Master Informatique M1

Introduction aux applications réparties

Sacha Krakowiak
Université Joseph Fourier
Projet Sardes (INRIA et IMAG-LSR)

http://sardes.inrialpes.fr/people/krakowia

Plan du cours

- **Motivations**
  - On a vu des mécanismes de base pour la construction d'applications client-serveur : *sockets*, RPC, Java RMI
  - On veut maintenant montrer quelques problèmes de la construction effective de telles applications
  - Pour cela on va examiner la conception et le fonctionnement de deux applications parmi les plus utilisées sur l'Internet
- **Points traités**
  - Objectifs et problèmes de la construction d'applications réparties
  - Exemple 1 : DNS (Domain Name System)
  - Exemple 2 : World Wide Web

Applications réparties : qualités requises (1)

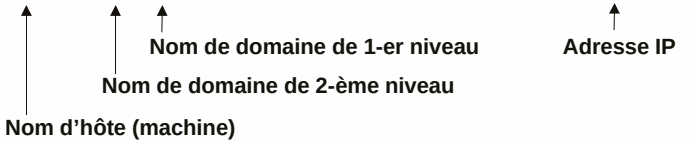
- **Qualité de service**
  - Recouvre plusieurs notions, qui doivent être précisées au cas par cas
    - Performances
      - Le critère dépend de l'application : latence (temps de réponse), débit d'information traitée, nombre de transactions par seconde, etc.
      - La stabilité des facteurs de performance peut être au moins aussi importante que leur valeur absolue (exemple : applications multimédia)
    - Tolérance aux fautes
      - Nécessite d'identifier les scénarios de fautes possibles
        - Matériel
        - Logiciel
        - Système de communication
    - Sécurité
      - Nécessite d'identifier les scénarios d'attaque possible
        - Violation de confidentialité
        - Violation d'intégrité
        - Déni de service

Applications réparties : qualités requises (2)

- **Capacité de croissance, ou passage à grande échelle (*scalability*)**
  - Les qualités d'un système ne doivent pas se dégrader en cas de croissance
    - Du nombre d'éléments du système (machines, réseaux, etc.)
    - Du nombre d'utilisateurs
    - De l'étendue géographique
- **Capacité d'évolution**
  - Architecture modulaire (composants)
    - Qualité de la décomposition
      - identifier les fonctions
    - Qualité de la conception des interfaces
      - identifier ce qui est montré et ce qui est caché
  - Découplage
    - L'impact des modifications locales doit si possible rester local
  - Qualité de la documentation
    - Documentation de conception
    - Documentation de réalisation

DNS : le service de noms de l'Internet

- **Fonction**
  - Toute machine connectée à l'Internet possède une adresse IP
  - La fonction de DNS est de réaliser la conversion (dans les deux sens) entre un nom symbolique (à usage humain) qui désigne une machine et l'adresse IP de cette machine
  - Les noms symboliques sont organisés en "domaines", d'où le nom DNS (*Domain Name Service*)
  - Exemple
    - goedel.imag.fr — 195.221.225.5



Place de DNS

- **DNS recouvre deux notions**
  - Une (grande) base de données répartie, réalisée par un ensemble de serveurs de noms qui communiquent entre eux
  - Un protocole (au niveau application) utilisant cette base
    - Fonctionne au-dessus d'UDP
    - Utilise par convention le port 53
- **Historique**
  - Jusqu'en 1988, le service de noms de l'Internet était centralisé
    - Acceptable pour quelques milliers de machines, mais ne passe pas à l'échelle (cf plus loin)
  - 1987-88 : Conception et mise en place de DNS
    - Documents : RFC 1034 et 1035 + compléments ultérieurs
      - Voir http://www.dns.net/dnsrd/
  - Depuis, a démontré sa capacité de croissance : de $x.10^3$ à $x.10^8$…

Limites d'un service de noms centralisé

- **Sensibilité aux défaillances**
  - Défaillance du serveur : perte du service
  - Palliatif : serveur dupliqué (mais nécessite mise à jour fréquente)
- **Goulot d'étranglement**
  - Point de passage obligé de toutes les communications : perte de performances
    - Pour le réseau
    - Pour le serveur lui-même (palliatif : serveur multiprocesseur)
  - Mise à jour fréquente, peut dégrader les performances d'accès
- **Accès majoritairement à distance**
  - Pas d'accès local (sauf sur le site du serveur)
  - Palliatif : caches locaux

DNS : noms de domaines (1)

- **La désignation sur l'Internet utilise un schéma hiérarchique (noms de domaines)**
  - Exemples : thales.e.ujf-grenoble.fr, cmu1.acs.cmu.edu, www.w3.org, firewall.ma02.bull.com
  - Un domaine est une unité pour la construction et la gestion des noms (analogue à un répertoire dans un système de fichiers)
- **La hiérarchie des noms a une racine multiple : les domaines du premier niveau**
  - domaines "géographiques" : fr (France), uk (Royaume Uni), de (Allemagne), jp (Japon), …
    - USA est souvent compris "par défaut", bien qu'il existe un domaine us
  - domaines "d'activité", génériques : com (commercial), org (organisations), edu (universités, aux USA), net (l'Internet), …

