

Architecture des applications internet - SR03

PRINTEMPS 2015

Table des matières

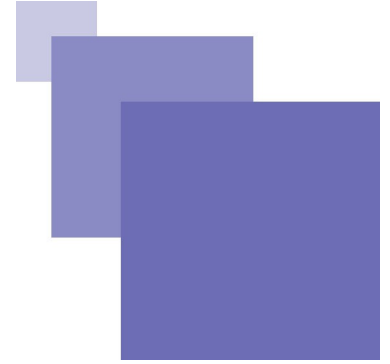
Objectifs	7
Introduction	9
I - Avant-propos	11
II - Introduction aux architectures des réseaux et techniques de communications	13
A. Introduction à l'internet.....	13
1. L'interconnexion.....	13
2. Protocoles et commutation.....	14
3. Organisation en couches.....	14
4. Description des couches.....	15
B. Les sockets.....	16
1. Introduction aux sockets.....	17
2. Un exemple d'interaction TCP Client/Server.....	19
C. Le modèle client-serveur et la communication entre applications.....	23
1. Le modèle Client-Serveur.....	23
2. Communication entre applications réparties.....	24
III - Le web	29
A. Web et ses composants.....	29
1. Les trois grandes briques du web.....	29
2. Le processus de téléchargement d'une page web.....	30
3. Description des trois briques du web.....	31
B. HTTP.....	33
1. Principes de fonctionnement.....	33
2. L'échange de messages.....	34
3. La version HTTP 1.1.....	39
4. Les cookies et les sessions.....	39
C. HTML.....	40
1. Principes de base.....	40
2. Un exemple.....	42
3. HTML 5.....	43
D. Les feuilles de style.....	45
1. Placement des feuilles de style.....	45
2. Les classes d'éléments.....	47
3. Les pseudo-classes d'ancrage.....	47
4. La notion de cascade.....	48
5. CSS3.....	49

IV - Programmation web côté client	51
A. Introduction au JavaScript.....	51
1. Les bases de la programmation en JAVASCRIPT.....	52
2. JavaScript : un langage orienté objet.....	53
B. JavaScript et le WEB.....	55
1. Intégration d'un script dans une page HTML.....	56
2. JavaScript et les cookies.....	56
C. JavaScript : un langage orienté événements.....	58
1. Qu'est ce que le DOM ?.....	58
2. Un langage orienté événement.....	61
3. HTML Dynamique (DHTML).....	66
D. Ajax.....	67
1. Motivation.....	67
2. Mode de fonctionnement.....	68
V - Programmation web côté serveur	71
A. Les scripts CGI.....	71
1. Principes de fonctionnement.....	71
2. Avantages et inconvénients des CGI.....	71
B. Introduction au PHP.....	72
1. Packages et Framework PHP.....	72
2. Principes de fonctionnement.....	73
3. Intégration du code PHP dans les pages html.....	74
4. Syntaxe du langage PHP.....	74
C. PHP, les fonctionnalités web.....	78
1. Traitement des formulaires.....	78
2. Chargement de fichier.....	79
3. Envoi de mails.....	80
4. Modification des entêtes HTTP.....	80
5. Gestion des cookies.....	80
6. Gestion des sessions.....	81
7. Les cookies versus les sessions.....	83
D. PHP un langage objet.....	84
1. POO pour PHP.....	84
2. PHP et l'interfaçage avec les bases de données.....	91
3. Fichier de configuration.....	94
VI - Programmation Java, les applets, les servlets et JSP	95
A. Introduction au Java et aux applets.....	95
1. Aspects généraux du langage java.....	95
2. Environnement.....	96
3. Les packages.....	97
4. Événement en Java.....	98
5. Les applets.....	99
B. Les servlets.....	104
1. Serveur d'application.....	105
2. Le descripteur de déploiement.....	106
3. Invocation des servlets.....	107
4. Création des servlets.....	107
5. Cycle de vie d'une servlet.....	109
6. Récupération des paramètres d'un formulaire.....	111
7. L'objet ServletContext.....	112
8. Gestion des cookies.....	113
9. Gestion des sessions.....	114
10. Redirection au niveau du serveur.....	114
11. Les limites des servlets.....	114

Objectifs

Les enjeux d'Internet sont à la fois technologiques, économiques, humaines et sociales. De nos jours, on se sert essentiellement de l'Internet via le WEB. Le WEB est devenu un immense service d'information mondial où il est indispensable de se faire connaître pour tout acteur de la vie économique et sociale. Grâce au WEB le développement du multimédia, des télécommunications et de l'informatique a connu une véritable révolution, comparable à celui des années 80 avec la naissance de la micro-informatique. Si le temps des pages personnelles est désormais révolu, les applications distribuées de grande échelle sont possibles et l'Internet procure un environnement propice. Internet a conduit aussi à des changements profonds du marché des terminaux ; on remarque en particulier l'augmentation des ventes des tablettes versus PC, des mobiles vers fixes, etc. L'objectif ultime du cours est de donner les éléments clés pour la mise en œuvre d'applications distribuées en grande échelle sur Internet. Visant cet objectif le cours est bâti en trois grandes parties. La première rappelle et introduit certaines notions de bases de l'architecture des réseaux, des protocoles de transport et des techniques de communication entre processus dans des machines distantes. La deuxième partie est consacrée à l'introduction de certaines briques de l'Internet, telles que le WEB, le protocole HTTP, le langage HTML, les feuilles de style CSS, l'API DOM etc. Cette partie reprend aussi l'architecture client-serveur appliquée à l'Internet et décrit certains langages de programmation situés coté client (Javascript / Ajax) et coté serveur (PHP, Servlets et JSP). La troisième partie est destinée à faire le lien entre les notions vues lors des deux premières parties pour la mise en œuvre d'architectures d'applications distribuées. On s'intéresse alors de près aux architectures réparties et à des notions telles que Java RMI, CORBA, XML-RPC et Web Service.

Introduction



L'UV SR03 présente les aspects architecturaux (infrastructures matérielles et logicielles) mis en oeuvre dans les systèmes d'informations de type Internet via des techniques de communication (sockets, RPC) aux objets distribués et aux serveurs d'application. Dans ce cadre, ce cours s'intéressera à la couche « application » et à la programmation pour Internet. Ainsi, après une présentation succincte de HTTP, WEB, HTML, CSS et DOM, on donnera les principes de fonctionnement de Javascript et de la technologie Ajax. Ensuite, des éléments de la programmation web coté serveur avec notamment PHP et Java (Servlets et JSP) feront l'objet de brèves présentations lors des cours et des TD. Enfin, nous continuons avec les architectures réparties, en mettant l'accent sur Java RMI. Ensuite nous verrons les concepts de CORBA, XML-RPC et Web Service. Tous ces cours seront illustrés lors des séances de Td machine. L'UV se termine par une introduction au MPI, qui est une API pour la communication entre les machines parallèles. Des interventions par des experts sont également prévues.

Avant-propos



Remerciements

Une aide précieuse pour la prise en main de ce cours a été l'ensemble des documents laissés par mes prédécesseurs, Monsieur Michel Vayssade et Monsieur Dritan Nace. Je tiens à leur rendre hommage pour tout le travail énorme qu'ils ont accompli pour la mise en place de cet enseignement. Je tiens également à remercier le corps des enseignants intervenant dans cette UV pour leur professionnalisme et leur enthousiasme.

Remarque

Les notes cours données ici comprennent la majorité des sujets traités. Le contenu peut évoluer au fur et à mesure pour intégrer surtout des interventions de spécialistes invités au SR03.

Introduction aux architectures des réseaux et techniques de communications



Introduction à l'internet	13
Les sockets	16
Le modèle client-serveur et la communication entre applications	23

Avant de décrire l'architecture de l'Internet, voici quelques éléments clés pour sa compréhension.

A. Introduction à l'internet

1. L'interconnexion

L'élément clé à l'origine de l'Internet a été le besoin d'assurer l'interconnexion de réseaux de nature diverse. Les différences entre ces réseaux ne devaient pas être visible à l'utilisateur final. Pour cela, il fallait assurer un niveau d'abstraction à chaque niveau de fonctionnalité (couches de protocoles) capable d'encapsuler les fonctionnalités de niveau inférieur. L'objectif était d'affranchir l'utilisateur des détails relatifs aux couches inférieures et finalement au réseau lui-même (la couche physique). On disposait alors de deux approches différentes. L'une était de faire appel à des programmes d'application chargés de gérer l'hétérogénéité du système (on parle alors d'interconnexion au niveau application). La deuxième consistait à masquer les détails de fonctionnement par le système d'exploitation (on parle alors d'interconnexion au niveau réseau). La première solution présentait plusieurs inconvénients :

- si les applications interfacciaient elles-mêmes les réseaux (aspects physiques), elles seraient victimes de toute modification de ces dernières,
- plusieurs applications différentes sur une même machine dupliqueraient l'accès au réseau,

- si le réseau devenait important, il serait impossible de mettre en œuvre toutes les applications nécessaires à l'interconnexion sur tous les nœuds du réseau.

Une alternative à cette solution était alors l'interconnexion au niveau réseau qui conduisait à la mise en œuvre de l'interconnexion au niveau des protocoles gérant la couche réseau de ces systèmes. Les avantages sont les suivants : les données sont routées par les nœuds intermédiaires sans que ces nœuds aient la moindre connaissance des applications responsables des ces données. De plus, le système est flexible puisqu'on peut facilement introduire de nouvelles interfaces physiques en adaptant la couche réseau alors que les applications demeurent inchangées, donc les protocoles peuvent être modifiés sans que les applications soient affectées.

2. Protocoles et commutation



Définition : Protocoles

Les protocoles définissent le format, l'ordre des messages envoyés et reçus entre les entités du réseau, ainsi que les actions à prendre sur les messages (transmission, réception...).

Commutation par paquet

Contrairement à la technologie de commutation de circuit (qui est propice aux communications téléphoniques) à la base du fonctionnement de l'Internet se trouve la technologie de commutation par paquet (mode datagramme). Ce mode est comparable avec la distribution des courriers par la poste. En fonction de la destination, le paquet est envoyé au voisin le mieux placé pour atteindre le destinataire. Ainsi, dans l'Internet, chaque paquet est individuellement routé sur la base des informations dans le table de routage des routeurs (des paquets différents peuvent suivre des chemins différents).



Rappel

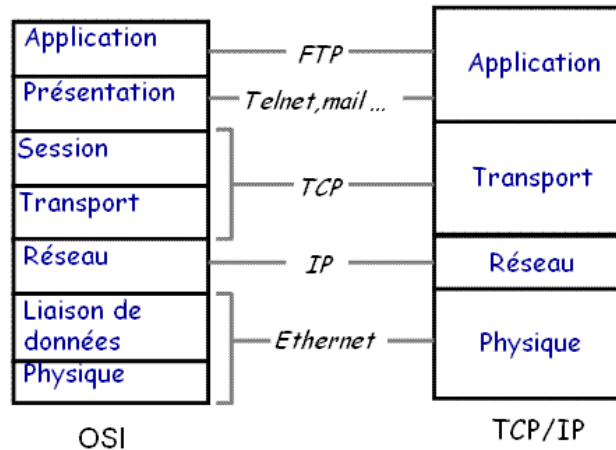
La commutation de circuit nécessite un chemin (circuit) dédié pour la durée de la transmission. Il a ainsi besoin de bande passante mais procure des délais d'acheminement faible. Le réseau téléphonique fonctionne sur ce modèle.

3. Organisation en couches

Internet est organisé en couches. Chaque couche implémente un service et se fait servir par les couches au-dessous. Trois éléments sont alors concernés : les réseaux, les ordinateurs et les applications. A ces derniers on peut mettre en face trois couches : le réseau d'accès, la couche transport, la couche application. Les couches communiquent via les protocoles. L'Internet suit le modèle TCP/IP qui s'inspire de la structuration en couches du modèle OSI, Figure 1, (voir *le site dédié*¹ pour plus de détails sur OSI).

1 - <http://www.frameip.com/osi>





Architecture OSI versus architecture TCP/IP

Mise en œuvre

Pour résumer, TCP/IP est destiné à l'interconnexion de divers réseaux sur une base planétaire : Ethernet, Token Ring, X25, Frame Relay, FDDI, etc. La technologie est constituée par des protocoles (suite TCP/IP) qui offrent les services de base du transfert des données.

- transport de datagrammes : service élémentaire de la commutation de paquets.
- transport de messages sécurisés : service orienté connexion permettant d'acheminer des données en garantissant leur intégrité.

Ces services de base sont indépendants du support de transmission et adaptables à toute sorte de média depuis les réseaux locaux jusqu'aux réseaux longue distance. L'interconnexion est universelle tandis que les machines ont une adresse unique sur l'Internet. Deux machines reliées au réseau, communiquent grâce aux autres nœuds du réseau qui routent de manière coopérative sur la base de l'adresse destinataire. Dans le cadre du transport sécurisé, les acquittements sont effectués entre les systèmes finaux (source et destinataire) plutôt que continuellement entre chaque nœud relayant les messages.

4. Description des couches

La couche physique

Cette couche regroupe la couche physique et la couche liaison de données du modèle OSI. La seule contrainte de cette couche, c'est de permettre à un hôte d'envoyer des paquets IP sur le réseau. L'implémentation de cette couche est laissée libre. De manière plus concrète, cette implémentation dépend fortement de la technologie utilisée sur le réseau local. Par exemple, beaucoup de réseaux locaux utilisent Ethernet.

La couche réseau (appelée aussi internet)

Cette couche est la clé de voûte de l'architecture. Elle réalise l'interconnexion des réseaux (hétérogènes) distants sans connexion. Son rôle est de permettre l'injection de paquets dans n'importe quel réseau et leur acheminement indépendamment les uns des autres jusqu'à destination. Cette couche ne garantit pas l'ordre d'arrivée des paquets, cette tâche incombe aux couches supérieures.

Du fait du rôle imminent de la couche réseau dans l'acheminement des paquets, le point critique de cette couche est le routage. La couche réseau possède une implémentation officielle : le protocole IP (Internet Protocol).

La couche transport

Son rôle est de permettre à des entités paires de soutenir une conversation. Officiellement, cette couche n'a que deux implémentations, le protocole TCP (Transmission Control Protocol) et le protocole UDP (User Datagram Protocol).

TCP est un protocole fiable, orienté connexion, qui permet l'acheminement sans erreur de paquets de bout en bout. Son rôle est de fragmenter le message à transmettre de manière à pouvoir le faire passer sur la couche réseau. A l'inverse, sur la machine destination, TCP replace dans l'ordre les fragments transmis sur la couche réseau pour reconstruire le message initial. TCP s'occupe également du contrôle de flux de la connexion : il assure le contrôle de congestion grâce à l'algorithme AIMD (Additive Increase - Multiplicative Decrease). Cet algorithme adapte le débit de la source en fonction de l'état (niveau de congestion) du réseau.

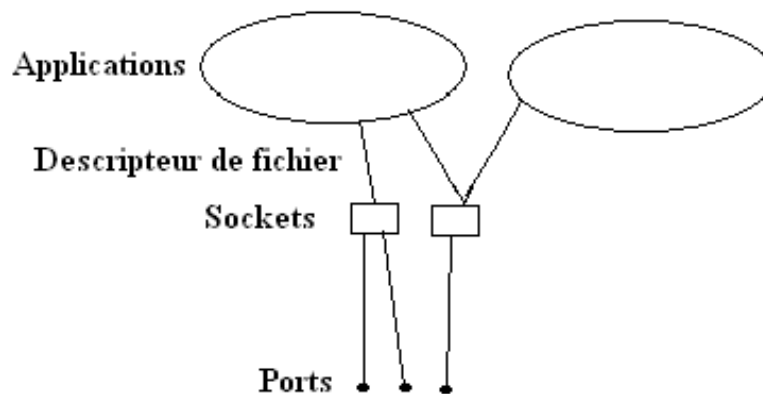
UDP est en revanche un protocole plus simple que TCP. Il est non fiable et sans connexion. Son utilisation présuppose que l'on n'a pas besoin ni de contrôle de flux, ni de la conservation de l'ordre de remise des paquets. Par exemple, on l'utilise lorsque la couche application se charge de la remise en ordre des messages. De manière plus générale, UDP est utilisé pour des applications temps-réels où le temps de remise des paquets est prédominant. Ainsi, UDP est utilisé pour transmettre la voix sur IP. Dans ce cas l'envoi en temps réel est primordiale et la retransmission des paquets perdus serait inutile. Même si UDP n'assure pas une transmission fiable des données, il procède à un contrôle simple : il utilise le champ de contrôle d'erreur Checksum pour détecter la corruption du contenu du paquet.

La couche application

C'est la couche immédiatement supérieure à la couche transport. Elle contient tous les protocoles de haut niveau, comme par exemple Telnet, TFTP (trivial File Transfer Protocol), SMTP (Simple Mail Transfer Protocol), HTTP (HyperText Transfer Protocol). Le point important pour cette couche est le choix du protocole de transport à utiliser. Par exemple, TFTP (surtout utilisé dans les réseaux locaux) utilisera UDP, car on part du principe que les liaisons physiques sont suffisamment fiables et les temps de transmission suffisamment courts pour qu'il n'y ait pas d'inversion de paquets à l'arrivée. Ce choix rend TFTP plus rapide que le protocole FTP qui utilise TCP. A l'inverse, SMTP utilise TCP, car pour la remise du courrier électronique, on veut que tous les messages parviennent intégralement et sans erreurs.

B. Les sockets

Nous venons de donner une description succincte des concepts fondamentaux de l'Internet et des protocoles TCP/IP sans spécifier la nature de l'interface placée entre les protocoles et les applications. Nous verrons dans ce chapitre l'interface de programmation socket qui remplit ce rôle.



Sockets, protocoles et ports

1. Introduction aux sockets

a) Rappels sur les entrées sorties UNIX

Les commandes d'entrées sorties (E/S) UNIX suivent la séquence ouverture-lecture-écriture-fermeture. P.ex. l'appel d'ouverture (open) renvoie un descripteur de fichier, et seulement après qu'on pourra exécuter des actions de lecture ou d'écriture. Les instructions pour les E/S réseau suivent une logique similaire mais sont plus élaborées que les E/S habituels. Cela nous mène aux sockets : une socket est une interface de communication introduite par les systèmes Unix pour la communication réseau. Elle est destinée à la communication entre les processus afin de leur permettre de communiquer aussi bien sur une même machine qu'à travers un réseau.

b) La notion des sockets

Comme évoqué ci-dessus, les opérations d'entrées-sorties réseau s'appuient sur une abstraction connue sous le nom de socket. Elle permet aux applications de recevoir/envoyer des données de la même manière que de lire/écrire sur un fichier. Un socket est donc un mécanisme qui procure un accès à l'extrémité d'un canal de communication bidirectionnel (full duplex). Les sockets sont créés par le système d'exploitation sur demande de programmes applicatifs. On identifie les sockets à l'aide d'un petit nombre entier (ou descripteurs de fichiers), qui est utilisé par les programmes pour les référencer.

c) Types de sockets

Le type du socket définit les propriétés de la communication. Les deux types principaux sont :

- le type datagramme, qui permet l'échange de messages complets et structurés, sans négociation préalable (communication sans connexion) ;
- le type connecté (stream), qui permet l'échange de flux de données sur un circuit virtuel préalablement établi.

Les deux types principaux de sockets correspondent donc aux protocoles sans connexion (i.e. UDP) et aux protocoles orientés connexion (i.e. TCP).

Vu "de l'intérieur" du programme applicatif (ou processus utilisateur), un socket est un descripteur de fichiers. On peut ainsi rediriger vers un socket les E/S standard d'un programme développé dans un cadre local.

d) Ouverture et fermeture d'un socket

Un socket est créé par la primitive `sd = socket(pf, type, protocole)`. L'argument `pf` spécifie la famille de protocoles à utiliser par le socket, c'est-à-dire la manière d'interpréter les adresses, p.ex. IPv4 (PF_INET), IPv6 (PF_INET6), etc. L'argument `type` spécifie le type de communication souhaitée (SOCK_STREAM pour une transmission fiable, SOCK_DGRAM pour une transmission sans connexion tandis que SOCK_RAW permet aux utilisateurs d'accéder à des protocoles de communication différents en même temps). Puisque la famille peut disposer plusieurs protocoles, alors le troisième argument permet de spécifier le protocole à utiliser. Dans le cas où il n'y a qu'un seul protocole pour un service donné, cet argument peut être 0.

La valeur de retour donne le descripteur sur lequel on va faire les opérations de lecture et d'écriture.

Pour la fermeture, on utilise la fonction `close (sd)`, où `sd` indique le socket à fermer.

e) Spécification d'adresse (locale et distante)

Notons que les sockets sont créés sans aucune association à une adresse locale ou distante. Du point de vue des protocoles TCP/IP, les nouveaux sockets naissent sans une adresse IP locale ou distante et sans numéro de port de protocole. Les programmes client généralement délèguent au protocole le choix de l'adresse IP ou du numéro de port. Une fois le socket créé, les serveurs utilisent la fonction `bind()` pour lui associer une adresse locale. Notons aussi qu'un socket créé est dans un état « sans connexion », donc il ne possède aucune destination. Pour lui associer une destination on utilise la fonction `connect()`.

f) Échange de données

L'application qui a créé un socket peut ensuite l'utiliser pour transmettre des données. Ceci peut se faire par cinq fonctions différents : `send`, `sendto`, `sendmsg`, `write` et `writetv`. Les trois premières ne fonctionnent qu'en présence de sockets connectés, tandis que les deux dernières permettent l'envoi de messages au travers de sockets non connectés.

La réception des données sur un socket se fait par l'intermédiaire de cinq fonctions : `read`, `readv`, `recv`, `recvfrom` et `recvmsg`.

g) Configuration d'options

D'autres fonctions sont proposées pour la configuration des options des sockets. Il est ainsi possible de préparer les sockets aux requêtes entrantes et spécifier la longueur de file d'attente (`qlength`) via la fonction `listen(fd, qlength)`. L'acceptation d'une connexion entrante par un serveur se réalise par la fonction `accept()`. Ci-dessous nous donnons le schéma basique de communication via les sockets à la fois dans le cas du protocole UDP et TCP.

h) Schéma de communication de serveur

Schéma de communication de serveur et de client TCP

Un serveur TCP et un client TCP

1. le serveur se met en attente sur un port TCP,
2. le client se connecte au serveur et se met en attente
3. le serveur accepte la connexion sur un nouveau socket,
4. le client boucle sur envoi de messages,
5. le serveur boucle sur la lecture du socket : pour récupérer des messages complets, il doit implanter une boucle (`while(attendu)`) pour chaque

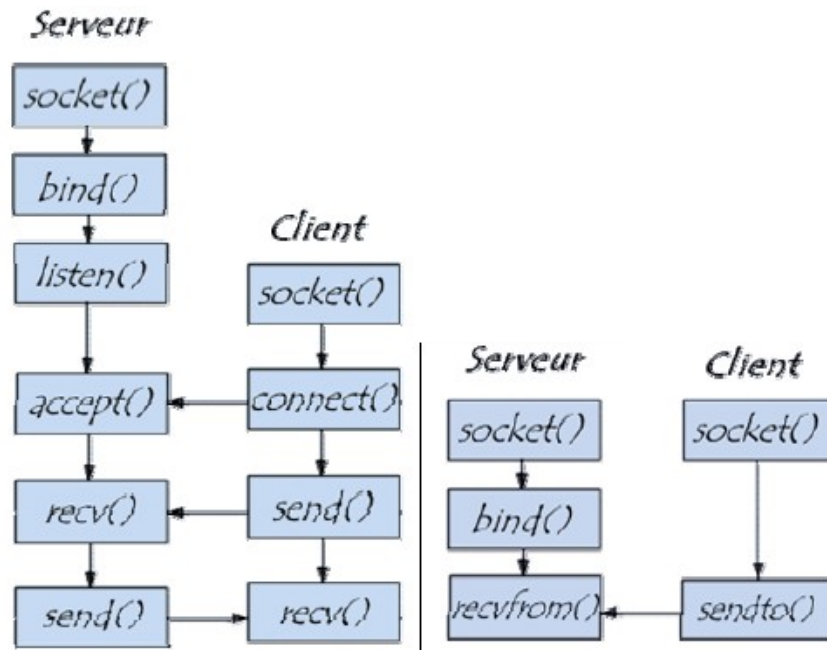
message.

Schéma de communication de serveur et de client UDP

Un serveur UDP et un client UDP

1. le serveur se met en attente sur un port UDP,
2. le client crée son socket local,
3. le client envoie un message sur le host et le port UDP du serveur,
4. à réception du message, le serveur récupère l'adresse (host, port) de l'émetteur et traite le message, puis envoie une réponse.

Ci-dessous les deux schémas illustrant les échanges client-serveur pour les deux cas évoqués ci-dessus.



Communication serveur client via les sockets : TCP versus UDP

2. Un exemple d'interaction TCP Client/Server

Dans la suite nous apportons un exemple d'écriture de socket^{Donahoo, Calvert} permettant au client et au serveur d'implémenter le protocole « echo ». Ce protocole est très simple, il fonctionne comme suit : le client se connecte au serveur et envoie ses données. Le serveur renvoie au client tout simplement ce qu'il a reçu. Ci dessous, le client envoie une « chaine » via un argument d'une commande en ligne.

TCPEchoServer.c

```
#include <stdio.h> /* for printf() and fprintf() */
#include <sys/socket.h> /* for socket(), bind(), and
connect() */
#include <arpa/inet.h> /* for sockaddr_in and inet_ntoa()
*/
#include <stdlib.h> /* for atoi() and exit() */
#include <string.h> /* for memset() */
#include <unistd.h> /* for close() */
#define MAXPENDING 5 /* Maximum outstanding connection
requests */
void DieWithError(char *errorMessage); /* Error handling
```

```

function */
void HandleTCPClient(int clntSocket); /* TCP client
handling function */
int main(int argc, char *argv[])
{
    int servSock; /* Socket descriptor for server */
    int clntSock; /* Socket descriptor for client */
    struct sockaddr_in echoServAddr; /* Local address */
    struct sockaddr_in echoClntAddr; /* Client address */
    unsigned short echoServPort; /* Server port */
    unsigned int clntLen; /* Length of client address data
structure */
    if (argc != 2) /* Test for correct number of arguments */
    {
        fprintf(stderr, "Usage: %s <Server Port>\n", argv[0]);
        exit(1);
    }
    echoServPort = atoi(argv[1]); /* First arg: local port */
    /* Create socket for incoming connections */
    if ((servSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP))
        < 0)
        DieWithError("socket() failed");
    /* Construct local address structure */
    memset(&echoServAddr, 0, sizeof(echoServAddr)); /* Zero out
structure */
    echoServAddr.sin_family = AF_INET; /* Internet address
family */
    echoServAddr.sin_addr.s_addr = htonl(INADDR_ANY); /* Any
incoming interface */
    echoServAddr.sin_port = htons(echoServPort); /* Local port
*/
    /* Bind to the local address */
    if (bind(servSock, (struct sockaddr *) &echoServAddr,
sizeof(echoServAddr)) < 0)
        DieWithError("bind() failed");
    /* Mark the socket so it will listen for incoming
connections */
    if (listen(servSock, MAXPENDING) < 0)
        DieWithError("listen() failed");
    for (;;) /* Run forever */
    {
        /* Set the size of the in-out parameter */
        clntLen = sizeof(echoClntAddr);
        /* Wait for a client to connect */
        if ((clntSock = accept(servSock, (struct sockaddr *)
&echoClntAddr, &clntLen)) < 0)
            DieWithError("accept() failed");
        /* clntSock is connected to a client! */
        printf("Handling client %s\n",
inet_ntoa(echoClntAddr.sin_addr));
        HandleTCPClient(clntSock);
    }
    /* NOT REACHED */
}

```

TCPEchoClient.c

```
#include <stdio.h> /* for printf() and fprintf() */
```

```

#include <sys/socket.h> /* for socket(), connect(), send(),
and recv() */
#include <arpa/inet.h> /* for sockaddr_in and inet_addr()
*/
#include <stdlib.h> /* for atoi() and exit() */
#include <string.h> /* for memset() */
#include <unistd.h> /* for close() */
#define RCVBUFSIZE 32 /* Size of receive buffer */
void DieWithError(char *errorMessage); /* Error handling
function */
int main(int argc, char *argv[])
{
    int sock; /* Socket descriptor */
    struct sockaddr_in echoServAddr; /* Echo server address */
    unsigned short echoServPort; /* Echo server port */
    char *servIP; /* Server IP address (dotted quad) */
    char *echoString; /* String to send to echo server */
    char echoBuffer[RCVBUFSIZE]; /* Buffer for echo string */
    unsigned int echoStringLen; /* Length of string to echo */
    int bytesRcvd, totalBytesRcvd; /* Bytes read in single
recv() and total bytes read */
    if ((argc < 3) || (argc > 4)) /* Test for correct number of
arguments */
    {
        fprintf(stderr, "Usage: %s <Server IP> <Echo Word> [<Echo
Port>]\n", argv[0]);
        exit(1);
    }
    servIP = argv[1]; /* First arg: server IP address (dotted
quad) */
    echoString = argv[2]; /* Second arg: string to echo */
    if (argc == 4)
        echoServPort = atoi(argv[3]); /* Use given port, if any */
    else
        echoServPort = 7; /* 7 is the well-known port for the echo
service */
    /* Create a reliable, stream socket using TCP */
    if ((sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
        DieWithError("socket() failed");
    /* Construct the server address structure */
    memset(&echoServAddr, 0, sizeof(echoServAddr)); /* Zero out
structure */
    echoServAddr.sin_family = AF_INET; /* Internet address
family */
    echoServAddr.sin_addr.s_addr = inet_addr(servIP); /* Server
IP address */
    echoServAddr.sin_port = htons(echoServPort); /* Server port
*/
    /* Establish the connection to the echo server */
    if (connect(sock, (struct sockaddr *) &echoServAddr,
sizeof(echoServAddr)) < 0)
        DieWithError("connect() failed");
    echoStringLen = strlen(echoString); /* Determine input
length */
    /* Send the string to the server */
    if (send(sock, echoString, echoStringLen, 0) !=
echoStringLen)
        DieWithError("send() sent a different number of bytes than
expected");
}

```



```

/* Receive the same string back from the server */
totalBytesRcvd = 0;
printf("Received: "); /* Setup to print the echoed string */
while (totalBytesRcvd < echoStringLen)
{
    /* Receive up to the buffer size (minus 1 to leave space
    for a null terminator) bytes from the sender */
    if ((bytesRcvd = recv(sock, echoBuffer, RCVBUFSIZE - 1, 0))
    <= 0)
        DieWithError("recv() failed or connection closed
        prematurely");
    totalBytesRcvd += bytesRcvd; /* Keep tally of total bytes */
    echoBuffer[bytesRcvd] = '\0'; /* Terminate the string! */
    printf("%s", echoBuffer); /* Print the echo buffer */
}
printf("\n"); /* Print a final linefeed */
close(sock);
exit(0);
}

```

HandleTCPClient.c

```

#include <stdio.h> /* for printf() and fprintf() */
#include <sys/socket.h> /* for recv() and send() */
#include <unistd.h> /* for close() */
#define RCVBUFSIZE 32 /* Size of receive buffer */
void DieWithError(char *errorMessage); /* Error handling
function */
void HandleTCPClient(int clntSocket)
{
    char echoBuffer[RCVBUFSIZE]; /* Buffer for echo string */
    int recvMsgSize; /* Size of received message */
    /* Receive message from client */
    if ((recvMsgSize = recv(clntSocket, echoBuffer, RCVBUFSIZE,
    0)) < 0)
        DieWithError("recv() failed");
    /* Send received string and receive again until end of
    transmission */
    while (recvMsgSize > 0) /* zero indicates end of
    transmission */
    {
        /* Echo message back to client */
        if (send(clntSocket, echoBuffer, recvMsgSize, 0) !=
        recvMsgSize)
            DieWithError("send() failed");
        /* See if there is more data to receive */
        if ((recvMsgSize = recv(clntSocket, echoBuffer, RCVBUFSIZE,
        0)) < 0)
            DieWithError("recv() failed");
    }
    close(clntSocket); /* Close client socket */
}

```

C. Le modèle client-serveur et la communication entre applications

Les échanges entre les applications peuvent se faire dans un contexte complètement distribué (ou réparti) et asynchrone, ou dans un mode plus conventionnel et synchrone comme le mode client-serveur. Le modèle client-serveur peut être vu selon deux angles, « environnement de développement » et « système ». Le premier concerne les outils de construction d'applications, en particulier **l'appel de procédure à distance** et le deuxième concerne plus la mise en œuvre de l'appel de procédure à distance dans un environnement distribué hétérogène. Ces services systèmes sont généralement regroupés dans une couche de logiciel interposée entre l'application et le système d'exploitation, on parle alors de « middleware ».

1. Le modèle Client-Serveur

Rappelons brièvement l'architecture client-serveur (CS). Il est composé du (processus) client, qui demande l'exécution d'un service, et un processus serveur, qui réalise ce service.

Évolution des architectures

Les premières architectures client-serveur sont parues dans les années 70 et elles étaient centralisées. Les architectures décentralisées sont ensuite parues dans les années 80 : on retrouvera essentiellement les architecture de type Client-serveur (2 tiers). Dans les années 90 ce sont les architectures distribuées et les architectures CS troisième génération, dite architectures à 3 niveaux (3 tiers).

Aujourd'hui le besoin en architecture a aussi évolué pour faire face au :

- échange de données entre des applications hétérogènes utilisant des données de formats différents ;
- répartition des données sur des sites géographique distants ;
- distribution des traitements effectués sur les données réparties ;
- interopérabilité des plate-formes de développement ;
- portabilité des applications (postes nomades).

Client serveur 1^{ère} génération

- Client lourd, traitement client

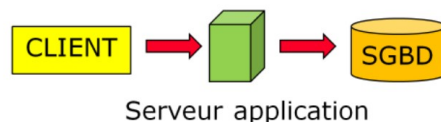


Client serveur 2^{ème} génération

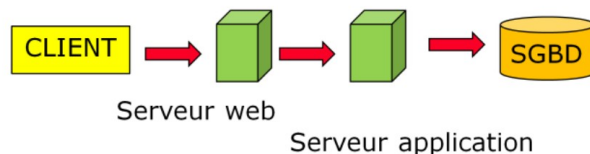
- Client lourd, évènementiel

-Accès aux ressources locales

-Accès mode dconnecté



Client Web
- Client léger



Évolution des architectures Client-Serveur

2. Communication entre applications réparties

Client et serveur sont localisés sur deux machines reliées par un réseau de communication. Néanmoins, plusieurs serveurs peuvent cohabiter sur la même machine et un serveur peut être aussi réparti sur plusieurs sites. Ce modèle a été à la base des premières applications réparties (transfert de fichiers, connexion à une machine distante, courrier électronique, etc.), réalisées chacune par un protocole applicatif spécifique. Ensuite, une construction commune, l'appel de procédure à distance, a été introduite pour fournir un outil général pour la programmation d'applications client-serveur.

Ce modèle fonctionne sur la base d'échange de messages (plutôt que par mémoire partagée). Le client, ne connaît rien d'un serveur sauf les services pour lesquels il connaît leur nom, les paramètres à fournir et les paramètres qui lui seront rendus après exécution du service. Le client est un consommateur de service et le serveur est le fournisseur de ces services. Le client (qui est une application) démarre l'échange qui est alors constitué de deux messages : le premier transmis par le client contient une requête pour le serveur en donnant le nom du service souhaité et les paramètres associés, le second est envoyé par le serveur et contient le résultat de l'exécution du service. La caractéristique la plus fréquente des connexions en mode "client-serveur" est d'être asymétrique : d'abord, un processus serveur se prépare et il se met en écoute sur un port de communication. Ce n'est encore qu'une "moitié" de la liaison. Ensuite, un processus client va envoyer à ce port serveur déjà existant une demande de connexion. La connexion est alors établie et les données peuvent circuler dans les deux sens.

L'échange client-serveur se réalise selon un format de messages (question-réponse) déterminé, c'est donc un protocole. On distingue les protocoles spécifiques à l'application et ceux à vocation générale. Remote Procedure Call (RPC) fait partie de ces dernières. Cette technologie regroupe des standards tels que CORBA, RMI, DCOM, .Net Remoting (Microsoft .net) et SOAP. L'idée de RPC est de faire réaliser une opération sur une machine distante. L'appel de procédure à distance est un mécanisme qui permet à un processus, implanté sur un site C, d'exécuter une procédure P sur un site distant S, avec un effet globalement identique à celui qui serait obtenu par l'exécution locale de P sur C. Il s'agit donc de faire un appel de procédure ou d'un sous-programme distant.



Remarque

Le rôle de la bibliothèque de RPC est de factoriser le code qui est commun à tous les appels de procédure à distance et en particulier le code nécessaire au traitement des erreurs.

a) Comment ça marche

La réalisation repose sur l'utilisation d'un processus s (ou serveur), qui exécute la procédure P sur le site S. Le processus client p reste bloqué pendant l'exécution de P et il est réactivé au retour de P. Voici comment une procédure P sur C est appelée à une machine distante S.

1. Sur S, un processus serveur se déclare en associant à une fonction P, un numéro de programme, un numéro de version et un numéro de fonction ;
2. Sur C, un client appelle la procédure P en donnant l'adresse de la machine S et le numéro de programme, numéro de version et numéro de fonction correspondant à la procédure P :
 - empilement de l'adresse de retour et des paramètres de l'appel sur la pile locale.
3. Sur C, recherche de la procédure: requête au serveur d'associations (portmap ou rpcbind) sur S.

4. Transfert depuis C vers S du nom de la fonction distante et des paramètres à la machine distante via le port donné par le processus portmap de S puis :
 - positionnement des paramètres dans des registres du processus serveur sur S
 - appel au noyau ou saut dans le code du processus distant devant réaliser cette opération.
5. Sur S, exécution de P par :
 - exécution du code distant
 - renvoi du résultat.
6. Transfert depuis S vers C du résultat via le réseau ;
7. Sur C, retour de la fonction :
 - dépilement des variables locales à la procédure et des paramètres d'appel,
 - écriture du résultat à l'adresse donnée par l'adresse de retour,
 - dépilement de cette adresse.

b) Description des actions

Sur le site client

- mettre les paramètres d'appel sous une forme permettant leur transport sur le réseau vers le site appelé, c'est la **fonction d'emballage** (marshalling) ;
- générer un **identificateur pour la requête en cours** et envoyer vers le site appelé un **message contenant l'identité de la procédure appelée et les paramètres d'appel**,
- mettre en attente le processus client, en attendant la réponse du site appelé,
- Lorsque la réponse arrive, le processus client est réveillé ; il poursuit son exécution dans le **talon** client en **extrayant du message de réponse les résultats**, s'il y en a, pour les transformer dans une forme compréhensible par le site local, c'est la **fonction de déballage** (unmarshalling),
- exécuter le retour de procédure, avec transmission des résultats. Tout se passe alors, pour le processus client, comme pour le retour d'un appel de procédure local.

Sur le site distant

- à partir du message reçu du client, **enregistrer l'identificateur de la requête**, déterminer la procédure appelée et déballer les paramètres d'appel ;
- **exécuter la procédure appelée en lui passant les paramètres** (il s'agit alors d'un appel de procédure local) ;
- au retour de cet appel, l'exécution se poursuit dans le talon serveur, pour préparer le message de retour vers le site appelant, en emballant les éventuels résultats ;
- **envoyer le message de retour au site appelant**, attendre l'**accusé de réception** du message de réponse et **terminer l'exécution** soit par autodestruction si le processus a été créé à la demande, soit par blocage s'il s'agit d'un processus pris dans un pool.

Les talons

Comme noté ci-dessous, on utilise deux procédures appelées talons. Le talon client (stub), qui se trouve sur le site appelant, fournit au processus client une interface identique à celle de la procédure appelée. De même, le talon serveur (skeleton) qui se trouve sur le site appelé, réalise pour le processus serveur une interface identique à celle d'un appel local. Chacun des talons, lié au programme du processus client ou serveur sur le site correspondant, fonctionne aussi comme un représentant de l'autre processus (client ou serveur) sur ce site.

