan);
  pt.Deplacer(25,25);
Fin.
```

On peut directement voir l'avantage de cette approche, qui fournit un code «source» beaucoup plus clair en créant une entité entièrement autonome comportant tout ce qui est nécessaire pour la représenter ainsi que pour la manipuler.

De plus, il faut savoir que la plupart des langages permettent de spécifier un type d'accès aux différents éléments d'une classe. Ainsi il est possible d'imposer l'utilisation de fonctions pour manipuler certaines variables contenues dans la classe. Par exemple, une classe représentant une fraction serait formée de deux variables «numérateur» et «dénominateur». En forçant l'utilisation de fonctions, on peut éviter qu'un programmeur puisse mettre accidentellement la valeur de la variable «dénominateur» à 0.

A.3 Héritage

Pour illustrer l'héritage, le plus facile est de comparer un cercle à un point. En fait, le cercle peut être considéré comme un point avec un rayon, en d'autres termes, on peut envisager un cercle comme une évolution du point. C'est exactement cela le principe de l'héritage. Voyons un peu comment on pourrait «dériver» un cercle d'un point :

```
Classe Cercle hérite de Point :
  Entier R
  Fonction Dessiner(couleur est Entier)
  Fonction Deplacer(nx est Entier,ny est Entier)
Fin.
Fonction Dessiner de Cercle(couleur est Entier)
  DessinerCercle(X,Y,R,couleur)
Fin.
```

```
Fonction Deplacer de Cercle(nx est Entier,ny est Entier)
  Dessiner(CouleurFond) // La fonction de Cercle
  X=nx;
  Y=ny;
  Dessiner(CouleurPlan) // La fonction de Cercle
Fin.
Principal
  Cercle c;
  c.X=10;
  c.Y=10;
  c.R=20;
  c.Dessiner(CouleurPlan);
  c.Deplacer(25,25);
Fin.
```

Comme on peut le voir dans la déclaration, Cercle hérite de Point, c'est-à-dire de toute les entités composant le point, en particulier les coordonnées X et Y. De plus, on redéfinit les fonctions afin de dessiner un cercle et non un point.

L'avantage de l'héritage est donc de pouvoir récupérer des entités existantes et d'en étendre les capacités de manière très aisée.

A.4 Polymorphisme

Pour introduire le concept de «polymorphisme», il suffit de regarder les deux fonctions de déplacement du cercle et du point. En fait, d'un point de vue du code source, ces deux fonctions sont identiques. En réalité, la fonction «Dessiner» n'est pas la même pour les deux classes, c'est la raison pour laquelle on a dû réécrire cette fonction pour le cercle.

Pour éviter cela, le polymorphisme permet de déclarer une fonction «virtuelle». Cela veut dire que lorsqu'on rencontre un appel à une fonction virtuelle, c'est au moment de l'exécution qu'on choisit la fonction qui sera appellée en fonction de la nature exacte de la classe. Reprenons le même code en utilisant le polymorphisme :

```
Classe Point :
  Entier X
  Entier Y
  Fonction Dessiner(couleur est Entier) est virtuelle
  Fonction Deplacer(nx est Entier,ny est Entier)
Fin.
Classe Cercle hérite de Point :
  Entier R
  Fonction Dessiner(couleur est Entier) est virtuelle
Fin.
Fonction Dessiner de Point(couleur est Entier)
  DessinerPoint(X,Y,couleur)
Fin.
Fonction Deplacer de Point(nx est Entier,ny est Entier)
  Dessiner(CouleurFond) // Appel dépend de la classe au moment de l'exécution
  X=nx;
  Y=ny;
  Dessiner(CouleurPlan) // Appel dépend de la classe au moment de l'exécution
Fin.
Fonction Dessiner de Cercle(couleur est Entier)
  DessinerCercle(X,Y,R,couleur)
Fin.
Principal
  Point pt;
  Cercle c;
```