Utilisation de DNS

- **DNS est utilisé par tous les protocoles d'application**
  - HTTP (Web) , FTP (transfert de fichiers), SMTP (mail), …
  - DNS est donc un élément essentiel du fonctionnement de l'Internet, bien qu'il ne soit pas (en général) directement visible par les utilisateurs
- **DNS est aussi accessible directement**
  - Exemple dans le *shell* Unix

```
unix> host boole.imag.fr
boole.imag.fr is an alias for ufrima.imag.fr
ufrima.imag.fr has address 195.221.225.1
```

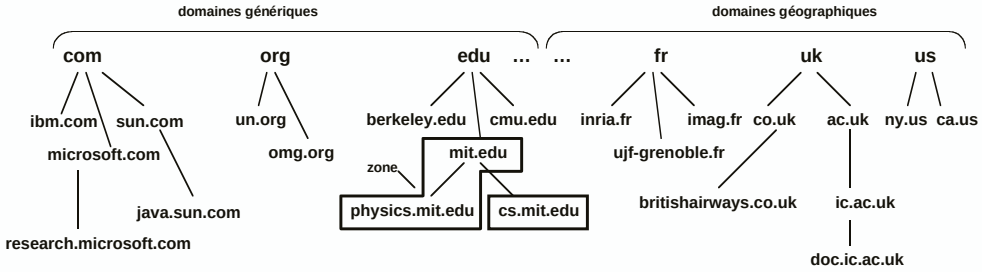
  - Exemple en Java

```
import java.net.*
…
InetAddress IPAddress = InetAddress.getByName("boole.imag.fr");
```


DNS : noms de domaines (2)

- **Dans chaque domaine, les noms sont attribués par une autorité responsable du domaine**
  - Pour établir la liste des domaines du premier niveau : l'Internet Society (ISOC) via un groupe technique ad hoc
  - Pour les domaines "publics" du premier niveau (com, org, net, …) : une autorité centrale, l'ICANN (www.icann.org), avec des autorités déléguées
  - Pour les domaines géographiques : une autorité nationale par pays - en France, l'AFNIC (www.nic.fr) - Association Française pour le Nommage sur l'Internet en Coopération)
  - Pour les domaines inclus : autorités locales (entreprise, administration, etc.)

DNS : noms de domaines (3)



- **Domaines et zones**
  - Domaine = unité de désignation (espace de noms)
  - Zone = unité de gestion administrative (serveur de noms propre à la zone)
  - Le plus souvent, un domaine est aussi une zone, mais une zone peut grouper plusieurs domaines, administrés en commun

Conception de DNS

- **Principes directeurs**
  - Algorithme de recherche décentralisé (pas de point de décision unique)
  - Hiérarchie de serveurs calquée sur la hiérarchie des zones (voisines des domaines)
  - Usage intensif de caches (informations dupliquées) et d'indicateurs (informations probablement valides permettant un accès rapide la plupart du temps)
    - toute information peut être obtenue par plusieurs voies
    - la validité de toute information peut être confirmée si nécessaire
  - Ces règles favorisent la capacité de croissance
  - Elles contribuent aussi à la disponibilité (service assuré même en en cas de panne locale)

Fonctionnement de DNS (1)

- **Toute zone (unité de gestion) comporte au moins deux serveurs de noms**
  - Motivation de la duplication : pour performances, et surtout tolérance aux fautes
  - Chaque serveur maintient une table de correspondance Nom de domaine-Adresse IP
  - Plus précisément, une table est une collection d'enregistrements de la forme :

    Nom, *Valeur*, *Type*, *Classe*, *Durée de vie*

Exemples

<inrialpes.fr, if.inrialpes.fr, NS, IN, …> --"if" est un serveur de noms du domaine "inrialpes.fr"

<if.inrialpes.fr, 194.199.18.65, A, IN, …> -- l'adresse IP de "if" est 194.199.18.65

- **Principe de la recherche**
  - Ou bien le serveur contient l'adresse recherchée, ou bien il contient l'adresse d'un autre serveur qui a davantage de chances que lui de la connaître

Fonctionnement de DNS (2)

- **Le serveur racine contient les noms des serveurs de noms des domaines du second niveau et les adresses IP de ces serveurs**

  <inrialpes.fr, if.inrialpes.fr, NS, IN, …> -- if est un serveur de noms du domaine inrialpes.fr

  <if.inrialpes.fr, 194.199.18.65, A, IN, …> -- l'adresse IP de if est 194.199.18.65

  - Le serveur racine est très largement dupliqué
    - Plus un serveur est "haut", plus souvent on en a besoin, plus il a de copies, et moins souvent il change (long TTL)
- **De même, chaque serveur de zone contient**
  - Des couples (noms, adresses) d'hôtes appartenant aux domaines contenus dans la zone
  - Éventuellement des couples (noms, adresses) de serveurs de noms de sous-domaines inclus
  - Par exemple, if.inrialpes.fr contient