Ainsi, à la compilation, un appel à une procédure distante est remplacé par un appel

à un talon client. A l'appel lors de l'exécution, les paramètres effectifs sont rassemblés dans un message, puis par un appel au noyau, et via des sockets, ce message est transmis au noyau de la machine distante. Enfin, le noyau de la machine distante transmet ce message au serveur du processus chargé du service qui désassemble les paramètres, appelle la fonction de façon locale et classique et le résultat fait le cheminement inverse.

c) Quelques exemples de RPC

Le **RPC de Sun/ONC** a été développé initialement pour la mise en œuvre du système de fichiers répartis NFS. Ce RPC, facile d'emploi, permet essentiellement de construire des appels de procédure entre stations Unix avec des programmes client et serveur écrits dans le langage C. Ce RPC a servi de référence pendant très longtemps pour l'écriture d'applications réparties selon le modèle client-serveur.

L'OMG, au travers de la spécification d'architecture Corba, a apporté deux contributions significatives au modèle client-serveur. La première concerne l'utilisation des propriétés bien connues de l'objet (modularité, encapsulation, typage, héritage, etc.) pour décrire les services accessibles à distance. La seconde contribution est le rôle clé donné au langage de définition d'interface (IDL) comme format pivot de représentation du contrat entre le client et le serveur. Ce langage permet l'écriture d'applications client-serveur multilingages.

L'environnement **DCOM**, disponible sur les diverses variations du système Windows de Microsoft, offre une fonction similaire pour écrire des applications à base d'objets/composants COM interconnectés selon un modèle client-serveur. Le RPC disponible dans Windows est une évolution du RPC de l'environnement DCE.

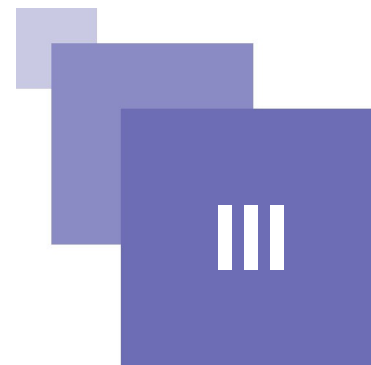
Java RMI a intégré au sein de la programmation objet la notion d'appel de procédure à distance. Il permet l'écriture d'applications client-serveur en Java avec la possibilité de charger dynamiquement le code du talon client. Cette facilité permet à un client de se lier au serveur uniquement lors de l'appel. Java RMI fera l'objet d'une étude plus approfondie dans le volume 2 du polycopié.

HTTP est un protocole client-serveur utilisé, entre autres, par les navigateurs des stations clientes pour dialoguer avec les serveurs Web à l'échelle de la planète. Dans sa version de base, HTTP permet l'accès distant à quatre services pour consulter ou modifier des données gérées par les serveurs. La consultation d'informations sur le Web est sans aucun doute l'application client-serveur la plus utilisée aujourd'hui.

d) Alternative de RPC

Les technologies d'échanges de messages constitue une solution alternative. Les applications communiquent entre elles via des messages véhiculés par l'infrastructure MOM. Ces messages ont une nature complètement générique et peuvent représenter tous types de données, binaire ou texte. L'exemple le plus représentatif de cette technologie est JMS.

Le web



Web et ses composants	29
HTTP	33
HTML	40
Les feuilles de style	45

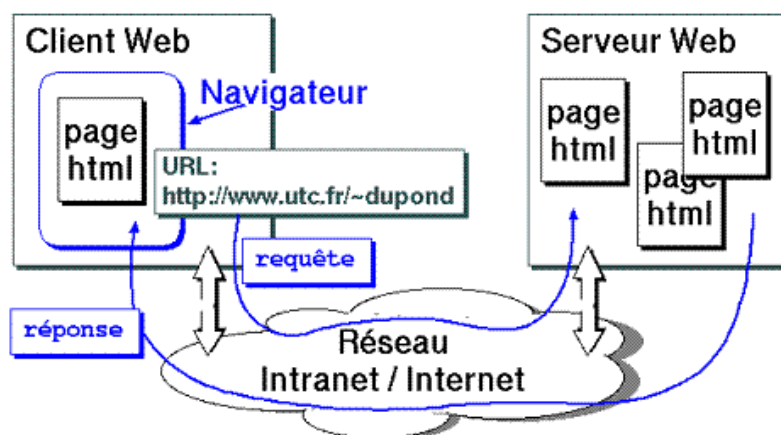
Ce chapitre donne une brève description de quelques notions fondamentales qui sont le Web, URL, HTTP, HTML et CSS.

A. Web et ses composants

Web est un système d'information hyper-média réparti qui contient du texte, des images, animées ou non, son etc. Les sources d'informations sont stockées sur des serveurs : www.orange.fr, www.cisco.fr, tuxa.sme.utc, etc. Pour pouvoir interroger les serveurs on utilise des clients tels que Netscape, Internet Explorer, Piano, Safari, etc. Ce sont ces clients qui traduisent nos demandes en requêtes HTTP pour dialoguer avec les serveurs. Les liens qui unissent les navigateurs avec les serveurs sont du type client-serveur.

1. Les trois grandes briques du web

Le Web est composé de trois grandes briques: HTML, URL; et HTTP, comme illustré dans la figure ci-dessous



Cheminement d'une connexion

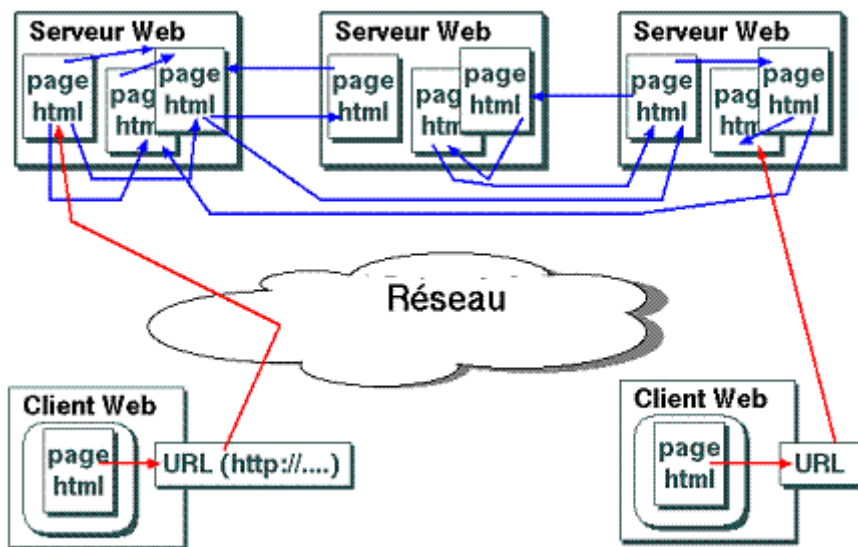
Expliquons le schéma de la Figure "cheminement d'une connexion" : le navigateur initialise une requête en tapant (cliquant) l'adresse (le lien) `HTTP://www.utc.fr/~dupond` et une requête est transmise par le protocole HTTP via l'internet, au serveur qui héberge cette page HTML. La transmission via Internet inclue évidemment le passage par les protocoles TCP, IP et la couche physique. Le serveur traite la requête et envoie une réponse HTTP incluant la page demandée dans le corps de la réponse.

2. Le processus de téléchargement d'une page web

Le processus pour télécharger une page web passe par les étapes suivantes :

- l'utilisateur clique sur un lien web ou tape une adresse web
- le navigateur (client HTTP) formule ensuite une requête à envoyer au serveur Web
- l'application HTTP passe cette requête à TCP (couche transport)
- TCP ajoute son en-tête, dans laquelle figure le numéro de port client, et le numéro de port serveur (ici 80), ensuite transmet l'ensemble (segment) à la couche réseau, c'est-à-dire IP
- IP ajoute son en-tête dans laquelle figure l'adresse IP source et IP destination
- IP transmet l'ensemble (paquet) à la couche physique
- la couche physique rajoute son en-tête (p.ex. Ethernet) pour obtenir une trame, et envoie les données (bits) sur le support physique

Au fur et à mesure des passages dans les switch/routeurs on fait le chemin inverse, c'est-à-dire on lit, élimine les entêtes, on récupère les segments/paquets, on rajoute une nouvelle entête et ainsi de suite.



Le web

En bref, le WEB est destiné à relier les documents à travers un réseau mondial (1989) en faisant intervenir les pages web, les adresses web et la navigation entre les pages via le réseau (Figure "Le web").

3. Description des trois briques du web

Examinons chacune des trois briques citées ci-dessus :

Le premier élément

Les pages web sont des documents construits avec un langage simple et accessible dans le but de favoriser une navigation aisée entre eux. On parle alors de Hypertext comme un ensemble de documents reliés par des « hyperliens », (tandis que Hypermedia étend le concept au delà du texte). L'idée de base consiste à utiliser des notations pour structurer le texte, et cela a été réalisé par le langage SGML en 1986, et ensuite par HTML qui est le langage universel pour écrire des pages web. HTML est réputé plus simple et moins formelle que SGML, ce qui a, en partie, déterminé son succès.

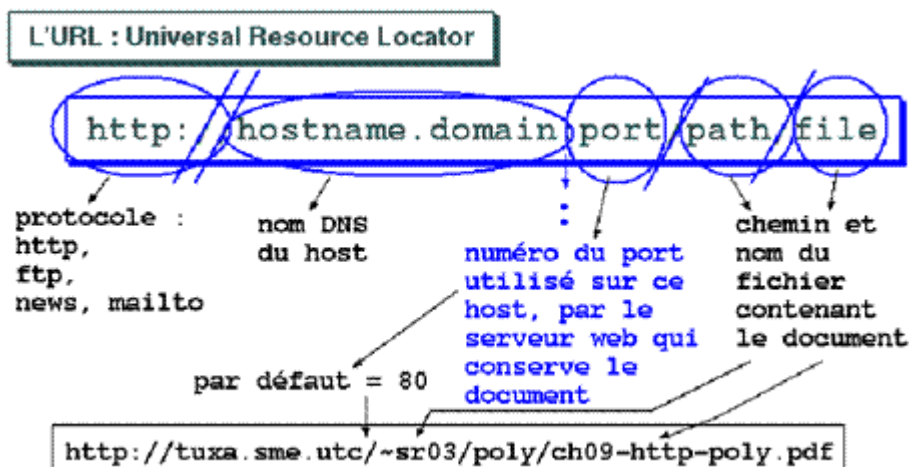
Le deuxième élément

Le deuxième élément clé du web concerne l'adressage. Pour cela on a l'adresse web URL.

URL désigne une ressource sur Internet dans un format de nommage universel. Cette adresse est composée de :

HTTP://nom d'hôte [: port]/chemin [;paramètres] [?requête],

où les termes entre crochets concernent des informations optionnelles. Le nom d'hôte est le nom du domaine ou l'adresse IP du serveur hébergeant la ressource demandée. Le numéro de port, (qui est associé à un service) permet au serveur de savoir quel type de ressource est demandé. Pour HTTP il est par défaut 80. Ensuite, on trouve le chemin d'accès et le nom du fichier contenant la ressource. Les éléments [;paramètres] [?requête], concernent des arguments optionnels précisant la requête soumise par le navigateur au serveur (voir plus loin la méthode GET avec passage de paramètres).



Composition d'une URL

Notons ici qu'il est impossible de se connecter directement sur le serveur qui héberge le fichier souhaité à partir de l'information marquée dans le lien. Cela n'est pas possible car la machine du client a besoin d'une adresse physique pour faire la connexion avec la machine de destination. On a donc besoin d'un serveur traducteur pour connaître l'adresse IP du serveur destination. On utilise pour cela le système appelé DNS qui permet d'associer des noms en langage courant aux adresses numériques. Il s'agit donc de serveurs DNS dédiés qui fournissent en échange d'un nom de domaine, une adresse IP plus compréhensible par les protocoles de transport et de routage. Clairement, une URL identifie en fait l'emplacement d'une ressource, plutôt que la ressource elle-même. Ainsi, lorsqu'une ressource est déplacée, par exemple mise sur un autre serveur Web, toutes les URL l'identifiant sont rendues obsolètes. Ce problème est à la base de la plupart des hyperliens cassés du Web. Pour remédier à cela il a été proposé d'utiliser les adresses URN qui identifient les ressources elles-mêmes. Néanmoins, les URN ne sont pratiquement guère utilisées.



Remarque

Plus récemment on entend parler d'URI. C'est une généralisation d'URL dont le nommage respecte une norme d'Internet mise en place pour le Web (RFC 3986).

Le troisième élément

Le troisième élément concerne la navigation web. Cette dernière se réalise grâce au protocole HTTP. Rappelons que Web fonctionne selon le modèle client-serveur, le client est le navigateur et le serveur est la machine qui héberge les services demandés par le navigateur de la machine client. L'utilisation du protocole HTTP définit le langage utilisé pour les échanges entre client et serveur Web. Les objets manipulés sont repérés par leur URL. Notons aussi que HTTP n'est pas un langage en soi, mais un protocole de communication, ce qui définit une syntaxe pour formuler des demandes et des réponses. Pour faire des requêtes HTTP et recevoir les réponses du serveur, il faut utiliser les sockets. On utilise alors un vrai langage de programmation, le C par exemple. Le langage se chargera de se connecter au serveur, d'envoyer / recevoir les requêtes et de fermer la connexion.

B. HTTP

HTTP est un protocole de la couche application destiné à assurer les échanges entre un serveur et un client dans l'Internet.

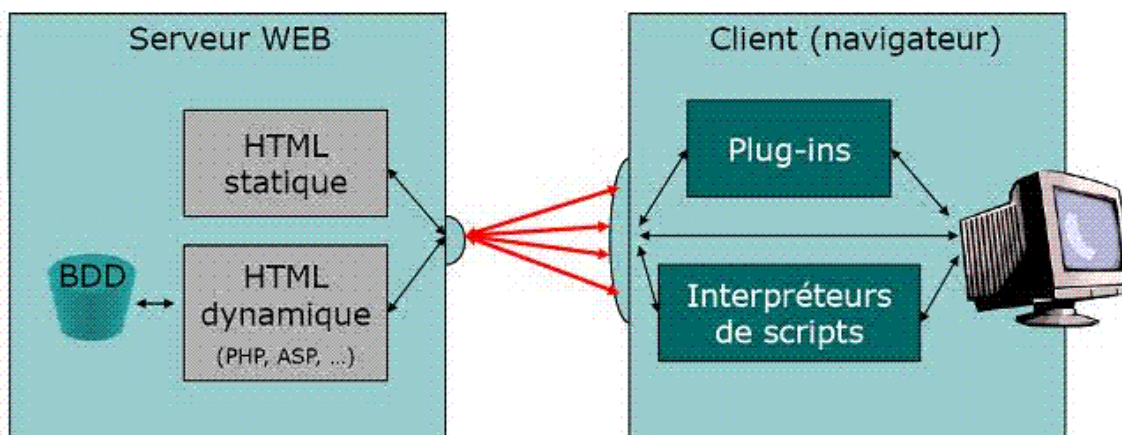


Schéma général du fonctionnement du protocole HTTP

Le protocole HTTP, normalisé par des RFC, a été défini en trois versions. La version 0.9 est le protocole défini à l'origine par Tim Berners-Lee (l'idéateur du WEB). La version 1.0 apporte de très nombreuses fonctionnalités (typage des documents, codage du contenu, identification, ...). La dernière, la version 1.1, est la plus avancée et permet de réaliser différentes négociations, mais surtout d'obtenir des sessions persistantes.

1. Principes de fonctionnement

1. HTTP se situe au niveau de la couche Application. Il s'appuie donc sur la couche transport et le protocole TCP en occurrence.
2. HTTP peut transporter du binaire brut (images...) sans encodage (pas d'encodage MIME par exemple). Le navigateur (le client) communique avec le serveur Web à travers une ou plusieurs connexions TCP en se connectant sur le port 80.
3. HTTP repose sur le système de requêtes/réponses. HTTP est un protocole très simple : le client établit une connexion TCP avec le serveur, il envoie une requête et attend une réponse du serveur.
4. HTTP est un protocole sans état : une fois la réponse envoyée, le serveur marque la fin de la réponse en fermant la connexion TCP. Donc, il n'y a pas de session permanente entre le client et le serveur.
5. HTTP assure le transfert bidirectionnel : d'habitude le transfert est du serveur vers le client mais dans certains cas (p.ex. envoi de données d'un formulaire), le transfert inverse peut avoir lieu.
6. Négociation de fonctionnalités : HTTP permet aux navigateurs et aux serveurs de négocier entre eux des détails de leurs sessions, type de caractère, etc. Le client précise ce qu'il peut accepter et le serveur ce qu'il propose.
7. Support de cache : pour réduire le temps de réponse, le navigateur place des copies des documents chargés en cache. Pour éviter l'affichage de documents périmés, lorsque l'utilisateur visite une page pour une deuxième fois, HTTP permet de vérifier auprès du serveur si le document est à jour.
8. Support de proxy serveur (un serveur situé au long du chemin qui agit comme serveur mandataire du serveur) : HTTP spécifie comment ce type de serveurs traite les requêtes et interprète les entêtes.

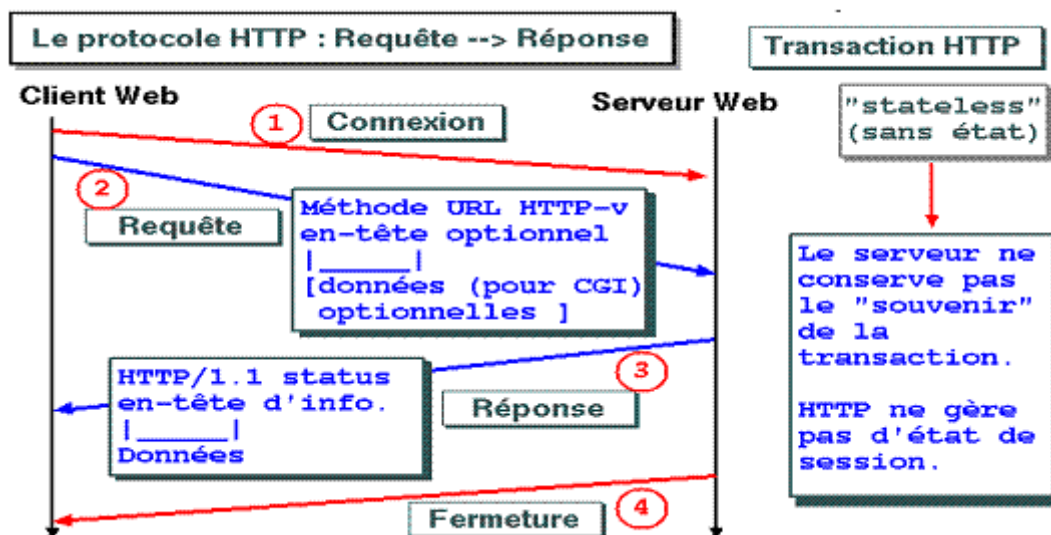
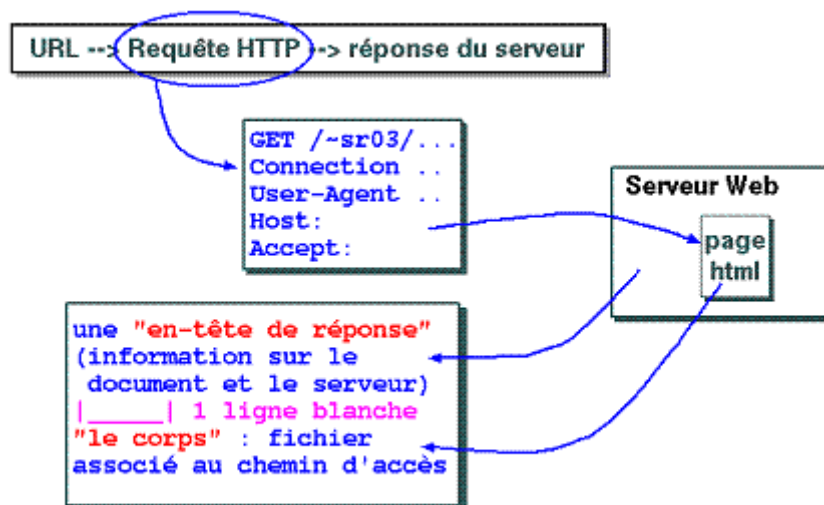


Schéma d'échange requête - réponse HTTP

2. L'échange de messages

L'échange entre le navigateur et le serveur se fait par des messages (des textes). Il y a deux types de messages, les requêtes et les réponses.



Requête/réponse HTTP

a) Les requêtes HTTP

Les requêtes HTTP, comme leur nom l'indique, constitue le moyen utilisé par HTTP pour prendre contact avec le site du serveur et lui préciser la demande (la requête).

Voici le format général d'une requête HTTP :

```
la ligne de requête ;
les en-têtes (0 ou plus) ;
<ligne blanche> ;
Le corps de la requête (qui est présent uniquement pour une
requête de type POST).
```

Le format de la ligne de requête est : « *requête URL-voulue HTTP-version* »

exemple : « *GET /sr03/td1.html HTTP/1.0* ».

Il existe trois requêtes distinctes :

la requête **GET**, qui renvoie le contenu de l'URL demandée (si elle existe) ;

la requête **HEAD**, où le serveur ne renvoie que l'en-tête (permet notamment de vérifier l'accessibilité) ;

la requête **POST**, qui permet d'envoyer des données au serveur (formulaires...).

On distingue les requêtes simples, GET et HEAD, et les requêtes avec envoi d'information et/ou modifiant le contenu du serveur (GET et POST).

Les requêtes simples, GET et HEAD, demandent respectivement un document ou une information sur le document. Ces requêtes peuvent ne pas contenir d'en-tête, elles ne contiennent que la ligne de commande.

Les requêtes avec envoi d'information concernent surtout les échanges via formulaires HTML. Les informations sont incluses dans le corps de la requête pour POST ou rajoutées dans l'URL de destination pour GET (Figure 9).

HTTP : Requêtes avec envoi d'informations

POST

On fournit de l'information au serveur
(par ex. issue d'un formulaire) par:
`POST /chemin/cgi-bin/form.php HTTP/1.1`
en-tête
|_____| ligne blanche
`var1=data1&var2=data2&....`

GET + infos

GET + Format "URL encodé"

<FORM>...method=GET>

l'info. saisie dans la forme est passée au serveur par:

`GET /ch.../form.php?var1=data1&var2=data2 HTTP/1.1`

fichier ? var = data & var = data ...

Les formats de requête HTTP

Regardons dans la suite les entêtes et les réponses pour le protocole HTTP 1.0, le principe étant le même pour HTTP 1.1.

Une entête se divise en un nom de champ, un caractère ':', un espace et une valeur de champ. Les en-têtes sont classés en trois catégories : ceux qui s'appliquent aux requêtes, aux réponses et ceux qui décrivent le corps du message.

b) Les entêtes

Les entêtes générales HTTP

Cache-Control = contrôle du caching.

Connection = listes d'option (close pour terminer une connexion).

Date = date actuelle.

MIME-Version = version MIME utilisé. (MIME permet le codage de données non ASCII).

Pragma = instruction pour le proxy.

Transfer-Encoding = type de la transformation appliquée au corps du message.

Via = utilisé par les proxys pour indiquer les machines et protocoles intermédiaires.

Les entêtes de requêtes client HTTP

Accept = type MIME visualisable par l'agent

Accept-Encoding = méthodes de codage acceptées compress, x-gzip, x-zip

Accept-Charset = jeu de caractères préféré du client

Accept-Language : liste de langues fr, en, ...

Authorization : identification du browser auprès du serveur

Cookie = cookie retourné

Content-Length = Longueur du corps de la requête

From = adresse email de l'utilisateur (rarement envoyé pour conserver l'anonymat de l'utilisateur)

Host = spécifie la machine et le port du serveur (un serveur peut héberger plusieurs serveurs)

If-Modified-Since = condition de retrait la page n'est transférée que si elle a été modifiée depuis la date précisée. Utilisé par les caches.

Max-Forwards = nombre maximal de proxy

Proxy-Authorization = identification

Range = zone du document à renvoyer bytes=x-y (x=0 correspond au premier octet, y peut être omis pour spécifier jusqu'à la fin)

Referer = URL d'origine page contenant l'ancre à partir de laquelle a été trouvé l'URL.

User-Agent = donner des informations sur le client, comme le nom et la version du navigateur, du système d'exploitation, etc.



Exemple

Demande client. Soit l'URL *HTTP://tuxa.sme.utc/sr03/td1.html*.

Le message envoyé au serveur est :

```
GET /sr03/td1.html HTTP/1.1 (1)
Accept: image/gif, image/x-xbitmap, image/jpeg, */* (2)
Accept-Language: fr (3)
Accept-Encoding: gzip, deflate (4)
User-Agent: MSIE 7.01 (5)
Host: tuxa.sme.utc (6)
Connection: Keep-Alive (7)
```

1. Le client demande au serveur (via GET) le document dont la localisation est */sr03/td1.html*. Le client indique sa version de protocole (HTTP1.1).
2. Le client indique au serveur quels sont les types de documents qu'il accepte.
3. Le client indique sa langue préférée (français).
4. Le client indique qu'il sait traiter une réponse compressée par gzip et/ou deflate.
5. Le client s'identifie comme étant Internet Explorer 7.01.
6. Le client fournit le nom du serveur.
7. Le client demande au serveur de conserver la connexion ouverte. Il s'agit d'une connexion TCP et non d'une connexion applicative.

c) Les réponses HTTP

De façon générale les réponses se présentent comme suit :

```
ligne de statut
en-têtes (0 ou plus)
```

```
<ligne blanche>
corps de la réponse (il contient le document demandé)
```

La première ligne de la réponse d'un serveur Web s'appelle la ligne de statut. La ligne de statut est de la forme :

« HTTP-version code-réponse phrase-réponse », p.ex. « HTTP/1.1 200 OK ».

Elle commence par la version du protocole HTTP et elle est suivie d'un code numérique de réponse de trois chiffres et d'une phrase :

100-199 Informationnel

- 100 : Continue (le client peut envoyer la suite de la requête), ...

200-299 Succès de la requête client

- 200: OK, 201: Created, 204 : No Content, ...

300-399 Redirection de la Requête client

- 301: Redirection, 302: Found, 304: Not Modified, 305: Use Proxy, ...

400-499 Requête client incomplète

- 400: Bad Request, 401: Unauthorized, 403: Forbidden, 404: Not Found

500-599 Erreur Serveur

- 500: Server Error, 501: Not Implemented, 502: Bad Gateway, 503: Out Of Resources (Service non disponible).

Les réponses, comme les requêtes, peuvent contenir un certain nombre de champs d'en-tête. Une ligne blanche permet de séparer ces derniers du document. Voici les en-têtes réponse.

Les entêtes réponse :

Accept-Range = autorisation ou refus d'une requête par intervalle

Age = ancienneté du document en secondes

Proxy-Authenticate = système d'authentification du proxy

Public = liste de méthodes non standards gérées par le serveur

Retry-After = date ou nombre de secondes pour un nouvel essai en cas de code 503 (service unavailable)

Set-Cookie = créer ou modifie un cookie sur le client

WWW-Authenticate = système d'authentification pour l'URI

Allow = méthodes autorisées pour l'URI

Content-Base = URI de base

Last-Modified = date de dernière modification du document utilisé par les caches

Content-Length = taille du document en octet utilisé par le client pour suivre la progression des chargements

Content-Location : URI de l'entité quand l'URI est à plusieurs endroits

Content-Range : position du corps partiel dans l'entité (bytes x-y/taille)

Content-Transfert-Encoding : transformation appliqué du corps de l'entité (7bit, binary, base64, quoted-printable)

Content-Type = type MIME du document renvoyé utilisé par le client pour sélectionner le visualisateur (plugin).

ETag : identifiant unique opaque assigné par le serveur web à chaque version d'une ressource accessible via une URL.



Exemple

Voici un exemple de réponse

```
Accept-Range = autorisation ou refus d'une requête par
```



```

intervalle
Age = ancienneté du document en secondes
Proxy-Authenticate = système d'authentification du proxy
Public = liste de méthodes non standards gérées par le
serveur
Retry-After = date ou nombre de secondes pour un nouvel
essai en cas de code 503 (service unavailable)
Set-Cookie = créer ou modifie un cookie sur le client
WWW-Authenticate = système d'authentification pour l'URI
Allow = méthodes autorisées pour l'URI
Content-Base = URI de base
Last-Modified = date de dernière modification du document
utilisé par les caches
Content-Length = taille du document en octet utilisé par le
client pour suivre la progression des chargements
Content-Location : URI de l'entité quand l'URI est à
plusieurs endroits
Content-Range : position du corps partiel dans l'entité
(bytes x-y/taille)
Content-Transfert-Encoding : transformation appliqué du
corps de l'entité (7bit, binary, base64, quoted-printable)
Content-Type = type MIME du document renvoyé utilisé par le
client pour sélectionner le visualisateur (plugin).
ETag : transformation appliqué du corps de l'entité (7bit,
binary, base64, quoted-printable).

```

3. La version HTTP 1.1

La nouveauté dans la version HTTP 1.1 consiste dans la conception de **connexions TCP persistantes**, utilisation de nouveaux entêtes et de nouvelles requêtes, possibilité d'utiliser de multiples serveurs http sur la même machine (impossible avec http 1.0), et enfin un meilleur fonctionnement des caches.

HTTP 1.0, comme FTP, nécessitait l'ouverture d'une connexion TCP pour chaque requête. HTTP 1.1 spécifie désormais la possibilité de maintenir une connexion TCP persistante, évitant ainsi d'inutiles ouvertures de connexions entre le même serveur et le même navigateur. La connexion est fermée sur la demande du navigateur ou serveur. L'intérêt principal est de réduire la charge. Un navigateur peut encore croître ses performances en *pipeliant* les requêtes. Le seul souci des connexions persistantes est la nécessité d'identifier le début et la fin de chaque objet envoyé. Cela se réalise soit en précisant la longueur de l'objet au début de l'envoi, soit en mettant un caractère spécial à la fin des données. On utilise la première méthode. Notons au passage que cette méthode est difficilement utilisable dans certains cas d'envoi de pages dynamiques. Le serveur informe le client, envoie l'objet et ferme la connexion.

Dans cette dernière version de HTTP on trouve aussi l'utilisation de nouveaux entêtes pour une meilleure négociation, de nouvelles requêtes, etc. Voici les nouvelles requêtes :

PUT : permet d'envoyer au serveur un document à enregistrer à l'URL spécifiée. Des entêtes permettent de contrôler que tout s'est bien passé.

DELETE : efface la ressource spécifiée.

OPTIONS : permet au client de savoir les options de communication utilisables pour obtenir la ressource. Il s'agit donc de négociation.

TRACE : méthode de contrôle. Demande au serveur de renvoyer la requête telle qu'elle a été reçue. C'est comme une sorte de Ping.

HTTP 1.1 est la version finale de HTTP. Énormément d'améliorations ont été apportées depuis la version initiale HTTP 0.9. Il est un protocole mature et stable et

on n'attend plus de modifications majeures. Le W3C, l'organisme chargé de sa définition, a fermé l'activité HTTP. Les efforts se portent maintenant plus vers des protocoles de plus haut niveau comme les Web services.

4. Les cookies et les sessions

Revenons maintenant à un aspect très important du protocole HTTP. HTTP est un protocole « *sans mémoire* », dit « *stateless* ». Donc, il n'y pas de possibilité d'établir une session avec HTTP, chaque communication étant indépendante. Pour remédier à cela sont proposés les cookies. Les cookies sont des informations stockées au niveau du client, concernant la navigation, i.e., domaine du serveur, une paire clé valeur, date d'expiration. Lors de la première connexion, un serveur HTTP envoie à un client HTTP, un cookie que ce dernier stocke en disque dur quand il le reçoit pour la première fois, et le retourne lors de chaque interrogation du même serveur HTTP.

Les **cookies** sont donc de petits fichiers textes stockés par le navigateur web sur le disque dur du visiteur d'un site web et qui servent à enregistrer des informations sur le visiteur ou encore sur son parcours dans le site. Il est alors possible au serveur web de reconnaître un visiteur et personnaliser la réponse pour chaque visiteur.

Un cookie a une durée de vie limitée, fixée par le concepteur du site. Ils peuvent aussi expirer à la fin de la session sur le site, ce qui correspond à la fermeture du navigateur. Les cookies sont largement utilisés pour simplifier la vie des visiteurs et leur présenter des informations plus pertinentes. Dans la pratique, le serveur web envoie un entête HTTP :

```
Set-Cookie : NOM=VALEUR;  
domain=NOM_DE_DOMAINE;  
expires=DATE;
```

Le navigateur envoie au serveur un entête

```
Cookie : NOM1=VALEUR1;
```





Fondamental

- Un cookie est en réalité un fichier stocké sur le disque dur de l'utilisateur ; virus impossible, pas d'accès au disque dur, lecture par notepad ;
- Ils permettent au serveur web de reconnaître les clients d'une page web à l'autre.
- Ils sont gérés dans les en-têtes HTTP échangés lors des communications HTTP. En fait, les cookies font partie des spécifications de HTTP, lequel les utilise pour remédier à la « perte de connexion » en faisant une suivie de sessions.
- Ne pas utiliser les cookies pour stocker des informations confidentielles à l'intérieur (mot de passe par exemple).

C. HTML

L'essentiel des documents du Web sont des fichiers textes. Ces fichiers ont un contenu et une **structure** formalisée à l'aide de **balises** ("*tags*") tels que <title>, <h2>, <head>, <p>, , , etc . Pour cela il est utilisé le langage HTML, un langage à balise issu d'un langage antérieur et bien plus complexe, qui est le SGML. Ce dernier est défini par la communauté des documentalistes et gestionnaires de bibliothèques pour décrire complètement des documents de tout type. Après la définition et le succès d'usage de XML, on a fait évoluer HTML en **XHTML** qui reprend et étend HTML.

1. Principes de base

Structuration d'une page web

Un document HTML doit commencer par une déclaration de type de document (<!DOCTYPE HTML PUBLIC "type_de_HTML" "adresse_de_DTD">) :

```
Strict : <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
Transitional : <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
4.01
Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
```

Cela donne une référence à la norme HTML utilisée permettant de spécifier le standard utilisé pour le codage de la page.

a) Les balises

Le langage HTML est construit sur le concept des balises. On distingue les balises contenant des méta informations, les balises de mises en forme de la page web, les balises de liens, les balises d'insertion multimédia (images, son, vidéos..), etc. Les balises ne sont pas sensibles à la casse (préférer en minuscule). On utilise la forme d'écriture suivante :

-<nom-balise> ... </nom-balise>

-<nom-balise /> (p.ex.
 pour être conforme au XHTML)

Les balises méta servent essentiellement à enseigner sur la langue utilisée du document web, le type de document, le codage utilisé, l'auteur. Ces balises servent également à rediriger automatiquement vers une autre page, interdire la mise en cache ou l'indexation par les moteurs de recherche.



Exemple

```
<meta name="Author" content="Michel Vayssade, Dritan Nace">
<meta name="Copyright" content="UTC">
<meta name="Keywords" content="SR03, Architecture Internet, Applications distribuées">
```

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta http-equiv="Content-Language" content="en, fr">
<meta http-equiv="Refresh" content="10; URL=HTTP://www.hds.utc.fr">
<meta http-equiv="Pragma" content="no-cache">
<meta http-equiv="Content-Style-Type" content="text/css"> .
```

Balises d'insertion pour la multimédia (images, son, vidéos..).

Ces balises s'écrivent : ``, `<object>`. Image : ``

D'autres balises sont :

Pour les commentaires on a `<! ... -->`.

Pour intégrer des scripts, styles etc., on utilise `<script>`, `<style>`.

Les liens web

Un des objectif de HTML est de faciliter la navigation entre les pages web, d'où l'utilisation des liens. On distingue un lien "absolu" :

`Aller page interne UTC`,

et un lien "relatif" :

`voir test3.html`.

Le formulaires

Les balises du formulaire dont les suivantes : balise d'ouverture `<FORM>` et fermeture `</FORM>`. La spécification de la méthode employée pour l'envoi d'informations au serveur (GET) + l'adresse de destination des informations (`cgi-bin/test.cgi`):

`<FORM METHOD=GET ACTION=cgi-bin/test.cgi>.. </FORM>`.

Parmi les éléments d'un formulaire on distingue les balises et les champs. Les Balises sont : `<input>`, `<select>`, `<option>`, `<textarea>`. Parmi les champs on citera Champ de texte, Zone de texte, Boutons radios, Cases à cocher, Liste déroulante, Liste ouverte.

2. Un exemple

```
<FORM METHOD=GET ACTION=cgi-bin/test9.cgi> .
<P>Entrez votre nom et vos choix :<BR>
<P>Nom : <INPUT TYPE=TEXT NAME=nom SIZE=40>
```

```
<P>Choisir 1 parmi 3 :
<INPUT TYPE=radio NAME=choix1 value="un">numéro un
<INPUT TYPE=radio NAME=choix1 value="deux">numéro deux
<INPUT TYPE=radio NAME=choix1 value="trois">numéro trois
<P>Cocher si urgent : <INPUT TYPE=checkbox Name=urg>
<p>choisir 1 dans menu :
<SELECT NAME=fichier SIZE=3>
<OPTION>fichier1.txt
<OPTION>fichier2.txt <OPTION>fichier3.txt
<OPTION>fichier4.txt <OPTION>fichier5.txt
</SELECT>
<P><INPUT TYPE=submit VALUE=Envoyer><INPUT TYPE=reset
VALUE=Effacer>
</FORM>
```

Qui donne le résultat suivant :

Entrez votre nom et vos choix :

Nom :

Choisir 1 parmi 3 : ☐ numéro un ☐ numéro deux ☐ numéro trois

Cocher si urgent : ☐

choisir 1 dans menu :

fichier1.txt
fichier2.txt
fichier3.txt

Formulaire

HTML décrit la structure du document, ainsi que des indications sur la façon d'afficher certains éléments. Mais cette façon dépend de la configuration du browser sur laquelle l'auteur du document ne peut pas agir. Ce mélange des genres entre structure et "aspect" est mauvais et est corrigé par l'utilisation des CSS et de XHTML.

3. HTML 5

HTML5 renforce la sémantique à une page web, c'est-à-dire il apporte de nombreux nouveaux éléments qui viennent renforcer le sens d'une page web. Les nouveaux éléments de découpage d'une page (section, article, aside, etc.) ainsi que les nouvelles balises (time, figure, figcaption, etc.) permettent un transport d'informations non négligeable qui est utilisée par les moteurs de recherche, et le

sera certainement davantage par les lecteurs d'écran ou d'autres terminaux dans un avenir proche.

Voici les principales nouvelles balises sémantiques (voir la figure ci-dessous):

- un en-tête de page : HEADER
- une barre de navigation : NAV
- une colonne de gauche : ASIDE
- une zone principale pour le contenu : ARTICLE
- un pied de page : FOOTER



Les nouvelles balises sémantiques de HTML 5

Mais HTML5 apporte aussi de nouvelles pièces à notre panoplie d'intégrateur web : les microdonnées. Ces informations ainsi émises par votre code source peuvent être utilisées par Google.

Grâce à de nouveaux attributs HTML5, il est désormais possible de donner la référence du document utilisé pour baliser l'information directement au sein de l'élément porteur de ces informations.

Un microformat (abrégé sous μ F ou uF) est une approche de formatage de données conçue pour permettre à l'information destinée aux utilisateurs finaux (comme le carnet d'adresses, les coordonnées géographiques, les événements et autres données en rapport) d'être traitée automatiquement par le logiciel de parcours (parseur).

Même si le contenu des pages web est déjà capable techniquement d'être « traité automatiquement », et cela depuis la conception du web, il existe certaines limites. Ceci parce que les balises traditionnelles de marquage étaient utilisées pour afficher l'information sur le Web et non pas pour décrire ce que voulait dire l'information. Les microformats sont destinés à combler ce fossé en attachant de la sémantique, et par conséquent éviter d'autres méthodes plus compliquées de traitement automatisé, comme le traitement du langage naturel ou le screen scraping.

Un autre élément qu'on retrouve dans HTML5 c'est le format SVG : un langage alternative XML à PNG ; Ses avantages sont : adapté au web ; portabilité pour les terminaux de petites taille (agendas, ordinateurs portables, tablettes, téléphones portables..) et imprimantes ; Intégration au langage XML et XHTML aisée; capacités graphiques élevées et contenu accessibles aux moteurs de recherches...

D. Les feuilles de style

HTML utilise des techniques complexes et lourdes : tableau, images, JavaScript,

applets, etc. Le résultat est que tous les éléments extra doublent la taille des pages ; on fait usage intensifs de graphiques, d'où des problèmes de maintenances par manque de lisibilité ; enfin, utilisation intensive de JavaScript, etc... qui rend encore plus difficile la maintenance.

L'objectif est de simplifier la présentation/design des pages web. Les standards définissant CSS sont publiés par le World Wide Web Consortium (W3C). Actuellement nous sommes au CSS3. CSS présente certains avantages : Il permet de gérer séparément la structure et la présentation, présenter de façon homogène, positionner rigoureusement les éléments, etc. En conséquence, le code HTML est allégé et gagne en lisibilité.

Considérons la balise `<style>`. Elle contient les attributs suivants :

```
type="..." : type de contenu Internet
media="..." : définit le média de destination (screen,
print, projection, braille, speech, all)
title="..." : titre de la feuille de styles
```

Pour les navigateurs ne supportant pas les feuilles de style on utilise :

```
<STYLE type="text/css">
<!--
.....
-->
</STYLE>
```

Un exemple de règle CSS est `H1 { color : blue }`. Une règle CSS est composée de deux parties : un sélecteur (ici `H1`) et une déclaration (`color:blue`). Une déclaration a elle-même deux parties : une propriété (`color`) et une valeur (`blue`).

1. Placement des feuilles de style

Le CSS peut être placé à trois positions différentes dans la page. Dans l'élément HTML lui-même, dans la page HTML ou dans un fichier indépendant. Voici les trois méthodes.

Dans l'élément HTML lui-même

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<HTML>
<HEAD>
<TITLE>
CCSinline
</TITLE>
</HEAD>
<BODY>
<p style="font-style: italic; size: 1.5em;"> CSS
directement dans la balise HTML </p>
</BODY>
</HTML>
```

Cette méthode est à éviter.

