IV

La toile mondiale

‘World Wide WEB’

Introduction
1 La désignation dans le WEB, les URL
2 La représentation des données HTML, XML
3 Le protocole de communication HTTP.
Conclusion

Introduction

A partie d'un objectif initial de **collecte d'informations** sous la forme de documents hypertextes le Web évolue vers un modèle client serveur très **général**.

Désignation et liaison

⇨ Désignation et liaison de ressources :
URL '**U**niform **R**esources **L**ocator'

Représentation des données

⇨ Format de présentation des données
MIME 'Multipurpose Internet Mail Extensions'
HTML '**H**yper**T**ext **M**arkup **L**anguage'

Interaction de communication

⇨ Protocole de communication (accès aux documents sous forme de méthodes) :
HTTP '**H**yper**T**ext **T**ransfer **P**rotocol'

Premier chapitre

La désignation dans le WEB

Les URL

Désignation de base dans le WEB: La notion de localisateur

URL 'Uniform Resource Locators' RFC 1738 Décembre 1994

- La désignation avec les URL est placée dans le cadre d'un ensemble de **schémas** associés à des applications Internet:

ftp	File Transfer protocol
http	Hypertext Transfer Protocol
gopher	Gopher protocol
mailto	Adresse de courrier électronique
news	USENET news
nntp	USENET news avec NNTP
telnet	Session interactive
wais	Wide Area Information Servers
file	Noms de fichiers spécifiques
….	Autres schéma proposés.

- Un schéma définit une construction et une interprétation des noms ou adresses.

Syntaxe commune aux schémas

Forme générale des URLs

<nom_schema>:
<partie-spécifique-schéma>

Forme générale des URL Internet

- Pour Internet la partie spécifique du schéma commence par // **(en fait pas d'autre architecture de réseau utilisée)**

- En Internet on définit un accès à distance en **TCP/IP en tant qu'usager d'un OS**.

//<user>:<password>@<host>:<port>/
<url-path>

Signification des champs d’une URL Internet

User : Un nom d'usager optionnel.

Password : Un mot de passe optionnel.

Host : Le nom DNS d'un hôte (FQDN fully qualified domain name) ou une adresse IP

Port : Un numéro de port TCP (optionnel les protocoles ont un port par défaut).

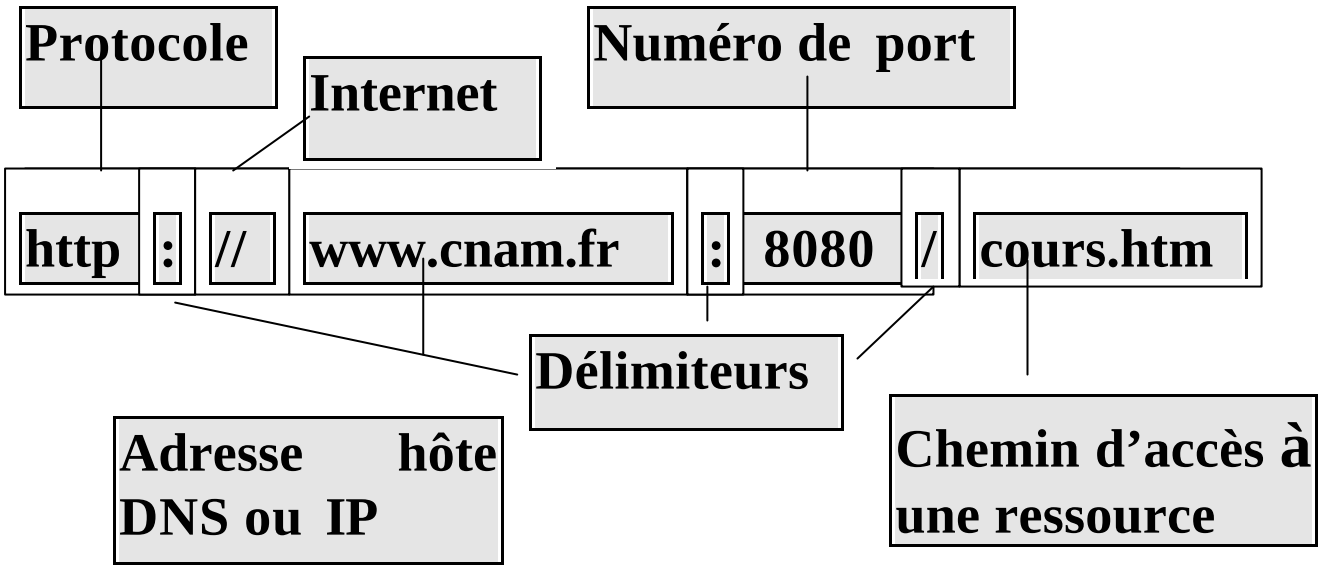
url-path: Le reste du nom est spécifique du schéma de désignation. Il s’agit d’un chemin d’accès à la ressource.