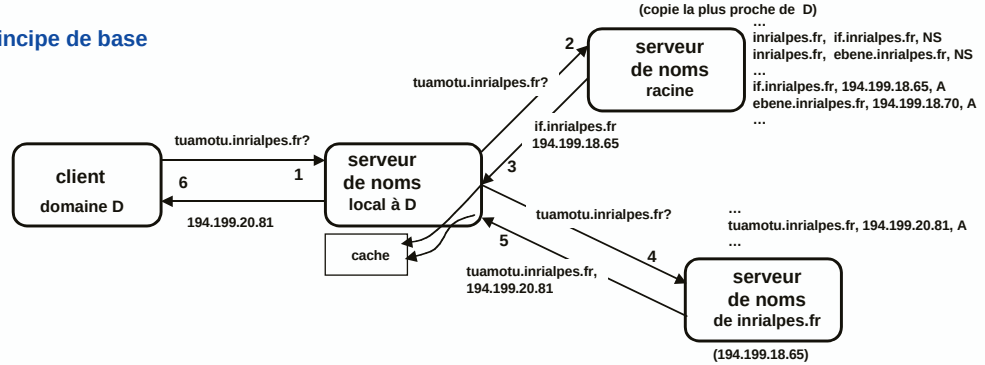
    <tuamotu.inrialpes.fr, 194.199.20.81, A, IN, …>

Fonctionnement de DNS (3)

- **Pour amorcer la résolution**
  - Tout hôte doit connaître l'adresse IP d'un serveur de noms local (dans sa zone) - il est recommandé de connaître deux ou trois adresses, pour la tolérance aux fautes
  - Ces adresses sont fournies aux utilisateurs de l'hôte par l'administrateur de sa zone (ou son fournisseur d'accès à l'Internet)
  - Elles sont inscrites "à la main" dans les tables de configuration pour l'accès à l'Internet

Fonctionnement de DNS (4)

Principe de base



Mécanismes d'accélération

Tout serveur conserve dans un cache les couples (nom, adresse IP) récemments résolus

Dans la pratique, il est rare de consulter plus de deux serveurs

Autorités

Tous les serveurs ne sont pas mis à jour en permanence de toutes les modifications.

Certains seulement le sont, et font autorité, d'ou les réponses "*authoritative answer*" ou "*non authoritative answer*"

Fonctionnement de DNS : compléments (1)

- La hiérarchie de serveurs de noms : 3 niveaux
  - Serveurs locaux (dans une organisation)
    - Le serveur local (par défaut) est le prmeier à recevoir les requêtes. Si la réponse n'est pas dans son cache, il transmet au niveau supérieur (racine)
  - Serveurs racine
    - En petit nombre, au sommet de la hiérarchie, mais dupliqués ; connaissent les adresses des serveurs “autorité”
  - Serveurs “autorités”
    - Un serveur “autorité” pour un hôte connaît l’adresse IP exacte de cet hôte
    - Tout hôte doit avoir au moins 2 serveurs “autorité” connaissant son adresse
    - Un serveur peut être à la fois local et autorité
- Trajet des messages
  - La réponse suit toujours le trajet inverse de celui de la requête
  - Exemple : client->local->racine->autorité->racine->local->client
  - On peut ainsi localiser une erreur “au plus près”

Fonctionnement de DNS : compléments (2)

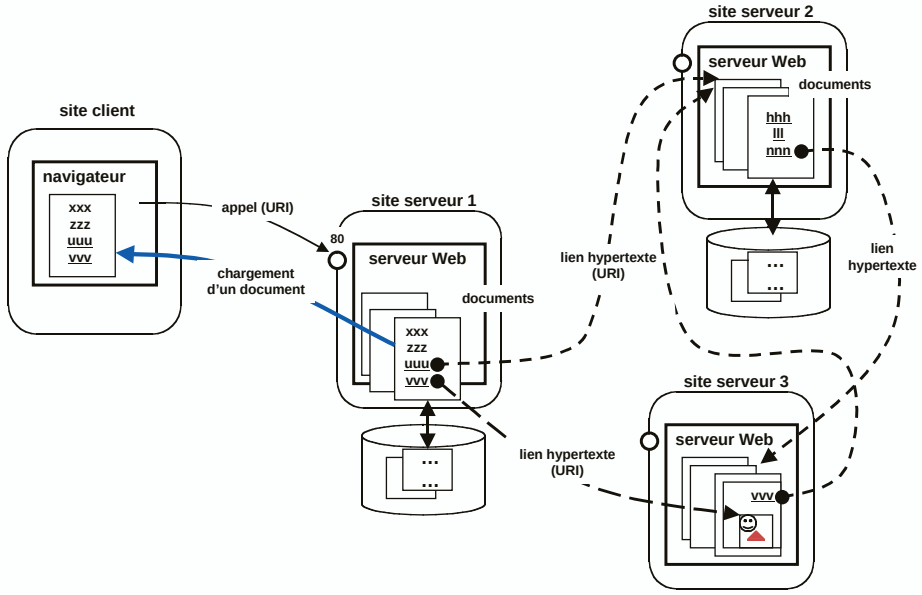
- Noms
  - DNS permet la création d’alias : un site a un nom principal ("canonique") et un nombre quelconque d’alias. Exemple : boole.imag.fr est un alias du nom canonique ufrima.imag.fr
  - Même chose pour les adresses email.
- Répartition de charge
  - Les sites très chargés utilisent des serveurs multiples, dont chacun a une adresse IP différente (pour un même nom de domaine). DNS répartit la charge entre ces serveurs (renvoie les adresses dans un ordre différent lors de chaque requête). Exemple : exécuter plusieurs fois host cnn.com