Dans le code dans la page HTML

```
<HTML>
<HEAD>
<TITLE>Exemple concret de Cascading Style Sheets</TITLE>
```

```
<STYLE TYPE="text/css">
BODY {
color: white;
background: black;
}
</STYLE>
</HEAD>
<BODY>
Texte en blanc sur un fond noir
</BODY>
</HTML>
```

Dans une feuille de style dans un fichier séparé

Le principe est très simple, il s'agit de placer la feuille de style dans un fichier séparé, et à y faire référence dans l'en-tête du document. Pour cela il y a deux façons, les balises `<link />` et `<@import>`. Pour la première la syntaxe utilisée est la suivante : `<link rel="stylesheet" type="text/css" href="style.css" />`. La méthode `<link href=..."` permet aussi de mettre en place des feuilles de styles destinées aux différents medias (*screen, projection, handheld, print, all*).

```
<..
<head>
<title><title>
<link rel="stylesheet" type="text/css"
href="style.css"media="screen, projection" />
</head>
<body> ...
```

Une version alternative de la balise `<link />`, est «@import». Il s'agit d'une propriété CSS2 qui doit être suivie de l'URL d'un fichier qui contiendra des styles à appliquer en plus de la feuille de style en cours.

```
<style type="text/css">
@import url(styles.css);
</style>
```

L'intérêt est de pouvoir importer des feuilles de style contenues dans d'autres feuilles de style. Il faut noter que certaines anciennes versions de Internet Explorer et Netscape (version 4 ou inférieures) ne reconnaissent pas "@import". Dans ces cas, l'affichage sera « brut », et il n'y aura aucune tentative d'interpréter des commandes CSS non supportées, ce qui pourrait être problématique.

2. Les classes d'éléments

Les classes d'éléments permettent de choisir entre plusieurs types de présentations pour un même élément HTML (ou tous). On peut ainsi avoir les paragraphes d'en-tête, des remarques et des paragraphes normaux.



Exemple

e.rouge { color: red } P.entete { font-style: italic }

```
<HTML>
<HEAD>
<TITLE> Les classes </TITLE>
<STYLE type="text/css">
<!--
.rouge { color: red }
P.entete { color: green; font-style: italic }
//-->
</STYLE>
</HEAD>
<BODY>
<H2 class=rouge> Un titre rouge </H2>
<P class=entete> Un titre italic en vert </P>
</BODY>
</HTML>
```

3. Les pseudo-classes d'ancrage

CSS permet de représenter différemment les liens qui n'ont pas été visités de ceux qui l'ont déjà été au travers des pseudo-classes ':link' et ':visited'.



Exemple

```
A:link { color : fushia; }
A:active { color : green; }
```

La pseudo-classe :link s'applique aux liens qui n'ont pas été visités ;

La pseudo-classe :visited s'applique lorsque le lien a été visité par l'utilisateur.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
<html>
<head>
<link rel="stylesheet" href="P.css" type="text/css">
<title>CSS Pseudo Classes</title>
</head>
<body>
<td>
<a href="HTTP://www.utc.fr">site UTC</a>
</td>
</body>
</html>
```

Fichier P.css

```
a:link
{background-color: YellowGreen;
color: white;
font-family: Arial;
font-size: .8em;}
a:hover
{background-color: cyan;
color: blue;
font-family: Arial;
font-size: .8em;}
```

4. La notion de cascade

Les propriétés des CSS peuvent être définies plusieurs fois. C'est toujours la dernière définition qui compte. Cela permet d'importer plusieurs feuilles de styles, et de redéfinir certains styles dans le document. Supposons qu'on dispose d'une première feuille de style, que nous appellerons style1.css qui contienne les propriétés suivantes :

```
H1 { color : red; font-size : 48pt }
H2 { color : blue; font-size : 12pt }
```

Nous utilisons aussi une autre feuille de style, nommée style2.css et contenant les propriétés suivantes :

```
H2 {color : green; }
H3 {color : pink; font-size : 12pt }
```

Dans une page donnée, nous incluons dans l'en-tête l'appel de ces deux feuilles, ainsi que la définition d'autres propriétés.

```
<HEAD>
<TITLE>...</TITLE>
```



```
<LINK rel=STYLESHEET href="style1.css" type="text/css">
<LINK rel=STYLESHEET href="style2.css" type="text/css">
<STYLE type="text/css">
<!--
H1 { color : fushia; } H2 { font-size : 16pt; } H3 { font-
size : 14pt; } --> </STYLE>
</HEAD>
```

Les valeurs utilisées pour ce document sont : *H1 : fushia, 48 points ; H2 : vert, 16 points, et H3 : rose, 14 points.*

Pour déterminer la valeur d'une propriété, on dispose donc de la notion de cascade. Dans les cas où la propriété n'a pas été définie, deux possibilités se présentent. La première est la propriété dite "héritée". Dans ce cas, c'est la valeur de l'élément "parent", c'est à dire de l'élément dans lequel est utilisé l'élément courant. Dans l'autre cas, on prend la valeur par défaut. Donc, la dernière règle lue est prioritaire.

5. CSS3

CSS3 permet, mais ne se limite pas, de créer de l'ombre portée sur vos éléments html comme div, header, article, footer, aside, nav, section, image avec la propriété CSS3 BOX-SHADOW. on peut aussi faire du text shadow. Il remplace d'une certaine façon photoshop...



Effet shadow avec CSS 3

* *
*

Notons que nous n'allons pas détailler d'avantage ni le langage HTML ni les feuilles de style. Notre objectif est de donner les notions de base de HTML et CSS dans le but de préparer l'introduction de DHTML. Ensuite, on verra comment peut-on rendre dynamique l'affichage d'une page web grâce au concours de CSS, Javascript et DOM.

Programmation web côté client

IV

Introduction au JavaScript	51
JavaScript et le WEB	55
JavaScript : un langage orienté événements	58
Ajax	67

A. Introduction au JavaScript

JavaScript est un langage de programmation de scripts principalement utilisé dans les pages web interactives, coté client. C'est un langage interprété qui est intégré aux pages HTML. Le langage a été créé en 1995 par Brendan Eich pour le compte de Netscape Communications Corporation. Le langage est actuellement à la version 1.8. A ses débuts JavaScript était surtout utilisé pour des fins graphiques et d'animations de sites. En plus, pour chaque nouvelle version de navigateur, une nouvelle version JavaScript verrait le jour. Seulement plus tard, avec l'arrivée des versions standardisées, un réel allégement de travail a été ressenti dans le camp des programmeurs. A nos jours, JavaScript est surtout destiné à aider les utilisateurs de web à mieux naviguer. D'autres applications peuvent concerner la conception d'outils de mesure d'audience, p.ex. on place des marqueurs dans les pages pour obtenir des informations sur les utilisateurs. Les questionnaires de bandes publicitaires sur Internet l'utilisent aussi pour appeler les bannières, les animations ou mêmes pour appeler les liens sponsorisés. Enfin, avec l'arrivée de AJAX, de nouvelles fonctionnalités ont vu le jour. On peut citer l'aide fournie aux utilisateurs lors du remplissage d'un formulaire.

Notons que JavaScript porte mal son nom. Il n'a pas grande chose en commun avec Java. En effet, JavaScript est interprété et Java un langage compilé. Ensuite, Java est un langage orienté objet notoire bâti complètement sur la notion de classe et fortement typé tandis que JavaScript est un langage orienté objet dit à prototype et il est faiblement typé. D'autres différences moins importantes peuvent être répertoriées, mais du point de vue structures et éléments de programmation procédurale, JavaScript est construit sur une base bien maîtrisée déjà, sur les éléments de Java et C.

Un inconvénient sérieux de JavaScript reste la compatibilité pour différents navigateurs et plateformes. JavaScript est toujours sensible aux versions des navigateurs ou des plateformes, même si les efforts de normalisation commencent à porter leurs fruits.

1. Les bases de la programmation en JAVASCRIPT

a) Principes de programmation en JavaScript

C'est un langage orienté objet un peu rudimentaire (dit à prototype). C'est un langage peu typé. La même variable peut prendre plusieurs types successivement. Pour les programmeurs avancés, il fonctionne également comme un langage orienté objet et comme un langage procédural pour lequel on trouve une syntaxe proche des langages de programmation comme le C et le Java. Le code peut être lu par tout le monde. Les liaisons sont dynamiques, donc la référence des objets n'est pas vérifiée au chargement.

b) JavaScript en tant que langage procédural

JavaScript reconnaît les types primitifs suivants :

- les nombres : 42 ou 3.1415 ;
- les booléens : true ou false ;
- les chaînes : "Bonjour !" ;
- les valeurs undefined ;
- les valeurs null.

JavaScript est faiblement typé. Il ne fait pas de distinction entre entiers et réels. Pire encore, les variables ne sont pas associées à un type particulier.

```
var i = 4.25 ; i = "Bonjour";
```

Dans une expression faisant intervenir des nombres et des chaînes les nombres sont convertis en chaîne :

```
message = " Il y a " + 109 + "étudiants inscrit au SR " + 03;
```

Noter que pour délimiter une chaîne on peut utiliser double guillemets (") ou simple guillemet (').

Comme pour d'autres langages, il y a deux niveaux de visibilité de variables :

- globale : la variable est visible depuis n'importe quel endroit du programme (avec 'use strict', les variables globales commencent à disparaître).
- locale : la variable n'est visible que dans la fonction courante. Pour déclarer une variable locale à une fonction il faut utiliser le mot-clé var.

```
<script language = "JavaScript">
Function test_visibilite() {
var test = 5;
}
test = 2;
test_visibilite();
alert (test);
```

Le résultat est bien évidemment 2.

c) Les opérateurs

Les opérateurs rencontrés en JavaScript sont :

- Opérateurs d'affectation : =, +=, -=, *=, etc.
- Opérateurs arithmétiques : +, -, /, *, %, --, ++
- Les opérateurs logiques : &&, || et !
- Les opérateurs de comparaison : ==, !=, >, <, >=, <=
- Opérateurs sur les chaînes : +, +=.

Les opérateurs spéciaux :

- new permet de créer un objet.

- typeof permet de déterminer le type de l'opérande.
- Void est utilisé pour exprimer l'absence de retour d'une fonction ou plus précisément pour retourner la valeur null.

Expressions conditionnelles « ? : ». La forme est condition ? val1 : val2 ;
m=(age>=18) ? "majeur" : "mineur" ;

Pour écrire des commentaires, deux formes sont possibles : // commentaire jusqu'en fin de ligne ou /* commentaire jusqu'à */.

La déclaration de variables suit la syntaxe suivante : var nom1 [= valeur1] [... , nomN [= valeurN]].

var i = 0, j = 2 ;

La déclaration de fonctions se fait par :

function nom() { instructions }/function nom(param1,...,paramN)
{ instructions }.

L'instruction return (sortir d'une fonction) est : return, return (expression).

d) Les boucles

En JavaScript on retrouve des éléments similaires des autres langages pour exprimer les boucles et les branchements.

- La boucle do ... while s'exprime par : do instruction-ou-bloc while (condition).
- La boucle while s'exprime par : while (condition) instruction-ou-bloc
- La boucle for : for (initialisation ;condition ;incrément) instruction-ou-bloc.
- La boucle for ... In : for (variable in objet) instruction-ou-bloc
- Le test if ... else : if condition instruction-ou-bloc else instruction-ou-bloc.
- Le branchement multiple switch:

```
switch(i) {
case 1: // qqe chose
break
case 2: // autre chose
break
default: // encore autre chose
break
}
```

2. JavaScript : un langage orienté objet

a) Introduction

JavaScript offre toutes les facilités d'un langage procédural et d'un langage orienté objet à différence près que ce dernier est basé sur l'objet prototype. Intéressons nous maintenant à certains aspects de JavaScript en tant que langage orienté objet.



Rappel

Un objet est une encapsulation de propriétés et de méthodes. Les propriétés sont des boîtes contenant des valeurs primitives, des méthodes et/ou des objets. Les valeurs primitives peuvent être prises dans les types primitifs suivant : Undefined, Null, Boolean, Number ou String. Une méthode est simplement une fonction associée à un objet.

Nous venons de citer plus haut que JavaScript est un langage orienté objet dit à prototype. La particularité réside dans le fait que JavaScript ne supporte pas la notion de classe habituelle des langages orientés objet. Les objets ne sont pas des instances de classes, mais sont chacun équipés de constructeurs permettant de générer leurs propriétés, et notamment une propriété de prototypage qui permet d'en générer des objets héritiers personnalisés. En d'autres termes, en JavaScript il y a un représentant d'un objet qui sert à créer d'autres. C'est un objet vivant, contrairement à une classe qui est un « objet » abstrait.

b) Création d'objets en JavaScript

On peut créer de nouveaux objets en JavaScript soit en définissant un type d'objet en créant une fonction, soit en créant une instance en utilisant l'opérateur new.

Un exemple :



Exemple

```
function eleve(nom, prenom) {
    this.nom = nom;
    this.prenom = prenom;
}
P = new eleve ("Goldman", "Jean-Jacques");
```

Il est possible de rajouter des propriétés dynamiquement. Pour accéder aux propriétés d'un objet on utilise : `<objet>.<propriété>`. Pour définir une propriété, on peut lui associer une valeur (`eleve.nom = "Lutton"`). Enfin, une autre façon d'accéder/affecter une valeur à une propriété est de le voir comme un élément d'un tableau. P.ex. (`eleve["nom"] = "Lutton"`). Notons qu'une propriété peut être aussi un autre objet. Une méthode est une fonction associée à un objet : `<objet>.<méthode> = <fonction>`. Pour accéder aux propriétés il est possible de passer par l'opérateur **this** qui est utilisé pour référencer l'objet courant. Pour plus de facilité, un autre opérateur, **with**, est utilisé quand on veut exécuter plusieurs instructions avec l'objet.

```
with(document)
{
    open();
    write("Cette page a pour titre " + title);
    close();
}
```

c) L'objet prototype

Intéressons nous maintenant à l'objet prototype. Ce dernier, comme son nom l'indique, est le représentant d'un type d'objet au lieu d'une classe. Notons que prototype est une propriété présente à chaque objet javascript, et cette propriété est en effet l'objet prototype. Regardons ci-dessous cette propriété à travers d'un exemple.

Soit `r` un objet défini avec deux propriétés : `r.largeur` et `r.hauteur`. Pour calculer la surface du rectangle, la logique procédurale conduit à la fonction suivante :

```
function calculerAireduRectangle(r) {return
    r.largeur*r.hauteur;}
```

Une autre façon de faire, cette fois suivant la logique orienté objet, est de la déclarer comme une méthode de l'objet `Rectangle` qui a servi comme base de construction de l'objet `r`.

```
function Rectangle(l,h)
{
    this.largeur = l;
    this.hauteur = h;
    this.aire= function () { return r.largeur*r.hauteur; }
}
```

Or, aucune des solutions n'est optimale. Notons que la méthode `aire` est la même pour tout objet de type `Rectangle` (elle peut être vue comme une méthode abstraite) et il est inutile de lui associer une place mémoire chaque fois qu'un nouveau objet `Rectangle` se crée. Il suffit alors de la déclarer via la propriété prototype que tout objet possède. L'objet prototype est destiné à détenir les méthodes et les propriétés qui peuvent être partagées par toutes les instances.

Voyons comme cela fonctionne :

```
function Rectangle(l,h) {  
  this.largeur = l;  
  this.hauteur = h;  
}  
Rectangle.prototype.aire= function ()  
{  
  return r.largeur*r.hauteur;  
}
```

L'objet prototype est associé au constructeur et tout objet que celui-ci initialise hérite exactement du même ensemble de propriétés du prototype.

d) Les objets prédéfinis

JavaScript offre des objets prédéfinis comme Array, Boolean, Date, Math, Number, Object, String, etc. Pour chaque objet il y a des constructeurs, propriétés et des méthodes prédéfinies, p.ex. pour Array :

Constructeur : Array(n), Array(e11,e12,...,eln-1),

Propriétés : index, input, length..

Méthodes : concat, join, pop, push, sort, toString...

Notons que l'objet Math est un objet noninstanciable possédant des propriétés et des méthodes pour les opérations mathématiques.

Enfin, comme nous allons voir plus loin, les objets DOM occupent une place importante dans les objets manipulés par JavaScript. Il y a aussi quelques fonctions prédéfinies comme eval(), escape()/unescape(), qui jouent un rôle important.

B. JavaScript et le WEB

A ses débuts, JavaScript était utilisé pour des fins d'animation, effets graphiques... Depuis JavaScript est principalement utilisé dans les pages web interactives, facilitant ainsi la vie de l'utilisateur mais aussi apporter une valeur ajoutée au fournisseur de pages web. Voici quelques différentes utilisations de JavaScript:

- Contrôle de formulaires html;
- Outils de mesure d'audience;
- Gestionnaire des bandes publicitaires;
- Ajax: aide au remplissage automatique de formulaire.

La sécurité

Quand on parle de WEB on doit aussi réfléchir en termes de sécurité. JavaScript est un langage client dont les échanges avec le client sont fortement surveillés. Ainsi, JavaScript ne peut pas accéder aux fichiers stockés sur l'ordinateur de client. Le seul moyen d'échanger avec le client est par l'intermédiaire d'un formulaire. Par défaut, il est impossible d'accéder aux pages html rapatriés au client par d'autres domaines, mais peut être autorisé via des headers CORS. JavaScript n'autorise pas l'installation sur le site du client de programmes exécutables. Cela est par contre possible avec les composantes d'ActiveX ou les Applets.

Les inconvénients

Les limites de JavaScript résident surtout dans l'incompatibilité du code à l'égard des navigateurs et des plateformes. Enfin, le code JavaScript est non visible par les moteurs de référencement.

1. Intégration d'un script dans une page HTML

Il y a trois méthodes pour intégrer du code JavaScript dans un fichier HTML. Utilisation de la balise <script>;

```
<script language="JavaScript"> (in HTML5 javascript est le défaut)
alert("mon premier texte en JavaScript !!!!")
</script>
```

Inclusion d'un fichier externe ;

```
<script language="JavaScript" src="monfichier.JavaScript">
```

Directement dans les balises en associant du JavaScript aux événements;

```
<INPUT TYPE="button" VALUE="ce paragraphe contient une phrase
importante"
onClick='alert("Nous sommes tous égaux")' >
```

Cette dernière méthode est à éviter. En effet, cela conduit à un mélange des gendres qui rend difficile la lisibilité du code.

2. JavaScript et les cookies



Rappel

Les cookies sont utilisés pour remédier à la non-persistance des communications Internet. Il s'agit d'informations stockées au niveau du client et qui permet au serveur d'identifier le client lors de la prochaine connexion grâce au cookie qu'il rapporte dans son en-tête de la requête HTTP.

JavaScript malheureusement ne possède pas de méthodes natives pour gérer les cookies. Il faudra donc créer des fonctions pour le stockage et l'accès aux cookies. Voici un exemple (extrait de « JavaScript le guide complet », par O. Hondemarck, éditions MicroApplication 2009).

```
function setCookie(name, value, expires, path, domain,
secure) {
document.cookie=name+"="+escape(value)+
((expires==undefined) ? "" : (";
expires="+expires.toGMTString()))+
((path==undefined) ? "" : ("; path="+path))+
((domain==undefined) ? "" : ("; domain="+domain))+
((secure==true) ? "; secure" : "");
}
```

```
function getCookie(name) {
if (document.cookie.length==0) { return null; }
var regCookies=new RegExp("(; )","g");
var cookies=document.cookie.split(regCookies);
for (var i=0; i<cookies.length ; i++) {
var regInfo=new RegExp("=","g");
var infos=cookies[i].split(regInfo);
if (infos[0]==name) {
return unescape(infos[1]); }
}
return null;
}
```

Noter que le fichier cookie.js contient les deux fonctions ci-dessus.


```
<script type="text/javascript" src="cookie.js"></script>
<script type="text/javascript">
function setPrefLangue(pref) {
var dtExpiration1An=new Date();
dtExpiration1An.setTime(dtExpiration1An.getTime()
+365*24*3600*1000);
setCookie("langue", pref, dtExpiration1An);
window.location.href=document.location;
}
function getPrefLangue() {
var txtIntro="Choisissez votre langue :";
var laLangue=getCookie("langue");
if (laLangue=="fr") {
txtIntro="Choisissez votre langue :";
}
if (laLangue=="us") {
txtIntro="Choose your language :";
}
if (laLangue=="es") {
txtIntro="Cual es su idioma :";
}
document.write(txtIntro+" <a
href=\"javascript:setPrefLangue('fr')\">Français</a> - <a
href=\"javascript:setPrefLangue('us')\">Anglais</a> - <a
href=\"javascript:setPrefLangue('es')\">Espagnol</a>") }
</script>
<script type="text/javascript">
getPrefLangue();
</script>
```

Dans le code ci-dessus on trouve certaines fonctions couramment utilisées dans JavaScript comme `escape`, `unescape`, `split` et `regexp`. Les fonctions `escape` et `unescape` permettent d'encoder et décoder des chaînes de caractères. En particulier, la fonction `escape` renvoie l'encodage hexadécimal du paramètre ASCII. La fonction `unescape` renvoie la chaîne ASCII correspondant à la valeur d'encodage hexadécimale spécifiée. L'objet `RegExp` est un objet permettant de manipuler des expressions régulières, c'est-à-dire des modèles créés à l'aide de caractères ASCII permettant de manipuler des chaînes de caractères. Cela permet de trouver des portions de la chaîne correspondant au modèle, tandis que la fonction `split()` permet de scinder une chaîne de caractère et de retourner les résultats dans un tableau, grâce à une chaîne définie comme séparateur.

C. JavaScript : un langage orienté événements

Cette partie introduira la notion de DOM avant de décrire les fonctionnalités "événements" du langage JavaScript.

1. Qu'est ce que le DOM ?

Document Object Model (ou DOM) décrit une interface indépendante de tout langage de programmation et de toute plate-forme, permettant à des programmes informatiques et à des scripts d'accéder ou de mettre à jour le contenu, la structure ou le style de documents HTML ou XML. DOM est souvent identifié par une

arborescence de la structure d'un document et de ses éléments.

P.ex. chaque élément généré à partir du balisage comme, dans le cas de HTML, un paragraphe, un titre ou un bouton de formulaire, y forme un nœud.

DOM est utilisé pour pouvoir modifier facilement des documents XML ou accéder au contenu des pages web. A partir d'un arbre DOM donné, il est aussi possible de générer des documents dans le langage de balisage voulu. Donc, Document Object Model est un API pour les documents HTML et XML. Selon le W3C: "DOM permet aux programmes et scripts d'accéder et de modifier dynamiquement le contenu, la structure et le style de documents XML ou HTML".

En résumé, DOM est composé de :

- un ensemble d'objets représenté par un arbre,
- un modèle pour la façon dont ces objets peuvent être combinés,
- une interface pour y accéder et les manipuler.

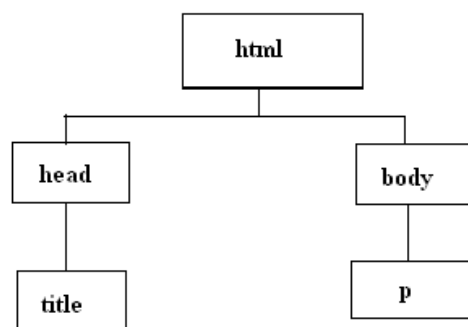
DOM fournit les interfaces fondamentales pour représenter tout document structuré HTML ou XML.

La représentation hiérarchique de DOM ressemble à un arbre de famille. On retrouve les parents (parent), les enfants (child), qui à leur tour sont parent de leur propres enfants... On peut également identifier les « first child », « last child » et « siblings ». Le type des nœuds peut être élément, texte ou attribut.

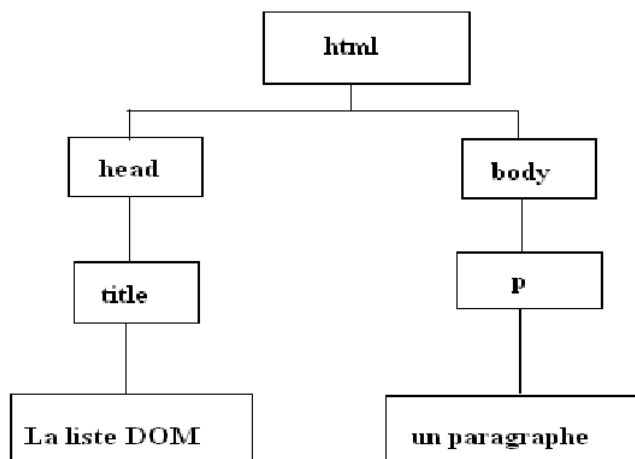
a) Le type des éléments

Les nœuds « Node » sont les éléments essentiels pour la représentation de l'arbre DOM. Les liens hiérarchiques entre les nœuds définissent la structure du document et sont représentés dans l'arbre DOM. L'étiquette associée au nœud donne le nom de l'élément, p.ex. pour <html> le nom est html, tandis que pour <p> il est p. Un nœud élément peut contenir d'autres nœuds éléments, ce qui permet de construire une arborescence. L'élément html est à la racine de l'arbre DOM. Voici un exemple simple et sa représentation par l'arbre DOM.

```
<html>
<head>
<title> La liste DOM </title>
</head>
<body>
<p> un paragraphe </p>
</body>
</html>
```



Les nœuds « Text ». Le texte contenu dans un élément apparaît comme un nœud texte dans l'arbre DOM. Ce nœud est toujours un « child » de l'élément html associé. Voici la représentation de l'arbre DOM intégrant les nœuds texte.

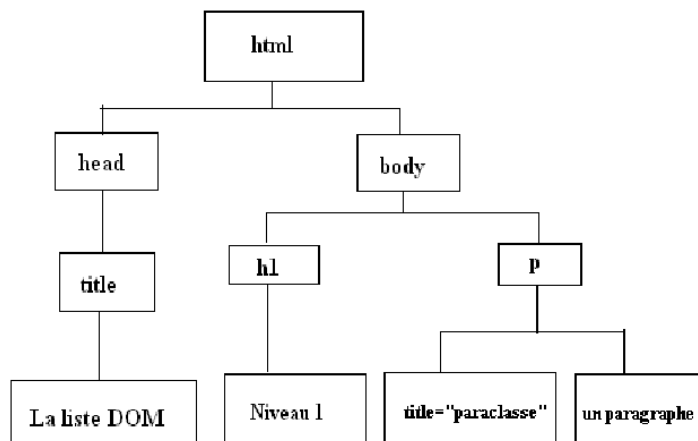


Les nœuds « attribut ». Les attributs d'un élément html sont représentés comme des nœuds attributs dans l'arbre DOM. Reprenons l'exemple ci-dessus :

```

<html>
<head>
<title> La liste DOM </title>
</head>
<body>
<h1> Niveau 1 </h1>
<p title ="paraclasses"> un paragraphe </p>
</body>
</html>
    
```

Ce qui donnerait :



b) Les objets de DOM

Les principaux objets de DOM :

L'objet **Window** (Fenêtre) est l'objet le plus élevé dans la famille des objets pour tout ce qui est affiché dans la fenêtre du navigateur. Par l'intermédiaire de l'objet Window on peut questionner et contrôler la fenêtre du document. De plus, on peut définir de nouvelles fenêtres et en spécifier librement les propriétés. Parmi les

éléments de Window on peut citer Location (c'est un objet coté-client, donc créé par le navigateur), Navigator, Document et History.

L'interface **Node** est définie dans la spécification DOM 2, elle permet d'accéder à la structure d'un document HTML vu comme une arborescence d'élément, et de modifier cette structure. L'élément racine est Document qui représente l'arbre de la page web. Ensuite, on pourra accéder et modifier tout élément dans l'arbre grâce aux méthodes associées à l'objet Document.

L'interface **NodeList** est un objet DOM 2, qui permet d'accéder aux éléments d'une page Web ou d'un fichier XML. C'est un élément dynamique, tous les changements de structure dans la page modifient le contenu de NodeList.

Considérons maintenant de façon plus détaillée l'objet Window :

Window est un objet qui correspond à la fenêtre dans laquelle s'affiche une page Web. Une telle fenêtre peut être créée dynamiquement.

Attributs de window :

frames[]. Les frames dans la fenêtre. Lecture seule.

length : Nombre de frames. Lecture seule.

name : Nom de la fenêtre.

status : Texte de la barre d'état.

parent : La fenêtre parent d'une fenêtre donnée

Parmi les objets contenus dans Window, on citera :

Document : Désigne une page, celle que contient une fenêtre.

History : Liste des pages précédemment vues dans la même fenêtre.

Location : dDésigne l'URL d'une page, celle que contient la fenêtre.

Navigator : désigne le type du navigateur du client

Screen : désigne l'écran et contient les propriétés: width, height, availWidth, availHeight, colorDepth.

Méthodes de window

open(url, nom [, liste de propriétés]) ouvre une nouvelle fenêtre. L'URL et le nom sont donnés en paramètres ainsi que la liste des options.

close() ferme la fenêtre. Exemple: x.close(); window.close();

alert(message) affiche un message dans une fenêtre pop-up.

Document, un sous-objet de Windows, est l'objet clé de DOM

Parmi les méthodes les plus utilisées citons : close(), getElementById(), getElementByName(), open(), write().

Parmi les attributs on citera en particulier cookie, qui donne les cookies associés au document.

Parmi les attributs DOM les plus utilisés on citera :

attributes, childNodes, documentElement, document.body, firstChild, innerHTML, lastChild, length, nextSibling, nodeName, nodeType, nodeValue, parentNode, previousSibling, tagName...

Parmi les méthodes DOM les plus utilisées on citera :

appendChild, createElement, createTextNode, getAttribute, getAttributeNode, getElementById, getElementsByTagName, removeAttribute, removeChild, replaceChild, setAttribute, setAttributeNode...

Pour plus d'information sur l'API DOM, les interfaces, les objets et les

méthodes proposés, voir *le site de W3C².*



Remarque

DOM est un outil permettant l'accès aux documents HTML et XML. Il permet au développeur de fournir une représentation structurée du document et de codifier la manière dont un script peut accéder à cette structure. Il s'agit donc essentiellement d'un moyen de lier une page Web, par exemple, à un langage de programmation ou de script.

Pour faire du DHTML, rappelons que la représentation interne d'une page web est déterminée par le document HTML et les informations de style CSS. Les modifications peuvent être effectuées via JavaScript, qui accède à la représentation interne à travers l'interface de programmation Document Object Model (DOM).



Attention

Les propriétés CSS du DOM ne correspondent pas nécessairement aux propriétés conventionnelles "CSS ". Ainsi pour accéder à la propriété font-style, il faut l'écrire fontStyle tandis que pour les propriétés ne contenant pas de tiret, on utilise la même notation.

2. Un langage orienté événement

JavaScript interagit avec des applications/pages web grâce à des fonctions prédéfinies (gestionnaire d'événement - event handler) associées à certains événements, (un événement est une action engendrée par l'utilisateur). La syntaxe d'appel de ces fonctions à l'intérieur d'une page web est comme suit :

```
Syntaxe <balise GestionnaireEvénement = "code/fonction  
JavaScript">
```

où GestionnaireEvénement est le nom de la fonction prédéfinie associée à l'événement, entre les cotes on met ou bien une fonction JavaScript déclarée préalablement ou du code JavaScript à exécuter lors que l'événement se produira. Le nom d'un gestionnaire d'événements commence par « on » et continue avec le nom de l'événement, i.e., onSubmit, onClick, etc.

2 - [HTTP://www.w3.org/DOM/](http://www.w3.org/DOM/)



Exemple : Exemple 1

```
<input type= "button" value "Cliquer ici"
onClick = "alert ('Merci !');">
<form name "f1" >
<input name= "b1" type= "button" value "Cliquer ici" >
</form>
```

```
alternativement :
Document.f1.b1.onClick = function(){alert ('Merci !' ); };
```

Rappelons que le navigateur web charge une page web comme une arborescence DOM. La manipulation des événements est alors effectuée à travers des fonctions JavaScript. Un des éléments principaux de cette manipulation est l'identification des éléments html via l'attribut « id » ou « name ». Une fois l'élément identifié, JavaScript permet de gérer dynamiquement les nœuds élément ou texte via les attributs ou méthodes DOM. Voici un exemple qui permet de remplacer un nœud de type élément par un nouveau.



Exemple : Exemple 2

```
<html>
<head>
<title> Liste DOM </title>
</head>
<body>
<div id="container">
<h1 name="hclasse" id="hid"> Niveau 1 </h1>
<p title="paraclasse" id="pid"> C'est un paragraphe </p>
<h3 name="hclasse"> Fin de la page </h3>
<script language = "javascript" type="text/javascript">
var parentID = document.getElementById("container") ;
document.write("afficher les noeuds de type element et
texte" + "<br>");
var newP = document.createElement("p") ;
parentID.appendChild(newP) ;
var pTag = document.getElementsByTagName("p") ;
document.write(pTag[1].nodeName + "<br>") ;
var newT = document.createTextNode("Bonjour") ;
newP.appendChild(newT) ;
document.write(pTag[1].firstChild.nodeValue + "<br>") ;
var oldH = document.getElementById("hid");
parentID.replaceChild(newP,oldH);
</script>
</body>
</html>
```

a) L'essentiel des événements traités par JavaScript

événements page et fenêtre

onabort - s'il y a une interruption dans le chargement

onerror - en cas d'erreur pendant le chargement de la page

onload - après la fin du chargement de la page

D'autres événements : onbeforeunload, onunload, onresize.

événements souris

onClick - sur un simple clic

ondblclick - sur un double clic

onmousedown - quand le bouton de la souris est enfoncé, sans le relâcher

D'autres événements : onmousemove, onmouseout, onmouseover, onmouseup.

événements clavier

onkeydown - lorsqu'une touche est enfoncée

onkeypress - lorsqu'une touche est pressée et relâchée

onkeyup - lorsqu'une touche est relâchée

événements formulaire

onsubmit - quand le formulaire est validé (via un bouton bouton de type "submit" ou une fonction submit())

onreset - lors de la remise à zéro du formulaire (via un bouton "reset" ou une fonction reset())

D'autres événements : onselect, onblur, onchange, onfocus.

b) Un exemple complet

Quand on parle d'associer du JavaScript aux événements, on pense tout de suite au contrôle de formulaires. Voici un exemple où on utilise des fonctions JavaScript pour contrôler comment les champs sont remplis :

- Tester si la civilité est renseignée,
- Tester le champ E-mail,
- Tester si dans les champs (code postal, téléphone, fax) est de valeur entière,
- Tester la date...
- Tester l'acceptation des conditions générales

Pour obtenir une application complète, il faut créer une nouvelle fonction testSaisie() conenant l'ensemble des fonctions de contrôle citées plus haut et l'associer à onSubmit(). Donc, cette fonction appellera toutes les autres fonctions JS définies pour traiter les champs des formulaires.