```
    c.X=pt.X=10;
    c.Y=pt.Y=10;
    c.R=20;
    pt.Dessiner(CouleurPlan);
    c.Dessiner(CouleurPlan);
    pt.Deplacer(25,25);   // Utilisera 'Deplacer' de Point
    c.Deplacer(25,25);    // Utilisera 'Deplacer' de Cercle
Fin.
```

On remarque directement que le polymorphisme permet de récupérer une plus grande partie du code existant. De plus, le programmeur qui dérive une classe de Point, ne doit pas savoir comment fonctionne réellement la fonction Deplacer, il doit juste se concentrer sur la partie qui distingue sa classe de celle dont il hérite.

Il existe quelques langages de programmation, comme le C++, qui permettent l'héritage multiple. Cette possibilité augmente la puissance de la programmation orientée objets, mais augmente également sa difficulté de conception.

A.5 Accès aux méthodes et aux variables d'objets

La plupart des langages offrent des moyens afin de réguler l'accès aux méthodes et variables des objets. Ils offrent généralement trois modificateurs d'accès :

public C'est l'accès public : les entités ainsi définies peuvent être accédées de n'importe où.

protected C'est l'accès protégé : les entités ainsi définies ne peuvent être accédées que de la classe qui les définit ainsi que de toutes les classes qui dérivent de cette dernière.

private C'est l'accès privé : les entités ainsi définies ne peuvent être accédées que de la classe.

En plus de cela, certains langages comme le C++, permettent de modifier l'accès d'une entité lors d'une dérivation, suivant le modificateur assigné à cette dérivation. Le TAB. A.1 montre l'influence sur l'accès lorsque l'on assigne un modificateur à une dérivation. Ainsi si on dérive une classe B d'une classe A de manière privée, aucune des variables et des méthodes de B héritées de A ne pourront être accèder en dehors de celle-ci.

TAB. A.1 – Modification d'accès

Accès	Modificateur lors de dérivation		
	public	protected	private
public	public	protected	private
protected	protected	private	private
private	private	private	private

A.6 Avenir des langages traditionnels

Les avantages de l'approche orientée objets sont nombreux :
- Représentation des entités plus proche de la réalité.
- Récupération facile de code existant.
- Faire facilement évoluer du code existant.
- Diminuer le nombre de bogues grâce à la séparation claire entre les intervenants.

En d'autres termes, une classe bien pensée vaut toujours mieux que des variables et des fonctions permettant le même type de manipulation. Au vu de ses nombreux avantages, il est clair que la programmation orientée objet, même si son utilisation signifie toujours une certaine surcharge de code par rapport à l'approche tradionnelle, va entraîner la disparition des langages traditionnels.

La meilleure preuve est que la plupart des langages, dont le C (qui est devenu le C++ ou Objectif C), le Pascal, Perl voir même le Fortran ont implantés les extensions nécessaires à la programmation orientée objets.

Annexe B

Syntaxe des expressions régulières Perl

La puissance des expressions régulières Perl constitue sans aucun doute un des atouts majeurs de ce langage, car elles permettent de manipuler des fragments de texte. En particulier, elles permettent d'isoler dans une chaîne de caractères un certain nombre de sous-chaînes ayant des caractéristiques précises. Depuis, de nombreux autres langages de programmation ont introduit un support pour ces expressions. Dans la suite de cette annexe, certains exemples seront étudiés. Pour une étude complète, il faut consulter [26].

```
/abc/
```

Chaque occurence de «abc» dans la chaîne.

```
/\abc/
```

«abc» en tête de la chaîne.

```
/abc$/
```

«abc» en fin de chaîne.

```
/abc|def/
```

Chaque occurence de «abc» ou «def».

```
/ab{2,4}c/
```

«a» suivi de 2 à 4 «b» puis suivi de «c». L'omission de la seconde valeur («ab{2,}c») se traduit par l'intervalle $\{2, \infty\}$.

```
/ab*c/
```

«a» suivi ou non d'un ou plusieurs «b», suivi de «c».

```
/ab+c/
```

«a» suivi d'au moins un «b», puis de «c».

```
/ab?c/
```

«a» suivi ou non d'un «b», puis de «c».

```
/./
```

N'importe quel caractère isolé à l'exception du retour de chariot («\n» dans la plupart des langages). Ainsi l'expression «/p..l/» représente la lettre p suivi d'une quelconque paire de caractères, suivi de la lettre l, comme «perl» ou «pull».

```
/[abc]/
```

N'importe lequel des trois caractères appartenant à la classe. Ainsi, «/[abc]+/» renvoie aux chaîne «abcab» ou encore »acbc».

```
/\d/
```

Un quelconque chiffre, correspond à l'expression «/[0-9]/».

```
/\w/
```

Tout caractère pouvant appartenir à un mot, correspond à l'expression «/[a-zA-Z0-9_]/».