World Wide Web : principes et composants

- Bref historique
  - Idée de base : ensemble de documents répartis reliés entre eux par des liens hypertexte. Objectif initial (Tim Berners-Lee, CERN, 1989-90) : créer un outil pour le travail en collaboration, sur des données communes, pour une communauté répartie de physiciens
    - en fin 1993, 250 serveurs, 1% du trafic de l’Internet (10 fois plus qu’en début 1993)
  - Le vrai démarrage (1994)
    - les premiers navigateurs : Mosaic (NCSA), puis Netscape
    - les premiers moteurs de recherche : AltaVista, Yahoo!
    - création du World Wide Web Consortium (W3C) <www.w3c.org>
    - en fin 1994, environ 10 000 serveurs
  - Depuis, croissance explosive (l’application la plus utilisée de l’Internet) ~10^9 pages web
- Éléments de base du Web
  - Un espace de noms global pour la désignation des ressources (URL, puis URI)
  - Un protocole (client-serveur) pour le transfert d’information : HTTP
- Un langage de balisage (*markup*) pour la description de documents hypertextes (HTML)
- Extensions
  - Utilisation de langages de script (activation chez le client - *applets*, ou le serveur - *servlets*)
  - Utilisation de types de données multiples ; descriptions génériques (XML), outils associés

Web : schéma global

(simplifié)



Web : désignation

- **Format d'un URI (Uniform Resource Identifier)**
  - <protocole>:<chemin d'accès>
  - Le format du chemin d'accès dépend du protocole. Par exemple (les parties entre crochets sont facultatives) :
    - Protocole `http` : <chemin d'accès> ::= //<identité de serveur> [:<numéro de porte>]
      [<chemin d'accès local>] [?requête] [#étiquette]
    - Protocole `ftp` : idem, sans requête ni étiquette
    - Protocole `file` : <chemin d'accès> ::= <chemin d'accès à un fichier local>
    - Protocole `mailto` : <chemin d'accès> ::= <adresse *email*>
    - Protocole `news` : <chemin d'accès> :: = <nom de *newsgroup*>
  - Exemples
    - http://sardes.inrialpes.frpeople/krakowia ftp://ftp.imag.fr
    - mailto:president@whitehouse.gov news:imag.38
  - Valeurs par défaut : http pour le protocole, 80 pour le numéro de porte, index.html pour le nom de fichier dans un répertoire, etc.
- **Utilisation des URI**
  - Ce sont les voies d'accès à toutes les ressources (documents, serveurs, programmes, etc.)
  - Ce sont les constituants des liens hypertexte

Web : HTTP (*HyperText Transfer Protocol*) (1)

- **HTTP : le protocole standard du World Wide Web**
  - Protocole client-serveur, construit au-dessus de TCP
  - Utilisation principale : entre navigateur et serveur Web, mais peut être utilisé de manière autonome par toute application
- **Principales commandes du protocole**
  - GET <URI> : demande au serveur indiqué dans l'URI d'envoyer la page désignée par l'URI.
    - option : n'envoyer la page que si elle a changé depuis une date spécifiée
  - HEAD <URI> : demande au serveur d'envoyer l'en-tête de la page (contenant des informations diverses : titre, date, etc.)
  - PUT <URI> <page> : envoie une page au serveur spécifié pour la rendre disponible sur ce serveur à l'URI indiquée ; replace le contenu courant de cet URI s'il existe
  - POST <URI> <page> : comme PUT, mais intègre les nouvelles données à celles existant déjà à l'URI (dépend de la nature des données)

Web : HTTP (*HyperText Transfer Protocol*) (2)

- **Principales commandes du protocole (suite)**
  - DELETE <URI> : supprime la page figurant à l'URI indiqué
  - Toutes ces commandes sont soumises à autorisation, en fonction des droits du client demandeur et des protections associées aux ressources sur le serveur
  - La réponse à une commande comporte un code (OK ou type d'erreur) et éventuellement un résultat (contenu de page pour GET, etc.)
  - Les commandes d'envoi de données utilisent une convention standard (MIME) pour les données non textuelles

Web : HTML (*HyperText Markup Language*)

- **HTML est un langage de "balisage" (*markup*)**
  - Un tel langage comporte des marques (balises) insérées dans le texte et destinées à donner des indications de formatage (présentation, interprétation du texte). Exemples plus loin
  - Un langage de balisage très général, utilisé dans l'édition de documents, est SGML (*Standard Generalized Markup Language*) ; HTML en est inspiré.
  - Intérêt du balisage : permet de séparer le contenu de la présentation ou de l'interprétation, et donc permettre des interprétations différentes selon (par exemple) les capacités d'affichage d'une station de travail
  - HTML est en évolution constante (version 4.0) - normalisé par le W3C <www.w3c.org>
- **Comment sont produits les documents HTML ?**
  - "À la main". Pas recommandé, il est préférable d'utiliser un des outils qui suivent
  - Par un éditeur de documents (pour l'écriture de pages Web)
    - directement (frappe du texte, insertion d'images, etc.)
    - par traduction depuis un autre format de document (LaTeX, Word, ou autre)
  - Par un générateur spécialisé, à partir (par exemple) du résultat d'une requête sur une base de données. Chaque application peut construire son générateur (exemples séance suivante)

HTML - quelques exemples (1)