Remarque

Le "/" entre host (ou port) et url-path ne fait pas partie de l'url-path.

Le schéma d'adressage HTTP

Schéma très tourné vers la localisation



Aspects spécifiques HTTP de la partie définition du chemin d'accès

url-path = <path>?<search-part>

<path> : un chemin d'accès absolu à la ressource demandée sur le site serveur.

<search-part> : une chaîne de caractère qui complète le nom par des paramètres complémentaires définissant une question.

Complément :
Accès interne aux documents

- Adressage à **l'intérieur** d'un document.

- Non défini dans la RFC 1738 URL mais dans le cadre des URI ('Uniform Resource Identifiers').

url-path = <path>#<fragment>

<fragment> : le nom d'une ancre locale d'un document permettant un accès interne.

- Balise A présente dans le document HTML pour définir une ancre locale :
<A NAME="etiquette_fragment"></A>

Exemples d'URL

- Une URL pour une **question posée à un moteur de recherche** (partie question de la forme symbole=valeur).

http://www.google.com/search
?q=URL+syntax

- Une URL **avec question et accès interne** dans un document.

http://home.netscape.com/assist/
extensions.html#topic1?x=7&y=2

Remarques complémentaires

- La chaîne question contient des couples (**nom de variable, valeur)** séparés par des 'esperluettes' (ampersand &).
- Si des caractères réservés dans la création des URL (";" "/" "?" ":" "@" "&" "="+"$" ",") doivent figurer dans les **données une procédure d'échappement** est définie: **"%" HEX HEX** (deux fois 4 bits).

Réalisation de la liaison avec les URLs
Mise en œuvre dans le protocole HTTP

- L'analyse syntaxique de l'URL permet **de délimiter les éléments** de la référence**.**
- La liaison est très facile car les éléments sont presque tous des **adresses => directement utilisables pour la liaison.**

Détermination de l'adresse IP serveur

- Soit elle est **en clair**.
- Soit la partie **nom de domaine DNS** est soumise au serveur d'annuaire DNS qui retourne l'adresse IP de l'hôte distant

Ouverture de connexion TCP sur le serveur

- Au moyen de **l'adresse IP** et **du numéro de port** (ou du port par défaut **80**).

Utilisation par le serveur de la partie chemin d'accès de l'URL

- Chemin **d'accès absolu** dans le système de fichier du site distant.

Conclusion: Avantages Inconvénients du schéma d'adressage de base URL HTTP

- Les URL HTTP définissent un adressage absolu qui est **pratique pour localiser des ressources** mais uniquement pour cela.

Limitations

- **Manque d'outils d'adressage relatif** (pour définir une ressource relativement à une autre et éviter des problèmes de relocation des liens lors du déplacement d'une page HTML).

- Les URL comportent essentiellement des informations **d'adresses ('codées en dur') :** Protocole, Adresse DNS, No Port
Accès en échec (destruction/migration de ressource). Seul mécanisme possible de migration: **les liens de poursuite**.

- Les URL n'offrent pas de description du **contenu (informations méta).**

Conclusion : les améliorations

URN ‘Uniform Resource Names’

Les URN ont été définis au départ pour résoudre le problème **de persistance** des noms que ne possèdent pas les URLs.

URC ‘Uniform Resource Characteristics’

Rajouter aux ressources WEB des descriptions de caractéristiques (contenu, date, auteur, ….).