```
/\s/
```

Tout caractère repésentant un espace, correspond à l'expression «/[\r\t\n\f]/».

```
/\b/
```

Un séparteur de mots ou un retour arrière. Ainsi, l'expression «/table\b/» isole le mot «table» sans pour autant retenir «tablette».

```
/[^abc]/
```

Tous les caractères n'appartenant pas à la classe.

```
/\D/
```

Tout caractère à l'exception des chiffres, correspond à l'expression «/[^0-9]/»

```
/\W/
```

Tout caractère non associé à un mot, correspond à l'expression «/[^a-zA-Z0-9_]/».

```
/\S/
```

Tout caractère à l'exception des espaces, correspond à l'expression «/[^\r\t\n\f]».

```
/\B/
```

Absence de séparateur de mots. L'expression «/transport\B/» retient «transport» mais pas «transport aérien».

```
/\*/
```

Le signe «*» remplace n'importe quel caractère. La barre oblique inversée s'emploie afin d'ôter leur signification particulière aux caractères et signes eux-mêmes utilisés en tant qu'opérateurs dans les expressions régulières.

```
/(abc)/
```

«abc» partout dans la chaîne, les parenthèses permettant dans ce cas d'enregistrer les chaînes assorties au sein des variables $1, $2 ...

Exemple 1

```
/nom=(.*)/
```

Assigne à la variable $1 tout caractère apparaissant éventuellement à la suite de la chaîne «nom=».

Exemple 2

```
/nom=(.*)&utilisateur=\1/
```

Stocke dans $1 tout caractère apparaissant éventuellement à la suite de «nom=», après quoi Perl substitue à \1 la valeur contenue dans $1 afin de vérifier l'égalité des chaînes.

Exemple 3

```
/nom=([^&]*)/
```

Consigne dans $1 tout caractère apparaissant éventuellement entre la fin de la chaîne «nom=» et avant le signe «&» suivant.

Exemple 4

```
/nom=([^&]+)&age=(.*)$/
```

Enregistre dans $1 tout caractère apparaissant éventuellement entre la fin de la chaîne «nom=» et avant le signe «&» suivant, puis isole ce dernier signe avant d'assigner à $2 tout caractère figurant à la suite de la chaîne «age=» jusqu'à la fin de la ligne courante.

On remarque dans ce dernier exemple, l'intérêt des expressions régulières afin de décoder des informations reçues par l'interface CGI.

Annexe C

Bases de données relationnelles et SQL

C.1 Base de données

Une base de données est par définition une collection de données sur laquelle on peut faire des requêtes, c'est-à-dire des demandes d'informations. Un exemple de base de données non-électronique est une bibliothèque. Celle-ci stocke des livres, des magazines et des articles. De même, une base de données électronique stocke des informations.

Lorsque l'on parle de base de données, on s'imagine toujours cela sous la forme de logiciels gourmands. Même si le plus souvent, on utilise en effet des logiciels afin de travailler avec des bases de données, ce n'est pas forcément le cas. L'extrait suivant d'un fichier texte peut être comparé à une base de données :