```
<BODY>
<FORM name="form1" onSubmit="return testSaisie()">
<P>Veuillez remplir les champs suivants :</P>
<P>Civilité* :
<INPUT type="radio" name="civ" value="Mr">Mr
<INPUT type="radio" name="civ" value="Mme">Mme
<INPUT type="radio" name="civ" value="Mlle">Mlle<BR>
...
<P>(*) Champs obligatoires</P>
<P>
<INPUT type="submit" name="Submit" value="Envoyer">
<INPUT type="reset" name="Submit2" value="Rétablir">
<BR>
</P>
</FORM>
```

function testNumerique

```
function testNumerique(valeur)
```

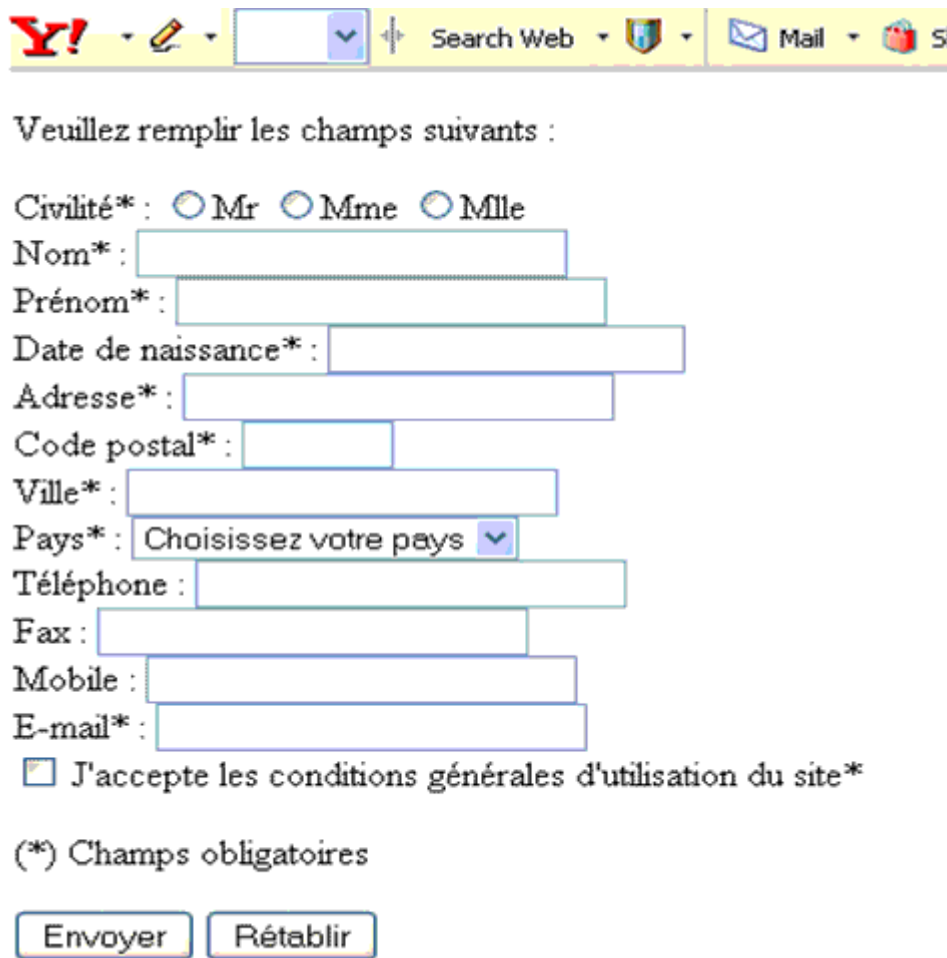
```
{
  if (valeur == parseFloat(valeur)) return true;
  else return false;
}
```

function testMail

```
function testMail(email)
{
  var posArobase;
  posArobase = email.indexOf("@");
  if (posArobase == -1) return false;
  var posPoint;
  posPoint = email.lastIndexOf(".");
  if ((posPoint == -1) || (posPoint < posArobase)) return
  false;
  return true;
}
```

function testRadio

```
function testRadio(nomForm,nomGroupe)
{
  var compteur;
  compteur = 0;
  while (compteur < nomForm.elements[nomGroupe].length)
  {
    if (nomForm.elements[nomGroupe][compteur].checked)
      return true;
    compteur++;
  }
  return false;
}
```

Y! Search Web Mail SI

Veuillez remplir les champs suivants :

Civilité* : ☐ Mr ☐ Mme ☐ Mlle

Nom* :

Prénom* :

Date de naissance* :

Adresse* :

Code postal* :

Ville* :

Pays* : Choisissez votre pays

Téléphone :

Fax :

Mobile :

E-mail* :

☐ J'accepte les conditions générales d'utilisation du site*

(*) Champs obligatoires

Formulaire HTM

c) Gestion des événements clavier



Exemple

```
<form name="leformulaire">
Code postal : <input type="text" name="codepostal"><br>
</form>
<script type="text/javascript">
document.forms["leformulaire"].elements["codepostal"].onkey
press = traiterCodepostal
function traiterCodepostal(e) {
var txtCarOk="0123456789";
var car="";
var isCarOk=false;
car=String.fromCharCode(e.charCode);
if (txtCarOk.indexOf(car)>=0) {
isCarOk=true;
}
if ((e.charCode==0)&&(e.keyCode>0)) {
return true;
}
if (isCarOk) {
if
```

```
(document.forms["leformulaire"].elements["codepostal"].value.length<5) {
document.forms["leformulaire"].elements["codepostal"].value+=car;
}
}
return false;
}
</script>
```

d) JavaScript et CSS

JavaScript permet de modifier les styles CSS: p.ex. changement de couleurs d'un élément. On peut accéder à un élément via `document.getElementById` et modifier la classe d'un élément. On peut accéder à une classe via l'attribut `className` de tout élément html. Tout élément CSS est accessible via JavaScript et DOM selon la règle : le nom ne change pas si absence de tiret. Sinon supprimer le tiret et mettre en majuscule la première lettre qui le suit. Par exemple, pour appeler l'attribut `font-size` il faudra écrire `fontSize`.

On peut aussi manipuler les feuilles de style elles-mêmes: Il s'agit d'activer et désactiver des feuilles de style (utiliser la propriété `disabled` pour les éléments `link` et `style`).

3. HTML Dynamique (DHTML)

DHTML est un nom générique donné à l'ensemble des techniques utilisées par l'auteur d'une page web pour que celle-ci soit capable de se modifier elle-même en cours de consultation dans le navigateur web. Ces technologies sont le HTML, nécessaire pour présenter le document, les feuilles de style (CSS), permettant de définir un style pour plusieurs objets et le positionnement de ceux-ci sur la page, le modèle objet de document (DOM), proposant une hiérarchie d'objets, afin de faciliter leur manipulation, et enfin le JavaScript, langage de script essentiel pour définir des événements utilisateur.

DHTML est essentiellement destiné à animer des éléments HTML, (animation = modification des propriétés (position, hauteur, largeur, visibilité, couleur ...) ou en utilisant leur méthodes (fonctions associées à un élément). Cela ne peut se faire qu'à l'aide d'un code JavaScript permettant de modifier les propriétés des éléments suite à des événements utilisateurs (clic sur la souris, déplacement de la souris, ...), combiné avec une structuration des éléments dans la page définie par le DOM (Document Object Model)

D. Ajax

1. Motivation

a) Présentation générale

Ajax est l'utilisation conjointe de plusieurs technologies comme XHTML, CSS, DOM et XML toutes liées par JavaScript qui fait appel au mécanisme XMLHttpRequest. Ajax permet l'échange d'informations entre le navigateur et le serveur web sans recharger la totalité de la page. Il permet ainsi de garder des pages web affichées et avoir des performances accrues. Principe de fonctionnement : l'exécution du code

JavaScript continue en parallèle du traitement de la requête Ajax (asynchrone). Pratiquement, une requête est envoyée au serveur (grâce à XMLHttpRequest). Cette requête est traitée par le serveur qui envoie une réponse. Enfin, le JavaScript, et non le navigateur, réceptionnera et traitera la réponse. Un élément important dans les requêtes AJAX est le choix du mode, synchrone ou asynchrone. La différence entre les deux modes se situe dans le fait d'autoriser ou non la poursuite du travail avec la page web pendant que le serveur traite la demande envoyée par l'objet XMLHttpRequest. Voyons cela à travers d'un exemple. Examinons la gestion d'un panier de commande sur un site de commerce électronique.

b) Mode d'exécution

En mode synchrone l'internaute ajoute un article dans le panier.

Une requête est envoyée au serveur ;

- Le navigateur attend la réponse du serveur ;
- Le serveur valide l'ajout ;
- Le serveur renvoi le contenu du nouveau panier ;
- Le navigateur rafraîchit la zone d'affichage du panier ;
- L'internaute continue ses "achats".

En mode asynchrone l'internaute ajoute un article dans le panier.

- Une requête est envoyée au serveur ;
- Le navigateur continue son exécution ;
- L'internaute continue ses "achats" ;
- Le serveur valide l'ajout ;
- Le serveur rappelle le navigateur et communique le contenu du nouveau panier ;
- Le navigateur rafraîchit la zone d'affichage du panier.

L'intérêt principal des requêtes asynchrones réside dans le fait que l'utilisateur peut continuer le travail sur un formulaire ou autre, tandis que le serveur se charge d'envoyer des informations complémentaires au client.

2. Mode de fonctionnement

a) Fonctionnement

Comment fonctionne AJAX ? Il s'agit d'une requête HTTP qui est traitée de façon transparente grâce à l'objet XMLHttpRequest. Cet objet possède des propriétés et des méthodes qui permettent de faciliter l'envoi, le traitement et la réception de la requête, tout en permettant de continuer le travail si cela est jugé possible par l'utilisateur. Les propriétés permettent de suivre l'état de la requête et de lui associer des actions spécifiques, tandis que les méthodes gèrent l'envoi et la réception de la requête et éventuellement des données associées à la requête.

Les propriétés

onreadystatechange : (on lui affecte une fonction avec les instructions de traitement de la réponse);

readyState : l'état d'avancement de la requête de 0 (non initialisée) à 4 (complétée).

responseText / responseXML : la réponse du serveur à la requête en texte (i.e. objet reconnu par JavaScript, on préfère alors le format JSON) ou formatée en un objet XML.

Status : état de la réponse HTTP du serveur. (p.ex. 200 OK).

Les méthodes

`open(method, url, bool)` : (si mode asynchrone bool vaut true et sinon false)

send(data) : Envoie la requête HTTP au serveur (quand il n'y a pas de données la valeur est null)



Remarque: Note

En fonction de la méthode d'envoi utilisée, (GET ou POST), il existe certaines subtilités. Ainsi, pour GET, l'argument url cité dans la méthode open, doit inclure les données à envoyer au serveur. Typiquement pour envoyer des données venant d'un formulaire, il faudra déclarer une variable pour récupérer la valeur du champ cible du formulaire et ensuite utiliser la fonction javascript escape pour l'encodage :

```
var var1 = document.getElementById("formulaire-nom").value;
...
var url = "http://adresse-web-serveur.nom-procedure?var1="
+ escape(var1);
```

Pour la méthode POST, il faudra utiliser la propriété `setRequestHeader` de l'objet `XMLHttpRequest` :

```
req.setRequestHeader( 'Content-Type', 'application/x-www-  
form-urlencoded')
```

Ensuite, préparer la donnée (même démarche que pour url ci-dessus) à envoyer au serveur :

```
var data= "id=" + escape (login) +"&password=" +
escape(password);
req.send(data);
```

b) Lancement d'une requête (asynchrone) HTTP



Exemple

```
if (window.XMLHttpRequest) {
    xhr_obj = new XMLHttpRequest();
} else if (window.ActiveXObject) {
    xhr_obj = new ActiveXObject("Microsoft.XMLHTTP");
}
...
xhr_obj.onreadystatechange = function() {
    // instructions de traitement de la réponse };
...
xhr_obj.open('GET', 'HTTP://url...', true);
xhr_obj.send(null);
```



Exemple : Gestion de la réponse du serveur

```
if (xhr_obj.readyState == 4) { // réponse reçue
}
else { // pas encore prête
}
...
if (xhr_obj.status == 200) { // OK !
```

```
}
else { // il y a eu un problème avec la requête HTTP
}
```

c) Format d'échange avec le serveur

Le navigateur reçoit la réponse du serveur et traite la réponse. Pour faciliter le traitement de la réponse, il existe deux modes d'échanges, soit en format XML et dans ce cas on utilise l'attribut `responseXML`, soit en format texte reconnu par JavaScriptON (JavaScript Object Notation, JSON), et on fait appel à l'attribut `responseText`.

JSON est un format récursif compatible avec JavaScript structuré comme un objet JavaScript et sauvegardé dans un fichier. Voici un exemple avec les deux formats pour l'envoi des mêmes informations.

Format XML	Format JavaScriptON
<pre><?xml version="1.0" ?> <root> <menu>Fichier</menu> <commands> <item> <title>Nouveau</value> <action>CreateDoc</action></item> <item> <title>Ouvrir</value> <action>OpenDoc</action> </item> <item> <title>Fermer</value> <action>CloseDoc</action> </item> </commands> </root></pre>	<pre>{ "menu": "Fichier", "commandes": [{ "title": "Nouveau", "action": "CreateDoc" }, { "title": "Ouvrir", "action": "OpenDoc" }, { "title": "Fermer", "action": "CloseDoc" }] }</pre>

Pour utiliser un fichier JavaScriptON, il suffit de charger le fichier en tant que texte, (méthode `XMLHttpRequest.responseText`) et utiliser la fonction `JSON.parse`.

d) Outils complémentaires

Plusieurs outils complémentaires à JavaScript ont été développées. Parmi les plus couramment utilisés on trouve JQuery, une bibliothèque JavaScript libre qui porte sur l'interaction entre JavaScript et Ajax et HTML. JQuery offre, parmi d'autres, des fonctions facilitant la création et le traitement de menus, formulaires de contact, utilisation d'Ajax...

Un autre outils très utilisé est AngularJS, qui est un framework libre et open-source. Il permet de simplifier la syntaxe javascript, et de combler les faiblesses de javascript en lui ajoutant de nouvelles fonctionnalités. AngularJS utilise la bibliothèque open source jquery. Si jquery n'est pas présent dans le chemin du script, AngularJS reprend sa propre implémentation de jquery lite. Si jquery est présent dans le chemin, AngularJS l'utilise pour manipuler le DOM2.

```
AJAX via jQuery
//code javascript
```

```
$.ajax({
  type: "GET",
  url: "test.htm",
  error:function(msg){
    alert( "Error !: " + msg );
  },
  success:function(data){
    //affiche le contenu du fichier dans le conteneur dédié
    $('#contenu_fichier_ajax').text(data);
  });
});
```

Programmation web côté serveur



Les scripts CGI	71
Introduction au PHP	72
PHP, les fonctionnalités web	78
PHP un langage objet	84

Nous allons étudier dans ce chapitre quelques éléments clés de la programmation web serveur. En particulier on s'intersectera aux CGI et au langage PHP.

A. Les scripts CGI

Un script CGI est un programme exécuté par le serveur web permettant d'envoyer au navigateur de l'internaute un code HTML créé automatiquement par le serveur. Il s'agit en fait d'un programme exécuté coté serveur qui traite des demandes spécifiques de l'utilisateur et des paramètres et fournit en retour des pages html au serveur qui les retourne à l'utilisateur. Donc c'est un interface/protocole d'échange, généralement, entre un formulaire et un serveur. L'application la plus fréquente de cette technique repose sur l'utilisation de formulaires HTML permettant à l'utilisateur de choisir ou de saisir des données, puis de cliquer sur un bouton de soumission du formulaire, envoyant alors les données du formulaire en paramètre du script CGI.

1. Principes de fonctionnement

Comment ça marche techniquement ? Les requêtes CGI se transmettent au serveur via les méthodes GET et POST d'un formulaire. Le serveur localise le CGI (précisé dans le formulaire, p.ex. `<form method=get action="cgi/nomFunction.pl" name="nomForm">`) et lui fournit les données en tant que variables d'environnement. Ainsi, pour une méthode GET, on récupérera QUERY_STRING, tandis que pour la méthode POST on lit les données dans l'entrée standard STDIN (pour cela on utilise la variable d'environnement CONTENT_LENGTH).

2. Avantages et inconvénients des CGI

L'intérêt des GCI consiste surtout dans l'utilisation de tous les langages, donc possibilités de traitement complexes. Cela permet une solution opérationnelle dans une grande majorité des cas. Cette solution est généralement très rapide puisque il

s'agit de langage compilé. Parmi les inconvénients on citera qu'il est peu portable et qu'il varie d'une plateforme à l'autre. Un autre inconvénient est qu'il consomme beaucoup de ressources système puisque un processus est créé pour chaque requête HTTP. Enfin, les CGI peuvent constituer des failles de sécurité. Ils peuvent laisser filtrer des informations sur le serveur. Ainsi, il est possible de faire exécuter des commandes sur le serveur en passant certains paramètres par les champs des formulaires. A cause de ces problèmes et de la peu de lisibilité du code CGI, les webmasters sont réticents d'accepter leur installation sur le serveur. Malgré l'apparition de FastCGI (qui remédie en partie aux défauts de CGI), cette solution est de plus en plus abandonnée au profit de solutions comme PHP ou JSP et servlets.

B. Introduction au PHP

La version actuelle de PHP est la 5.6.6 (19 février 2015). Le PHP est utilisé couramment avec Microsoft IIS ou Apache HTTPD comme serveur HTTP, MySQL, PostgreSQL, Oracle ou MS SQL serveur comme SGBDR, Windows, Linux ou Mac OS X comme système d'exploitation. Le quadruple Linux, Apache, MySQL et PHP (connu sous le nom de LAMP) constitue la majorité des plateformes en production. PHP est un langage compilé à la volé, qui fonctionne grâce à un module rajouté au serveur web téléchargeable sur [//fr.php.net](http://fr.php.net).

1. Packages et Framework PHP

Il existe plusieurs produits « **packages** » disponibles gratuitement sur Internet, qui installent PHP, Apache et d'autres éléments. Parmi les plus utilisés citons Easy PHP, WAMP et TSW (The Saint Wamp).

- EasyPHP installe et configure automatiquement Apache, PHP et MySQL. C'est le premier package complet, même si ce n'est plus à nos jours le produit de référence.
- WAMP est une alternative française constituée de Apache, PHP et Mysql. Il permet l'installation de deux environnements PHP (version 4 et version 5). Il donne la possibilité de gérer des scripts PERL. Il est aussi possible de rajouter le framework ZEND Optimizer pour accélérer les performances du serveur.
- TSW est un serveur modulaire ultracomplet constitué de Apache, Mysql, PHP (version 5), Perl, Python, Zend Optimizer, phpMyAdmin, phpConf (outil de configuration PHP), OpenSSL, FileZilla (serveur FTP), Hamster (serveur de mail), etc.

Les frameworks PHP

Une solution pour faciliter le codage PHP (et tout autre langage) est d'adopter des composants standards. Il s'agit alors de faire appel à un framework. Parmi les frameworks PHP les plus connus citons

Cake : [HTTP://cakephp.org/](http://cakephp.org/),

PEAR : [HTTP://pear.php.net/](http://pear.php.net/),

ZEND : [HTTP://framework.zend.com/](http://framework.zend.com/),

Symphony : [HTTP://www.symfony-project.org/](http://www.symfony-project.org/),

Ez Components : [HTTP://ezcomponents.org/](http://ezcomponents.org/), etc.



Rappel

Un framework est un ensemble d'un langage de programmation, d'une bibliothèque de fonctions, d'outils externes, de bonnes pratiques à suivre destinées à faciliter la programmation et optimiser l'exécution du code.

2. Principes de fonctionnement

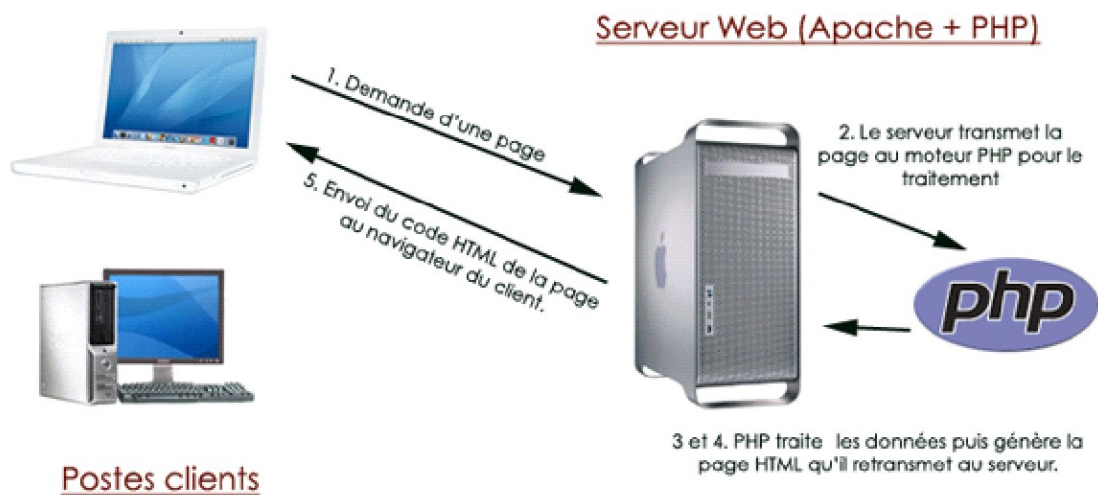
PHP est un langage de script qui peut être utilisé de diverses manières :

- Pour une interface Web, via le module intégré au serveur, ou bien via une interface de type FastCGI ou CGI. C'est l'utilisation la plus courante.
- En ligne de commandes "CLI",
- Pour produire une interface desktop "GUI".

Il y a également divers modes d'exécution :

- Mode interprété (compilé à la volée) : c'est l'utilisation la plus courante.
- Mode pré compilé (nécessite une extension, souvent payante).
- Mode compilé (nécessite certaines manipulations, non adapté au Web).

Dans la suite nous nous concentrons surtout sur l'utilisation de PHP dans un environnement WEB. Les principes de fonctionnement dans un tel environnement sont résumés dans la figure "fonctionnement de PHP".



Fonctionnement de PHP

Lorsqu'un visiteur d'un site web réclame l'affichage d'une page dynamique écrite en PHP, voici les étapes principales qui se déroulent :

- le serveur web voit d'après le type MIME du fichier demandé que c'est au PHP de le traiter et lui passe la main.
- le PHP charge le script depuis le système de fichiers.
- le PHP compile le script chargé.
- le PHP exécute le script compilé.
- le PHP rend la main au serveur web.
- le serveur complète et envoie la page au client.

Voici quelques éléments clés à retenir :

- Le code PHP est placé directement dans le code source HTML, encadré par des balises spécifiques.
- Un fichier source php peut contenir une alternance de parties "html" et de parties "php".
- La syntaxe de PHP est très proche du C.

Après traitement par le serveur, la source HTML envoyée au navigateur (par le serveur) ne contient aucune trace du code PHP.

3. Intégration du code PHP dans les pages html

Il existe différentes façons (balises) pour permettre au serveur HTTP de passer la main au module PHP :

- `<?php insérer ici le code PHP ?>` (ce sont les balises à préférer)
- `<script language = "php">`
- `<% insérer ici le code PHP %>`

Pour les commentaires, il faut utiliser « `//` » ou « `/* */` ».

Ne pas oublier de séparer les instructions PHP par « `;` ».



Exemple

```
<?php echo 'Hello World !'; ?>
```

Ce code permet d'afficher sur la sortie standard de l'appelant.



Exemple

Les deux codes ci-dessous donne une page simple html qui affiche « Hello World », (trouver la différence)

<pre><html> <head> <title>Hello World en PHP</title> </head> <body> <p> <?php echo 'Hello World !'; ?> </p> </body> </html></pre>	<pre><html> <head> <title>Hello World en PHP</title> </head> <body> <?php echo '<p>Hello world !</p>'; ?> </body> </html></pre>
---	---

4. Syntaxe du langage PHP

Les variables et les types de données.

Les variables PHP sont définies en les précédant par le signe dollar « `$` » suivies d'une lettre puis d'une suite de lettres, chiffres et traits soulignés (`_`).

On distingue quatre types de données simples

- Entiers (integer), en type php `int`.
- Booléennes, en type php `bool`. Elle prend les valeurs `true` et `false`.
- Nombres flottants : connus comme réels, floats, ou doubles, type php `float`.
 - Exemple. `$a = 1.234` (non `1,234`).
- Les chaînes, type php `string`. Les chaînes, type php `string`.
`$machaine = 'bonjour tous le monde';` ou `$machaine = "bonjour tous le monde";`
- Les fonctions utilisées pour l'affichage sont `echo`, `printf`. La concaténation se fait par « `.` ». Le formatage se fait par (`strtolower()`, `strtoupper()`), la comparaison par (`strcmp()`), la longueur de la chaîne est donnée par `strlen()`, des recherches dans la chaîne se font par `strpos()`, `stripos()`, le découpage de chaînes avec `trim()`, la substitution par `str_replace()`, etc



Attention

Avec double guillemets il est possible au PHP de comprendre aussi des chaînes spéciales \n, \r... et de lire les variables (précédés de \$).

Types de données composés

Les tableaux : Le type php pour les tableaux est array. On distingue les tableaux indexés numériquement et les tableaux associatifs.

```
- Déclaration et initialisation :
<?php
$fruits = array();
$legumes = array('carotte','poivron','aubergine','chou');
$identite = array(
    'nom' => 'Hamon',
    'prenom' => 'Hugo',
    'age' => 19,
    'estEtudiant' => true
);
?>
```

```
- Ajouts d'éléments :
<?php
$legumes[] = 'salade';
$identite['taille'] = 180;
?>
```

```
- Accès aux éléments :
echo $legumes[2];
echo 'Nom : ', $identite['nom'] ;
```

Voici quelques fonctions liées aux tableaux : comptage du nombre d'éléments count(), parcours d'un tableau reset(), current(), next(), prev(), each(), trier un tableau asort(), arsort()...uasort().

Notons que les deux types de tableaux sont dans les faits la même chose. Dans le premier tableau nous avons :

'0' => 'carottes', '1' => 'poivron'...

Deux types de données spéciaux

Les ressources : le type php est resource. Ce type concerne les fichiers, bases de données, connexions... etc.

Parmi les fonctions de manipulations de fichiers on trouvera fopen(), feof(), fgets(), fread(), fwrite(), fputs(), file_exists(), copy(), rename(), opendir(), readdir(), mkdir(), rmdir().

La valeur NULL, est connue comme un type php null. Elle représente l'absence de valeur.

Un autre type de donnée spécial est l'objet (object) utilisé en Programmation Orientée Objet. Nous y reviendrons plus tard.



Remarque

Noter que PHP est un langage à types dynamiques, en d'autres termes il est possible de faire changer de type à une variable selon les besoins

PHP propose tous les opérateurs classiques et standards, mathématiques et logiques (similaire à C).

Les constantes :

L'utilisation des constantes est similaire à des variables. La différence est qu'il ne faut pas utiliser le signe dollar (\$) car il s'applique uniquement aux variables. Une fois que la valeur d'une constante est définie, elle ne peut plus être modifiée jusqu'à la fin du script. La déclaration se fait par la commande `define (nom_constante, valeur)`.

Exemple.

```
define('PI', 3.14159265);  
echo (PI) ;
```

Il existe des constantes prédéfinies, dites magiques. Voici certains :

- `__LINE__` : La ligne de code en cours ;
- `__FILE__` : Le nom complet du script en cours ;
- `__FUNCTION__` : La fonction en cours ;
- `__CLASS__` : La classe en cours.

Les instructions :

Les instructions conditionnelles :

- `($a > $b) ? $a : $b;`
- `If (expression) instruction1 else instruction2;`
- `elseif`, imbriqué...
- `switch... case cas1: instruction... default : instructions.`

Les boucles :

- `while`, `do ... while`, `for`, `foreach`;
- les sorties de boucles : `break`, `continue`, `return`...

Les fonctions :

Voici un exemple simple de définition d'une fonction :

```
function($arg1, $arg2, ..)  
{  
    $valeur_retour = $arg1+$arg2;  
    return $valeur_retour ;  
}
```



Attention

Si on veut passer les arguments par leur adresse il faut les précéder de `&` (sauf pour les objets).

La visibilité d'une variable dépend de sa première utilisation. Une variable n'est visible que dans la fonction dans laquelle elle a été définie. Il est possible d'utiliser "global" pour éviter cela. Il existe certaines fonctions qui peuvent être utiles lors de la manipulation des fonctions :

- `function_exists()` : permet de vérifier que la fonction existe.
- `get_defined_functions()` : donne la liste des fonctions définies.
- `func_num_args()` : donne le nombre de paramètres transmis à la fonction courante.
- `func_get_arg()` : récupère un paramètre de la fonction courante.
- `func_get_args()` : récupère tous les paramètres de la fonction courante.

D'autres fonctions utiles sont :

- les fonctions `echo()` ou `print()`, utilisées pour l'affichage.
- la fonction `var_dump()` affiche le type d'une variable et son contenu (ainsi que sa taille si c'est une chaîne). Elle s'applique aussi bien aux variables scalaires qu'aux objets, tableaux, ressources.
- exemple. `var_dump(1.2);` .. donne pour résultat float (1.2).
- la fonction `gettype()`, qui permet de déterminer le type d'une valeur, ou `get_resource_type()` utilisée en cas de type ressource.

Enfin, quelques mots sur les variables superglobales. Ces variables sont prédéfinies et accessibles de partout. Voici les plus utilisées :

- `$_GET` : Les paramètres provenant de la requête HTTP, méthode GET ;
- `$_POST` : Les valeurs envoyées par un formulaire, méthode POST ;
- `$_FILE` : Les fichiers envoyés par un formulaire ;
- `$_SESSION` : Les valeurs correspondant à des sessions ;
- `$_COOKIE` : Les valeurs transmises au moyen de cookies par le navigateur ;

Nous y reviendrons dans la suite à chacune des variables superglobales citées ci-dessus.

C. PHP, les fonctionnalités web

PHP facilite énormément la génération des pages html dynamique. Ainsi, il est possible d'automatiser la création d'un formulaire html avec des informations récupérées à temps réel dans une base de données (ou même un fichier texte), grâce à l'utilisation des boucles, tableaux, etc.

1. Traitement des formulaires

Les variables superglobales citées plus haut procurent une grande aide dans le traitement des formulaires. Ainsi, toutes les données envoyées par un formulaire sont automatiquement disponibles au script PHP spécifié dans le traitement du formulaire grâce au tableaux superglobaux `$_POST[]` ou `$_GET[]`. Voici un exemple simple.

```
<html><body>
<form method="post" action="verif.php">
Nom : <input type="text" name="nom" size="12"><br>
Prénom : <input type="text" name="prenom" size="12">
<input type="submit" value="OK">
</form></body></html>
```

Code pour le traitement PHP

Formulaire

```
<?php
$prenom = $_POST['prenom'];
$nom = $_POST['nom'];
print("<center>Bonjour $prenom $nom</center>");
?>
```

Traitement PHP

Résultat :

Bonjour James Bond

2. Chargement de fichier

Il est très facile en PHP de charger les fichiers sélectionnés dans un formulaire. Un tableau associatif superglobal (\$_FILES[]) permet de récupérer les fichiers placés dans un formulaire, champ (input) de type (file), méthode POST.



Attention

Il s'agit ici de ce qui s'appelle « upload » et non « download ». En fait, il s'agit de charger sur le site du serveur des fichiers stockés coté client.

3. Envoie de mails

Une autre fonctionnalité web est liée à la possibilité d'envoyer des courriels. Typiquement on peut utiliser la fonction mail(destinataire(s), sujet, message, entetesOptionnels). La fonction mail où PHP dialogue directement avec le serveur SMTP du destinataire est à proscrire (pas de signature DKIM de mail, pas de retry en cas de serveur indisponible, pas de SSL pour l'envoi de données sensibles...)

Voici un exemple simple :

```
<html>
<body>
<?php
$dest = "sr03@adm.utc.fr";
echo "Ce script envoi un mail à $dest";
mail($dest, "test ", "Hello SR03 ");
?>
</body>
</html>
```

4. Modification des entêtes HTTP

Il est très aisée de modifier les entêtes HTTP des requêtes en utilisant la fonction header(). Une utilisation très fréquente de cette fonction concerne la redirection des pages html. Par exemple, si on ne souhaite pas donner l'accès à une page web (accueil.php) sans authentification, on redirige la page vers une page login.php qui procède à l'authentification. Pour cela il suffit d'intégrer au début de la page accueil.php (avant toute code html) la commande

```
<?php header("Location: login.php"); exit();?>.
```

5. Gestion des cookies

PHP gère les cookies de façon transparente. Ainsi, la fonction `setcookie()`, placée au début de la page (donc avant le code HTML), permet d'écrire dans l'entête de la requête HTTP les informations du cookie à destination du client. Ce dernier les stocke et les rajoute à chaque requête lors des prochaines visites. Plus précisément, lorsqu'un internaute interroge un site web repéré par un nom de domaine, son navigateur envoie au serveur la liste des cookies disponibles pour ce domaine. PHP les réceptionne puis construit le tableau superglobal `$_COOKIE`. Les clés correspondent aux noms des cookies et les valeurs aux valeurs inscrites dans ces derniers. On pourra alors accéder à la valeur d'un cookie en appelant le tableau `$_COOKIE` avec la clé correspondant.



Exemple

Création d'un cookie :

```
<?php
setcookie('couleurPreferee','cielbleu', time()+3600*24*31);
```

Lecture d'un cookie :

```
<html>
<head>
<title>Lecture d'un cookie !</title>
</head>
<body>
<p>
Votre couleur favorite est :
<?php
echo $_COOKIE['couleurPreferee'];
?>
</p>
</body>
</html>
```

Suppression d'un cookie (coté client et serveur):

```
<?php
setcookie('couleurPreferee'); // permet de supprimer le
cookie coté client.
unset($_COOKIE['couleurPreferee']); // permet de supprimer
la valeur coté serveur.
?>
On peut supprimer un cookie coté client également par :
<?php
setcookie ('couleurPreferee', '', time( ) - 1);
?>
```

6. Gestion des sessions

Une session est un système permettant de conserver des valeurs de pages en pages, permettant ainsi de faire vivre une transaction. Les sessions remplissent le même rôle que les cookies, (il s'agit d'un cookie côté serveur contenu dans un répertoire tmp). Les sessions sont particulièrement utilisées pour des types d'applications sensibles ou complexes, comme les espaces membres et accès sécurisés avec authentification (i.e. des forums, ou même des interfaces d'administration), gestion d'un caddie sur un site de vente en ligne, ou formulaires

éclatées sur plusieurs pages.

Principe de fonctionnement

Au lieu de stocker les informations chez le visiteur (comme pour les cookies), on les stocke sur le serveur. Un identifiant est attribué à chaque client et lui est envoyé. Chaque fois que le client revient et annonce cet identifiant, PHP récupère toutes les informations sauvegardées dans le tableau superglobal `_SESSION` sur la session du client.

Les informations sont stockées dans des fichiers sur le serveur tel qu'à chaque session correspond un fichier. Chaque session est désignée par un nom et un identifiant. Lorsque le visiteur accepte les cookies, l'identifiant de la session est stocké dans un cookie, dans le cas contraire, il existe un autre moyen de stocker l'identifiant (PHPSESSID ajouté à la fin de l'url, ou par des champs cachés d'un formulaire, selon la configuration du serveur). Pour pouvoir travailler avec les sessions, on a besoin de la variable globale `$_SESSION`, des fonctions comme `session_start()`, `session_destroy()`, (démarrer/arrêter une session) et `session_unset()` (détruit certaines/toutes les variables de session).

Normalement, la session est détruite ou à la fermeture du navigateur, ou au bout de 30 minutes. Cette valeur peut être modifiée dans le fichier `php.ini` du serveur (voir le fichier de configuration plus loin).

Initialisation (et restauration) d'une session :

- `session_start()`



Attention

La fonction `session_start()` doit obligatoirement être appelée avant tout envoi au navigateur puisqu'elle écrira dans l'entête HTTP.

Écriture et lecture d'une session :

- Le tableau `$_SESSION` : tableau superglobal associé à la session concernée ;
- L'écriture de session : `$_SESSION['login'] = 'dupond'` ;

Destruction d'une session

```
<?php
session_start();
$_SESSION = array();
session_unset();
session_destroy();
?>
```

Il faut ne pas perdre de vue que le serveur peut avoir besoin de stocker des informations sur le client autre que de type chaîne de caractères. Il est ainsi possible de stocker un tableau entier dans un cookie. Pour réaliser cette opération, il faut obligatoirement transformer ce dernier en chaîne de caractères puis le reconstruire lors de sa réception. C'est ce que l'on appelle la sérialisation (ou *marshalling*) et la désérialisation (ou *unmarshalling*). Ces deux opérations sont réalisées au moyen des fonctions `serialize()` et `unserialize()`. Un schéma similaire est utilisé quand il faut stocker des types de données composés ou spéciaux dans les échanges via une session.

7. Les cookies versus les sessions

Pour résumer, voici quelques éléments permettant de comparer les sessions avec les cookies.

- Les cookies sont stockés côté client et les sessions côté serveur.
- Les cookies sont moins sécurisés que les sessions.
 - Les cookies font circuler les infos client entre le serveur et le poste client tandis que les sessions font circuler qu'un identifiant qui n'est valide que pour un temps limité.
- Les cookies sont destinés à durer plus longtemps que les sessions. Si on veut des sessions qui durent plus il faudra alors faire appel à des bases de données pour stocker au serveur les informations des clients.
- Côté client : les cookies sont stockés en disque dur et les cookies-sessions en mémoire vive.
- Côté serveur : les informations sessions sont stockés dans des fichiers temporaires.
- Les sessions ont souvent besoin des cookies pour fonctionner correctement mais il est possible de se passer en utilisant le passage par l'URL.

Enfin, PHP permet aussi de gérer les erreurs, comme d'autres langages avancés le font. Ainsi PHP permet d'afficher plusieurs niveaux d'erreur (`E_ERROR`, `E_WARNING` ...) et de les afficher grâce à la fonction `error_reporting()`. Pour gérer les exceptions PHP propose les méthodes standard `try { ... } catch () { ... }`.

D. PHP un langage objet

Rappels. Les éléments de base de l'approche objet sont :

- L'abstraction correspond à extraire du monde réel les propriétés et les méthodes importantes associées à un objet ;
- L'encapsulation consiste à regrouper les méthodes en deux grands groupes (noyau et interface) ;
- La modularité consiste à décomposer un système complexe en sous-systèmes plus simples ;
- La hiérarchisation permet de classer et ordonner les abstractions.

Voici les cinq concepts fondateurs objets :

- L'objet est le résultat d'une abstraction : un paquet logiciel regroupant des variables (propriétés) et des fonctions (méthodes) qui lui sont propres.
- La classe regroupe des propriétés et des méthodes qui seront partagées par tous ses objets. Possède un constructeur/destructeur.
- La généralisation et/ou spécialisation (héritage) est destiné à faire correspondre à une classe une classe plus générale appelée superclasse. La spécialisation est l'inverse...
- Le polymorphisme transpose l'héritage défini pour les classes, aux méthodes des objets.
- Les messages sont destinés à faire communiquer les objets entre eux.

1. POO pour PHP

La possibilité de POO (Programmation Orientée Objet) pour PHP est apparue avec la version 5. Nous n'allons pas nous attarder sur la POO puisqu'en PHP5, le modèle objet est tout à fait conforme aux grands langages classiques de la POO. Nous voyons donc apparaître les mots clés suivants :

- **private** : Propriété ou méthode uniquement accessible au sein de la classe où elle a été définie. L'objet instancié ne pourra pas référencer directement cette propriété/méthode.
- **protected** : Propriété ou méthode uniquement accessible au sein de la classe et au sein des classes héritant de celle-ci. L'objet instancié ne pourra pas référencer directement cette propriété/méthode.
- **public** : Propriété ou méthode accessible à tous
- **interface** : Permet de définir des interfaces
- **implements** : Permet d'implémenter une interface préalablement définie
- **abstract** : Classe ne pouvant être instanciée. Toute classe contenant une méthode de type abstract doit elle-même être de type abstract
- **final** : Empêche les classes filles de reimplémenter une méthode de la classe mère
- **static** : Permet de définir des propriétés/méthodes accessibles en dehors du contexte d'objet.
- ...

a) Les classes en PHP 5

Définir une classe

```
class voiture{
public $type = "mercedes";
public function afficheType () {echo $this->type}
}
```

Instancier une classe :

```
$mavoiture = new voiture();
$mavoiture->afficheType(); //affichera mercedes
```

Des constructeurs/destructeurs par défaut ou les fonctions `__construct()` et `__destruct()`;

b) Héritage en PHP

Pour l'héritage il faut utiliser le mot-clé **extends** :

class classe-fille extends classe-mère.

On peut accéder aux méthodes/attributs de la classe-mère par `parent::`. Les appels de méthodes et l'accès aux membres peuvent être surchargés via les méthodes magiques `__call`, `__get` et `__set`. Ces méthodes ne seront déclenchées que si l'objet, hérité ou non, ne contient pas le membre ou la méthode à laquelle on souhaite accéder (voir les méthodes magiques plus loin).



Attention

Pas d'héritage multiple pour PHP.



Exemple

```
class voiture{
public $type = "mercedes";
public function afficheType () {echo $this->type}
final public function dejaPlein() {
echo "personne ne me surchargera !"
}
}
```

```
Class voitureBreak extends voiture
{
//surcharge
public $type = "break";
//specialisation
Public $volumeCoffre = "695 dm3 <BR>";
//surcharge
Public function afficheType(){
parent ::afficheType() ;
echo "<BR>" ;
echo "avec un grand coffre de <BR>" ;
echo $this->volumeCoffre ;
}
}
```

```
$maVoiture = new voiture () ; // voiture mère
$monBreak = new voitureBreak() ; //la fille
echo "MERE :<BR>" ;
maVoiture->afficheType(); // cela affichera « mercedes »
```