- **Principe du balisage**
  - Les balises vont en général par paires, encadrant un texte à interpréter
    balise début : <xxx paramètres éventuels> - balise fin : </xxx>
- **Structure d'un document HTML** (indicatif)

```
<HTML>
<HEAD> en-tête </HEAD>      -- contient le titre, la date, d'autres méta-informations.
<BODY> corps </BODY>        -- contient le document proprement dit
</HTML>
```

- **Quelques balises de présentation (exemples)**
  - Présentation de caractères
    - <B> texte </B> : caractères gras (*bold*) ; <I> texte </I> : caractères italiques
    - Caractères accentués. Exemples : é = é à = à Ê = Ê etc.
  - Titres
    - <H1> texte du titre </H1> : titre de 1-er niveau (idem pour H2, H3, …)
  - Paragraphage
    - <P> texte </P> : paragraphe ;
 retour à la ligne
    - <HR> coupure du texte (trait horizontal)
  - En fait, l'interprétation précise des balises de présentation peut être définie séparément (feuilles de style). En modifiant la feuille de style, on modifie la présentation sans changer le document

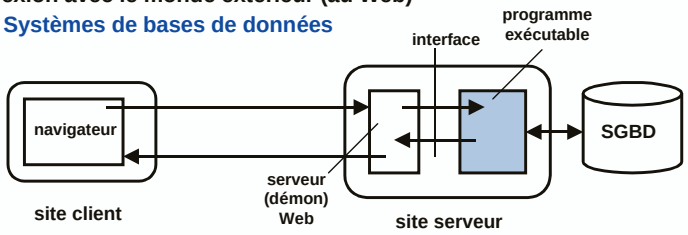
Le Web comme support d'exécution d'applications

Au delà de sa fonction d'accès à l'information, le Web peut servir de support à l'exécution d'applications réparties.

- **Intérêt du Web comme support d'applications**
  - Une interface familière, intuitive et présente partout (le navigateur, les liens)
  - Des outils de base
    - Espace universel de désignation (les URI)
    - Protocole universel de transfert d'information (HTTP)
    - Gestion d'information sous un format standard (HTML), ou extensible (XML)
  - Le Web comme un "système d'exploitation" primitif pour applications réparties ?
- **Problèmes à résoudre**
  - Où et comment sont exécutés les programmes ?
    - sur le site serveur : scripts CGI, *servlets*
    - sur le site client : scripts dans extension du navigateur (*plugin*), *applets*
  - Comment est assurée la sécurité ?
    - problème majeur, non entièrement résolu
      - protection des sites
      - chiffrement de l'information
      - restriction sur les conditions d'exécution

Exécution de programmes sur un serveur Web

- **Intérêt**
  - Exécution de programmes interactifs
    - Formulaires (inscriptions, enquêtes, etc.)
    - Requêtes (moteur de recherche, etc.)
  - Connexion avec le monde extérieur (au Web)
    - Systèmes de bases de données



- **Mécanismes**
  - Script CGI *(Common Gateway Interface)*
    - le plus ancien
    - langages multiples, interface commune standard
  - *Servlets*
    - programmes Java activés sur le serveur, interface spécifique Java

Servlets (1)

Consulter http://java.sun.com/docs/books/tutorial/servlets/

- **Une alternative à CGI, utilisant le langage Java**
  - La classe `Servlet` (et ses extensions) fournissent des outils pour traiter les requêtes HTTP et pour construire les réponses (documents HTML)
    - `doGet` et `doPost` : traitent les opérations GET et POST de HTTP
    - `ServletRequest` : objet permettant de récupérer les paramètres fournis par le client
    - `ServletResponse` : objet permettant de préparer une réponse HTML au client
  - Autres outils
    - synchronisation entre clients multiples d'un *Servlet*
    - communication entre servlets
    - communication avec d'autres sites
- **CGI versus Servlets**
  - Avantages des *Servlets* : sécurité meilleure (JVM) ; plus grande efficacité d'exécution (*threads* au lieu de processus lourds)
  - Avantage de CGI : pas de restriction sur le langage : possibilité de développement rapide avec langages interprétés, en particulier prototypage rapide
  - Utilisation préférable des *Servlets* pour application importante nécessitant connexions externes (accès à un SGBD)

Servlets (2)

Exemple très simple de *servlet*

Répond à l'opération GET en renvoyant un message prédéfini dans une page HTML

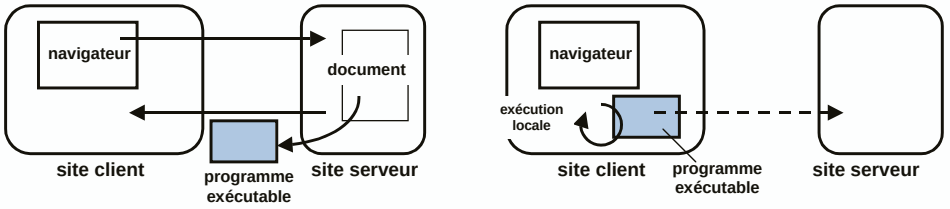
Objets définis:
- **HttpServletRequest représente la requête du client (n'est pas utilisée)**
- **HttpServletResponse représente la réponse construite par le serveur et renvoyée au client**