```
John, Smith, 28/01/1970, Atlanta
Rick, Brand, 17/02/1968, Chicago
Frederic, Dupond, 03/11/1981, Bruxelles
```

Dans ce cas, on parle de fichier séquentiel, car il faut parcourir le fichier ligne après ligne pour produire le résultat de n'importe quelle requête. Il est évident que ce type de base de données ne serait pas très efficace pour gérer des grands volumes de données. De plus, supposons qu'un second fichier contienne d'autres données sur les personnes, il serait impossible de relier ces informations, car il n'y a pas de notion de relation entre les données. C'est la raison pour laquelle pour représenter des volumes d'informations importants, on utilise des bases de données relationnelles.

C.2 Bases de données relationnelles

Dans une base de données relationnelle, les informations sont stockées dans des tables pouvant éventuellement être reliées entre elles. Les TAB. C.1 et TAB. C.2 montrent deux tables représentant respectivement les clubs de football et les joueurs de ceux-ci.

TAB. C.1 – Table des clubs

Club #	Nom	Ville	Coach
1	Anderlecht	Bruxelles	Antheunis
2	Standard de Liège	Liège	Preud'homme
3	Club Brugge	Bruges	Solied

On voit également que ces deux tables sont reliées entre elles par l'intermédiaire d'un champ commun «Club #». Grâce à ce lien, on peut faire une requête du type «liste des joueurs dont le club auquel ils appartiennent se trouve dans la ville de Bruxelles ou de Liège». Ce champ est appellé «clé primaire», car il permet de désigner chaque entrée de la table des clubs de manière unique. De même, le champ «Joueur #» est la clé primaire de la table des joueurs, puisque chaque joueur a un numéro unique.

TAB. C.2 – Table des joueurs

Joueur #	Nom	Position	Club #
1	Stocia	Avant	1
2	Goosens	Avant	2
3	Bassegio	Milieu	1
4	Vermant	Milieu	3

D'une manière générale, on peut définir trois types de liens entre une table «Voitures» et table «Employés» :

Lien 1-1. A un élément de la table «Voitures» correspond un seul élément de la table «Employés» et réciproquement, en d'autres termes chaque employé utilise sa propre voiture. Pour implanter un tel lien, il suffit qu'il y ait dans la table «Voitures» un champ correspondant à la clé primaire de la table «Employés».

Lien 1-n. A un élément de la table «Voitures» correspond plusieurs éléments de la table «Employés» et à un élément de la table «Employés» correspond un seul élément de la table «Voitures», en d'autres termes chaque employé utilise une voiture et chacune d'elle peut être utilisée par plusieurs employés. Pour implanter un tel lien, il suffit qu'il y ait dans la table «Employés» un champ correspondant à la clé primaire de la table «Voitures».

Lien n-n. A un élément de la table «Voitures» correspond plusieurs éléments de la table «Employés» et réciproquement, en d'autres termes chaque employé peut utiliser plusieurs voitures et chacune d'elles peut être utilisée par plusieurs employés. Pour implanter un tel lien, il faut créer une table séparée contenant un champ correspondant à la clé primaire de la table «Voitures» et un champ correspondant à la clé primaire de la table «Employés».

C.3 Stuctured Query Language

Pour travailler avec une base de données, il faut pouvoir créer des tables, insérer et effacer des données, faire des requêtes, etc... Pour faire cela, on a besoin d'une interface entre la base de données et les utilisateurs. Le but du Stuctured Query Language (SQL) est de fournir un moyen d'interagir. La plupart des bases de données relationnelles utilisent SQL, avec l'une ou l'autre adaptation propre, comme langage de manipulation standard.

Les instructions SQL peuvent être séparées en trois groupes :
- les instructions de définition de données, comme **CREATE** ou **DROP** ;
- l'instruction de recherche **SELECT** ;
- les instructions de manipulation de données : **INSERT**, **DELETE** et **UPDATE**.

Les instructions de définitions de données sont celles qui différent les plus d'une base de données à l'autre, c'est pourquoi on ne va pas les étudier ici. Pour trouver des renseignements les concenant, il vaut mieux se reporter à des livres spécialement écrits pour la base de données utilisée.

C.3.1 L'instruction SELECT

SELECT est l'instruction principale de SQL. Elle permet d'effectuer des recherches dans une ou plusieurs tables. Sa forme la plus simple à la forme :

```
SELECT champs FROM tables WHERE condition;
```

où

champs Correspond aux noms des champs demandés, séparés par des virgules. La cas particulier est le caractère '*' qui signifie tous les champs.

tables Correspond aux noms des tables contenant les champs demandés, séparés par des virgules.

condition Correspond à une condition sur les données renvoyées. On peut utiliser les opérateurs **=**, **<>**, **OR** et **AND** ainsi que des parenthèses pour construire des requêtes complexes.

La requête suivante renvoit tous les champs de la table «documents», dont le champ «auteurid» est 2 :

```
SELECT * FROM documents WHERE auteurid=2;
```

La requête suivante renvoit les champs «titre» et «parution» de la table «documents», dont la champ «auteurid» est soit 2 soit 3 :

```
SELECT titre, parution FROM documents WHERE (auteurid=2 OR auteurid=3);
```

La requête suivante renvoit les champs «titre» et «parution» de la table «documents» ainsi que le champ «nationalité» de la table «auteurs», tels que le champ «auteur» de la table «auteurs» vaut 'Emile Zola', le résultat devant être classé suivant le titre :