```
echo "FILLE :<BR>" ;
monBreak->afficheType(); // cela affichera « mercedes break
avec un grand coffre de 695 dm3 »
$monbreak->dejaPlein() ;
/* cette dernière ligne est possible car monBreak est un
descendant de voiture. Elle affichera « personne ne me
surchargera ! » */
```

c) Le polymorphisme en PHP

En PHP on peut utiliser

- les classes abstraites :
 - On peut déclarer une classe abstraite (non-instanciable, mot-clé `abstract`).
- les interfaces :
 - Une interface décrit les services que pourront réaliser certaines classes.
 - Se définit par le mot-clé `interface`. Les fonctions sont public.
 - Une classe qui implémente les méthodes d'une interface se définit par : `class nom-classe implements nom-interface`.

d) Les fonctions magiques

Une des particularités de PHP est l'utilisation des fonctions magiques :

- Les fonctions `__construct`, `__destruct`, `__call`, `__get`, `__set`, `__isset`, `__unset`, `__sleep`, `__wakeup`, `__toString`, `__invoke`, `__clone`... sont connues comme des fonctions magiques en PHP.
- Les méthodes magiques sont appelées automatiquement lorsque certaines actions sont effectuées;

Ces méthodes se reconnaissent par leur nommage (réservé) qui commence toujours par «`__` ». Voici les principales méthodes magiques :

`__construct` Cette méthode est appelée sitôt que la classe est instanciée, ce qui permet de lancer une initialisation.

`__destruct` Cette fonction est appelée lorsque toutes les références à un objet sont effacées ou lorsque l'objet est explicitement détruit (`unset()`).

`__call` est lancé lorsque l'on invoque des méthodes inaccessibles dans le contexte de l'objet. Cela permet de gérer l'absence d'une méthode sans provoquer d'erreur bloquante.

`__get` Cette méthode sert à lire une donnée normalement inaccessible, par exemple une variable private dans le parent d'une classe.

`__set` Cette méthode sert à écrire une donnée normalement inaccessible.

`__isset` Cette méthode est lancée en appelant la fonction `isset()` ou la fonction `empty()` sur des membres inaccessibles.

`__unset` Cette méthode est appelée lorsque `unset()` est appelé sur des membres inaccessibles.

`__sleep` La fonction `serialize()` vérifie si la classe a une fonction avec le nom magique `__sleep`. Si c'est le cas, cette fonction sera exécutée avant toute linéarisation. Elle peut nettoyer l'objet et retourner un tableau avec les noms de toutes les variables de l'objet qui doivent être linéarisées.

`__wakeup` Réciproquement, la fonction `unserialize()` vérifie la présence d'une fonction dont le nom est le nom magique `__wakeup`. Si elle est présente, cette fonction peut reconstruire toute ressource que l'objet possède.

`__clone` Cette méthode crée la copie d'un objet. Elle ne peut pas être appelée directement, elle réagit à l'appel de la fonction `clone $object`.



Exemple

```
<?php
class MaClasse{
private $vars = array();
public function __construct(){
}
public function __set($var, $val){
$this->vars[$var] = $val;
}
public function __get($var){
if(isset($this->vars[$var])){
return $this->vars[$var];
} else {
throw new Exception("attribut '$var' n'existe pas");}
}
}
$maClasse = new MaClasse();
$maClasse->hello = "world"; // rajoute l'attribut "hello" et l'initialise avec la valeur
"world".
echo $maClasse->hello; // affiche "world"
echo $maClasse->resto; // lance une exception
?>
```

L'accès à une BDD peut se faire soit au moyen d'une API spécialisée pour le SGBD concerné (MySQL, PostgreSQL, Oracle...), soit à travers une API générique (aussi appelée couche d'abstraction). Puisque une API générique laisse davantage de liberté et de flexibilité que les APIs, cette solution est préférée. Son plus gros avantage est de se connecter à la plupart des SGBD avec un minimum de modifications sur le code de l'application PHP. L'API générique fournie en standard depuis PHP 5.1 est le PHP Data Objects (PDO¹).

L'accès à une base de données est soumis à diverses étapes :

- 3 - http://fr2.php.net/manual-lookup.php?pattern=nom_function&lang=fr&scope=404quickref

Dans le cas de l'utilisation d'une API pour l'accès à une BD (on s'intéresse ci-dessous à mysql), PHP permet un interfaçage très simple en 3 étapes :

- Déclarer les variables qui vont permettre la connexion à la base de données :
 - \$user : Le nom d'utilisateur
 - \$passwd : Le mot de passe
 - \$host : L'hôte (ordinateur sur lequel le SGBD est installé)
 - \$bdd : Le nom de la base de données
- Utiliser des fonctions permettant de manipuler les bases de données (mysql etc...)
 - La fonction de connexion au serveur (mysql_connect..). Traiter les erreurs
 - mysql_connect(\$host,\$user,\$passwd) or die("erreur de connexion au serveur \$host");
 - La fonction de choix de la base de données (mysql_select_db, ...)
 - La fonction de requête (mysql_query, ...)
 - La fonction de déconnexion (mysql_close, ...)
- Traiter les erreurs et récupérer les résultats (mysql).
 - Ex. mysql_connect(\$host,\$user,\$passwd) or die("erreur de connexion au serveur \$host");
 - Stocker l'ensemble des enregistrements dans une variable tableau qui sera exploitable avec la fonction mysql_fetch_row():


```
$result = mysql_query($query);
while($row = mysql_fetch_row($result)){...
... }
```
 - Enfin : mysql_close();



Remarque

Il y a aussi mysqli (i pour improve) au lieu de mysql qui permet de profiter des nouvelles fonctionnalités de MySQL5 et une approche objet. L'utilisation est très semblable à mysql.

b) PDO

Comme cela a été dit plus haut, PDO est une interface uniforme pour la connexion et l'exploitation des SGBD. L'extension PDO (PHP Data Object) arrivée avec PHP 5.1, constitue le socle commun d'accès vers les SGBD. Trois classes permettent de réaliser cela : **la classe PDO** destinée à la connexion avec les bases, **la classe PDOStatement** gère les requêtes et le résultat et **la classe PDOException** permet de traiter les erreurs.

La connexion au serveur et la sélection de la BDD se font lors de l'instanciation de l'objet PDO :

```
$db = new PDO('mysql:host=localhost;dbname=developpez',
'utilisateur', 'motdepasse');
```

Pour envoyer une requête, il y a plusieurs solutions. La plus sécurisée est de passer par des requêtes préparées (*prepared statements*) :

```
$select_users = $db->prepare('SELECT id, name FROM user');
$select_users->execute();
$users = $select_users->fetchAll();
```

Une requête se prépare donc une seule fois pour l'ensemble du script (méthode prepare()). Chaque exécution de la requête se fait alors par la méthode execute(), et la récupération des résultats par la méthode fetchAll().

3. Fichier de configuration

Le comportement de PHP est dicté par sa configuration établie dans le fichier `php.ini`. Ce fichier standard de configuration est habituellement placé dans le même répertoire que PHP et peut être modifié à l'aide de n'importe quel éditeur de texte.

Le fichier est composé de directives qui peuvent prendre des valeurs booléen (0,1), numérique (p.ex. pour exprimer le temps en secondes), ou même en string (souvent entre guillemets). Les directives ne sont pas figées. Ainsi de très nombreuses directives de ce fichier peuvent être modifiées pour la durée d'une requête par le fichier « `httpd.conf` », par un fichier « `.htaccess` » ou au moment de l'exécution du script par la fonction « `ini_set()` ».

Voici quelques directives qui peuvent se révéler utiles. Il est ainsi possible de paramétrer comment répertorier les erreurs mais aussi fixer le niveau des erreurs grâce à l'option « `error_reporting` » (PHP dispose de divers niveaux d'erreur). Il est aussi possible de préciser la taille maximum des fichiers que PHP accepte depuis un formulaire POST. En particulier, pour faciliter la gestion des sessions plusieurs paramètres peuvent être définis dans `php.ini`. Citons par exemple « `session.use_cookies` » qui doit être activé pour utiliser la transmission par cookie, (conseillée), « `session.auto_start` » permet de démarrer la session à chaque requête, etc.

Programmation Java, les applets, les servlets et JSP

VI

Introduction au Java et aux applets	95
Les servlets	104
Java Server Pages	115

A. Introduction au Java et aux applets

JAVA est un langage développé par SUN pour être orienté objet et simple, robuste et sûr, indépendant des architectures matérielles, orienté réseau et distribué, et multitâches. Java est souvent associé au monde Internet car il permet de programmer des applications à la fois coté client (applets) et serveur (servlets, JSP). Néanmoins, il est entièrement possible de réaliser des applications autonomes. Nous allons nous intéresser dans ce cours seulement aux applications Java pour l'Internet.

1. Aspects généraux du langage java

Examinons de plus près quelques propriétés du langage Java.

Langage orienté objet et simple

En effet, Java permet de faire de la POO tout en apportant une certaine souplesse et simplicité. Ainsi, même si Java se rapproche beaucoup de C++, certaines lourdeurs de ce dernier ont été supprimées. En particulier il offre une gestion mémoire transparente grâce à un *garbage collector* efficace.

Robuste et sûr

Notons d'abord que le typage des données est extrêmement strict. De plus, l'interpréteur Java vérifie le code chargé avant de l'exécuter afin d'éviter les problèmes liés aux virus ou autres tentatives de violation du système. Pour les applets, il est en principe impossible d'accéder aux ressources de la machine hôte

Indépendant des architectures matérielles

Le compilateur génère un code universel le « byte-code ». Un interpréteur spécifique à l'ordinateur hôte, « la machine-virtuelle » JVM, permet l'exécution des

programmes.

Orienté réseau et distribué

Java offre la possibilité de charger des parties de l'application via le réseau.

Multitâches

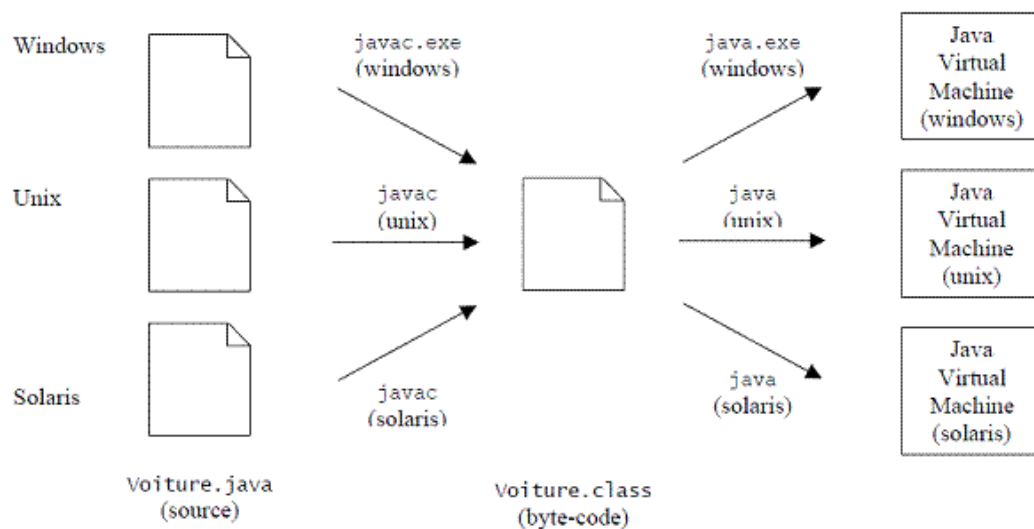
JAVA permet l'exécution en apparence simultanée de plusieurs processus. En réalité on accorde de façon séquentielle un peu du temps processeur à chaque processus (multithread). Illustrons cela par le cas d'une application simple, celle d'un applet. Une applet est vue comme composée d'un thread destiné à la tâche principale, un thread pour écouter les événements, et un autre pour la gestion du mémoire.

2. Environnement

On distingue J2SE (standard edition) et J2EE (entreprise edition). J2SE (actuellement J2SE v1.6, ou V6) contient les composantes basiques utilisables aussi bien coté client que serveur. J2EE (actuellement J2EE v1.4), est d'une certaine façon une extension du SE, mais beaucoup plus riche et dédiée à la programmation coté serveur. Il permet aussi de concevoir de façon plus aisée des applications distribuées.

Compilation et interprétation du code Java

Le fichier byte-code est le même quel que soit le système d'exploitation.



Compilation et interprétation du code Java

Le compilateur compile des sources java en byte code selon la commande `javac voiture.java` qui donne `voiture.class`. Ensuite on fait appel au programme `java` qui va interpréter le byte code généré. Java pousse l'indépendance du code plus loin. En effet, Java suit le principe « *write once, run anywhere* ». La figure ci-dessus résume cette logique.

3. Les packages

Java offre de nombreuses fonctions prédéfinies rangées dans des packages fonctionnels. Il suffit ensuite de les signaler dans le programme pour pouvoir les utiliser.

- java.lang // Classes et interfaces fondamentales du langage. Il est par défaut.
- java.util // Utilitaires (collections, internationalisation, logging, expressions régulières,...).
- java.io // Gestion des flux entrant et sortant
- java.math // Gestion des opérations mathématiques
- java.net // Accès aux réseaux
- java.security // Mise en oeuvre de fonctionnalités concernant la sécurité
- java.sql // API JDBC : Accès aux bases de données
- java.rmi // API RMI (invocation de méthodes distantes)
- java.awt // Création d'interfaces graphiques avec AWT
- java.applet // Création d'applets
- ...

Les commandes import et classpath

Pour inclure des classes distantes dans un programme, on utilise la commande `import`. Cela permet d'éviter de spécifier le package de la classe à chaque fois qu'on l'utilise. La commande `classpath` permet de spécifier à la machine virtuelle Java les emplacements à partir desquels les ressources (bytecode et autres) devront être recherchés. On peut ainsi spécifier le répertoire des classes (p.ex. `/project/classes`) ou une archive (p.ex. `/project/lib/archive.jar`).



Remarque

Java n'est pas un JAL¹ mais à la fois un langage et une plateforme destinés à véritablement faciliter le travail et la complexité de la programmation. Le JVM y contribue largement. Le système JAVA ne contient pas seulement une librairie avec des fonctions utilisables mais d'un ensemble hiérarchique rendu cohérent par la nature objets du langage.

4. Événement en Java

Un élément important dans les langages orientés web est la gestion des événements. En JAVA, les événements se gèrent par AWT² et plus récemment par SWING. Dans les versions plus récentes de AWT ainsi qu'avec SWING, la gestion se fait par un pattern design *observer* ou *listener*.



Rappel

En génie logiciel, **un design pattern** est un concept destiné à résoudre les problèmes récurrents suivant le paradigme objet.

La gestion des événements en Java

La gestion des **événements** (clic souris, appuyer sur un bouton, etc...) se fait sur le principe du pattern Observateur (ou listener). Certains objets (**source**) sont susceptibles de générer des événements.

Tout objet désireux d'être averti lors de la survenance d'un événement d'une source doit implémenter une interface et les méthodes pour le traitement `<Type événement>Listener` et s'abonner à cette source par les méthodes de l'objet source. L'abonnement/retrait se fait par `add/remove<Type événement>Listener()` avec en paramètre l'objet cible. Il devient alors une **cible** (de la source). Lorsqu'une source déclenche un événement, concrètement elle va simplement parcourir la liste de ses abonnés et appeler la méthode associée.

5. Les applets

Une applet est une application Java particulière qui s'exécute dans un document HTML visualisé par un browser. Une applet tourne donc dans une JVM spécifique, la JVM d'un navigateur web. L'objectif est de transmettre au client du code exécutable, auquel on peut passer des paramètres.

Une applet possède des caractéristiques particulières relatives à la sécurité :

- Une applet ne peut communiquer qu'avec la machine dont elle est issue.
- Une applet chargée dans un browser à travers le réseau n'a aucun accès au système de fichiers local. Elle ne peut pas accéder aux méthodes natives de la machine locale. Si cet accès était possible, elle pourrait notamment accéder par ce biais au système de fichier local.

Une applet est vue comme un conteneur de composant graphique et la classe **Applet** est une sous classe de la classe **Panel** (du package `java.awt`). Une applet va donc pouvoir répondre aux événements souris et claviers et utiliser des composants graphiques comme les boutons, cases à cocher etc.

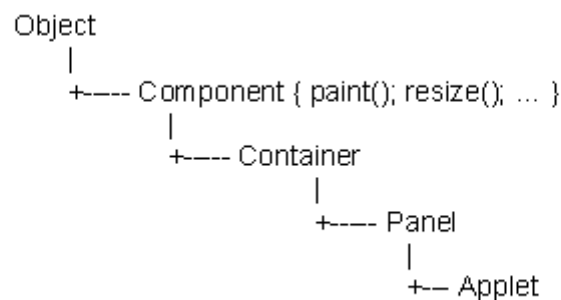


Remarque

Très souvent ce sont les Swing qui sont utilisés pour faire des interfaces graphiques. Les Swing sont des composants (bouton, fenêtre, label, zone de texte ...), regroupés dans le package Swing. Les Swing sont des JavaBeans. Grâce à Swing, il est possible de créer des applets plus attrayantes. Une applet Java standard hérite de la classe `java.applet.Applet`, mais dans le cas d'une **applet Swing**, elle hérite de la classe `javax.swing.JApplet`.

a) Le package `java.applet`

Le package `java.applet` permet aux programmeurs d'intégrer des applications Java (applets) dans des documents Web. Ce package contient la classe `Applet` et les interfaces `AppletContext`, `AppletStub` et `AudioClip`.



La classe `Applet` dérive de la classe `java.awt`

Pour construire une applet, il faut définir une classe héritant de la classe `Applet` et redéfinir certaines méthodes. Elle possède quatre méthodes particulières qui sont : `public void init()` ; `public void start()` ; `public void stop()` ; `public void destroy()`. Elle hérite aussi de la méthode `public void paint(Graphics g)` déclarée dans la classe `Component`.

`public void init()`

La méthode `init()`, appelée par le browser, permet l'initialisation de l'applet, (le constructeur sera appelé avant cette méthode). Cette méthode est exécutée une seule fois au démarrage de l'applet. Elle permet la récupération de paramètre et l'instanciation d'objets dépendant de paramètre, l'initialisation de valeurs dépendant de paramètre, le chargement de polices ou d'images dépendant de paramètre...

`public void start()`

Elle est exécutée juste après la méthode `init()` et à chaque fois que le browser revient sur la page HTML qui contient cette applet.

`public void stop()`

La méthode `stop()` est exécutée chaque fois que l'utilisateur quitte la page web contenant cette applet ou quand la page n'est plus visible.

`public void destroy()`

Cette méthode est appelée à la fin de l'applet, c'est à dire lorsque l'utilisateur quitte la fenêtre ou le browser de pages Web.

`public void paint(Graphics g)`

La méthode `paint()` est obtenue par héritage, elle est déclarée dans la classe

Component. Elle est appelée chaque fois que le gestionnaire de fenêtre doit dessiner le contenu de cette applet. D'autres méthodes permettent d'obtenir des informations sur l'applet en cours d'exécution.

public String getParameter(String name

Récupère les paramètres passés dans le document HTML.

Intégration d'une applet dans une page html

La balise <applet> permet d'inclure dans un document HTML un espace pour l'exécution d'une application java.

```
<APPLET
CODE="UnNomDeClasse"
HEIGHT= unEntier
WIDTH= unAutreEntier
>
<PARAM
NAME="premierParametre"
VALUE="valeurDuPremierParametre"
>
<PARAM
NAME= "secondParametre"
VALUE="valeurDuSecondParametre"
>
...
</APPLET>
```

Notons qu'on ne peut définir qu'un constructeur par défaut dans une applet. Le browser ne pourrait pas savoir quel paramètre transmettre à un autre constructeur. En revanche, il est possible de passer des paramètres à une applet. A titre d'exemple, ces paramètres peuvent être destinés à l'instanciation d'objets, l'initialisation de valeurs, chargement de polices ou d'images...

Le fichier HTML a donc la possibilité de passer des paramètres à une applet (la balise PARAM). La récupération de ces paramètres s'effectue dans la source de l'applet en utilisant la méthode String getParameter(String name) de la classe Applet.

Structure d'une applet :

```
import java.applet.*;
import java.awt.*;
public class <NomApplet> extends Applet {
public void init() {
<Initialisations>
<Démarrage de processus>
}
public void start() {
<Démarrer l'applet, la page Web est visitée ou redevient visible>
}
public void paint(Graphics g) {
<Dessiner le contenu actuel de l'applet>
}
public void stop() {
<Arrêter l'applet, la page Web n'est plus visible ou l'utilisateur quitte le navigateur>
}
public void destroy() {
<Relâcher les ressources, l'applet va quitter la mémoire>
}
```

```
}  
}
```

Cycle de vie d'une applet : init() (start() paint() stop()) destroy()

```
import java.applet.*;  
import java.awt.*;  
public class HelloSR03 extends Applet  
{  
    String texte;  
    public void init()  
    {  
        texte = getParameter("texte");  
    }  
    public void paint(Graphics g)  
    {  
        g.drawString(texte, 30, 30);  
    }  
}
```

Une page HTML minimale pour logger l'applet :

```
<HTML>  
<body>  
<APPLET code="HelloSR03.class" width="500" height="200">  
<param name="texte" value="Hello SR03 !">  
</applet>  
</BODY>  
</HTML>
```



Remarque

Pour l'exécuter on peut soit utiliser le navigateur en ouvrant le fichier HTML, soit utiliser un appletviewer.

Appel

```
<applet  
code = "fichier.class"  
archive="fichier.jar"  
width= "200" height= "200" >  
</applet>
```



Exemple : Un exemple d'une applet Swing

```
import java.awt.* ;  
import java.awt.event.* ;  
import javax.swing.event.* ;  
import javax.swing.* ;  
public class App2Bout extends JApplet implements  
    ActionListener  
{ public void init ()  
{ pan = new JPanel ();  
  panCom = new JPanel();  
  Container contenu = getContentPane() ;  
  contenu.add(pan) ;  
  contenu.add(panCom, "Bas") ;  
}
```

```

rouge = new JButton ("rouge") ;
jaune = new JButton ("jaune") ;
rouge.addActionListener(this) ;
jaune.addActionListener(this) ;
panCom.add(rouge) ;
panCom.add(jaune) ;
}
public void actionPerformed (ActionEvent e)
{ if (e.getSource() == rouge) pan.setBackground (Color.red)
;
if (e.getSource() == jaune) pan.setBackground
(Color.yellow) ;
}
private JPanel pan, panCom ;
private JButton rouge, jaune ;
}

```

Voici la page html qui héberge l'applet.

```

<HTML>
<BODY>
<APPLET
CODE = "App2Bout.class"
WIDTH    = 250
HEIGHT   = 100
>
</APPLET>
</BODY>
</HTML>

```

Dans cet exemple il y a un panneau appelé "bas", où on incorpore deux boutons permettant d'agir sur la couleur (rouge ou jaune) du panneau supérieur.

Le format de fichier archive jar permet d'agglomérer un ensemble de fichiers utilisables par une applet (".class", sons et images) en un seul fichier au format JAR qui sera chargé en une seule requête par le protocole HTTP. La création se fait par la commande `jar cfv fichier.jar fichier_1 ... fichier_n`. (l'option `c` indique qu'il faut créer une archive Jar, l'option `f` indique que le résultat sera redirigé dans un fichier l'option `v` fait afficher les commentaires associés à l'exécution de la commande).

B. Les servlets

Différemment des applets qui sont du code java exécutable sur le navigateur du client, les servlets sont l'alternative de la technologie Java à la programmation des CGI. Elles permettent par exemple de réaliser les actions suivantes :

- Lire les données envoyées par l'utilisateur. Elles proviennent généralement d'un formulaire d'une page Web, d'une Applet ou de tout programme client HTTP.
- Acquérir des informations supplémentaires sur la requête telles que identification du navigateur, type de méthode employée, valeurs des cookies, etc.
- Générer les résultats. Cela implique généralement l'accès à une couche métier service.

- Formater le résultat dans un document. Dans la plupart des cas les résultats sont incorporés dans une page HTML.
- Définir les paramètres de la réponse HTTP appropriés. Il faut indiquer au navigateur le type de document renvoyé, définir des cookies, établir une session etc.
- Renvoyer le document au client. Le format peut être du texte (HTML), un format binaire (image, par exemple), un zip etc.

1. Serveur d'application

Un serveur d'application permet de charger et d'exécuter les servlets dans une JVM. C'est une extension du serveur web. Ce serveur d'application contient entre autre un moteur de servlets qui se charge de manager les servlets qu'il contient. Pour exécuter une servlet, il suffit de saisir une URL qui désigne la servlet dans un navigateur.

- Le serveur reçoit une requête HTTP qui nécessite une servlet.
- Si c'est la première sollicitation de la servlet, le serveur l'instancie. Les servlets sont stockées (sous forme de fichier .class) dans un répertoire particulier du serveur et reste en mémoire jusqu'à l'arrêt du serveur.
- Le serveur crée un objet qui représente la requête HTTP et un objet qui contiendra la réponse et les envoie à la servlet.
- La servlet crée dynamiquement la réponse sous forme de page html transmise via un flux dans l'objet contenant la réponse. La création de cette réponse utilise la requête du client mais aussi un ensemble de ressources incluses sur le serveur tels que des fichiers ou des bases de données.
- Le serveur récupère l'objet réponse et envoie la page html au client.

Comme on peut le remarquer ci-dessus, il y a deux objets clés qui régulent les échanges client - serveur. Il s'agit des objets dédiés à la requête (*HttpServletRequest*) et à la réponse (*HttpServletResponse*).

L'environnement nécessaire pour le développement des servlets comprend la standard Edition J2SE v1.6 (ou V6) ou l'Entreprise Edition J2EE v1.4 (qui contient entre autres les classes pour servlets à la spécification 2.4), et un serveur de servlets (Tomcat) d'Apache⁴ v5.x.

Dans le serveur Tomcat on trouve la configuration suivante :

<code>/webapps</code>	
<code> /votre-application</code>	
<code> /images</code>	pour les .gif et autres .jpeg
<code> /WEB-INF</code>	
<code> /classes</code>	pour les .class
<code> /lib</code>	pour les .jar
<code> web.xml</code>	le fichier de déploiement
<code> /répertoires spéciaux</code>	
<code> les fichiers HTML et JSP</code>	

Hiérarchie des fichiers dans Tomcat

Le fichier de déploiement (web.xml) joue un rôle très important dans la configuration des servlets.

2. Le descripteur de déploiement

Un descripteur de déploiement est un fichier XML contenant des informations sur la WebApp nécessaires au serveur Web. Le nom standard est **web.xml**. Dans Tomcat, il doit être placé dans `webapps\your-application\WEB-INF` et est lu au démarrage de Tomcat. On peut y décrire les paramètres de contexte, les identifiants de servlet (**<servlet-name>**), des définitions de paramètres (**<param-name>**) associés à des identifiants de servlet, les mapping entre identifiant de servlets et url (**<servlet-mapping>** et **<url-pattern>**). Pour traiter la requête, le serveur fait un matching simple entre l'URL de la requête arrivée et la servlet stockée dans le serveur.



Exemple

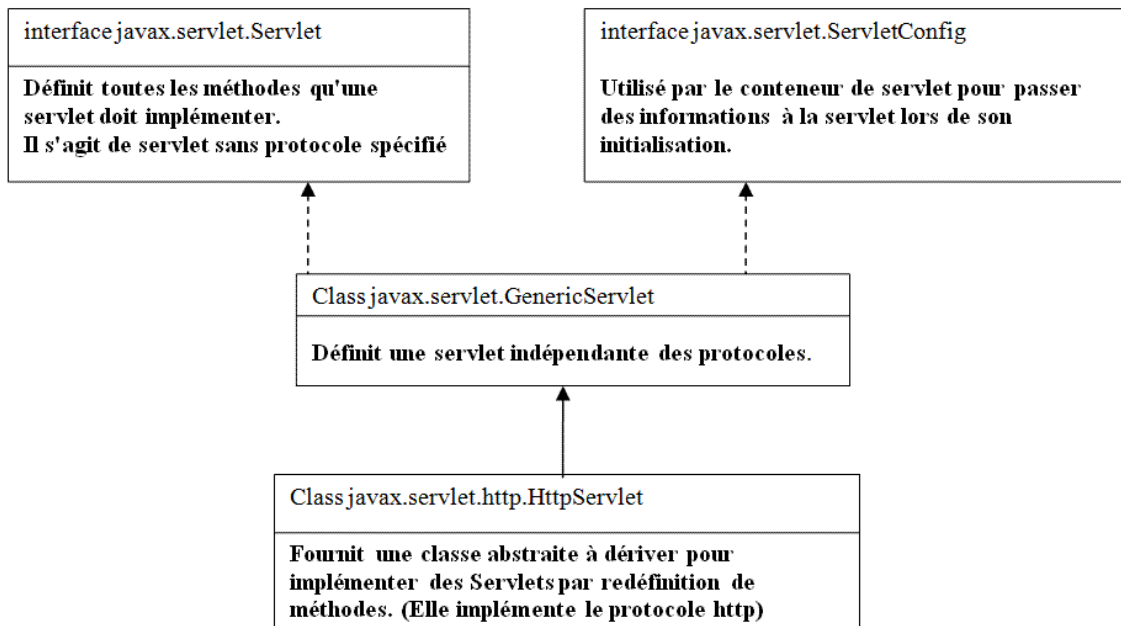
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN"
"http://java.sun.com/dtd/web-app_2_4.dtd">
<web-app>
<!--===== -->
<!-- Definition des paramètres WEB -->
<!-- Valable pour toutes les servlets -->
<context-param>
<param-name>UV</param-name>
<param-value>SR03</param-value>
</context-param>
<context-param>
<param-name>Cours</param-name>
<param-value>Servlet et Jsp</param-value>
</context-param>
<!--===== -->
<!--===== -->
<!-- Definition pour la servlet 1 -->
<servlet>
<servlet-name>Hello</servlet-name>
<description>C'est un premier exemple </description>
<servlet-class>Hello</servlet-class>
<servlet-mapping>
<servlet-name>Hello</servlet-name>
<url-pattern>/servlet/Hello</url-pattern>
</servlet-mapping>
<init-param>
<param-name ... </param-name>
<param-value> ...</param-value>
</init-param>
</servlet>
<!-- Definition pour la servlet 2 -->
<servlet>
...
```

3. Invocation des servlets

On peut appeler une servlet depuis un navigateur avec son mapping url, nom de classe ou son identifiant. Il est préférable d'utiliser le mapping url. On peut aussi appeler une servlet depuis une page html soit sur une balise de lien `<a >...`, soit sur une balise `<form>`, (dans ce dernier cas il s'agit de faire **`<form action="URL de la servlet" method="post"> </form>`**).

4. Création des servlets

On va maintenant s'intéresser à la création des servlets. Voici les classes et interfaces en jeu pour le fonctionnement des servlets.



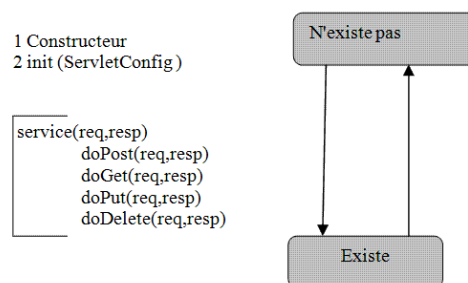
Classes et interfaces pour le fonctionnement des servlets

Package javax.servlet Classe interface	Contient les classes et interfaces définissant les servlets de manière générique. C'est-à-dire, sans protocole imposé.
GenericServlet	Définit une servlet générique indépendante des protocoles.
RequestDispatcher	Définit un objet qui reçoit les demandes d'un client et les renvoie à une ressource quelconque (Servlet , JSP ou HTML) sur le serveur. Il s'agit d'une redirection au niveau du serveur.
Servlet	Définit les méthodes que toutes les servlets doivent implémenter. Elle définit ainsi le cycle de vie des servlets.
ServletConfig	Un objet utilisé par un conteneur de servlet pour transmettre des informations (paramétrage) à une servlet pendant son initialisation.
ServletContext	Définit un ensemble de méthodes utilisées par une servlet pour dialoguer avec son conteneur. Un ServletContext représente une zone commune à toutes les servlets et Jsp d'une application Web.
ServletException	Définit l'exception générale que peut déclencher une servlet lors de problèmes.
ServletRequest	Définit un objet pour fournir à la servlet des informations sur les demandes du client.

ServletResponse	Définit un objet qui assiste la servlet dans sa réponse au client
Package javax.servlet Classe interface	Contient les classes et interfaces définissant les servlets supportant le protocole HTTP.
Cookie	Permet de gérer les cookies.
HttpServlet	Fournit une classe abstraite à dériver pour créer une servlet HTTP.
HttpServletRequest	Dérive de ServletRequest pour fournir les informations des demandes pour les servlet HTTP.
HttpServletResponse	Dérive de ServletResponse pour fournir les fonctionnalités spécifiques HTTP dans l'envoi d'une réponse.
HttpSession	Fournit le moyen d'identifier un utilisateur, d'une demande de page à l'autre et de stocker des informations sur cet utilisateur.

5. Cycle de vie d'une servlet

La servlet passe par plusieurs phases comme affiché dans la figure ci-dessous.



Cycle de vie d'une servlet

La méthode init

- La méthode **init(**ServletConfig**)** est appelée une seule fois après la création de la servlet, (c'est un peu l'équivalent de la méthode init() de la classe Applet). On peut y récupérer les paramètres de la servlet mis dans web.xml et stockés dans l'objet ServletConfig.

La méthode service

- Chaque fois que le serveur reçoit une requête pour une servlet il crée un thread et y appelle la méthode service. Celle-ci vérifie le type de la requête HTTP (GET, POST, PUT, etc ...) et appelle la méthode correspondante (doGet, doPost, doPut, etc.)

Les méthodes doXxx (doGet, doPost...)

Ce sont ces méthodes qu'il faudra redéfinir selon le type de requête à traiter. Elles suivent la forme suivante :

```
public void doXxx(HttpServletRequest request,
                HttpServletResponse response)
{
    //Récupération des demandes du client sur request.
    ...= request.getCookies( );
    ...= request.getParameterNames( );
    ...
    //Renseigner les en-têtes de la réponse si nécessaire.
    response.setXXX(...);
    //Indication de la nature de la réponse (type mime)
    response.setContentType("text/html");
    //Récupération d'un flux de réponse à partir de response
    PrintWriter out = response.getWriter( );
    //Ecriture de la réponse (création du document dynamique)
    out.println("<p>Le servlet <b>Hel ..... ");
    ...
}
```



Exemple

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
public class Hello extends HttpServlet {
    private static final String CONTENT_TYPE = "text/html";
    //=====
    /**Initialiser les variables globales*/
    public void init() throws ServletException {
    }
    //=====
    /**Traiter la requête HTTP Get*/
    public void doGet(HttpServletRequest
    request,HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
        out.println("<p>La servlet <b>Hello</b> a reçu un <b>"+
        +"<font color=\"red\">GET</font></b>. </p>");
        out.println("<h1>Hello. Vous avez le bonjour de
        Hello.class.</h1>");
    }
    //=====
    /**Traiter la requête HTTP Post*/
    public void doPost(HttpServletRequest
    request,HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
        out.println("<p>La servlet <b>Hello</b> a reçu un <b>"+
```

```
"<font color=\"red\">POST</font></b>.</p>");
out.println("<h1>Hello. Vous avez le bonjour de
Hello.class.</h1>");
}
//=====
=====
public void destroy() { }
}
```

6. Récupération des paramètres d'un formulaire

Les paramètres envoyés par un formulaire sont récupérés via les méthodes définies dans les interfaces *HttpServletRequest* et *ServletRequest* affichées ci-dessous.

Interface <i>HttpServletRequest</i>	Méthodes en jeu dans la récupération des en-têtes.
getCookies(), getContentType(), getProtocol (), ...	getContentLength(), getMethod (), ...
Interface <i>ServletRequest</i>	Méthodes en jeu dans la récupération des paramètres de la requête
Enumeration getParameterNames()	Renvoie une énumération sur les noms de paramètres.
String getParameter (String)	Renvoie la valeur du paramètre nommé.

Voici un formulaire html avec les champs *Nom*, *Prenom*, *AnneeNaissance* et *Sexe* saisis par l'utilisateur que la servlet (à laquelle la requête est destinée) peut récupérer grâce à la méthode *getParameter*.

Nom	<input type="text"/>
Prenom	<input type="text"/>
AnneeNaissance	<input type="text"/>
Sexe	<input type="text"/>
<input type="button" value="Soumettre"/>	

Formulaire de soumission

La servlet traite alors la requête HTTP comme ci-dessous :

```
doPost(HttpServletRequest req , HttpServletResponse rep)
{
String nom;
String prenom;
int anneeNaissance;
char sexe;
...
nom = req.getParameter ("Nom");
prenom = req.getParameter ( "Prenom" );
anneeNaissance =
Integer.parseInt(req.getParameter("AnneeNaissance"));
}
```