URI ‘Uniform Resource Identifier’ RFC 2396 Août 1998

Désignation WEB destinée à définir dans le même cadre toutes les améliorations des mécanismes d'adressage **en vigueur dont l’adressage relatif**.

Second chapitre

La représentation des données

Introduction aux formats de documents

Le format : XML

I Introduction aux formats de documents: un bref historique

SGML (‘Standard Generalized Markup Language’) ISO 8879 (année 1986)

- Un langage de balisage **très général**.

- Pour définir des documents électroniques, **indépendants de la forme visualisée** : Définir le contenu d'un document (rapport technique, livre, …).

- Un contenu est **transformé en autant de formats imprimés** que nécessaire**.**
Définir par ailleurs la forme imprimée.

- Le but principal reste de **transmettre, stocker des documents à imprim**er avec économies de volume et souplesse de formatage

SGML à un usage restreint à des spécialistes de la gestion documentaire.

HTML (‘HyperText Markup Language’)

- Développement de la documentation accessible en ligne.
=> Possibilité de liens **hypertextuels**.

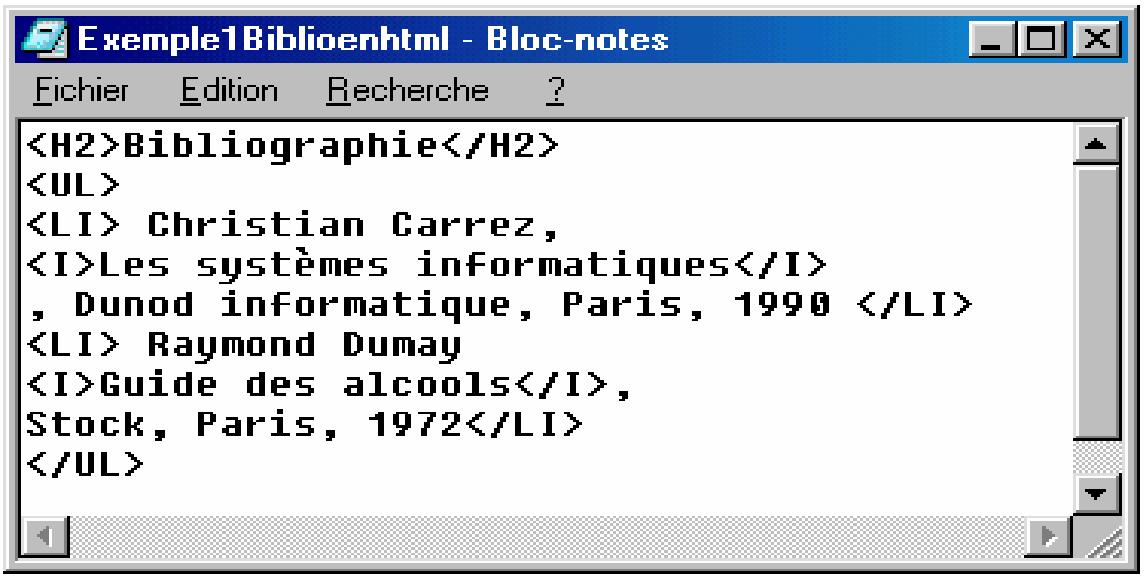
- Première études sur **le langage de balises hypertextes HTML** 1989 conçu à partir de SGML comme une version très simplifiée.