```
SELECT documents.titre, documents.parution, auteurs.nationalite
FROM documents, auteurs
WHERE (documents.auteurid=auteurs.auteurid AND auteurs.auteur='Emile Zola')
ORDER BY documents.titre;
```

On voit tout d'abord que le nom des champs peut être préfixé par le nom de la table correspondante. Cela permet de manipuler dans une même requête des champs ayant le même nom mais provenant de tables différentes. Lorsque le nom du champ est unique dans l'ensemble des tables d'une requête, il n'est pas nécessaire de le préfixer.

Une autre partie intéressante de cette requête est l'expression «documents.auteurid=auteurs.auteurid». Cela permet de faire le lien entre deux tables, dans ce cas cela signifie que la table «documents» possède un champ «auteurid» qui correspond à la clé primaire «auteurid» de la table «auteurs». Dès que, dans une requête quelconque, il faut utiliser un lien entre tables, il faut une expression similaire dans la clause **WHERE**.

C.3.2 L'instruction INSERT

L'instruction **INSERT** permet d'ajouter des enregistrements dans une table. Cette instruction peut opérer de deux façons distinctes : ajouter explicitement des valeurs (**INSERT...VALUES**) ou récupérer des valeurs d'une table existante (**INSERT...SELECT**). La syntaxe de base de l'instruction **SELECT** est :

```
INSERT INTO documents (doc_id, titre, auteurid, parution)
VALUES (8, 'Maîtrise de PHP', '2', '2001-01-08');
```

ou

```
INSERT INTO documents (doc_id, titre, auteur, parution)
SELECT doc_id, titre, auteur, parution
FROM anciens_documents
WHERE parution>='2001-01-01';
```

C.3.3 L'instruction UPDATE

L'instruction **UPDATE** permet de modifier un ou plusieurs champs dans un enregistrement comme le montre l'exemple suivant :

```
UPDATE documents
SET titre='Maîtrise totale de PHP', auteur='L.Auquière'
WHERE doc_id=8;
```

La clause **WHERE** n'est pas obligatoire, mais si elle ne se trouve pas, tous les records de la table en question seront modifiés.

C.3.4 L'instruction DELETE

L'instruction **DELETE** permet d'effacer un enregistrement ou un ensemble d'enregistrements.