```

sexe = req.getParameter("Sexe").charAt(0);
...
}
    
```

7. L'objet ServletContext

Il y a un objet ServletContext par application Web. Il est partagé par toutes les Servlets. Sa durée de vie est celle du conteneur Web. Il contient des paramètres définis et initialisés dans web.xml. Les servlets peuvent également y stocker des objets qui seront partagés par toutes les servlets de l'application.

Interface ServletContext	
String getInitParameter(String)	Retourne la valeur du paramètre.
Enumeration getInitParameterNames()	Renvoie une énumération sur l'ensemble des paramètres du contexte.
setAttribute(String, Object)	Stocke un objet référencé par un nom
Object getAttribute(String)	Retourne l'objet nommé.
void removeAttribute(String)	Retire l'objet nommé
Enumeration getAttributeNames()	Retourne une énumération sur les noms des objets stockés

Il est aussi possible de récupérer des paramètres stockés dans web.xml et destinés à une servlet particulière.

Classe GenericServlet	
String getInitParameter(String)	Retourne la valeur du paramètre nommé.
Enumeration getInitParameterNames()	Renvoie une énumération sur l'ensemble des paramètres de la servlet.
String getServletName()	Retourne le nom de la servlet

8. Gestion des cookies

C'est le protocole HTTP qui embarque dans ses en-têtes la demande de création et la valeur d'un cookie. Le système de stockage est laissé au navigateur. Le cookie est géré en Java sous la forme d'une instance de la classe Cookie. Voici les éléments les plus utilisés de la classe Cookie.

Classe Cookie	
Cookie(nom, valeur)	Le nom ne peut pas être changé après création, sinon il s'agit d'un nouveau cookie.
String getDomain () void setDomain(domaine)	Définir et lire le domaine associé.
int getMaxAge() void	Définir et lire la durée de vie

setMaxAge(dVie)	en secondes. Si dVie > 0, le cookie sera stocké sur disque. Si dVie < 0, (valeur par défaut) le cookie sera stocké en mémoire jusqu'à ce que le client quitte le navigateur (il s'agit d'un cookie de session). Si délai = 0, le cookie sera supprimé.
String getName ()	Lire le nom d'un cookie.
String getValue ()	Définir et lire la valeur d'un cookie
void SetValue (valeur)	

Pour manipuler les cookies on a besoin des méthodes qui les lisent sur la requête et les rajoutent dans la réponse.

Insérer des cookies dans les entêtes de réponse :

Interface HttpServletResponse	
addCookie (cookie)	Ajoute le cookie à la réponse.

Récupérer les cookies du client :

Interface HttpServletRequest	
Cookie [] getCookies ()	Récupère un tableau contenant tous les cookies envoyés par le client

9. Gestion des sessions

Les servlets offrent une solution à travers l'api *HttpSession*. Cette interface se situe au-dessus des cookies et de la réécriture d'URL. Chaque requête entrante est associée à une session active créée par le client (le navigateur). La session est détruite s'il n'y a pas de nouvelles requêtes dans les 30 minutes (valeur par défaut dans tomcat).

Interface HttpServletRequest	Interface HttpSession
HttpSession getSession (boolean create)	Object getAttribute (String name)
HttpSession getSession ()	Enumeration getAttributeNames ()

10. Redirection au niveau du serveur

Dans le traitement des requêtes clients et la préparation de la réponse, le serveur peut décomposer le travail sur plusieurs servlets. Ainsi, pour rediriger vers une page d'authentification ou confier la construction de la réponse, la servlet utilise la méthode forward. La servlet fait appel à include quand elle a besoin qu'une servlet insère sa réponse dans la sienne. Notons que les deux méthodes servent à des fins différents :

Forward

Séparer le traitement de la requête du client (effectué sur la servlet appelante) de la génération de la réponse (effectuée sur la servlet appelée).

- `getServletContext().getRequestDispatcher(urlCible).forward(req,resp);`

(urlCible, p.ex. une page JSP, fournira la réponse)

Répartir le traitement de la requête sur plusieurs servlets.

Include

Répartir la génération de la réponse sur plusieurs servlets.

11. Les limites des servlets

Les servlets sont très souples et possèdent de nombreux avantages : objet, portabilité, performance, extensibilité, gratuité, etc. Néanmoins on remarquera que :

- quand il s'agit de créer directement une réponse en html cela devient extrêmement fastidieux. Le développeur Java n'a pas forcément envie de maîtriser html, Javascript et les outils afférents.
- De même, le concepteur de page html n'est pas forcément un spécialiste de l'objet.
- La préparation de la réponse html a été alors confiée au JSP^{SP}. Cette technique permet au code Java et à certaines actions prédéfinies d'être ajoutés dans un contenu statique. Depuis la version 2.0 des spécifications, la syntaxe JSP est complètement XML.

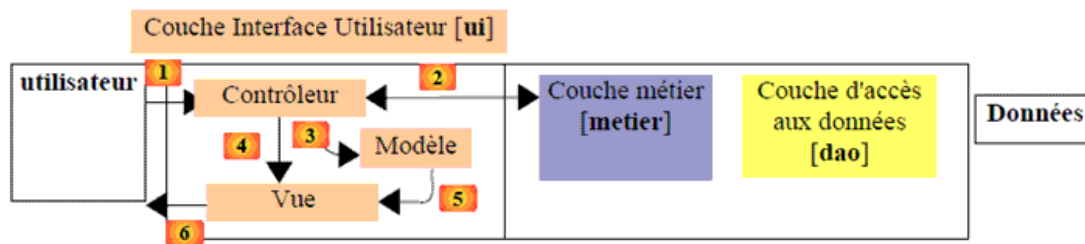
En complément de ci-dessus, notons que le traitement d'une requête peut conduire à faire appel à une BD pour récupérer les informations nécessaires. Or, un des principes d'une bonne architecture est qu'une application doit être conçue sur 3 grandes couches : Présentation – Métier – Persistance. Les servlets font bien évidemment partie de la couche Présentation, donc, en principe, on ne devrait pas permettre d'accès JDBC dans les servlets. Les accès JDBC sont traités dans la couche Persistance.

C. Java Server Pages

1. Le pattern MVC

La solution proposée par Sun est de diviser les applications web en 3 parties :

- Le traitement (le contrôleur) qui ne contient que la partie contrôle, chaînage, etc. C'est une servlet Java qui ne contient plus de génération de code HTML, difficile à créer "à la main". Sa création est du ressort d'un développeur Java.
- La présentation (une page JSP Java Server Pages, c'est-à-dire une page HTML d'extension JSP enrichie de quelques balises JSP). Elle ne devrait pas contenir de code métier (sous forme de "scriptlet"). En revanche, elle peut faire appel à des "scriptlets" pour de la mise en forme. Sa création est du ressort d'un développeur Web.
- L'accès aux données et au métier se fait dans des Beans et /ou des EJB. Leur création est du ressort d'un développeur Java.



Le pattern MVC

Le traitement d'une demande d'un client se déroule selon les étapes suivantes :

1. le client fait une demande au contrôleur. Celui-ci voit passer toutes les demandes des clients. C'est la porte d'entrée de l'application. C'est le **C** de MVC.
2. le contrôleur C traite cette demande. Pour ce faire, il peut avoir besoin de l'aide de la couche métier. Une fois la demande du client traitée, celle-ci peut appeler diverses réponses. Un exemple classique est d'envoyer une page d'erreurs si la demande n'a pu être traitée correctement et une page de confirmation sinon.
3. le contrôleur choisit la réponse (= vue) à envoyer au client. Choisir la réponse à envoyer au client nécessite plusieurs étapes:
 - choisir l'objet qui va générer la réponse. C'est ce qu'on appelle la vue **V**. Ce choix dépend en général du résultat de l'exécution de l'action demandée par l'utilisateur.
 - lui fournir les données dont il a besoin pour générer cette réponse. En effet, celle-ci contient le plus souvent des informations calculées par le contrôleur. Ces informations forment ce qu'on appelle le modèle **M** de la vue.
4. le contrôleur **C** demande à la vue choisie de s'afficher. Il s'agit le plus souvent de faire exécuter une méthode particulière de la vue V chargée de générer la réponse au client.
5. le générateur de vue **V** utilise le modèle **M** préparé par le contrôleur **C** pour initialiser les parties dynamiques de la réponse qu'il doit envoyer au client.
6. la réponse est envoyée au client. La forme exacte de celle-ci dépend du générateur de vue. Ce peut être un flux HTML, PDF, Excel, ...

2. Cycle de vie d'une JSP

La page Jsp est stockée au même niveau que les pages html.

Les composants principaux d'un framework JSP sont :

- Un générateur de source qui prend en entrée la Page.JSP et la transforme en une Servlet.
- Un compilateur java pour compiler la servlet générée.
- Un ensemble de classes de support d'exécution.
- Des outils pour faire le lien entre différents éléments comme des bibliothèques de balises.

Au final, une JSP est transformée en une Servlet. Une JSP est donc transformée en une servlet qui produit du html. La transformation de JSP en servlet est complètement transparente. Voici quelques éléments de compréhension de ce processus :

- La classe (la servlet) générée dérive de **javax.servlet.jsp.HttpJspBase**.
- La méthode **_jspService(HttpServletRequest, HttpServletResponse)**

recupère le code HTML sous forme de chaîne de caractères et l'écrit sur un Writer (JspWriter), comme le ferait une servlet. Elle remplace les méthodes **service** et **doXXX** des servlets .

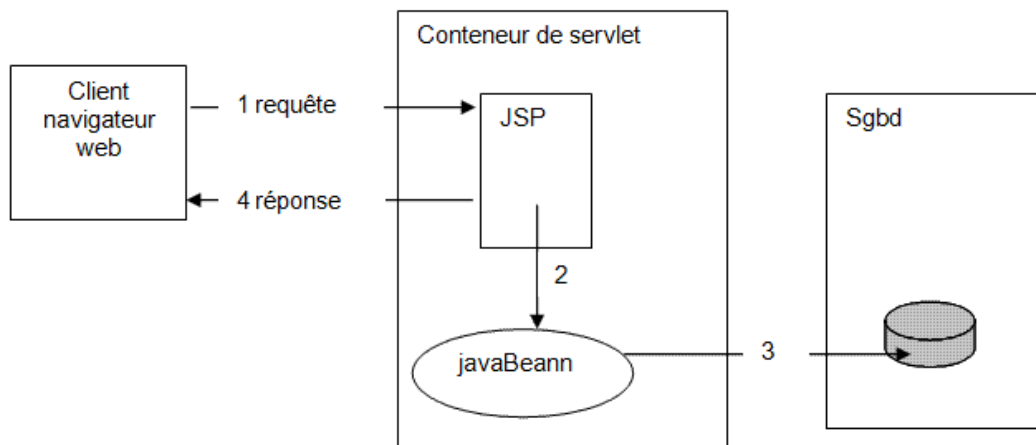
- Les méthodes héritées **jspInit()** et **jspDestroy()** peuvent être redéfinies (dans la page jsp). Elles sont appelées respectivement après la création et avant la suppression de la servlet.

3. Les architectures JSP

Architecture simple

- Une requête venant du browser est envoyée à la page JSP. Elle examine la requête (faire les contrôles comme les authentications) et envoie la réponse au client.
- La séparation entre la présentation et les données est respectée puisque ce sont des JavaBeans qui traitent le contenu.

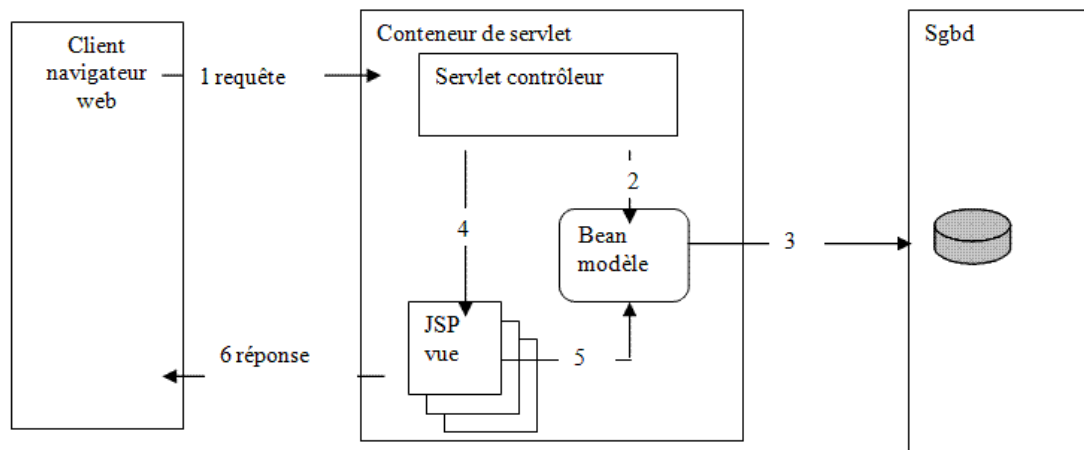
Ce modèle convient aux applications "très" simples. Il est déconseillé car la JSP ne doit pas avoir un rôle de contrôleur mais seulement de vue.



Architecture simple

Architecture complète

- Le contrôleur est une servlet qui examine la requête HTTP (authentications, sessions, ...) et qui instancie les JavaBeans ou les objets utilisés par la vue (présentation). Le contrôleur redirige la requête vers un de ces objets suivant les actions utilisateurs. Il n'y a aucune génération de code HTML dans le contrôleur.
- Le modèle est représenté généralement par des JavaBeans (qui réalisent aussi la couche métier). Ils ne connaissent ni le contrôleur ni les vues.
- La vue (présentation) est représentée par des JSP qui génèrent l'interface utilisateur en HTML/XML. Il n'y a aucun traitement métier java dans la vue.



Architecture complète

4. Structure d'une page JSP : les différentes balises JSP

Une page JSP peut être séparée en plusieurs parties

les données statiques comme le HTML,

- Les données statiques sont écrites dans la réponse HTTP exactement comme elles apparaissent dans le fichier source.

les directives

- Les directives contrôlent la manière dont le compilateur doit générer la servlet
 - `<%@ page import="java.util.*" %> // import`
 - `<%@ page contentType="text/html" %> // contentType`

les variables et code java

les actions

- les actions JSP sont des balises qui appellent des fonctions sur serveur HTTP.
 - `jsp:useBean`, `jsp:include`, `jsp:forward`

les balises personnalisées

- Bibliothèques de balises JSP
 - En plus des actions JSP pré-définies, on peut ajouter des actions personnalisées en utilisant l'API d'extension de balises JSP (JSP Tag Extension API).

Types	Syntaxe	Fonction
Commentaire	<code><!-- Html --></code>	Commentaires visibles dans le Html généré.
	<code><%-- JSP --%></code>	Commentaires visibles dans la JSP seulement.
Directive	<code><%@ page ...%></code>	Définition de la structure de la page JSP
	<code><%@ include ...%></code>	Inclusion d'une page html et/ou JSP. Il s'agit ici d'une copie de code.
Déclaration	<code><%! attribut java %></code>	Déclaration d'attributs pour la servlet générée.

	<code><%! méthode java %></code>	Déclaration de méthodes pour la servlet générée.
Scriptlet	<code><% code java ; %></code>	Insertion de code java dans la méthode service de la servlet générée.
Action	<code><jsp:useBean .../></code>	Instanciation ou récupération de Beans pour utilisation.
	<code><jsp:include .../></code>	Chaînage côté server pour inclure la réponse d'une autre JSP ou servlet dans la jsp appelante.
	<code><jsp:forward .../></code>	Chaînage côté server pour traitement supplémentaire ou génération de la réponse dans une autre jsp ou servlet.



Exemple : Exemple JSP : table de multiplication

```

<html>
<head>
<title>Exemple de JSP : Table de multiplication</title>
</head>
<body>
<!-- Recuperation du parametre table -->
<h1> Table de multiplication des <%=
request.getParameter("table") %> </h1>
<!-- Scriptlet - Java code -->
<table>
<tr><th>Facteurs</th><th>Produit</th></tr>
<%
int t=Integer.parseInt(request.getParameter("table"));
String couleurLigne;
boolean flagCouleur= true;
for(int i = 1; i<11; i++) {
if (flagCouleur) couleurLigne="#eeeeee" ;
else couleurLigne= "#dddddd" ;
flagCouleur = ! flagCouleur;
%>
<tr bgcolor="<%= couleurLigne %>">
<td><%= i %> x <%= t %> </td>
<td><%= i*t %> </td>
</tr>
<%
}
%>
</table>
</body>
</html>

```

Avec le paramètre table = 27 passé à la jsp, le résultat obtenu est le suivant.

Table de Multiplication des 27

Facteurs	Produit
1 x 27	27
2 x 27	54
3 x 27	81
4 x 27	108
5 x 27	135
6 x 27	162
7 x 27	189
8 x 27	216
9 x 27	243
10 x 27	270

Table de multiplication des 27

5. La redirection d'une page JSP

Comme pour les servlets, on retrouve des commandes pour la redirection d'une page JSP.

```
<jsp:forward page="uneAutrePage.jsp" />
```

Cette directive permet d'arrêter l'exécution de la page JSP et de rediriger la requête vers une autre page JSP (ou une servlet).

```
<jsp:include page="unePageJSP.jsp" />
```

Cette directive agit de façon similaire à l'appel d'une sous-routine. Le contrôle est temporairement donné à une autre page, soit un autre fichier JSP, soit un fichier statique. Après le traitement de l'autre page, le contrôle est redonné à la JSP en cours d'exécution. En utilisant cette fonctionnalité, le code Java peut être partagé entre deux pages plutôt que dupliqué.

6. Les javabeans

Un bean est un objet java remplissant des conventions d'écriture et contraintes structurelles. Un bean doit être sérialisable pour garantir la persistance, avoir un constructeur sans argument, permettre les méthodes *get* ou *set* et contenir les méthodes d'écouteurs d'événements nécessaires. Les propriétés d'un bean sont les aspects du beans qui pourront être modifiés lors du déploiement du bean. Elles possèdent un nom, un type et une valeur. Les beans permettent la récupération d'objets de présentation de données en provenance de servlet.

La déclaration

```
<jsp:useBean id="monBean" class="exemples.LesBeans"
scope="session" />
scope="page|request|session|application"
```

La **récupération** des attributs de beans se fait avec l' action :

```
<jsp:getProperty name="monBean" property="cours" />
```

La **modification** des attributs de beans se fait avec l'action :

```
<jsp:setProperty name="monBean" property="cours"
```

```
value="SR03"/>
```

Il y a 4 types de propriétés. Les propriétés simples qui servent à consulter/modifier une valeur. Les propriétés indexées permettent de consulter/modifier une collection de valeurs. Les propriétés liées quand elles sont modifiées, déclenchent un événement prédéfini de type *PropertyChangeEvent*. Enfin, on trouve les propriétés contraintes qui permettent de recevoir une exception en retour du déclenchement de l'événement.

7. Les bibliothèques de balises personnalisées, TagLib

La **bibliothèque de tags** JSTL^{JSTL}, étend la spécification JSP en ajoutant de nouvelles fonctionnalités. JSTL propose une manière de développer différents traitements dans une page JSP sans utiliser du code Java directement. Dans les faits, un tag JSP est une simple balise XML à laquelle on associe une classe Java. Cela croit la lisibilité des JSP.

Principe de fonctionnement des tags

- remplacer les scriptlets (code Java) dans les JSP, par des tags (balises) associés à des classes Java implémentant des interfaces particulières (interface *Tag*, *IterationTag*, *BodyTag*). À la compilation ces tags sont remplacés par ces classes Java.
- un fichier descripteur de taglib (p.ex. *taglib.tld*) fait le mapping entre les tags et les classes Java. (pour JSP et taglib 2.0 on utilise les fichiers DTD).
- Les taglib sont un moyen d'étendre les capacités des serveurs de servlets. Ils permettent de mutualiser la génération de code HTML commun à plusieurs JSP.



Conseil

Il est conseillé d'utiliser les bibliothèques standard développées.

P.ex. *Apache Jakarta TagLib*⁵ est une implémentation open source de JSTL 1.1. Elles proposent des fonctionnalités souvent rencontrées dans les JSP comme les tags de structure (itération, conditionnement ...), internationalisation, exécution de requêtes SQL ou utilisation de documents XML.

5 - [HTTP://jakarta.apache.org/taglibs/doc/standard-doc/intro.html](http://jakarta.apache.org/taglibs/doc/standard-doc/intro.html))

Corba : Common Object Request Broket Architecture

VII

Généralités	125
Interface et le langage IDL	128
Souche et squelette	130
L'ORB (Object Request Broker)	131
L'adaptateur d'objets	131
Invocation d'une méthode dans CORBA	135
Exemple d'une application CORBA : Hello World !	136
Interopérabilité entre ORB	149
Les services CORBA	150

A. Généralités

Corba est une spécification d'un "bus à requêtes". C'est une architecture complexe car elle permet d'assurer la portabilité, et l'interopérabilité entre environnements de fournisseurs différents. Mais aussi de permettre l'écriture d'applications hétérogènes : invoquer depuis du code C++ par exemple des traitements effectués en Java sur la même machine ou sur une autre machine. Ceci en pouvant changer le client ou le serveur de machine, sans avoir à recompiler.

La force de Corba est d'avoir défini des spécifications précises (+ de 1000 pages) de tous les formats d'échange de données. C'est ceci qui permet d'invoquer un objet sur un serveur utilisant l'ORB du fournisseur "A" depuis un code lié avec l'ORB du fournisseur "B". Le client et le serveur pouvant se trouver sur des machines différentes et être écrits dans des langages différents.

1. Objectifs

On peut résumer les objectifs de CORBA en les points suivants :

- Permettre l'interopérabilité entre composants / applications par

l'intermédiaire d'un mode de coopération unifié: Appel d'objets distants

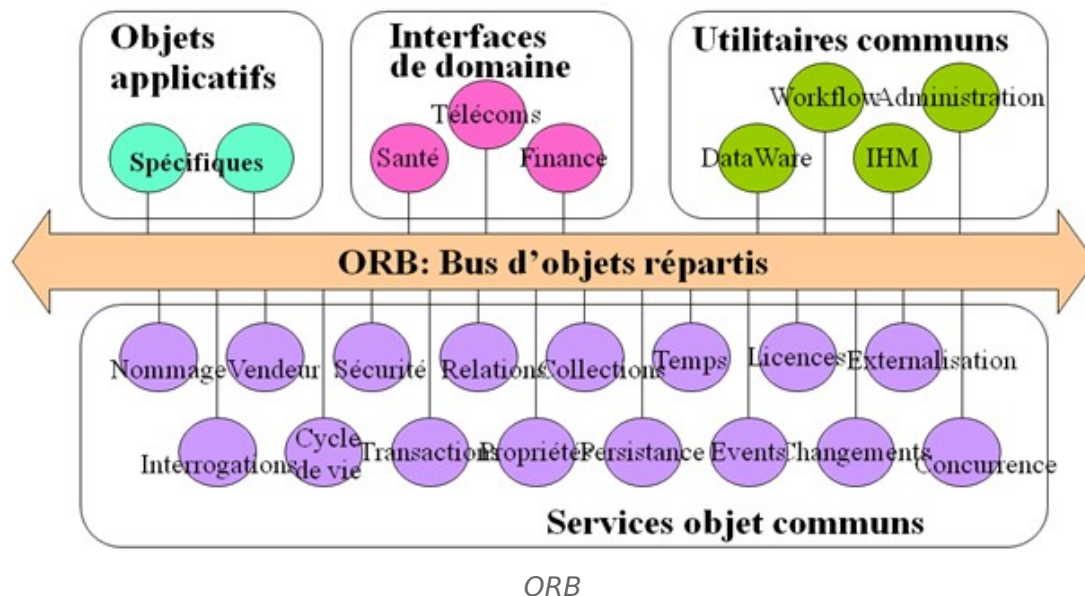
- Gérer l'hétérogénéité des réseaux, machines, systèmes et langages.
- Vision utilisateur = Langage pivot commun IDL
- Offrir une architecture globale.

2. Le modèle d'Objets distribués

En 1989 fut créé le consortium OMG pour promouvoir l'idée:

Orienté Objet + Client Serveur = Objets Distribués

Ainsi fut adopté un standard pour gérer les objets distribués en publiant des spécifications. En outre, l'OMG a développé une architecture globale du modèle d'Objets distribués appelé l'OMA :



L'architecture OMA comporte 4 parties :

- un composant principal l'**ORB** : Object Request Broker (mandataire de requêtes) ;
- des services pour développer les objets distribués : CORBA Services ;
- des **services utilisés par les objets** des applications distribuées : CORBA Facilities ;
- les **applications distribuées** elles-mêmes (clients et serveurs).

Dans ce modèle d'objets distribués, ce sont les objets eux-mêmes qui deviennent **serveur ou client** d'un autre objet. Un objet côté serveur peut lui-même être client d'un objet côté client ! Tous ces objets peuvent résider dans des processus différents sur des **machines différentes, n'importe où sur l'Internet**.

CORBA a défini des protocoles **d'échanges de messages entre** objets. Le respect de ces protocoles permet l'**interopérabilité** entre fournisseurs et entre langages.

3. Gestion des objets distribués

Les objets d'un environnement distribué doivent pouvoir :

- **accepter des requêtes** : c'est l'**interface** IDL qui décrit les requêtes acceptées : l'interface fait office de **contrat** pour les clients en indiquant les **services fournis** par un objet.
- fournir un moyen de créer de nouvelles instances : un constructeur ou *object*

factory.

- gérer leur espace de stockage ;
- gérer leur cycle de vie : création, stockage/persistence, destruction.

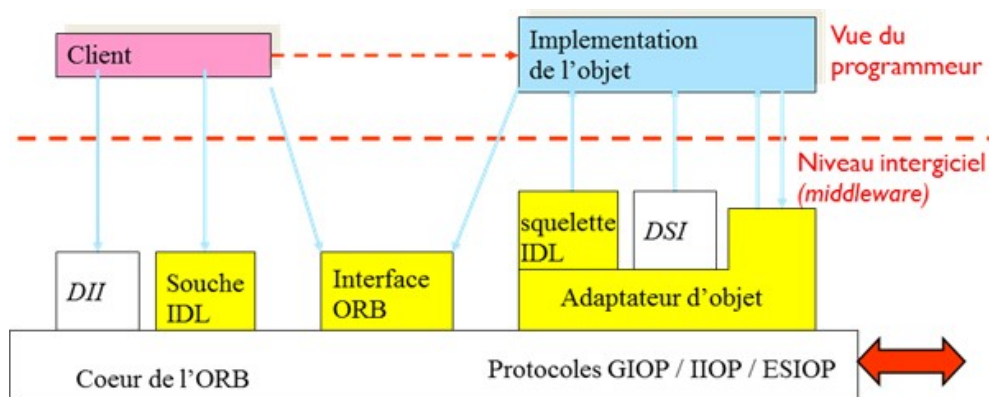
Les objets d'un environnement distribué doivent aussi pouvoir supporter :

- l'**encapsulation** : attributs accessibles seulement à travers les méthodes ;
- l'**héritage** : spécialiser une nouvelle classe en ajoutant ou surchargeant une classe existante ;
- le **polymorphisme** : comportements différents en fonction du type effectif de l'objet dans la hiérarchie d'héritage ;

En bref, conserver toutes les caractéristiques de la programmation objet.

4. Anatomie d'un bus à objets CORBA

La figure suivante résume l'anatomie d'un bus CORBA



Anatomie d'un bus CORBA

Un objet client dispose de deux types d'interfaces pour invoquer des services sur un objet serveur : une interface statique (Souche) générée à partir d'une interface spécifiée avec le langage IDL, et une interface dynamique (DII) permettant la construction des requêtes dynamiquement. De même ces invocations parviennent à l'objet serveur via deux types d'interfaces : un squelette statique généré à partir d'une interface spécifiée avec le langage IDL, et une interface dynamique (DSI). L'adaptateur d'objet a pour rôle de gérer le cycle de vie des objets serveur. L'interface ORB offre des services communs aux objets CORBA. Enfin, le cœur de l'ORB et les protocoles de communication permettent le transport des invocations de méthodes à distance et l'acheminement des résultats tout en garantissant l'interopérabilité.

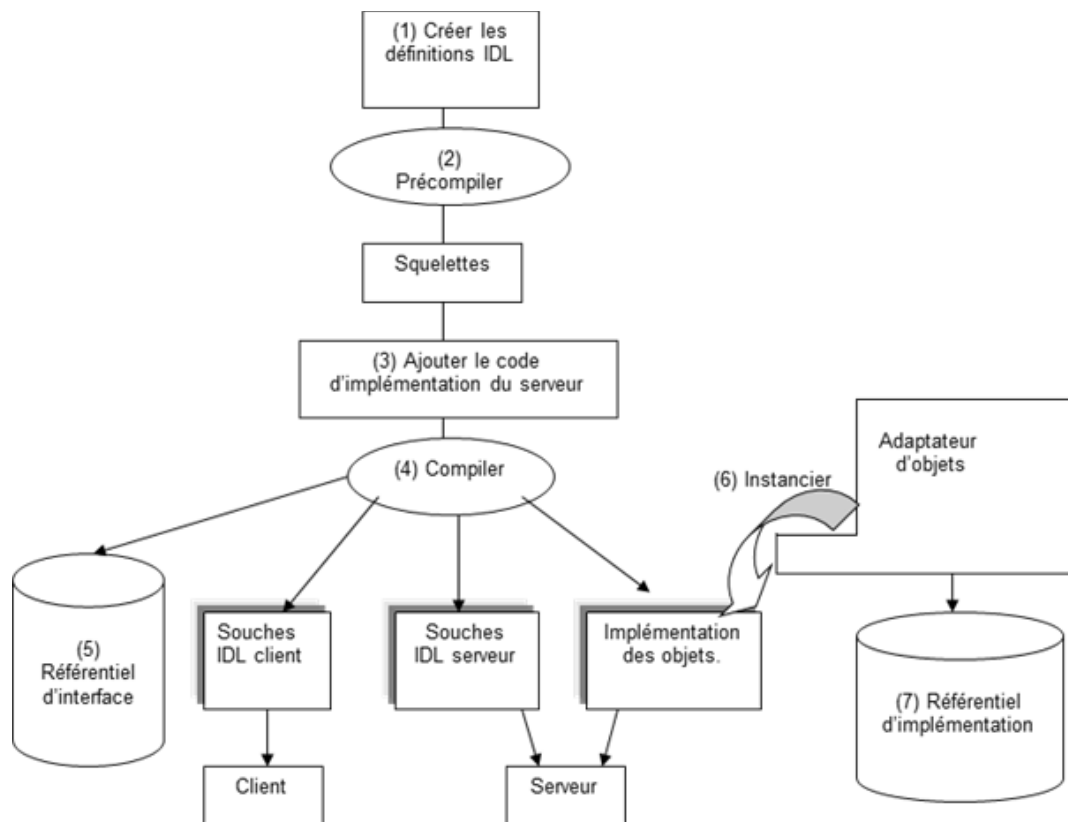
B. Interface et le langage IDL

L'interface d'un objet spécifie son comportement. Elle déclare les services que l'objet est prêt à rendre et cache les détails de l'implémentation grâce à l'encapsulation. Donc, tant que l'objet a le comportement spécifié par son interface, son implémentation peut changer. Ce principe est fondamental pour CORBA. L'OMA adhère au modèle objet "classique" : toutes les méthodes sont contenues dans une classe. Dans CORBA les objets sont connus exclusivement par leur interface, sans rien savoir de leur implémentation.

Pour cela, CORBA inclut un langage IDL qui permet de décrire les interfaces de tous les objets. L'interface est l'équivalent CORBA d'une classe. Les requêtes à un objet

CORBA sont toujours relayées à travers une référence à un objet. Toutes les invocations sont faites sur la référence "ObjRef" d'un objet. IDL n'est PAS un langage de programmation : c'est un langage pour exprimer des types, en particulier des types "interface". Il n'a ni construction de contrôle, ni itérateurs.

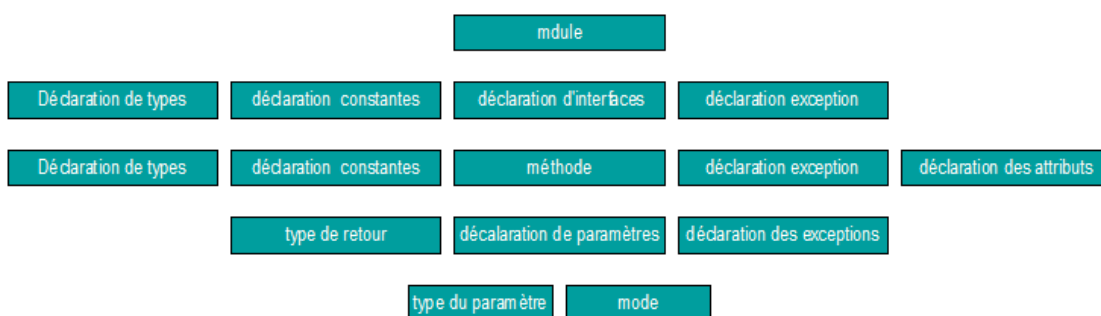
Comme illustré sur la figure suivante, il permet de faciliter le développement d'une application CORBA grâce aux pré-compilateurs qui génèrent les souches clients et squelettes serveurs avec un mappage vers différents langages.



interface et le langage IDL

La figure suivante illustre la structure d'un fichier IDL:

Structure d'un fichier IDL



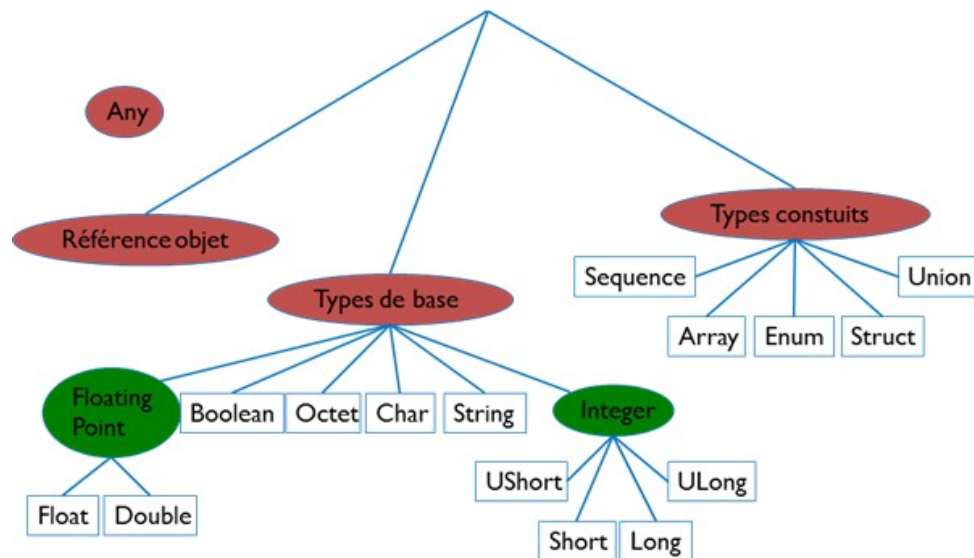
Structure d'un fichier IDL

IDL définit plusieurs types de base et permet d'en définir d'autres plus complexes.

Types de base: boolean, char, octet, enum, short, unsigned short, long, unsigned long, float, double, any, object.

Types construits: string, struct, union, array, sequence.

La figure suivante récapitule les différents types possible avec le langage IDL.



Types possibles avec le langage IDL



Exemple

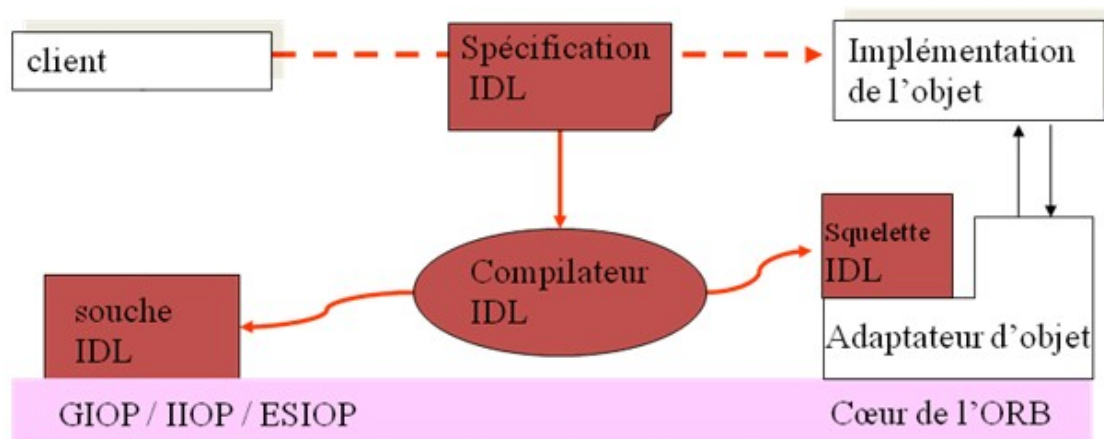
```

interface Person {
    readonly attribute Gender sex;
    readonly attribute Date birthdate;
    attribute string name;
};
interface Employee:Person {
    readonly attribute ssn;
    void addEmployer(Employer emp);
    void deleteEmployer(Employer emp);
    short numEmployers();
    Employer Employer(short index) throws exOutofBounds;
};
    