- Disponibilité et **première normalisation** en 1992

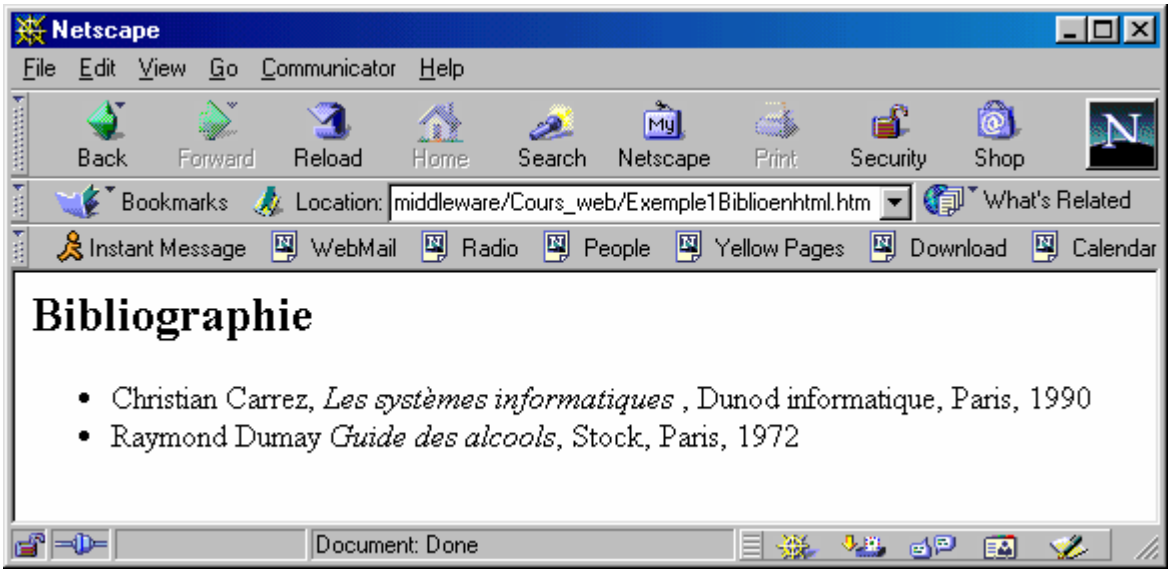
- **Démonstration de la très grande capacité** d'accès à l'information rendue possible par le WEB (HTML + HTTP + URL).

Le format de présentation: HTML

Exemple d'une bibliographie en HTML:



Une structuration de document complètement **orientée vers la présentation graphique.**



Quelques caractéristiques de HTML

- HTML est un **langage de balises** comme SGML :

  Exemple <H2>

- Normalisation **d'un ensemble figé** de symboles encadrés par "<" et ">".

- Les balises HTML définissent **des directives de mise en page du text**e encadré par un couple balise ouvrante, balise fermante:

<H2> xxxx </H2> un titre de niveau 2,
<UL> yyyy </UL> une liste sans numéros,
<LI> zzzzzz </LI> un item de la liste,
<I> ttttttttt </I> un texte en italique.

- **Pas de type de document** (pas de structuration) prédéfinie permettant de se mettre d'accord sur la structure d'un document.

Discussion des choix de HTML (1)
HTML: un langage trop graphique

- A l'origine HTML définissait des balises **indépendantes de la représentation** graphique des informations (Ex: H2 définit une tête de chapitre, UL définit une liste.

- Logiquement, la présentation graphique (listes, têtes de chapitres, …) **dépend d'une adaptation du navigateur à un contexte**.
Ex: présenter des informations en Braille pour des non voyants.

- Au fil du temps les fabricants ont introduits pour leurs **besoins des éléments graphiques**. (Ex: I pour italique, (FONT, CENTER, BGCOLOR). De même UL a le sens d'une indentation, pas d'une liste).

HTML est devenu spécifique des différents navigateur. C'est un langage graphique pour Netscape et Explorer.

Discussion des choix de HTML (2)
HTML: un langage non extensible

- Le langage HTML a été conçu pour être **assez simple** de sorte que le nombre et la signification des balises est **limité**.
Ex: Il n'existe pas de balisage pour la représentation des données en chimie (molécules, formules, valeurs numériques)

- Le langage HTML est **figé**. Toutes les balises utilisables sont définies au départ ce qui est intenable si l'on voulait prendre globalement en compte les besoins d'un grand nombre de métiers.

HTML est figé et assez simple : pas de possibilité d'introduire de nouvelles balises.

Discussion des choix de HTML (3)
HTML: un langage de description de documents non structurés

- HTML permet de définir de façon beaucoup trop **limitée** la **structure** d'un document.

- Pas de **vérification d'une structure** pour le document que l'on peut définir.
Ex: on peut créer un document commençant par une tête de chapitre H2 et poursuivant par une tête de chapitre H1.

HTML définit en fait un univers de documents plats.

Une recherche doit considérer un document HTML comme **une chaîne de caractères**.

Pas de moyen de partager **une structure de document** préétablie entre communicants.

HTML ne type pas les documents.