Exemple

```
public class SimpleServlet extends HttpServlet
 {
     /**
      * Repondre à l'opération HTTP GET en  renvoyant
      * une page web simple.
      * Surcharge la méthode doGet de la classe HttpServlet
      */
      public void doGet (HttpServletRequest request,
          HttpServletResponse response)
          throws ServletException, IOException
        {
          PrintWriter         out;
          String    title = " Message de SimpleServlet ";

              // fixer le type de la réponse
              response.setContentType("text/html");
              // écrire le texte de la éeponse
              out = response.getWriter();

              out.println("<HTML><HEAD><TITLE>");
              out.println(title);
              out.println("</TITLE></HEAD><BODY>");
              out.println("<H1>" + title + "</H1>");
              out.println("<P>Bonjour de SimpleServlet!");
              out.println("</BODY></HTML>");
              out.close();
         }
 }
```


Exécution de programmes sur un client Web

- **Intérêt**
  - **Décharger le serveur, en utilisant la capacité de traitement locale**
  - **Rendre l'exécution indépendante de la charge du réseau (en particulier pour affichage, animation)**
  - **Réaction rapide pour les programmes interactifs**



- **Mécanismes (nécessitent extension du navigateur)**
  - **Scripts**
    - **Tcl, Perl, JavaScript**
  - ***Applet* Java**

Applets

Consulter http://java.sun.com/docs/books/tutorial/applet/

- **Définition**
  - **Une *applet* est un programme Java inclus dans une page HTML. Quand la page est chargée par un client, l'*applet* est exécutée sur la machine virtuelle Java incluse dans le navigateur**
- **Forme**

  … `<applet code="MyApplet.class" width=120 height=150></applet>` …

  Le programme (bytecodes) est dans MyApplet.class, dans le même répertoire que le document HTML qui contient l'*applet*

  Le programme d'une *applet* doit dériver de la classe standard Applet

  ```
  public class MyApplet extends Applet {
          … }
  ```
- **Restrictions**
  - **Les capacités d'une *applet* sont limitées pour des raisons de sécurité. Actions interdites**
    - **charger des bibliothèques (importer du code nouveau)**
    - **accéder aux fichiers locaux sur le site du client (possible sous conditions depuis JDK 1.2)**
    - **ouvrir une connexion Internet, sauf vers le serveur d'où elle provient**
    - **lancer un nouveau processus sur le site du client**
    - **lire certaines caractéristiques du système d'exploitation local**

Applets

Exemple d'*applet*

Cet exemple montre une *applet* qui réagit aux principaux événements de son cycle de vie (premier chargement de la page HTML support, démarrage, changement de page, destruction) en affichant un message approprié dans une fenêtre.

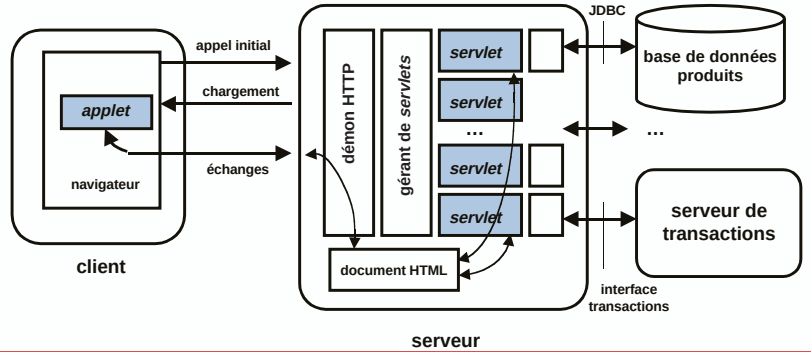
Les méthodes `init`, `start`, etc. sont définies dans la classe `Applet` et surchargées dans cet exemple. Elles sont automatiquement appelées par le navigateur ou l'`AppletViewer` (inspecteur d'*applets*) lors des événements correspondants

```
import java.applet.Applet;
import java.awt.Graphics;

                public class Simple extends Applet {

    StringBuffer buffer;

    public void init() {
        buffer = new StringBuffer();
        addItem("initializing... ");
    }

    public void start() {
        addItem("starting... ");
    }

    public void stop() {
        addItem("stopping... ");
    }

    public void destroy() {
        addItem("preparing for unloading...");
    }

    void addItem(String newWord) {
        System.out.println(newWord);
        buffer.append(newWord);
        repaint();
    }

    public void paint(Graphics g) {
        //Draw a Rectangle around the applet's display area.
        g.drawRect(0, 0, size().width - 1, size().height - 1);

        //Draw the current string inside the rectangle.
        g.drawString(buffer.toString(), 5, 15);
    }
}
```

extrait de http://java.sun.com/docs/books/tutorial/applet/

Schéma (très simplifié) d'une application sur le Web

- Exemple : application de commerce électronique
  - Sur le site client : des *applets* assurent la présentation des produits et la saisie des commandes ; on peut aussi utiliser directement des documents HTML
  - Sur le site serveur : des *servlets* assurent l'interfaçage avec d'autres composants (base de données de produits, gestion des stocks, gestion de transactions, etc.) et renvoient les réponses sous forme de documents HTML



Matériau complémentaire (non présenté en cours)

HTML - quelques exemples (2)