```

Il existe plusieurs spécification de mappage de l'IDL vers toutes sortes de langages orientés objets et non orientés objets.

C. Souche et squelette

La souche côté client et le squelette côté serveur sont générés (complètement ou partiellement) grâce à un compilateur IDL à partir de la spécification IDL du serveur :

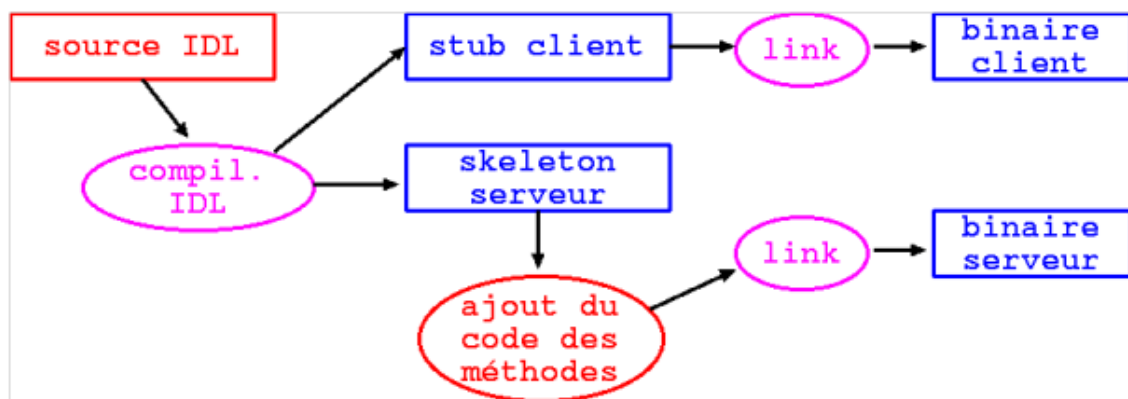


Génération des la souche et du squelette

Les stubs (souches) constituent le mécanisme d'invocation d'un objet, côté client.

Une souche contient les *proxies* des objets distants auxquels le client accède. Elle génère les requêtes à partir de l'invocation de méthode locale, encapsule requête et arguments, et envoie la requête sur le bus (ORB Core). Une souche est *liée* statiquement au code client, les stubs sont produits complets par le compilateur IDL (cf. figure suivante).

Les *skeletons* (squelettes) constituent l'interface côté serveur recevant les requêtes. Un squelette contient un aiguilleur de requêtes compilé (*dispatcher*). Il extrait l'invocation de méthode et les paramètres de la requête reçue du bus et transmet l'invocation de méthode vers le bon objet. Un squelette est *lié* statiquement au code serveur, après avoir été "enrobés" par le code d'implémentation de chaque méthode (d'où leur nom) (cf. figure suivante).



Souche - Squelette

L'implémentation de l'objet serveur peut être dans le même processus, dans un autre processus de la même machine ou dans une autre machine. CORBA prend en charge le transport des requêtes aux objets serveurs quelque soit l'endroit où ils sont situés.

D. L'ORB (Object Request Broker)

ORB[®] est l'intermédiaire qui établit les relations de type Client/Serveur entre des objets. L'ORB a comme rôle de :

- Intercepter l'appel du client.
- Trouver un objet pouvant satisfaire la requête.
- Lui fournir les paramètres.
- Faire appel à sa méthode.
- Renvoyer les résultats.

L'ORB garantit l'interopérabilité: il est indépendant des plateformes matérielles ou logicielles, de sorte que différents ORB peuvent communiquer à travers un protocole "Inter ORB" (xIOP). Par exemple : Visibroker (Microsoft) <--> ORBacus/Linux.

En pratique, l'ORB se présente comme une bibliothèque liée avec le client et avec le serveur, il va effectuer les opérations de:

- transcodage des arguments de l'appel,
- l'envoi des données sur une socket internet (après connexion au serveur),
- récupération des résultats,
- etc.

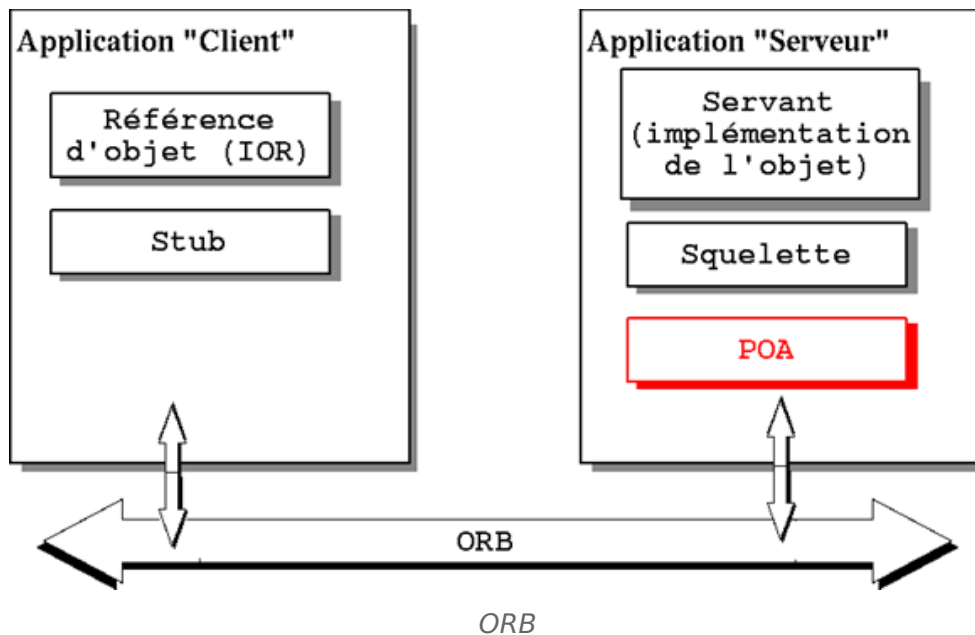
E. L'adaptateur d'objets

Côté serveur, un composant particulier, l'adaptateur d'objets est chargé d'**activer l'implémentation des objets**. Les requêtes passent d'abord par cet adaptateur avant d'être renvoyées vers les méthodes des objets une fois que ceux-ci sont prêts.

Si une interface est responsable de décrire le type de l'implémentation, l'adaptateur d'objet est responsable du "style" de cette implémentation, c'est-à-dire de la façon dont elle va être activée. Il y a 4 styles d'activation :

- **activation par méthode** (*per method*) : un nouveau serveur est démarré à chaque fois qu'une méthode d'un objet est invoquée.
- **activation partagée** (*shared*) : un serveur supporte plusieurs objets actifs simultanément.
- **activation non partagée** (*unshared*) : un serveur ne supporte qu'un seul objet actif. Il peut gérer de nombreuses invocations de méthodes tant que ces méthodes appartiennent à un même objet.
- **serveur persistant** (*persitant*) : le serveur est toujours actif et ne requiert pas d'activation. Il est supposé toujours disponible tant que la machine est opérationnelle.

Le BOA[®] fut l'adaptateur d'objets des anciennes versions de CORBA. En février 1998, la spécification CORBA 2.2 a retiré le BOA au profit du POA. Le POA[®] fournit en plus un ensemble d'outils permettant d'affiner le comportement du serveur



Il a pour rôle d'assurer la **gestion des différents objets** qui se situent à l'intérieur de l'application serveur:

- activer les objets (en fonction des demandes des clients),
- transmettre une invocation d'un client vers un objet particulier,
- créer et détruire des objets,
- déterminer l'état d'un objet,
- ...

Avec le POA on utilise une nouvelle terminologie pour prendre en compte les fonctions supplémentaires. Une application utilisant un POA possède donc les éléments suivants :

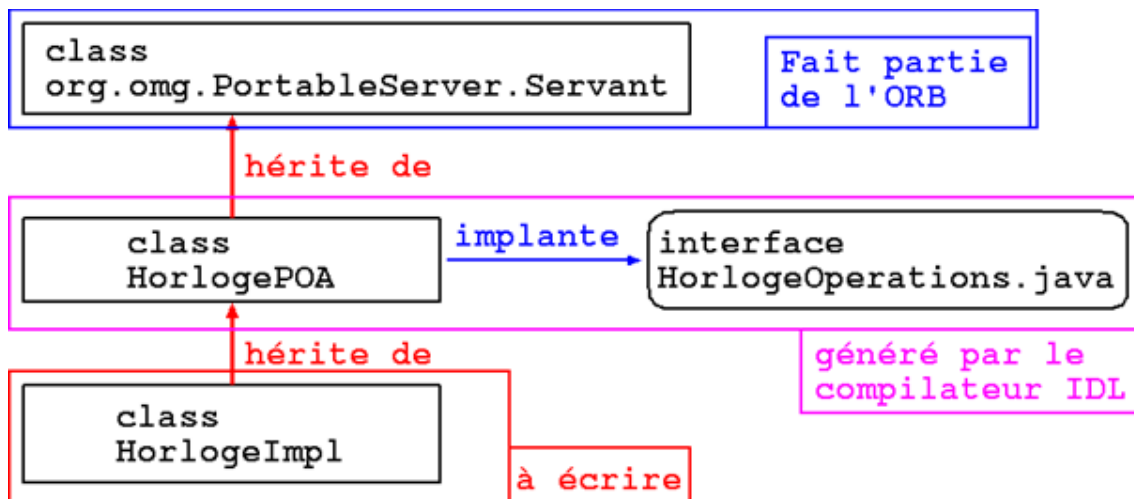
- **servant** : représente l'objet créé sur/par le serveur (anciennement appelé "implémentation de l'objet" ;
- **serveur** : c'est l'application qui loge un ensemble de servants et permet aux applications clientes d'invoquer ces objets ;
- **client** : c'est l'application qui invoque des services proposés par les objets servants ;
- **référence d'objet** : ce que le client voit d'un objet servant par l'intermédiaire de l'interface IDL ; le client utilise une référence d'objet pour accéder à un servant par l'intermédiaire du "stub", de l'ORB, du POA et du squelette qui, ensemble, masquent au client la non-proximité de l'objet réel. Cette référence est appelée IOR.

1. Implémentation d'un servant par héritage

Dans le cas du POA, les fichiers générés par le compilateur IDL sont pour XXX.idl :

- - _XXXStub.java : la souche du client,
- - XXX.java : l'interface XXX,
- - XXXOperations.java : l'interface d'opérations de XXX,
- - XXXPOA.java : le squelette côté serveur,
- - XXXHelper.java et XXXHolder.java les classes Helper et Holder de XXX.

L'implantation du servent va hériter du squelette :



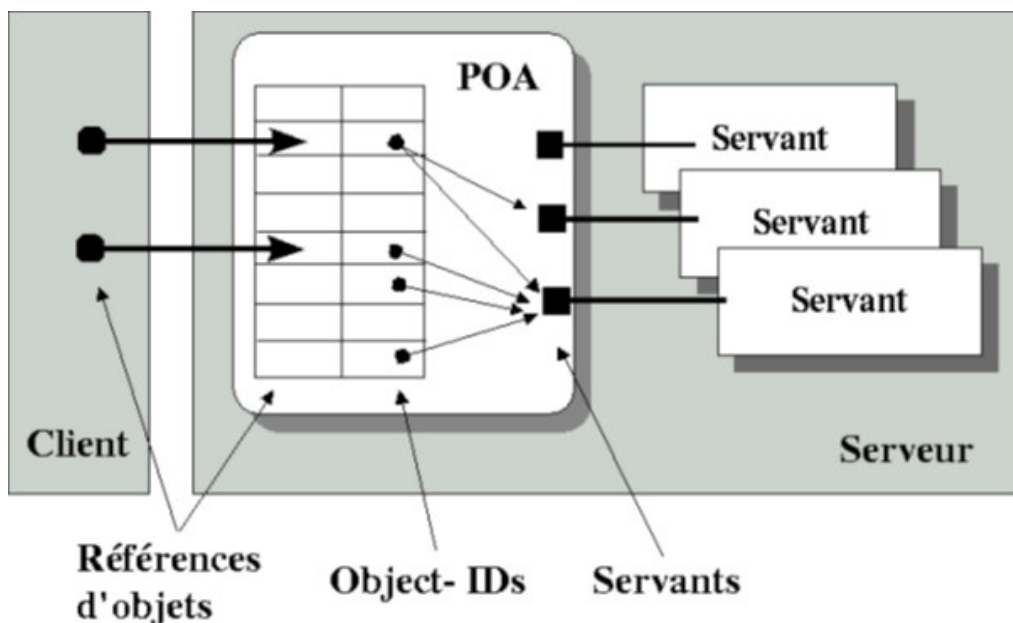
Implantation du servent

Le but est d'introduire une séparation entre la définition fonctionnelle de l'objet Corba (symbolisée par l'interface XXX) et l'implémentation qui en est faite (XXXImpl). Ceci, afin de pouvoir proposer différentes politiques de gestion des instantiations des objets servants au sein du serveur.

Ceci sera fait par les **politiques du POA** :

- politique de threading,
- politique de durée de vie,
- politique d'unicité d'identité d'objet,
- politique d'assignation d'identité,
- politique de rétention des servants,
- politique de gestion des requêtes,
- politique d'activation implicite.

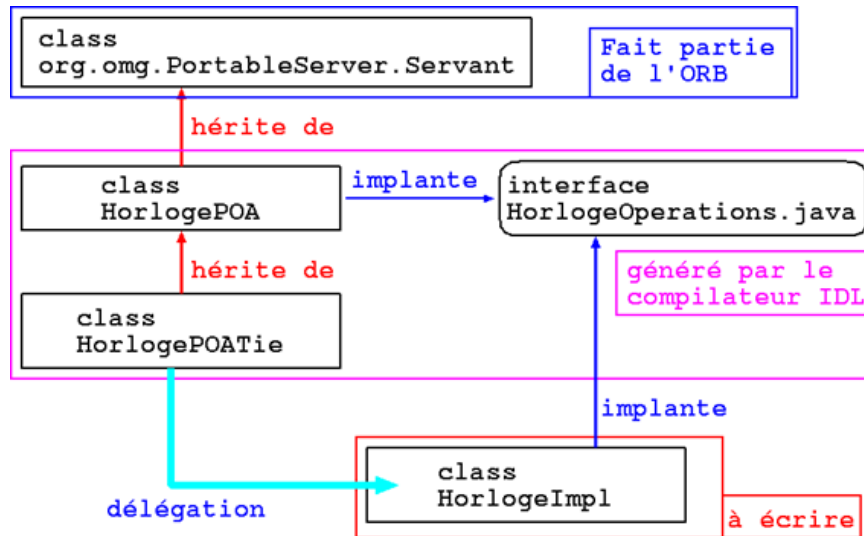
Le POA gèrera une table d'indirection pour localiser les servants :



Localisation des servants

2. Implémentation d'un servent par délégation

Pour les servants développés en Java, on peut utiliser l'approche par délégation :



Approche par délégation

On fait alors passer une référence sur une instance de XXXImpl à une instance de la classe de délégation :

```
HorlogeImpl hor = new HorlogeImpl();
HorlogePOATie tie = new HorlogePOATie(hor);
```

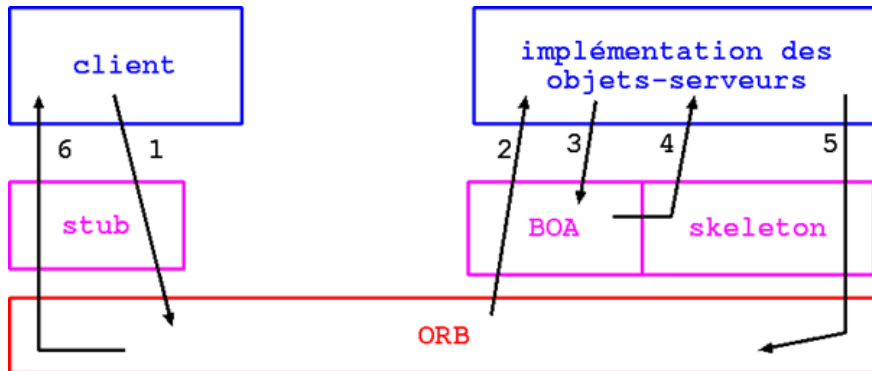
3. Arborescence de POA

L'une des nouveautés du POA est que l'on puisse en avoir plusieurs, organisés en une hiérarchie à partir du RootPOA. Chaque POA peut avoir plusieurs POA-fils.

L'intérêt est lié au fait que chaque POA peut avoir des règles ou politiques de fonctionnement différentes. Chaque POA gère un ensemble de servants et leur transmet les invocations en provenance des clients. La façon dont les servants et les invocations sont gérées est totalement paramétrable. Chaque POA est géré par un **gestionnaire de POA**. Il y a au moins un gestionnaire de POA et il peut être unique et gérer tous les POAs du serveur. Le gestionnaire de POA ("**POAManager**") contrôle le traitement des requêtes en provenance des clients à destination du POA. Suivant l'état dans lequel il est, les requêtes seront rejetées, mises en attente ou traitées. Avant que des requêtes puissent être traitées il faudra donc faire passer le POA dans l'état actif. La classe org.omg.PortableServeur fournit les méthodes pour convertir de l'un vers l'autre les trois types d'identifiants d'objets et pour associer un servent à un POA.

F. Invocation d'une méthode dans CORBA

Une invocation suit le chemin suivant :



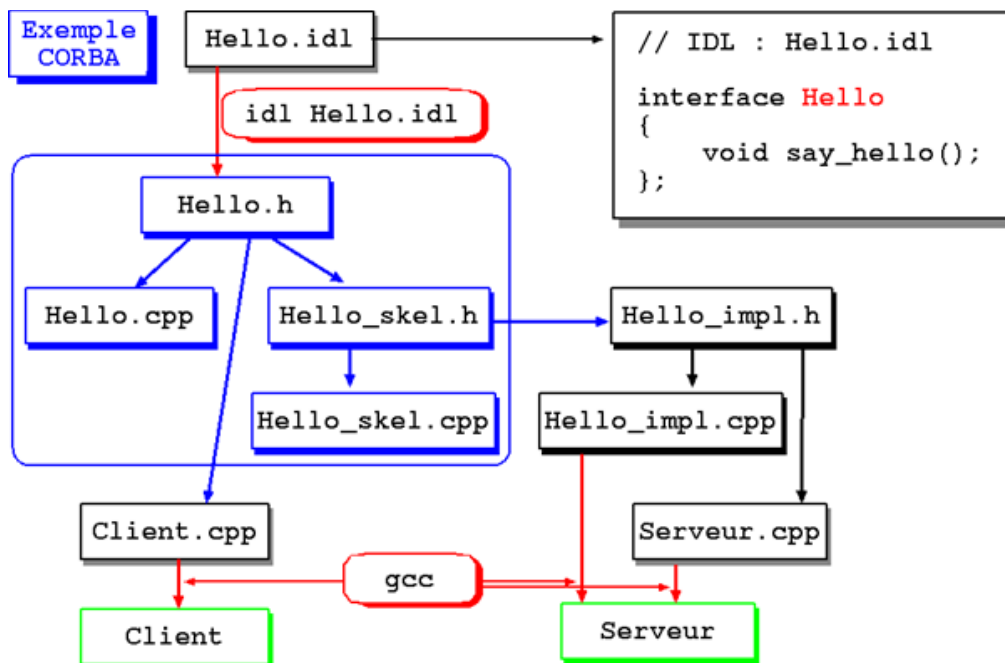
Chemin d'une invocation

1. le client appelle une méthode à travers un stub,
2. l'ORB passe la requête à l'adaptateur qui active l'implémentation,
3. l'implémentation appelle l'adaptateur pour lui indiquer qu'elle est prête,
4. l'adaptateur passe la requête à une méthode de l'implémentation, via le skeleton,
5. l'implémentation renvoie les résultats ou les exceptions au client, à travers l'ORB.

G. Exemple d'une application CORBA : Hello World !

1. Version C++

Nous souhaitons développer l'application décrite avec la spécification IDL suivante :



Spécification IDL

a) Implémentation du serveur

```

// Serveur.cpp
#include <OB/CORBA.h>
#include <Hello_impl.h>
#include <fstream.h>
#include <iostream.h>
int run(CORBA::ORB_ptr);

int main(int argc, char* argv[])
{
    int status = EXIT_SUCCESS;
    CORBA::ORB_var orb;
    try
    {
        orb = CORBA::ORB_init(argc, argv);
        status = run(orb);
    }
    catch(const CORBA::Exception&)
    {
        status = EXIT_FAILURE;
    }
    //
    if(!CORBA::is_nil(orb))
    {
        try { orb -> destroy(); }
        catch(const CORBA::Exception&)
        {
            status = EXIT_FAILURE;
        }
    }
    return status;
}

```

inclusion des symboles de la bibliothèque ORB et des définitions générées par le compilateur IDL

initialiser l'ORB

puis, appeler notre fonction de travail

détruire la connexion à l'ORB

```
//----- run() -----
int run(CORBA::ORB_ptr orb)
{
    CORBA::Object_var poaObj =
        orb -> resolve_initial_references("RootPOA");
    PortableServer::POA_var rootPoa =
        PortableServer::POA::_narrow(poaObj);
    PortableServer::POAManager_var manager =
        rootPoa -> the_POAManager();
    1

    Hello_impl* helloImpl = new Hello_impl();
    PortableServer::ServantBase_var servant = helloImpl;
    Hello_var hello = helloImpl -> _this();
    2

    CORBA::String_var s = orb -> object_to_string(hello);
    const char* refFile = "Hello.ref";
    ofstream out(refFile);
    out << s << endl;
    out.close();
    3
    Créer la forme
    "stringifiée" de
    l'identifiant de
    l'objet serveur.

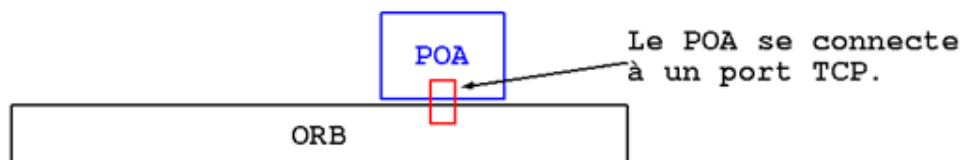
    manager -> activate();
    orb -> run();
    4

    return EXIT_SUCCESS;
}
```

Dans ce qui suit nous donnons des explications des quatre étapes composant le code du serveur :

```
//----- run() -----
int run(CORBA::ORB_ptr orb)
{
    CORBA::Object_var poaObj =
        orb -> resolve_initial_references("RootPOA");
    PortableServer::POA_var rootPoa =
        PortableServer::POA::_narrow(poaObj);
    PortableServer::POAManager_var manager =
        rootPoa -> the_POAManager();
    1
```

.....Récupérer une référence sur le RootPOA et la transformer en objet de type POA.
Puis récupérer la référence sur le manager du POA.



Etape 1

```
//----- run() -----
int run(CORBA::ORB_ptr orb)
{ .....
```

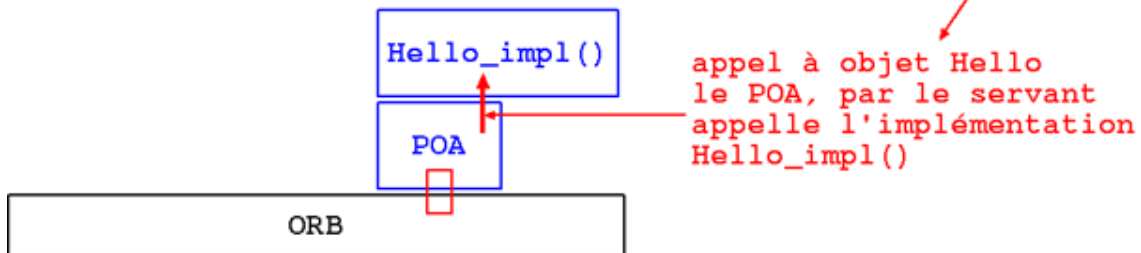
```
    Hello_impl* helloImpl = new Hello_impl();
    PortableServer::ServantBase_var servant = helloImpl;
    Hello_var hello = helloImpl -> _this();
```

2

```
    ...
```

```
}
    Instancier un objet d'implémentation.
    Déclarer que cet objet est un servant du POA.
```

Ce servant va incarner un "objet CORBA" par l'appel à `_this()`.



Etape 2

```
//----- run() -----
int run(CORBA::ORB_ptr orb)
{ .....
```

```
    CORBA::String_var s = orb -> object_to_string(hello);
    const char* refFile = "Hello.ref";
    ofstream out(refFile);
    out << s << endl;
    out.close();
```

3

```
    ...
```

```
}
    Créer la forme "stringifiée" de l'identifiant de l'objet serveur.
```

```
ex-echo> ls Hello.ref
Hello.ref
```

```
ex-echo> more Hello.ref
```

```
IOR:01aeefbf0e00000049444c3a48656c6c6f3a312e30000000010000
```

```
....
```

Pour accéder à un objet CORBA sur le serveur, le client a besoin d'une référence CORBA. Ici, on passe cette référence au client par l'intermédiaire de sa forme "string" dans un fichier. Dans une "vraie" application, on utilise un service de nommage (annuaire).

Etape 3

```
//----- run() -----
int run(CORBA::ORB_ptr orb)
{ .....
    manager -> activate();
    orb -> run();
    return EXIT_SUCCESS;
}
```

On active le manager pour lui indiquer qu'il peut accepter les requêtes des clients.

4

On appelle la boucle d'attente de l'ORB.

Il n'y a pas de retour de cette fonction, sauf si une fonction exécute un "shutdown" de l'ORB.

Après cet appel, le serveur est en attente des appels des clients.

Tout le traitement se fait dans les fonctions des objets serveurs déclarés (comme les callbacks d'une interface graphique ou comme un serveur RPC ou un serveur JavaRMI).

Etape 4

b) Implémentation du Servant

```
// C++ Hello_impl.h
// dérivée du skeleton POA_Hello définie dans Hello_skel.h
#include <Hello_skel.h>

class Hello_impl : public POA_Hello, public
    PortableServer::RefCountServantBase
{
public:
    virtual void say_hello()
        throw(CORBA::SystemException);
};
```

```
// C++ Hello_impl.cpp
// code de la classe d'implémentation
#include <iostream.h>
#include <OB/CORBA.h>
#include <Hello_impl.h>

void Hello_impl::say_hello()
    throw(CORBA::SystemException)
{
    cout << "Hello World!" << endl;
}
```

Implémentation du servant

c) Implémentation du Client

```
// C++ Client.cpp
#include <OB/CORBA.h>
#include <Hello.h>
#include <fstream.h>
int run(CORBA::ORB_ptr);

int main(int argc, char* argv[])
{ // Same as the server
  int status = EXIT_SUCCESS;
  CORBA::ORB_var orb;
  try
  { orb = CORBA::ORB_init(argc, argv);
    status = run(orb);
  }
  catch(const CORBA::Exception&)
  { status = EXIT_FAILURE;
  }
//
  if(!CORBA::is_nil(orb))
  { try { orb -> destroy(); }
    catch(const CORBA::Exception&)
    { status = EXIT_FAILURE; }
  }
  return status;
}
```

inclusion des symboles de la bibliothèque ORB et des définitions générées par le compilateur IDL

initialiser l'ORB

puis, appeler notre fonction de travail

détruire la connexion à l'ORB

```
//----- run() -----
int run(CORBA::ORB_ptr orb)
{
  const char* refFile = "Hello.ref";
  ifstream in(refFile);
  char s[2048];
  in >> s;
  CORBA::Object_var obj = orb -> string_to_object(s);
  Hello_var hello = Hello::_narrow(obj);
  hello -> say_hello();
  return 0;
}
```

Récupération de la forme "stringifiée" de l'identifiant de l'objet serveur.

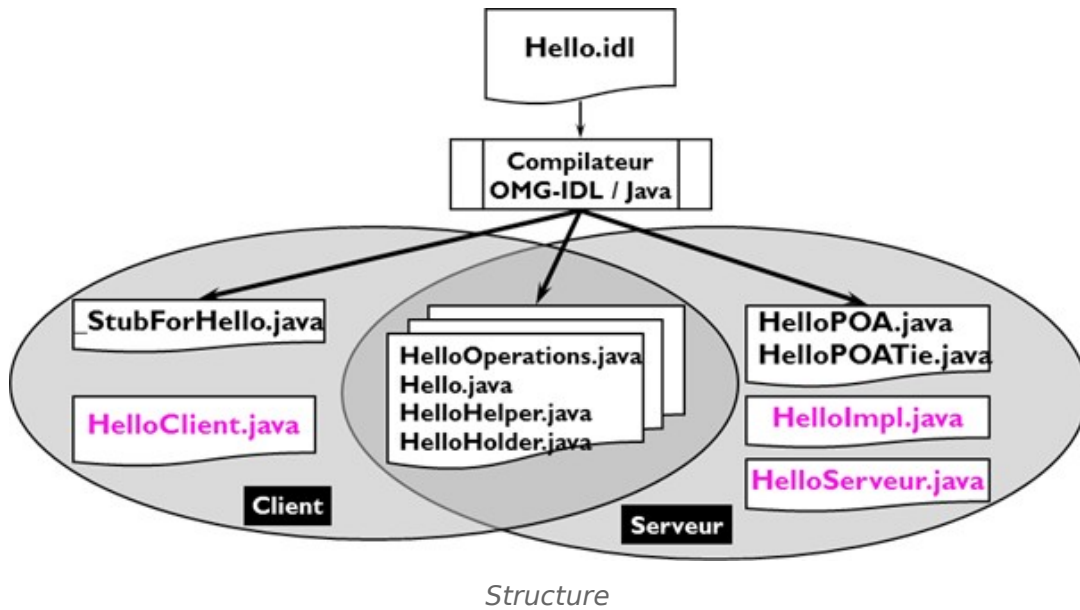
Cet identifiant stringifié est transformé en référence sur un objet

La référence sur un objet est convertie en une référence sur un objet de type Hello

Sur cet objet de type Hello on peut appliquer la fonction déclarée dans l'interface.

Les variables Xxx_var sont des "smart pointers" qui relâchent la mémoire allouée dès que la variable n'est plus visible ("out of scope").

2. Version en Java



a) Implémentation du Serveur

```
// Java hello/Server.java
package hello;

public class Server
{
    public static void main(String args[])
    {
        java.util.Properties props = System.getProperties();
        props.put("org.omg.CORBA.ORBClass",
                  "com.ooc.OBServer.ORB");
        props.put("org.omg.CORBA.ORBSingletonClass",
                  "com.ooc.CORBA.ORBSingleton");
        int status = 0;
        org.omg.CORBA.ORB orb = null;
        try
        { orb = org.omg.CORBA.ORB.init(args, props);
          status = run(orb);
        }
        catch(Exception ex)
        { ex.printStackTrace(); status = 1; }
        if(orb != null)
        { try
          { orb.destroy(); }
          catch(Exception ex)
          { ex.printStackTrace(); status = 1; }
        }
        System.exit(status);
    }
}
```

```

//----- run() -----
static int run(org.omg.CORBA.ORB orb)
    throws org.omg.CORBA.UserException
{
    org.omg.PortableServer.POA rootPOA =
    org.omg.PortableServer.POAHelper.narrow(
        orb.resolve_initial_references("RootPOA"));

    org.omg.PortableServer.POAManager manager =
        rootPOA.the_POAManager();

    Hello_impl helloImpl = new Hello_impl();
    Hello hello = helloImpl._this(orb);

    try
    { String ref = orb.object_to_string(hello);
      String refFile = "Hello.ref";
      java.io.PrintWriter out = new java.io.PrintWriter(
          new java.io.FileOutputStream(refFile));
      out.println(ref);
      out.close();
    }
    catch(java.io.IOException ex)
    { ex.printStackTrace(); return 1; }

    manager.activate();
    orb.run();

    return 0;
}

```

Diagram annotations:

- A blue arrow points from `orb.object_to_string(hello)` to a box labeled **client**.
- A blue arrow points from `manager.activate(); orb.run();` to a box containing the text: **mettre le serveur en attente des appels des clients**.

b) Implémentation du Servant

```
// Java Hello_impl.java
package hello;

public class Hello_impl extends HelloPOA
{
    public void say_hello()
    {
        System.out.println("Hello World!");
    }
}

// Generated by the ORBacus IDL to Java Translator
public abstract class HelloPOA
...
public org.omg.CORBA.portable.OutputStream
    _invoke(String opName,
    .....
        final String[] _ob_names =
        { "say_hello" };
        while(_ob_left < _ob_right)
            int _ob_res = _ob_names[_ob_m].compareTo(opName);

        switch(_ob_index)
        { case 0: // say_hello
        .....

```

Implémentation du servant

c) Implémentation du Client

```
// Java 2 Client.java
package hello;

public class Client
{
    public static void main(String args[])
    {
        // Same as the server
        java.util.Properties props = System.getProperties();
        props.put("org.omg.CORBA.ORBClass",
                  "com.ooc.OBServer.ORB");
        props.put("org.omg.CORBA.ORBSingletonClass",
                  "com.ooc.CORBA.ORBSingleton");
        int status = 0;
        org.omg.CORBA.ORB orb = null;
        try
        {
            orb = org.omg.CORBA.ORB.init(args, props);
            status = run(orb);
        }
        catch(Exception ex)
        {
            ex.printStackTrace(); status = 1; }

        if(orb != null)
        {
            try
            {
                orb.destroy();
            }
            catch(Exception ex)
            {
                ex.printStackTrace(); status = 1; }
        }
        System.exit(status);
    }
}

//----- run() -----

static int run(org.omg.CORBA.ORB orb)
{
    org.omg.CORBA.Object obj = null;
    try
    {
        String refFile = "Hello.ref";
        java.io.BufferedReader in = new java.io.BufferedReader(
            new java.io.FileReader(refFile));
        String ref = in.readLine();
        obj = orb.string_to_object(ref);
    }
    catch(java.io.IOException ex)
    {
        ex.printStackTrace();
        return 1; }

    Hello hello = HelloHelper.narrow(obj);

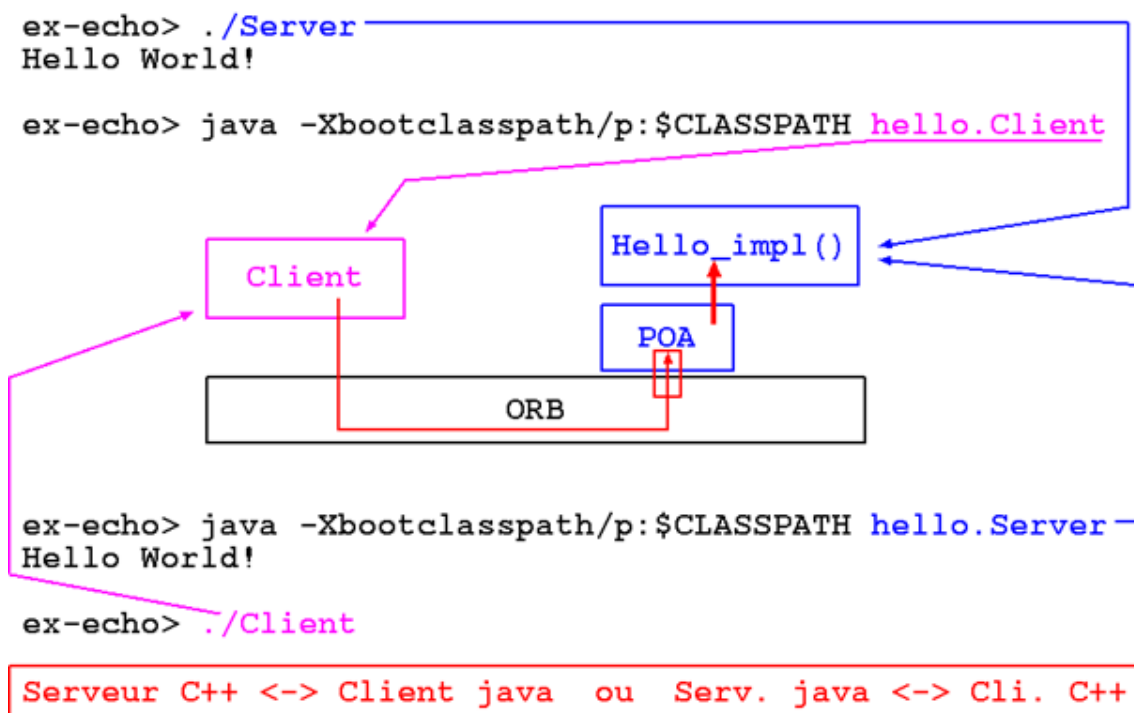
    hello.say_hello();

    return 0;
}
}
```

Diagram annotations:

- Annotation for `obj = orb.string_to_object(ref);`: récup. réf sur objet
- Annotation for `obj`: convertir référence en objet de type Hello
- Annotation for `hello.say_hello();`: invoquer la fonction de l'objet Hello instancié par le serveur

d) Exécution de l'application côté Serveur et Client



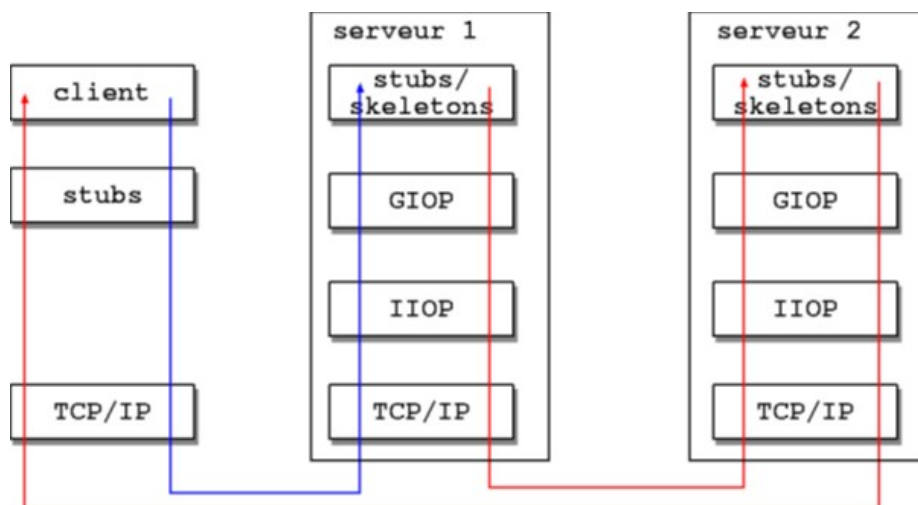
Execution

H. Interopérabilité entre ORB

L'implémentation de CORBA étant libre, il faut un ensemble de protocoles bien spécifiés pour que deux implémentations puissent interagir. C'est le rôle de GIOP, qui spécifie le type et le format des messages que les ORBs doivent s'envoyer pour coopérer.

La sous-couche IIOP permet de "projeter" GIOP sur TCP/IP.

GIOP + IIOP => obligatoires pour une interopération entre ORB "out-of-the-box".



I. Les services CORBA

Un ORB CORBA doit fournir plus que le simple mécanisme de passage de messages dont les objets ont besoin pour communiquer entre eux à travers des langages, des outils, des plateformes et des réseaux, tous hétérogènes. Il fournit aussi l'environnement pour gérer ces objets, annoncer leur présence sur le réseau et décrire leurs métadonnées.

Tout ceci est mis en oeuvre par une série de "**services CORBA**", que presque tous les objets CORBA seront amenés à utiliser. On y trouve par exemple : **notification d'évènements**, persistance, cycle de vie, **nommage**, contrôle des accès concurrents, **transactions**, collections, temps, sécurité, etc ...

Web services

VIII

Architecture des Web Services	152
SOAP : Simple Object Access Protocol	155
WSDL : Web Service Description Language	162
UDDI : Universal Description Discovery and Integration	167

Un web service est une entité qui peut **échanger des documents** avec le monde extérieur. Cette entité est **auto descriptive** et possède une **identité unique** (Systinet).

Le consortium W3C définit un web service par : un système logiciel identifié par un URI, dont les interfaces publiques et les liaisons sont définies et décrites en XML. Sa définition peut être découverte [dynamiquement] par d'autres systèmes logiciels. Ces autres systèmes peuvent ensuite interagir avec le web service d'une façon décrite par sa définition, en utilisant des messages XML transportés par des protocoles Internet.

Cette définition peut être complétée par la définition des termes clés suivants :

- Système logiciel : un programme
- URI (RFC2396)
 - URL: <http://www.utc.fr>
 - <mailto:info@utc.fr>
 - <ftp://diabolo.gi.utc>
- Interface : une description des opérations proposées par le composant logiciel.
- Liaisons : spécification du protocole et du format des données utilisés pour échanger des messages en vue de l'utilisation d'une interface.
- Découverte (dynamique) : **obtention de la description d'un service web.**
- XML (<http://www.w3.org/XML/>).
- Protocoles internet :
 - HTTP (web)
 - SMTP (mail)
 - etc.

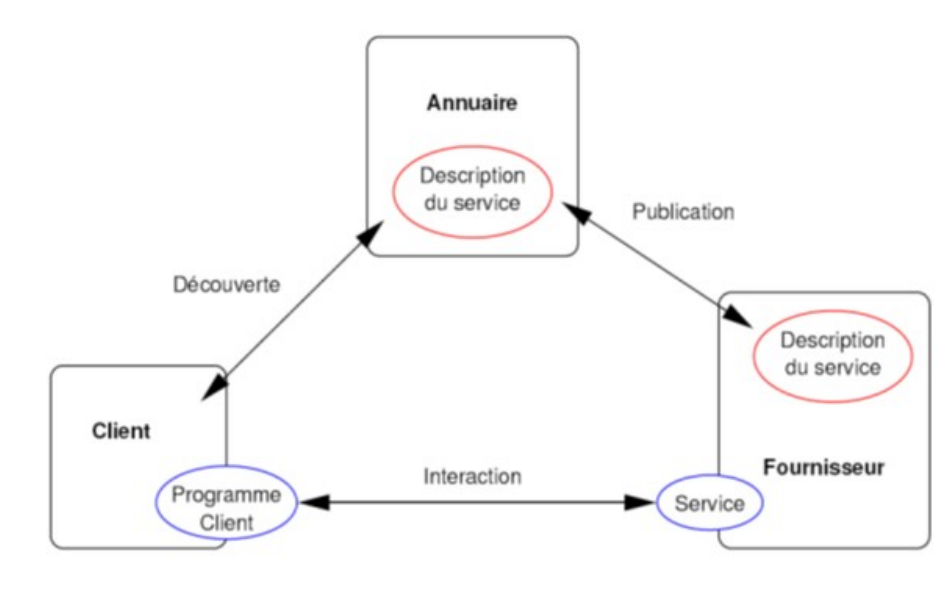
Le but des web services est de fournir une architecture générale pour les applications réparties sur internet qui soient :

- Interopérables : basées sur des standards ouverts sans composant spécifique à un langage ou un système d'exploitation
- faiblement couplées : limiter au maximum les contraintes imposées sur le modèle de programmation des différents éléments de l'application. Par exemple ne pas imposer un modèle objet

- « Déployables » à large échelle : le web.

A. Architecture des Web Services

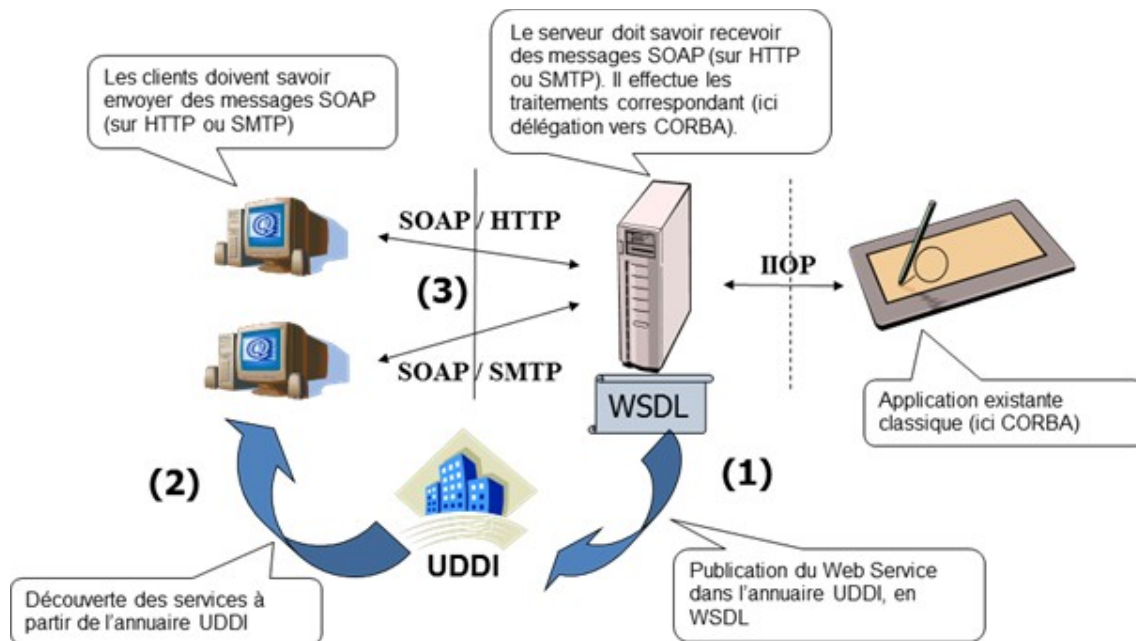
La figure suivante illustre les trois acteurs principaux d'une architecture d'un web service :



Principaux acteurs d'une architecture web-service

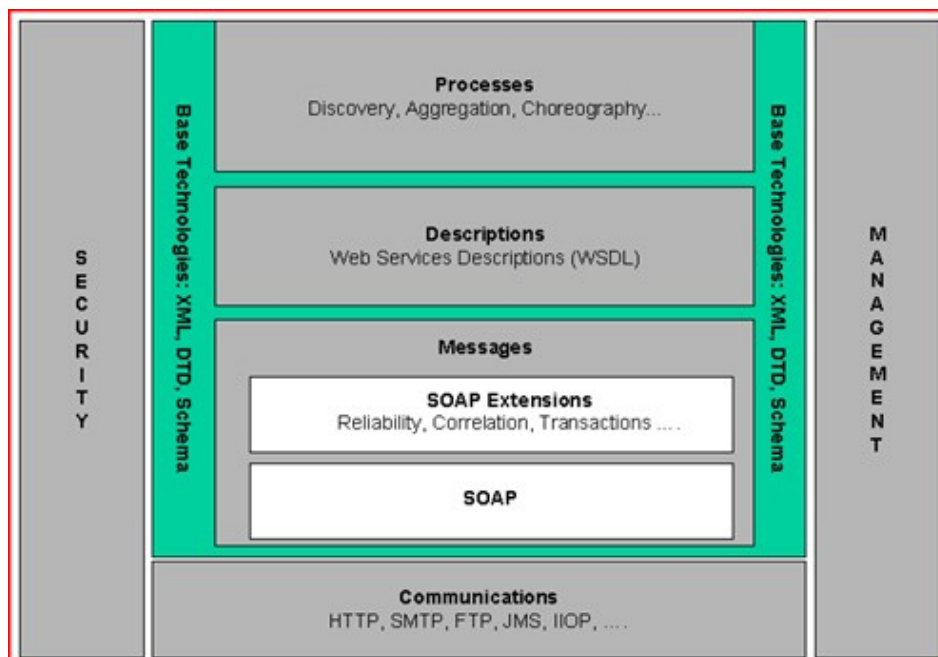
Le fournisseur de service publie la description du service en WSDL sur un annuaire public ou privé UDDI. Le client peut alors effectuer une recherche et découvrir le service. Cette description permettra au client d'interagir avec le service à travers des messages SOAP. Ces messages SOAP sont basés sur le langage XML et transportés par des protocoles Internet comme HTTP, SMTP, etc.

La figure suivante illustre une telle interaction entre ces trois acteurs principaux d'un web service.



Interactions entre ces acteurs

La figure suivante donne une description de la pile de l'architecture des web services telle qu'elle est définie par W3C :



Pile de l'architecture des web services

Chaque couche constitue une brique dont le rôle est bien précis [2]. Libero Maesano, Christian Bernard, Xavier Le Galles :

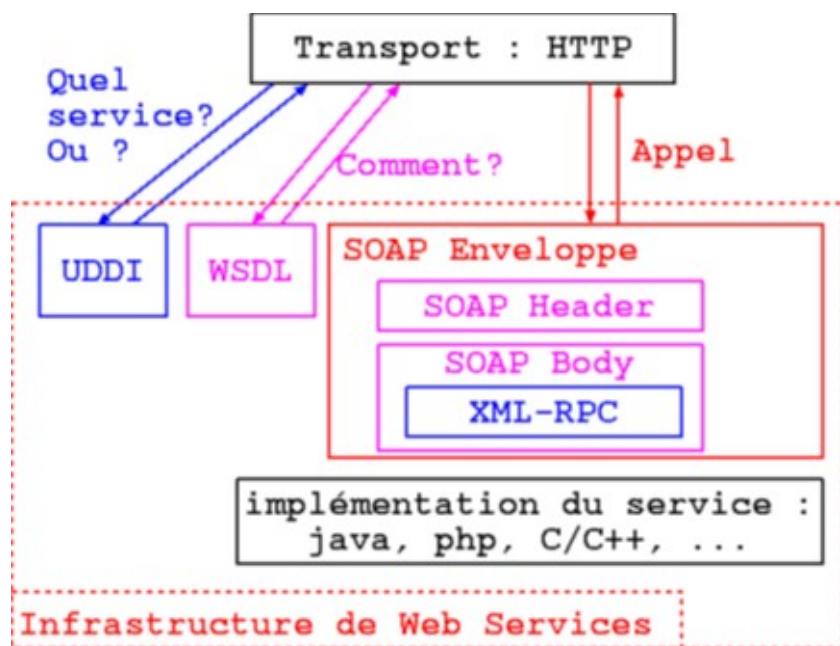
En bas de la pile se trouvent les protocoles de transport Internet pour l'acheminement des messages. La couche messages, permet de mettre en œuvre l'interaction entre les clients et serveurs. Ces messages SOAP sont basés sur le langage XML représente une technologie de mise en œuvre des web services.

Le niveau qui vient juste après concerne la description des services en termes d'interfaces, formats des données, opérations et liaisons implémentant ces services. Ceci est fait par le biais d'un langage de description. Bien qu'aucun langage

particulier ne soit imposé par W3C, le plus utilisé reste WSDL et les applications se voulant être des web services doivent avoir des descriptions en WSDL [2].
Libero Maesano, Christian Bernard, Xavier Le Galles :[]

Puis vient la couche se chargeant de la publication des interfaces et points d'accès des web services dans des annuaires tels qu'UDDI (Le plus utilisé de nos jours mais qui n'est pas "le standard"). Cela permet aux clients de les découvrir et de les localiser pour les utiliser. Dans cette même couche peuvent se faire des opérations de composition des services, de définition des relations entre les différents services associés à des rôles (choregraphy) en se basant sur la description de la couche d'en dessous (la couche de description).

La figure suivante récapitule les composants d'une infrastructure de web services et leurs interactions.



Cette pile peut intégrer des mécanismes de gestion et de sécurité qui sont mis en œuvre à travers toutes les couches ou presque.

Nous donnerons, dans ce qui suit, une description de ces différentes technologies et protocoles utilisés dans les web services.

B. SOAP : Simple Object Access Protocol

1. Définition

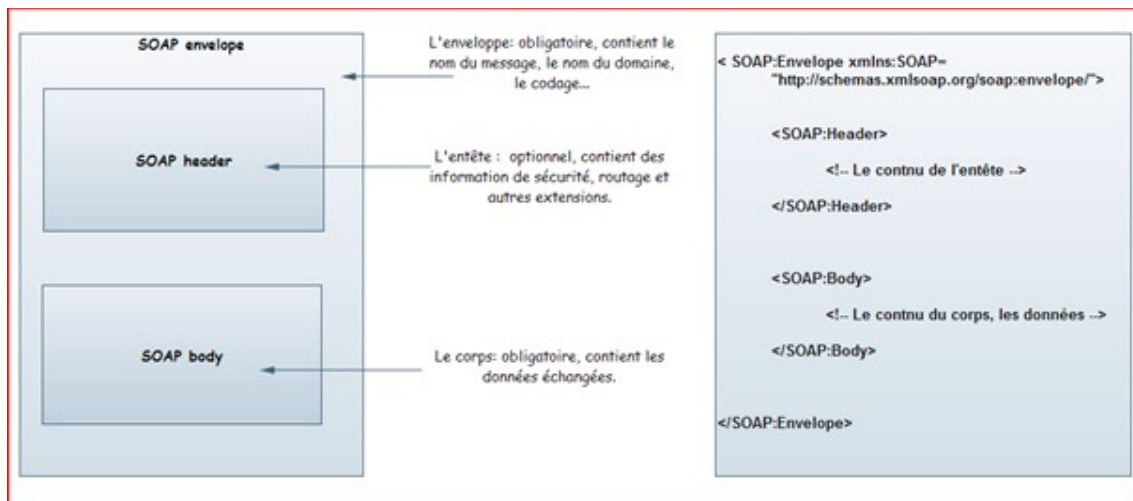
Le protocole SOAP est un standard du consortium W3C se basant sur XML permettant les échanges d'informations dans des environnements décentralisés. SOAP est basé sur XML et consiste en :

- un cadre de description du contenu du message et la manière avec laquelle il doit être traité,
- un ensemble de règles d'encodage pour interpréter des types définis par les applications,
- et une convention pour représenter les appels RPC et leur réponses.

SOAP peut être utilisé *potentiellement* avec une combinaison variée de protocoles de transport Internet comme http, FTP, SMTP.

2. Structure d'un message SOAP

Un message SOAP est composé de deux parties, un "*header*" facultatif (ou entête) pour des besoins d'extension (mécanismes de sécurité, de routage dans le cas où le message passe par des intermédiaires, ...) et un "*body*" obligatoire pour contenir le message proprement dit (le nom de la procédure d'un RPC avec les paramètres,...). Le tout est mis dans une enveloppe ou "*envelope*" qui permet de spécifier un codage spécifique, standard ou défini par l'utilisateur mais compréhensible par les entités l'utilisant. SOAP donne aussi la possibilité de signaler des erreurs lors du traitement des messages dans la partie "*body*". La figure suivante illustre la structure d'un message SOAP.



Structure d'un message SOAP



Exemple

StockQuote est un ensemble de services qui permet d'obtenir des informations sur des actions boursières. GetLastTradePrice est l'opération qui permet de connaître la dernière valeur d'une action. Cet exemple présente un échange de messages entre un client qui veut savoir la valeur de l'action « DIS ».

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Propre au portage sur HTTP

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse xmlns:m="Some-URI">
      <Price>34.5</Price>
    </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Propre au portage sur HTTP



Exemple : Signalement d'une erreur avec SOAP Fault

C'est une balise permettant de signaler des cas d'erreur. La balise Fault contient les balises suivantes :

- faultcode : un code permettant d'identifier le type d'erreur
 - Client, Server, VersionMismatch, MustUnderstand
- faultstring : une explication en langage naturel
- faultactor : une information identifiant l'initiateur de l'erreur
- detail : Définition précise de l'erreur.