```
DELETE FROM documents WHERE parution<'1999-01-01';
```

Annexe D

Acronymes

Pour trouver des renseignements sur d'autres acronymes, il est toujours possible de consulter le site http :///www.whatis.com.

ANP Assistants Numériques Personnels
Ce sont par exemple les Palm Pilot.

API Application Programmer Interface
Il s'agit de la manière dont un programmeur voit l'ensemble des ressources disponibles pour un système donné.

AWT Abstract Windowing Toolkit
Il s'agit d'un ensemble de classes standardisé permettant de construire une interface graphique.

BER Bit Error Rate
Moyenne du nombre de bits envoyés pour un bit envoyé en erreur.

CA Certification Authority
Autorité délivrant des certificats digitaux et assurant le lien entre un certificat et une personne ou une société. Aussi appelée Trusted Third Party.

CERN Centre Européen de Recherche Nucléaire
Il s'agit d'une institution de recherche européenne basée à Lausanne. On lui doit l'invention du langage HTML ainsi que les débuts des serveurs web modernes.

CGI Common Gateway Interface
Interface permettant à des programmes de communiquer avec le serveur web.

CML Chemical Markup Language
Une application XML permettant de traiter les molécules chimiques.

CSS Cascading Style Sheets
Représente les styles introduits par le W3C.

DHTML Dynamic HTML
Une évolution du HTML introduisant des éléments dynamiques dans le HTML.

DMZ De-Militarized Zone
Réseau tampon entre un réseau protégé et un réseau extérieur.

DNS Domain Name Service
Service permettant d'associer un nom à une adresse IP.

DTD Document Type Declaration
Permet de décrire la structure d'un langage XML.

FTP File Transfert Protocol
Protocole basé sur TCP/IP et permettant l'échange de fichiers.

GIF Graphics Interchange Format
Un format d'image utilisé dans le World Wide Web.

GUI Graphical User Interface
Intreface graphique permettant à l'utilisateur d'interagir avec les programmes.

HTML HyperText Markup Language
Il s'agit du format des documents standards sur le World Wide Web.

HTTP HyperText Transfert Protocol
Protocole basé sur TCP/IP permettant d'échanger le contenu de documents.

ICMP Internet Control Message Protocol
Protocole permettant la collecte d'informations concernant la gestion d'un réseau TCP/IP.

IP Internet Protocol
Il s'agit d'un des protocoles utilisés par le réseau Internet.

JDBC Java DataBase Connectivity
Interface Java permettant d'accèder de manière transparente et indépendante à des bases de données.

JPEG Joint Photographic Experts Group
Un format d'image utilisé dans le World Wide Web.

JSSS JavaScript Sytle Sheet
Extension de JavaScript introduit par Netscape pour supporter les feuilles de styles et abandonné depuis.

JVM Java Virtual Machine
Machine virtuelle utilisée par Java et devant être implantée par les différents systèmes d'exploitation.

LAN Local Area Network
Réseaux d'ordinateurs locaux à un site.

MAN Metropolitan Area Network
Réseaux d'ordinateurs se trouvant sur des sites différents.

MathML Mathematical Markup Language
Une application XML permettant de traiter les formules mathématiques.

MIME Multipurpose Internet Mail Extensions
Méthode permettant d'associer des applications avec des types de documents sur le réseau Internet.

NIC Network Interface Card
Carte réseau utilisée dans les ordinateurs.

NIS Network Information Service
Protocole permettant de distribuer des informations sur le nom des machines en interne.

NNTP Network News Transfert Protocol
Protocole basé sur TCP/IP et permettant la création de «newsgroup».

NTP Network Time Protocol
Protocole permettant de distribuer l'heure et la date exacte sur un ensemble de machines.

ODBC Open DataBase Connectivity
Convention permettant d'intérroger des bases de données sans de devoir utiliser directement du code natif pour celles-ci.

OSI Open Systems Interconnection
Modèle permettant une standardisation des différents éléments concernant les réseaux.

PDA Personal Digital Assistant
Ce sont par exemple les Palm Pilot.

PFR Portable Font Resource
Format proposé par Netscape pour des polices de caractères indépendantes des différentes plateformes.

PHP PHP Hypertext Processor
Langage de programmation server-side orienté serveurs web.

PNG Portable Network Graphics
Un format d'image utilisé dans le World Wide Web.

QoS Quality of Service
Représente la qualité de service d'un réseau.

RDBMS Relational DataBase Management System
Les systèmes de gestion de bases de données relationnelles.

RPC Remote Procedure Calls
Protocole permettant l'appels de procédures à distance.

SGML Standard Generic Markup Language
Format générique de documents utilisant des balises.

SQL Stuctured Query Language
Langage d'intérrogation de base de données standardisé.

SMTP Simple Mail Transfer Protocol
Protocole basé sur TCP/IP et permettant l'échange de messages électroniques.

SNMP Simple Network Management Protocol