- Autres informations de présentation
  - Tables (nombreux attributs possibles : disposition relative des cases, couleur du fond, épaisseur des traits, etc.)
  - Listes (numérotées ou non)
- Inclusion d'images
  - <IMG SRC = "le fichier ou l'URL contenant l'image" - autres paramètres (échelle, alignement par rapport au texte, affichage de texte alternatif si l'image ne peut être affichée, etc.) >
- Liens hypertexte
  - <A HREF="l'URI associée au lien" NAME="le nom" autres paramètres (affichage dans une fenêtre autonome, etc.) > le texte ou l'image qui constitue l'hyperlien </A>
  - Ce lien est affiché de manière particulière par le navigateur (par exemple souligné en bleu)
  - Un "clic" de souris sur ce lien est interprété par le navigateur comme : demander le chargement (GET) du document désigné par l'URI du paramètre HREF

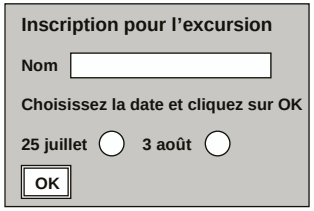
HTML - quelques exemples (3)

- Un premier exemple d'interaction
  - Jusque là ont été décrites des caractéristiques uniquement liées à l'affichage
  - On souhaite aussi permettre l'interaction entre client et serveur
  - Exemple : remplir un formulaire simple

le texte HTML

```
<HTML> <HEAD> <TITLE> Inscription </TITLE></HEAD>
<BODY>
<H1>Inscription pour l'excursion</H1>
<FORM ACTION = "http://sejourvacances.com/cgi-bin/choix" METHOD = POST>
Nom <INPUT NAME ="client" SIZE = 20><BR>
Choisissez la date et cliquez sur OK<BR>
25 juillet <INPUT NAME="date" TYPE=RADIO VALUE="2507"
3 ao&ucirc;t  <INPUT NAME="date" TYPE=RADIO VALUE="0308" <BR>
<INPUT TYPE=SUBMIT VALUE = "OK">
</FORM></BODY></HTML>
```

ce qui est affiché



ce qui est envoyé (par exemple)

client=Dupont&date=2507

c'est le programme (script) indiqué dans le paramètre ACTION qui traitera cette entrée

CGI : principes

Consulter le standard de facto (NCSA) : http://hoohoo.ncsa.uiuc.edu/cgi/

- Interface commune pour l'exécution de programmes sur le site d'un serveur Web
  - Multi-langages : scripts (Perl, Tcl, *shell* Unix, etc.), langages compilés (C, C++, etc.)
  - Localisation des programmes dans des répertoires spécifiques
    - /cgi-bin (exécutables), /cgi-src (sources des programmes compilés)
  - Conventions communes pour le passage de paramètres via HTTP
- Exemples de conventions (présentation simplifiée)
  - Récupération des paramètres d'entrée envoyés depuis le navigateur
    - par la commande HTTP GET
      - dans variable $QUERY_STRING, tout ce qui se trouve derrière le "?" dans l'URI transmise
    - par la commande HTTP POST
      - dans flot d'entrée standard `stdin`, encodage selon standard (~MIME)
  - Composition des résultats pour envoi au navigateur
    - dans un document (HTML ou autre, selon standard MIME
    - sous forme d'une URI (pointant vers le résultat)

CGI : problèmes

- Le principal problème de l'utilisation de CGI est la sécurité
  - CGI permet (potentiellement) à un client de faire exécuter un programme quelconque sur un serveur en lui passant des paramètres quelconques
  - Dangers
    - pénétration dans le système du serveur (via exécution de scripts Unix)
    - extraction d'informations confidentielles
- Mesures pour améliorer la sécurité
  - Les répertoires `/cgi-bin` et `/cgi-src` doivent être protégés (accès réservé à l'administrateur)
  - Il faut éviter d'y placer des commandes qui exécutent leurs paramètres comme un programme - cela revient à laisser le client écrire son propre programme
    - Exemples de telles commandes: `system` (dans *shell* Unix), `eval` (dans Tcl, Perl, etc.)
  - Il faut filtrer les paramètres des programmes (filtre spécifique à chaque programme)
    - Attention aux paramètres qui provoquent une exception dans le programme (type illégal, valeur hors limites, chaîne trop longue, etc.)
  - Le processus qui exécute le programme CGI doit avoir des droits limités
    - Il ne doit en particulier pas s'exécuter avec les droits (`uid`) de `root`, même s'il est lancé par `root`

Javascript

Consulter http://developer.netscape.com/tech/javascript/

- Qu'est-ce que JavaScript ?
  - Langage de script (interprété) destiné à être exécuté sur un site client Web
    - il y a aussi une utilisation possible côté serveur
  - Un programme JavaScript est inclus dans une page HTML (comme une *applet*)
    - <script language="JavaScript"> … texte du programme …</script>
  - L'interprète JavaScript est inclus dans le navigateur (JavaScript est issu de Netscape)
  - Malgré son nom, JavaScript ≠ Java. JavaScript a un modèle à objets rudimentaire
- Usages de JavaScript
  - Interactions locales sur le site client (évite la consultation du serveur)
    - modifier dynamiquement le contenu d'une page HTML
    - afficher sélectivement des images
    - afficher des boîtes de dialogue, contrôler la validité des données d'un formulaire
    - gérer un historique des documents visités
- JavaScript versus applets Java
  - Domaine plus restreint pour JavaScript (formulaires, images, affichage local)
  - Langage de script vs langage compilé (développement plus rapide, interactif)
  - Sécurité moindre pour JavaScript (pas d'équivalent de la JVM)