Conclusion : avenir de HTML

HTML peut rester très longtemps utilisé comme langage graphique assez simple pour les navigateurs WEB.

HTML présente des limitations trop importantes au plan du typage pour rester à terme le support de la représentation des données échangées sur le WEB.

- Non **séparation** du document et de sa présentation graphique.

- Non **extensibilité**.

- Non **structuration**, **typage**.

Représentation des données

XML (‘eXtended Markup Language’)
Recommandation W3C février 1998

Introduction à XML
Principales caractéristiques de XML
Corps d’un document XML
Prologue d’un document XML
- **a) Déclaration XML**
- **b) Les instructions de traitement**
- **c) Déclaration de type de document**

Introduction à XML

Convergence SGML HTML

- **SGML** riche, lourd, mal adapté au Web. **HTML** adapté à la navigation, mais limité par un ensemble de balises figé et réduit.

- **Groupe de travail XML** à partir de août 1996 qui est composé essentiellement de membres du groupe de travail SGML.

- Recherche d'un langage **assez facile** à la HTML avec une richesse proche de SGML.

- XML : un **sous-ensemble de SGML**, qui élimine des points trop ciblés sur certains besoins.

- Un document XML est **conforme** SGML.

Les principes directeurs du groupe de travail XML (1)

1. XML doit être un langage de présentation de l'information utilisable dans **Internet**.

2. XML doit supporter la **variété** la plus large possible **d'applications**.

3. XML doit être **compatible** avec **SGML**.

4. Il doit être **facile** d'écrire des codes qui **traitent** les documents XML.

5. Les caractéristiques **optionnelles** de XML doit être **minimum**.

Les principes directeurs du groupe de travail XML (1)

6. Un document XML doit **être lisible par un être humain** et raisonnablement **clair**.

7. La **conception** d'un document XML doit être **rapide**.

8. XML doit être défini **formellement** et de façon concise.

9. Les documents XML doivent être **faciles à créer.**

10. **La concision** des documents XML **n'est pas jugée importante** => mise en œuvre de mécanismes de compression si nécessaire.

Quelques conséquences pour XML

Extensibilité: pouvoir définir de nouvelles balises.

Structuration : pouvoir modéliser des données d'une complexité quelconque.

Validation : pouvoir vérifier la conformité d'une donnée avec un modèle de structure.

Indépendance du média: pouvoir formater un contenu selon des représentations diverses.

Interopérabilité : Pouvoir échanger et traiter une donnée en utilisant de nombreux types de logiciels.

Autres différences XML HTML

- **Aucune balise** ne peut être omise.

- Le balisage doit être '**bien formé**' .

- Toutes les valeurs d'attributs doivent être **délimitées** (entre ' ou ").

- Les éléments vides sont **explicités.**

- **Les symboles** (noms d'éléments et d'attributs) sont **sensibles à la casse** ("case-sensitive").

- **Interopérabilité** : Codage caractère **unicode** pour le contenu et les balises.

I Principales caractéristiques de XML

XML : langage de niveau méta

XML n'est pas un langage de balises de plus (comme HTML).

XML permet de définir de nouveaux langages de balises (comme SGML).

Notion de langage de niveau méta

- Un langage qui permet de **redéfinir des éléments d'un autre langage**,

- Un langage qui permet de **créer complètement de nouveaux éléments** dans un autre langage (ici essentiellement des balises).

XML: de la syntaxe (pas de sémantique)

On définit un ensemble aussi varié que possible de balises mais qui ne sont associées à aucun comportement.

Z1 XML n'est pas aussi abstrait que la **BNF 'Backus Naur Form'**mais est du même genre
Z2 En fait il y a **un tout petit peu de sémantique** dans XML (éléments, attributs)

Les spécifications de comportement doivent venir d'ailleurs.

- Dans la publication de documents il s'agit de feuilles de styles ("style sheets" = **formes déclaratives** de présentation graphique**)**.
Un document XML n'est pas directement publiable de façon élégante sur un navigateur.

- Dans d'autres domaines il pourra s'agir d'une application spécifique **(forme impérative**) offerte par un composant logiciel jouant le rôle d'un interpréteur ou processeur XML.