```
<?xml version="1.0" encoding='ISO-8859-1'?>
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  <env:Body>
    <env:Fault>
      <env:faultcode>env:Server</env:faultcode>
      <env:faultstring>'xx' is not a valid integer value</env:faultstring>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

3. Encodage

Un message SOAP contient des données typées. Il faut donc définir un moyen d'encoder ces données. On distingue les valeurs de type simple, et les valeurs de type complexe.

SOAP définit une représentation en XML de son modèle de données (un *encoding*) : pour l'utiliser, il faut définir l'attribut `env:encodingStyle` et lui donner la valeur `http://schemas.xmlsoap.org/soap/encoding/` (noté `enc`). Le modèle est facultatif : on peut utiliser une autre représentation, mais il faut que l'émetteur et le récepteur la comprennent. La représentation est celle des schémas du W3C, avec des extensions pour la notion de type composé. Toute valeur est représentée comme le contenu d'un élément.

a) Encodage des valeurs de type simple

Quand la valeur correspond à un type de base, celui-ci est précisé directement ou indirectement par l'élément :

- directement par l'attribut `type` du namespace des schémas `http://www.w3.org/1999/XMLSchema-instance` (noté `xsi`) dont la valeur correspond à un type fondamental des schémas `http://www.w3.org/1999/XMLSchema` (noté `xsd`)
Exemple: `<cost xsi:type="xsd:float">29.5</cost>`
- directement par le nom de l'élément (choisit dans le namespace `enc`)
`<enc:int>45</enc:int>`
- indirectement par le type du tableau (le cas échéant)

b) Les types composés

On distingue deux types principaux : les structures et tableaux.

Une structure est un type composé dans lequel les membres sont accessibles uniquement grâce à des noms différents.



Exemple : Structure

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<pers:personne
  xmlns:pers="http://apiacoa.org/teaching/xml/personnes"
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  <nom xsi:type="xsd:string">Jean</nom>
  <prenom xsi:type="xsd:string">Dupont</prenom>
</pers:personne>
```

Un tableau est un type composé dans lequel les membres sont accessibles uniquement grâce à leur position. Le type des éléments n'a pas besoin d'être uniforme. Sa traduction XML consiste en:

- un élément de type dérivé de `enc:Array`
- portant un attribut `enc:arrayType` qui précise le type des éléments et la taille du tableau (sous la forme `[n]`)
 - le type `xsd:ur-type` peut servir de joker pour avoir des types différents pour chaque élément
- des sous-éléments dans l'ordre du tableau, avec si besoin le type pour chaque élément



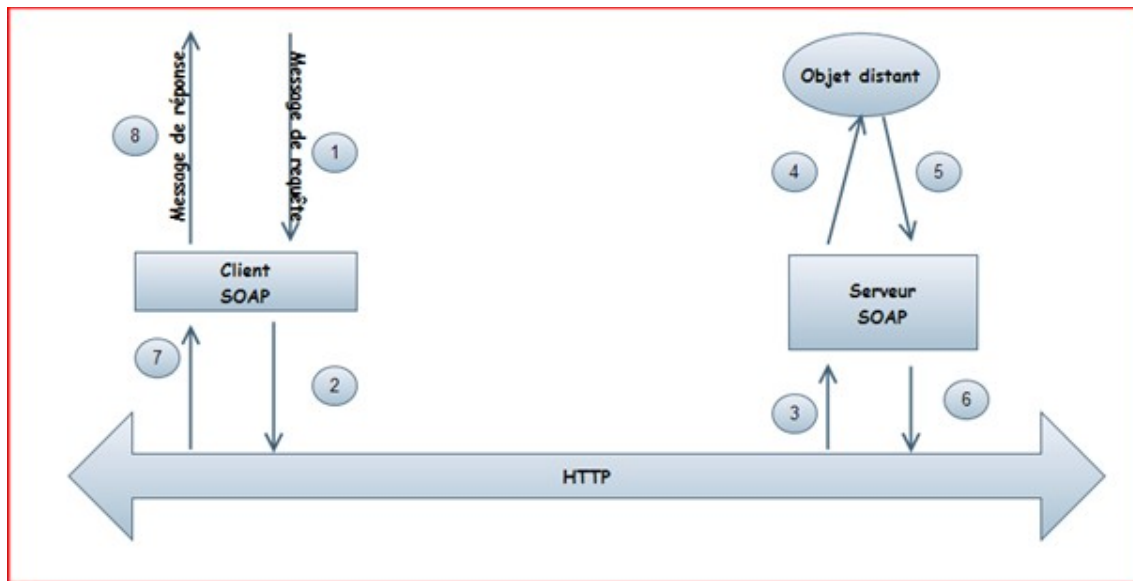
Exemple : Encodage d'un tableau

```
<enc:Array
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
  enc:arrayType="xsd:int[4]"
    <x>1</x>
    <y>2</y> <!-- les noms n'importent pas -->
    <x>3</x>
    <x>4</x>
</enc:Array>
```

```
<enc:Array
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
  enc:arrayType="xsd:int[4]"
    <x>1</x>
    <y>2</y> <!-- les noms n'importent pas -->
    <x>3</x>
    <x>4</x>
</enc:Array>
```

4. SOAP et RPC

SOAP permet aussi des appels de procédures à distance (RPC) en passant le nom de la procédure ou méthode avec le nom de l'objet suivi de la liste des paramètres d'appels dans un message au format XML. Le serveur concerné répond, dans le cas où le message transmis est correct, après exécution de la procédure, par un autre message contenant le résultat de l'appel au format XML toujours si il n'y pas eu d'erreurs, et par un message contenant l'erreur dans le cas contraire. La figure suivante illustre ce mécanisme dans le cas d'un appel de méthode d'un objet distant.



Etapes d'invocation d'objets à distance

Ce modèle RPC autorise 3 modes de passage pour les paramètres :

- mode in : paramètre utilisé mais non modifié, apparaît seulement dans l'appel
- mode in/out : paramètre utilisé et modifié, apparaît dans l'appel et la réponse
- mode out : paramètre de retour uniquement, apparaît dans la réponse seulement



Exemple

Appel de la fonction CtoF (Celsius vers Fahrenheit) :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <env:Body>
    <ns1:CtoF
      env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:ns1="urn:TempConverterIntf-ITempConverter">
      <temp xsi:type="xsd:int">30</temp>
    </ns1:CtoF>
  </env:Body>
</env:Envelope>
```

Résultat de l'appel :

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
<SOAP-ENV:Body>
  <NS1:CtoFResponse
    xmlns:NS1="urn:TempConverterIntf-ITempConverter"
    enc:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

    <return xsi:type="xsd:int">86</return>

  </NS1:CtoFResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

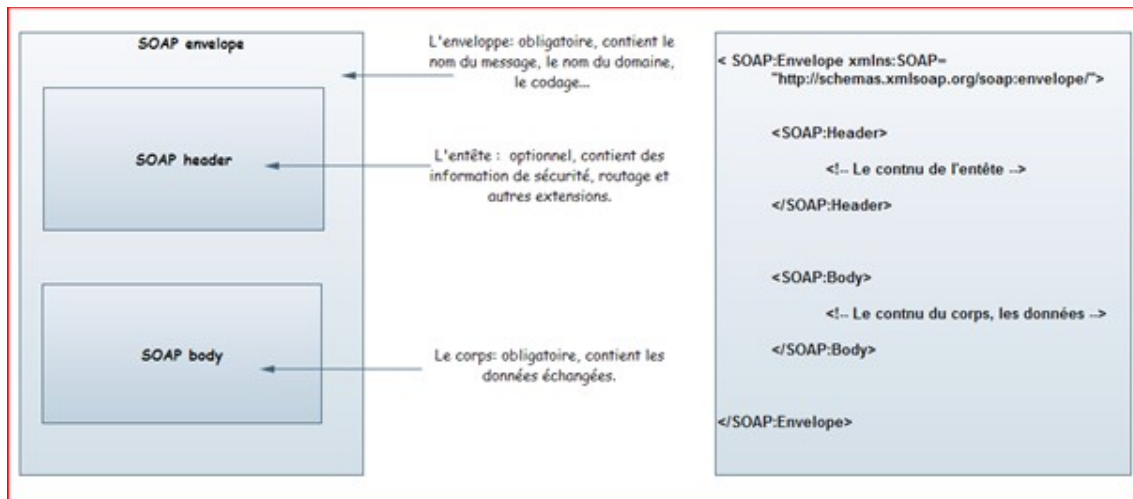
C. WSDL : Web Service Description Language

WSDL est un langage permettant de décrire les Web Services et les données manipulées par ces web services. Il a été développé conjointement par IBM, Microsoft et Ariba et a été présenté pour analyse au W3C qui l'a accepté comme une notice [3]^[Gilbert Babin & Michel Leblanc]. Il se base sur une description en XML du web service et permet de séparer la définition abstraite de la fonctionnalité offerte et les détails la concernant (W3C).

Il repose sur XML et sur SOAP, HTTP et MIME pour l'invocation de méthodes à distance. Il est utilisé dans la pratique pour spécifier les services applicatifs fournis par un composant applicatif et l'ensemble des opérations constituant ces services [1].^{Hubert Kadima, Valérie Monfort} Il permet ainsi de faire connaître le service auprès des clients pour une utilisation correcte.

Un document WSDL permet de décrire un web service comme étant un ensemble de points finaux appelés *ports*. Chaque port est associé à une liaison spécifique et qui détermine comment un ensemble abstrait d'opérations et de messages spécifiques sont associés à ce port selon un protocole particulier. Cette liaison établit une correspondance entre un protocole donné et un *port type*. Un *port type* est, quant à lui, un ensemble abstrait d'opérations que peut réaliser le service. Chaque opération est constituée d'un ensemble de messages abstraits qui représentent les données communiquées au cours de l'opération. Chaque message est constitué à son tour par un ensemble de données de types standards ou décrits par l'utilisateur [1].^{Hubert Kadima, Valérie Monfort}

Ainsi, un document WSDL se présente comme un ensemble de définitions comme dans le schéma de la figure suivante.



Structure d'un message SOAP

Il est constitué des différentes parties suivantes [1]Hubert Kadima, Valérie Monfort [2]Libero Maesano, Christian Bernard, Xavier Le Galles :

- **types** : fournissent les définitions des types de données utilisées pour constituer les messages échangés. Ils reposent sur un système de typage standard pour assurer une interopérabilité accrue, comme xsd des schémas XML.
- **message** : représente une définition abstraite des données échangées pendant les transmissions. Un message comporte des parties logiques, chacune étant définie dans un type donné.
- **operation** : C'est la définition abstraite d'un ensemble cohérent de messages (des messages en entrée et d'autres en sortie) et qui va former l'unité d'interaction avec le web service.
- **portType** : C'est un ensemble d'opérations abstraites. Les portTypes sont utilisés pour définir les traitements offerts par un web service.
- **binding** : spécifie un protocole réel et les spécifications de format de données pour les opérations et les messages définis par un type de port donné.
- **port** : spécifie une adresse pour une liaison définissant un simple point terminal de communication. Il s'agit d'une combinaison entre une adresse réseau et une liaison.
- **service** : C'est un ensemble de ports exposés permettant l'accès au web service.



Exemple : Synthèse

Considérons le web service défini par la classe suivante :

```

package test.ping;

/**
 * @author Administrateur
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class Ping {
    public Ping() {
    }

    public String are_u_alive(String msg) {
        return "Yes Im : "+msg;
    }
}

```

Les différentes parties qui composent le WSDL correspondant à ce service seront les suivantes :

Types : cette partie est donnée par un élément `wsdl:types`. Le contenu de cet élément peut être un schéma du W3 : espace de noms <http://www.w3.org/2001/XMLSchema> (préfixe classique `xsd`).

```

<?xml version="1.0" encoding="UTF-8"?>
<definitions name="test.ping.Ping"
  targetNamespace="http://systinet.com/wsdl/test/ping/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:map="http://systinet.com/mapping/"
  xmlns:ns0="http://systinet.com/xsd/SchemaTypes/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://systinet.com/wsdl/test/ping/"
  <types>
    <xsd:schema elementFormDefault="qualified"
      targetNamespace="http://systinet.com/xsd/SchemaTypes/"
      xmlns:tns="http://systinet.com/xsd/SchemaTypes/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="msg" nillable="true" type="xsd:string"/>
      <xsd:element name="string_Response" nillable="true"
        type="xsd:string"/>
    </xsd:schema>
  </types>

```

Messages : les messages échangés par les services sont décrits par des éléments `wsdl:message`. Chaque message possède un nom (attribut `name`) et est constitué de parties. Chaque partie est décrite par un fils `wsdl:part` précisé par :

- son nom (attribut `name`)
- soit son type (attribut `type`)
- soit directement le nom de l'élément qui la constitue (attribut `element`)

```
<message name="Ping_are_u_alive_Request_Soap">
  <part element="ns0:msg" name="msg"/>
</message>
<message name="Ping_are_u_alive_Response_Soap">
  <part element="ns0:string_Response" name="response"/>
</message>
```

PortType : un portType permet d'identifier (nommer) de manière abstraite un ensemble d'opérations.

```
<portType name="Ping">
  <operation name="are_u_alive" parameterOrder="msg">
    <input message="tns:Ping_are_u_alive_Request_Soap"/>
    <output message="tns:Ping_are_u_alive_Response_Soap"/>
  </operation>
</portType>
```

Bindings : WSDL permet de lier une description abstraite (portType) à un protocole.. Chacune des opérations d'un portType pourra être liée de manière différente. Le protocole SOAP est un des protocoles qui peuvent être utilisés.

Un *binding* propose une réalisation concrète d'un type de port

- élément racine `wsdl:binding`
 - précisé par un attribut `name` : le nom du *binding*
 - et par un attribut `type` : le type de port concerné par le *binding*
- contient un élément `wsdl:operation` pour chaque opération du type de port (attribut `name` pour indiquer l'opération concernée)
- chaque élément `wsdl:operation` contient des éléments définissant le *binding* des messages associés (à chaque fois précisé par un attribut `name`) :
 - `wsdl:input`
 - `wsdl:output`
 - `wsdl:fault`

Pour préciser que le binding est de type SOAP, il faut inclure la balise suivante :

```
<soap:binding transport="uri" style="soap_style" />
```

- `transport` définit le type de transport : `http://schemas.xmlsoap.org/soap/http` pour utiliser SOAP/HTTP
- `style` définit la façon dont sont créés les messages SOAP de toutes les opérations
 - `rpc` : Encodage RPC défini par SOAP RPC
 - `document` : Encodage sous forme d'élément XML

Pour chaque opération :

- il faut préciser l'URI de l'opération : `soapAction`
- Pour chaque message de chaque opération, il faut définir comment sera interprété le message SOAP, moyennant `soap:body`


```

<binding name="Ping" type="tns:Ping">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="are_u_alive">
    <map:java-operation name="are_u_alive"
      signature="KExqYXZhL2xhbmcvU3RyaW5nOylHamF2YS9sYW5nL1N0cmIuZzs="/>
    <soap:operation
      soapAction="http://systinet.com/wsdl/test/ping/Ping#are_u_alive?KExqYXZhL2xhk
      style="document"/>
    <input>
      <soap:body parts="msg" use="literal"/>
    </input>
    <output>
      <soap:body parts="response" use="literal"/>
    </output>
  </operation>
</binding>

```

Service : un service est une collection de ports

- un élément `wsdl:service` portant un nom (attribut `name`)
- contenant un élément `wsdl:port` par port, précisé par :
 - un attribut `name` donnant le nom du port
 - un attribut `binding` donnant le nom du binding associé
- pour SOAP, un élément `soap:address` précise l'URI du port grâce à son attribut `location`

```

<service name="Ping">
  <port binding="tns:Ping" name="Ping">
    <soap:address location="http://172.17.2.11:6060/Ping/" />
  </port>
</service>

```

D. UDDI : Universal Description Discovery and Integration

UDDI est une réponse à deux problèmes classiques des systèmes répartis :

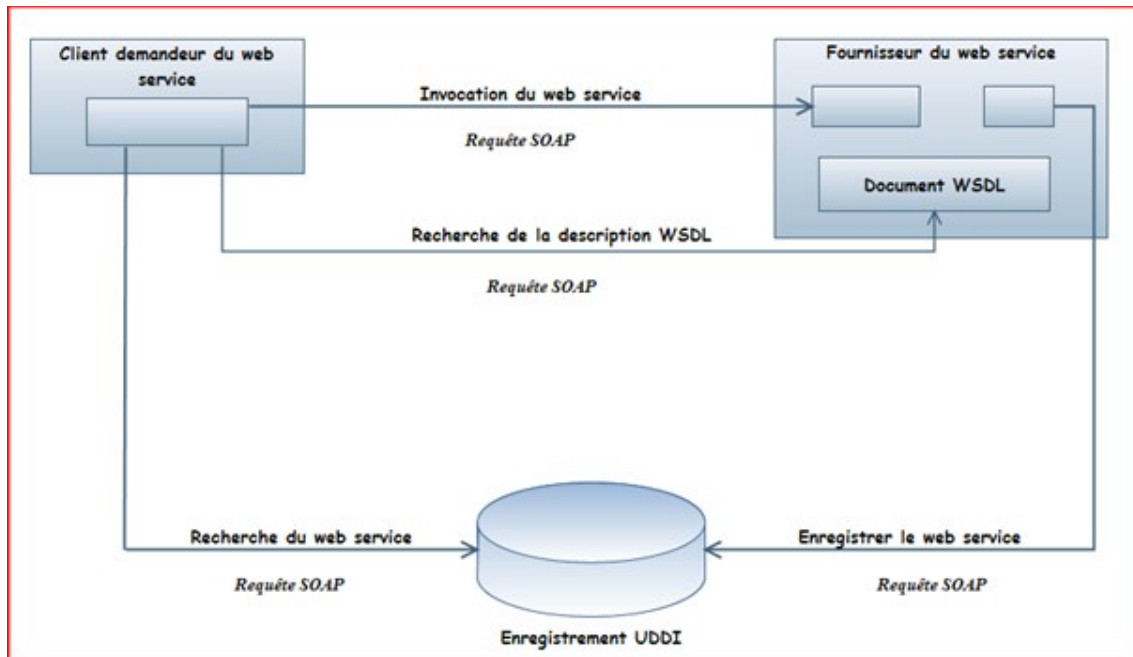
- comment indiquer qu'un service est disponible
- comment découvrir un service

UDDI définit les mécanismes permettant de répertorier des web services. Ce standard régit, donc, l'information relative à la publication, la découverte et l'utilisation d'un Web service [3] Gilbert Babin & Michel Leblanc. Il permet de publier et découvrir un ensemble de données relatives aux entreprises et organismes ainsi que les services qu'ils fournissent [1] Hubert Kadima, Valérie Monfort. Il découle de l'initiative du trio IBM, Microsoft et Ariba avant d'être adopté par OASIS pour être une référence en la matière pour les entreprises. Il vient pour répondre aux besoins croissants des échanges B2B [1] Hubert Kadima, Valérie Monfort [3] Gilbert Babin & Michel Leblanc.

UDDI est lui-même un web service et il est accessible par le biais de messages SOAP tout comme tout autre web service. L'API de UDDI est donc un service décrit par WSDL et se situe dans la couche des web services dans la pile architecturale [2]. Libero Maesano, Christian Bernard, Xavier Le Galles :

Des requêtes SOAP pour la découverte des web services publiés sont envoyées par les clients et des réponses SOAP sont par la suite envoyées par l'annuaire UDDI pour

donner les descriptions et coordonnées des services et/ou organismes voulus à la manière d'un web service normal. La Figure suivante illustre le mécanisme d'accès à un web service fourni par un nœud d'un registre UDDI.



Mécanisme d'accès aux services d'un nœud annuaire UDDI

L'annuaire UDDI est consultable sur plusieurs facettes [1]Hubert Kadima, Valérie Monfort[2]Libero Maesano, Christian Bernard, Xavier Le Galles :

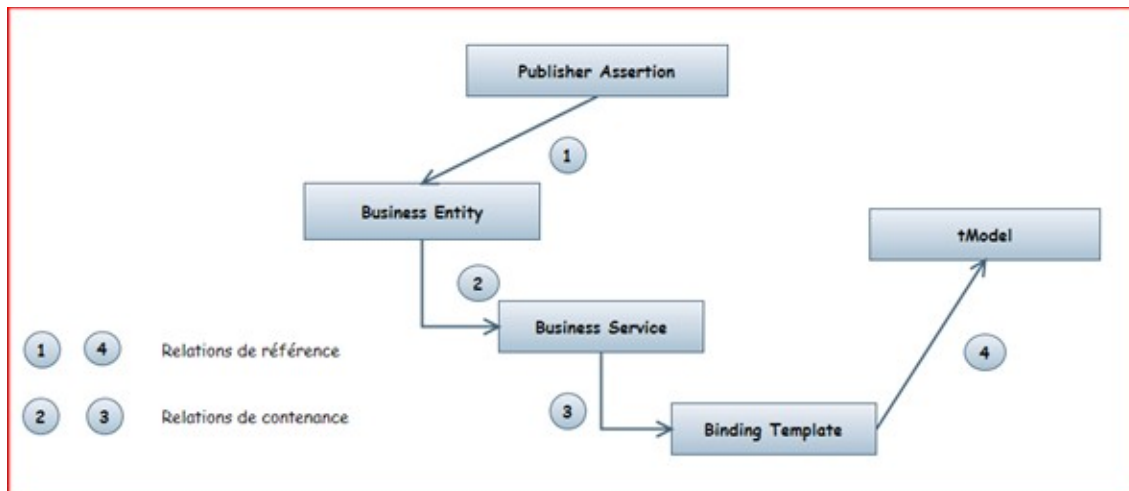
- **Les pages blanches** : Elles représentent les entreprises en contenant les informations qui leurs sont relatives telles que le nom, l'identité complète, siège social, site web..., chacune des entreprises étant identifiée de façon unique par un identifiant.
- **Les pages jaunes** : Elle comporte les descriptions en WSDL des différents services qui y sont publiés.
- **Les pages vertes** : Elles fournissent des informations techniques détaillées sur les services fournis. Elles comportent notamment des informations de description et de liaison des services publiés.

Un annuaire UDDI a deux types de clients ou demandeurs du service UDDI, ceux qui recherchent des services et ceux qui les publient. Il offre la possibilité d'interaction dynamique pour ces deux types de clients à savoir la découverte et la publication [1]Hubert Kadima, Valérie Monfort.

UDDI utilise un modèle d'information défini sous forme de schéma XML comportant cinq structures de données et qui sont [1]Hubert Kadima, Valérie Monfort[2]Libero Maesano, Christian Bernard, Xavier Le Galles :

- **Business Entity** : (ou entité métier) Ce sont les équivalents des pages blanches. Elles représentent les informations relatives aux entreprises qui y publient les services.
- **Business service** : (ou service métier) Ce sont les équivalents des pages jaunes. Elles représentent les services offerts par les entreprises.
- **Binding template** : (Ou modèle de liaison) Ce sont les équivalents des pages vertes. Ils comportent des informations relatives au lieu d'hébergement des web services.
- **tModel** : (ou service type) Ils définissent le mode d'accès aux services.
- **Publisher Assertion** : (ou assertion d'administrateur) Elles définissent les

assertions de contrats entre les partenaires pour l'accès aux services.
Les relations entre ces différentes structures peuvent être résumées dans le schéma de la figure suivante.



Relations entre les différentes structures de données UDDI

Les interfaces de l'annuaire UDDI comportent :

- la publication des services par les fournisseurs.
- la découverte des services publiés par les consommateurs ou clients.
- la gestion des jetons d'accès pour des besoins de sécurité afin d'éviter des suppressions et modifications malicieuses ou non autorisées.
- L'abonnement d'un client à un ensemble d'informations et le tenir au courant en cas de leurs modification.
- la réplication interne permettant de synchroniser un ensemble de nœuds d'un même annuaire et la réplication externe qui n'est pas indépendante mais qui utilise les précédentes et permettant à un annuaire d'interroger les autres annuaires, de publier et de s'abonner auprès d'eux. [1].

Hubert Kadima, Valérie Monfort

* *
*

En guise de conclusion, on peut citer les avantages et inconvénients suivants des web services :


Avantages :

- Des standards « simples » (SOAP, WSDL, UDDI)
- Multi Protocole / Multi OS / Multi Langage
- Paradigme de Service
- Des outils (éditeurs et moteurs)

Inconvénients :

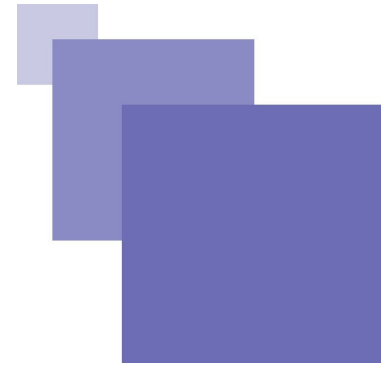
- Typage (pas de consensus)
- Performance

Signification des abréviations



- Ajax	Asynchronous JavaScript + XML
- AWT	Abstract Windows Toolkit
- B2B	Business to Business
- BOA	Basic Object Adapter
- CGI	Common Gateway Interface
- CLI	Command Line Interface
- DII	Dynamic Invocation Interface
- DNS	Domain Name System
- DSI	Dynamic Skeleton Interface
- GIOP	General Inter-ORB Protocol
- GUI	Graphical User Interface
- HTML	HyperText Markup Language
- HTTP	HyperText Transfer Protocol
- IDL	Interface Definition Language
- IIOP	General Inter-ORB Protocol
- IOR	Interoperable Object Reference
- JAL	Just Another Language
- JMS	Java Message Service
- MOM	Message Oriented Middleware
- OMA	Object Management Architecture
- OMG	Object Management Group
- OMG	Object Management Group
- ORB	Object Request Broker - Négociateur de requêtes objet
- PDO	PHP Data Objects
- PHP	Hypertext Preprocessor
- POA	Portable Object Adapter
- POO	Programmation Orientée Objet
- SGML	Standard Generalized Markup Language
- SOAP	Simple Object Access Protocol
- URI	Uniform Resource Identifier
- URL	Uniform Ressource Locator
- URN	Uniform Ressource Name
- WSDL	Web Service Description Language
- XML	eXtensible Markup Language

Références



- [A. Ploix] Cours Réseaux IP : Internet Réseaux d'entreprise, Université de Technologie de Troyes.
- [A list apart] [HTTP://www.alistapart.com/articles/cssatten](http://www.alistapart.com/articles/cssatten)
- [Annick Fron] Architectures réparties en Java, Dunod, Paris, 2007
- [Apache taglibs] [HTTP://jakarta.apache.org/taglibs/](http://jakarta.apache.org/taglibs/)
- [Apprendre PHP] [HTTP://www.apprendre-php.com/tutoriels](http://www.apprendre-php.com/tutoriels)
- [B. Catteau N. Faugout] « AJAX : le guide complet » B. Catteau N. Faugout, édition Micro Applications, 2009
- [C. Delannoy] Programmer en Java, édition Eyrolles, 2009.
- [Comment ça marche] [HTTP://www.commentcamarche.net/](http://www.commentcamarche.net/)
- [Cours de PHP 5 - Developpez] [HTTP://g-rossolini.developpez.com/tutoriels/php/cours](http://g-rossolini.developpez.com/tutoriels/php/cours)
- [D. Comer] " TCP/IP Architectures, protocoles et applications", Pearson Education, 5eme édition, 2009
- [D. Flanagan, ED. O'Reilly] « JavaScript: La référence » D. Flanagan, ED. O'Reilly, 5eme édition, 2007
- [Donahoo, Calvert] "TCP/IP Sockets in C: Practical Guide for Programmers" by Michael J. Donahoo et Kenneth L.Calvert *disponible en ligne*⁶
- [Douanes K. Fields, Mark A. Kolb] JSP - Java Server Pages, editions Eyrolles, 2001.
- [E. Daspet, C.P. de Geyer] PHP 5 avancé, 5eme edition, E. Daspet, C.P. de Geyer, ed. Eyrolles 2009

6 - <http://cs.baylor.edu/~donahoo/practical/CSockets/textcode.html>.

- [*Eclipse*] <http://www.eclipse.org/>
- [*F. BOYER, R. BALTER, M. RIVEILL,*] Communication synchrone entre programmes par RPC et RMI, BASE DOCUMENTAIRE : Technologies logicielles Architectures des systèmes.
- [*F.X. Sennesal*] JSP avec Eclipse et Tomcat, Editions ENI, 2009.
- [*Fabrice Rossi*] Reprise de contenus notamment graphiques de la présentation de Fabrice Rossi : « web services », « xml », Paris IX Dauphine.
- [*Gilbert Babin & Michel Leblanc*] "Les web services et leur impact sur le commerce B2B ". Rapport de BOURGOGNE. CIRANO. Août 2003.
- [*Hubert Kadima, Valérie Monfort*] "Les web services". Dunod, Paris, 2003.
- [*J.L Déléage*] JSP et Servlets efficaces : Production de sites dynamiques. Cas pratiques, editions Dunod, 2009.
- [*J. Pauli G. Poncon*] Les cahiers du programmeur, Zend framework, bien développer en PHP. J. Pauli G. Poncon, ed. eyrolles, 2009.
- [*JSP*] JAVA Server Pages
- [*JSTL*] JavaServer Pages Standard Tag Library
- [*L. Isolda et S. Magne*] PHP & MySQL, L. Isolda et S. Magne, eds. MicroApplication, 2005.
- [*Libero Maesano, Christian Bernard, Xavier Le Galles :*] "Services Web avec java et .NET, conception et implémentation". Editions EYROLLES, Paris. 2003.
- [*M. Vayssade*] "SR03 : ARCHITECTURES INTERNET", 2007
- [*Mozilla developer network*] [HTTP://developer.mozilla.org/en/docs/Gecko_DOM_Reference](http://developer.mozilla.org/en/docs/Gecko_DOM_Reference)
- [*O. Hondemarck*] « JavaScript : le guide complet », O. Hondemarck, édition Micro Applications, 2009
- [*PHP France*] [HTTP://www.phpfrance.com/tutoriaux](http://www.phpfrance.com/tutoriaux)
- [*S. Cateloin*] Cours HTTP, l'Université Louis Pasteur de Strasbourg;

[*S. Maccari et S. Martin*] "HTML et javascript", ed. MicroApplication, 2004.

[*Serge Tahé, site developpez*] [HTTP://tahe.developpez.com/](http://tahe.developpez.com/)

[*Site Developpez*] [HTTP://www.developpez.com](http://www.developpez.com)

[*W3C*] <http://www.w3.org/>

[*W3C*] [HTTP://www.w3.org/](http://www.w3.org/)

[*W3C - SOAP*] <http://www.w3.org/TR/SOAP/>

[*W3C - WSDL*] <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>

[*W3C - XML*] <http://www.w3.org/TR/REC-XML>

[*Wikipédia*] [HTTP://fr.wikipedia.org](http://fr.wikipedia.org)

[*Xavier Blanc*] Reprise de contenus notamment graphiques de la présentation de Xavier Blanc : « web services », Lip6