Protocole permettant de fournir des informations concernant la gestion du réseau, et qui n'est pas limité au TCP/IP.

SSL Secure Sockets Layer
Protocole basé sur le cryptage PK pour une utilisation avec un réseau TCP/IP.

TCP Transmission Control Protocol
Il s'agit d'un des protocoles utilisés par le réseau Internet.

TSL Transport Security Layer
Ce protocole , en cours de standardisation, intègre les spécificités du SSL.

TTP Trusted Third Party
Tierce partie de confiance aussi appelée CA.

UDP User Datagram Protocol
Protocole en mode non connecté pouvant être utilisé avec le protocole IP.

URL Universal Ressource Locator
Service permettant de définir des adresses pour l'ensemble des ressources d'un réseau TCP/IP.

WAN Wide Area Network
Réseaux d'ordinateurs mondiaux, comme Internet par exemple.

WYSIWYG What You See It's What You Get
Dénomination permettant d'indiquer que se que vous voyez à l'écran par l'intermédiaire d'un programme correspond au résultat final qui sera obtenu.

WYSIWYM What You See It's What You Mean
Dénomination permettant d'indiquer que se que vous voyez à l'écran par l'intermédiaire d'un programme permet d'identifier la structure et les éléments qui seront obtenus au résultat final.

WWW World Wide Web
L'ensemble du monde Internet.

W3C World Wide Web Consortium
Association modiale travaillant sur plusieurs standards comme HTML, CSS ou encore XML.

XBM X BitMaps
Format d'images du système X-Windows.

XHTML eXtended HyperText Markup Language
Spécification du langage HTML respectant les conventions des applications XML.

XPM X PixelMaps
Format d'images du système X-Windows.

XML eXtensible Markup Language
Nouvelle spécification permettant de structurer l'information sur Internet.

XSL eXtensible Style Language
Langage permettant d'intérroger un document XML et d'afficher les attributs des balises.

XSLT eXtensible Style Language Transformation
Langage permettant de transformer un fichier d'une structure XML donnée vers une autre.

XSLFO eXtensible Style Language Formatting Object
Langage permettant de décrire le layout d'un document XML.

YP Yellow Pages
Protocole permettant de distribuer des informations sur le nom des machines en interne.

Bibliographie

[1] Leon Atkinson. *Core PHP Programming*. Prentice Hall PTR, 2 edition, 2000.

[2] Hakon Wium Lie & Bert Bos. *Cascading Style Sheets : Designing for the Web*. Addison-Wesley, 1996.

[3] Jerry Bradenbaugh. *JavaScript Application Coobook*. O'Reilly, September 1999.

[4] Douglas E. Comer. *Réseaux et Internet*. Campus Press, 2000.

[5] Jon Meyer & Troy Downing. *Java Virtual Machine*. O'Reilly, March 1997.

[6] David Flanagan. *JavaScript : The definitive Guide*. O'Reilly, 3 edition, June 1998.

[7] Mark Grand. *Java Language Reference*. O'Reilly, 2 edition, July 1997.

[8] Shishir Gundavaram. *CGI Programming on the World Wide Web*. O'Reilly, March 1996.

[9] Elliotte Rusty Harold. *Java I/O*. O'Reilly, March 1999.

[10] Elliotte Rusty Harold. *XML Bible*. IDG Books Woldwide, Inc., 1999.

[11] Elliotte Rusty Harold. *Java Network Programming*. O'Reilly, 2 edition, August 2000.

[12] S. Schumann C. Scollo & D. Veliath J. Castagnetto, H. Rawat. *PHP Professionnel*. Eyrolles, 2000.

[13] Ben Laurie & Peter Laurie. *Apache : Installation et mise en oeuvre*. O'Reilly, 2 edition, September 1999.

[14] Netscape, developer.netscape.com. *Core JavaScript Guide*, 1998.

[15] Netscape, developer.netscape.com. *JavaScript Client-Side Guide*, 1999.

[16] Netscape, developer.netscape.com. *JavaScript Client-Side Reference*, 1999.

[17] Netscape, developer.netscape.com. *Server-Side JavaScript Guide*, 1999.

[18] Scott Oaks. *Sécurité en Java*. O'Reilly, 1999.

[19] Patrick Niemeyer & Joshua Peck. *Java par la pratique*. O'Reilly, 1997.

[20] Thomas A. Powell. *HTML : The Complete Reference*. Osborne/McGraw-Hill, 2 edition, 1999.

[21] George Reese. *Database Programming with JDBC and Java*. O'Reilly, 2 edition, August 2000.

[22] Eric Rescorla. *SSL and TLS – Designing and Building Secure Systems*. Addison Wesley, 2000.

[23] Simon Garfunkel & Gene Spafford. *Pratical UNIX & Internet Security*. O'Reilly, 2 edition, apr 1996.

[24] Simon Garfunkel & Gene Spafford. *Web Security & Commerce*. O'Reilly, jun 1997.

[25] George Reese & Randy Jay Yarger Tim King. *MySQL and mSQL*. O'Reilly, July 1999.

[26] Randal L. Schwartz & Larry Wall Tom Christiansen. *Programming Perl*. O'Reilly, 2 edition, September 1996.

[27] Tom Christiansen & Nathan Torkington. *Perl en action*. O'Reilly, 1999.

[28] W3C, www.w3.org. *Cascading Style Sheets, Level 2*, May 1998.

[29] W3C, www.w3.org. *HTML 4.01 Specification*, December 1999.

[30] W3C, www.w3.org. *Extensible Stylesheet Language (XSL) Version 1*, March 2000.

[31] Scott Oaks & Henry Wong. *Java Threads*. O'Reilly, 2 edition, January 1999.

[32] John Zukowski. *Java AWT Reference*. O'Reilly, April 1997.

[33] D. Brent Chapman & Elizabeth D. Zwicky. *Firewalls : La sécurité sur l'Internet*. O'Reilly, 1996.