Javascript : exemples

Exemple 1 : alternance entre images multiples

```
…
<SCRIPT language="JavaScript">
var temp ="";
var image1 = image.gif;
var image2 = Images/une_autre_image.gif
</SCRIPT>
…
<A HREF="htpp://..."
onMouseOver="temp=image1;
image1=image2; image2=temp;
document.mon_image.src=image1;">
<IMG SRC="image.gif" NAME=mon_image></A>
…
```

Dans cet exemple, une image est affichée (et sert d'ancre pour un lien hypertexte). Quand le curseur de la souris entre ou sort du cadre de l'image, une image différente est affichée (on alterne entre 2 images)

Exemple 2 : une horloge simple (affiche heure courante)
(extrait de : http://www.multimania.com/dliard/Sciences/Informatique/Langages/Scripts/Javascript)

```
<HTML>
  <HEAD>
    <SCRIPT LANGUAGE="javascript">
      <!--
      function clock(){
        var date=new Date();
        string=" "+date.getHours()+' : '+
               date.getMinutes()+' : '+
               date.getSeconds();
        document.forms[0].elements[0].value=string;
        setTimeout('clock()',150);
      }
      //-->
    </SCRIPT>
  </HEAD>
  <BODY onLoad="clock();">
    ...
    <FORM>
      Il est très exactement :
      <INPUT Type="text" Value="hh : mm : nn">
    </FORM>
    ...
  </BODY>
</HTMLd
```

Le type Date et les fonctions getHours, etc. sont prédéfinis

Web - aspects système : caches Web

- **Fonctions générales d'un cache (rappel)**
  - Introduire un niveau intermédiaire, d'accès rapide, entre le lieu de stockage d'une information et celui de son utilisation.
  - Objectifs :
    - réduire le temps moyen d'accès, en conservant les informations les plus utilisées
    - réduire le trafic entre les niveaux de stockage (pour le web : trafic sur l'Internet)
  - Hypothèse de travail (souvent vérifiée) : localité d'accès (réutilisation des informations)
- **Le web se prête bien à l'usage de caches**
  - Les informations changent relativement peu souvent
  - On peut travailler sur des regroupements de demandes (à plusieurs niveaux)
    - cache individuel sur disque
    - cache local pour un département, une entreprise, etc.
    - cache régional pour un ensemble de réseaux
- **Problèmes à résoudre**
  - Choix des informations à conserver
  - Politique d'élimination des informations quand le cache est plein
  - Rafraîchissement des informations supposées périmées
  - Coopération entre caches

Un cache populaire (gratuit) : Squid.
Voir http://squid.nlanr.net/Squid/

Problèmes des caches web (1)

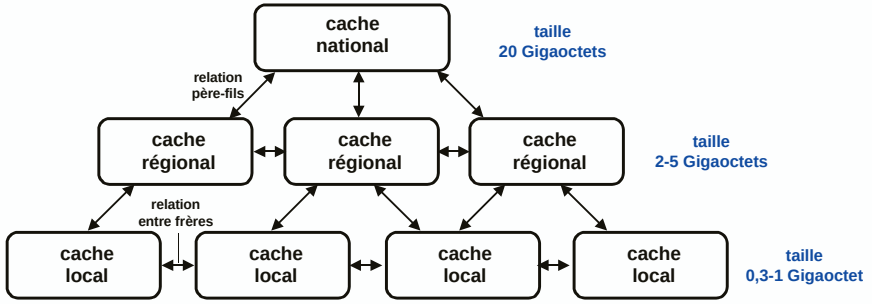
- **Politique de remplacement** (quels documents éliminer quand on a besoin de place)
  - FIFO (dans l'ordre des arrivées) : simple à réaliser, peu intéressant
  - SIZE : éliminer le document le plus gros (pour gagner de la place) : gain à court terme, mais risque de perte si le document éliminé était très demandé
  - LRU (*Least Recently Used*) : fondé sur l'hypothèse de localité, souvent utilisé
- **Cohérence** (comment garantir que les documents du cache sont à jour)
  - Invalidation : le serveur prévient le cache quand l'original est modifié
    - idéal, mais grosse charge de gestion pour le serveur (doit garder trace des copies)
  - TTL (*Time To Live*) : durée de vie limitée ; élimination ou rappel serveur à l'expiration
  - Autre solution : durée de vie proportionnelle à l'âge du document

Problèmes des caches web (2)

- **Coopération entre caches**
  - Hiérarchie : tout cache a un "parent", auquel il transmet la requête s'il ne peut la résoudre
    - Le parent fait de même (ou contacte le serveur s'il n'a pas de parent), puis répond au fils
  - Entre égaux : un cache transmet la requête aux autres caches "frères" et au serveur ; il prend la première réponse qui arrive
  - Le mode de coopération entre deux caches n'est pas fixé a priori et peut dépendre de la nature des requêtes

Exemple de hiérarchie de caches Web

- **Projet de caches du réseau Renater**



Rendement espéré : local 25%, régional 20%, national 15%

Voir : http://www.serveurs-nationaux.jussieu.fr/cache/