XML : un langage verbeux

Choix délibéré: XML définit tout en format caractère (lisible).

- Avantage : pour le **développement** des codes, **la mise au point, l'entretien** et pour **l'interopérabilité** (choix des caractères UCS 'Universal Character Set' ISO Unicode).

- Inconvénient: Les données codées en XML sont **toujours plus encombrantes** que les mêmes données codées en binaires.

L'espace mémoire, l'espace disque et la bande passante sont devenus **moins chers**.
Les **techniques de compression** sont accessibles **facilement et gratuitement**.
Elles sont applicables **automatiquement** dans les communications par modems ou en HTTP/1.1 et **fonctionnent bien** avec XML.

=> On atteint les mêmes encombrements qu'avec du format binaire.

XML : une famille de technologies

XML 1.0, la spécification de base décrivant la structuration des documents.
Xpath, un langage permettant de naviguer dans des documents XML.
XML Namespaces, une spécification pour l'unicité de la désignation des éléments XML.
DOM, une API pour interroger et manipuler des documents XML (et HTML)
RDF, Description meta des données.
XML Schéma, Types de documents.
Xlink, la spécification des liens en XML.
CSS, un langage de feuilles de styles applicable à XML comme à HTML.
XSL, un langage pour définir la présentation graphique des documents.
XSLT, un langage de transformation de documents XML.

Etc ….

Structure d'un document XML

Notion de structure logique

- Un document est composé de :

1) **Un prologue** qui comporte:
   . Déclarations
   . Instructions de traitement
   . Type du document

2) **Un ensemble d'éléments** en arbre.

3) **Des commentaires**

Les commentaires sont définis par les balises ouvrantes <!-- et fermantes -->

Exemple: <!-- Un commentaire -->.

Notion de structure physique

- Un document est composé d'unités de source appelées **entités**.

- Les **entités sont emboîtées** les unes dans les autres en commençant par l'entité racine (document) => l'ensemble forme un arbre.

- Une entité peut référencer une autre entité pour forcer **l'inclusion** à la place dans le document.

- Une entité est **nommée** (alias).

- Une entité peut être **interne** (son nom est un raccourci pour sa valeur).

- Une entité peut être **externe** et représenter un fichier XML local ou distant.

II Corps d'un document XML:

XML définit un cadre ('framework') pour baliser n'importe quel type de données structurées par un ensemble quelconque de balises ('tags').

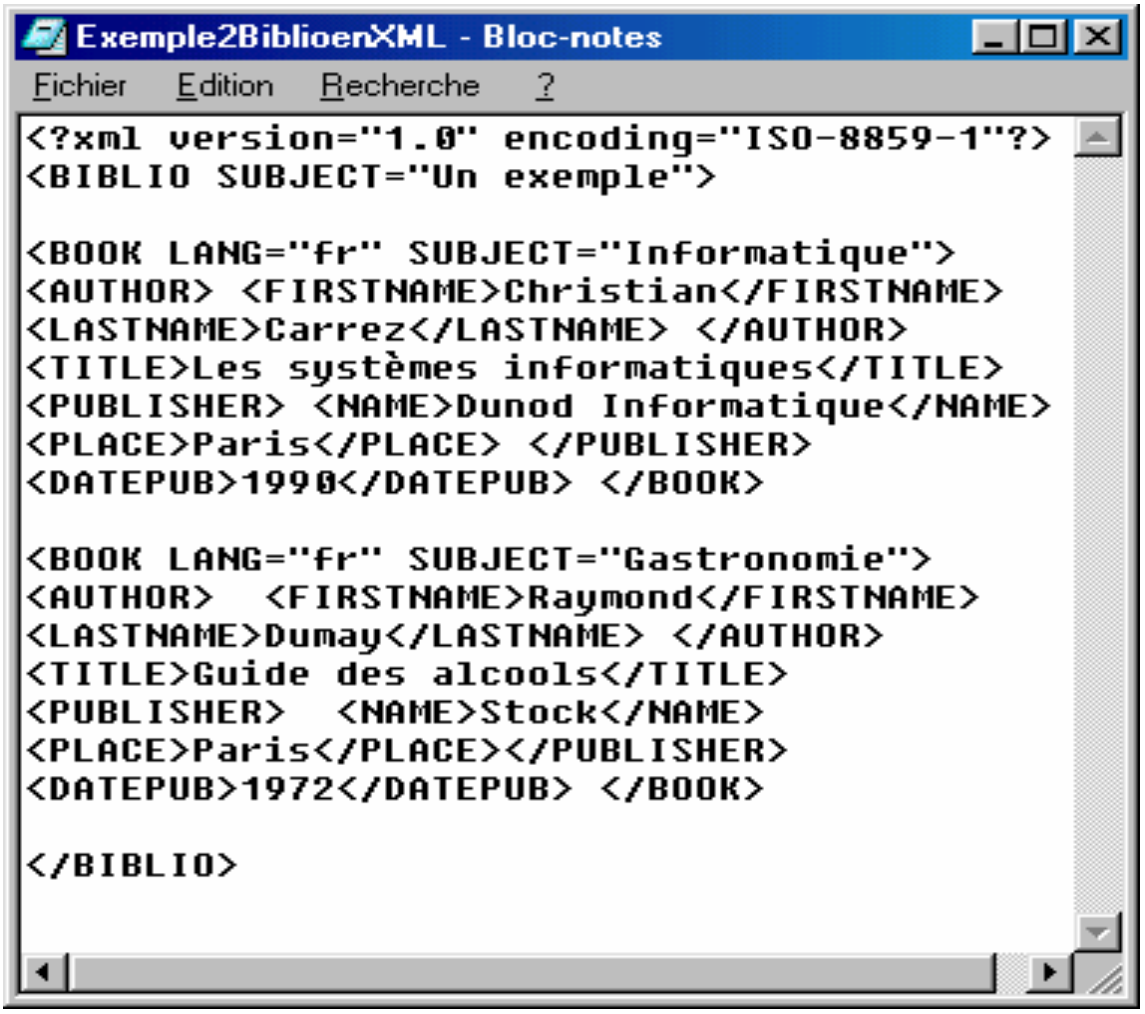
=> **Notion d'élément XML**: définit une donnée sous la forme d'une chaîne de caractères comme ayant un sens indiqué par une balise (un prix, un titre, …)

```
<message>
    <de>Jean</de>
    <pour>Jacques</pour>
    <objet>Rappel</objet>
    <texte>On se voit demain à 10h</texte>
</message>
```

=> **Notion d'attribut XML**: permet d'associer une donnée à une balise sous la forme d'un attribut (une localisation, un codage, ….)

Un exemple complet

Une bibliographie en XML



Exemple après passage en analyse syntaxique

Explorer version 5 contient un analyseur xml baptisé MSXML . Il produit l'arbre d'analyse syntaxique suivant
(Il existe de très nombreux autres parseurs).

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <BIBLIO SUBJECT="Un exemple">
  - <BOOK LANG="fr" SUBJECT="Informatique">
    - <AUTHOR>
        <FIRSTNAME>Christian</FIRSTNAME>
        <LASTNAME>Carrez</LASTNAME>
      </AUTHOR>
      <TITLE>Les systèmes informatiques</TITLE>
    - <PUBLISHER>
        <NAME>Dunod Informatique</NAME>
        <PLACE>Paris</PLACE>
      </PUBLISHER>
      <DATEPUB>1990</DATEPUB>
    </BOOK>
  - <BOOK LANG="fr" SUBJECT="Gastronomie">
    - <AUTHOR>
        <FIRSTNAME>Raymond</FIRSTNAME>
        <LASTNAME>Dumay</LASTNAME>
      </AUTHOR>
      <TITLE>Guide des alcools</TITLE>
    - <PUBLISHER>
        <NAME>Stock</NAME>
        <PLACE>Paris</PLACE>
      </PUBLISHER>
      <DATEPUB>1972</DATEPUB>
    </BOOK>
  </BIBLIO>
```

Sur l'arbre d'analyse syntaxique:
- \- signifie qu'une branche est dépliée détaillée
- \+ signifie qu'une branche est repliée.

Quelques caractéristiques de la syntaxe XML au travers de l'exemple

L'entête XML

- Définit la **version xml** utilisée: ici **1.0**.

Les éléments XML

- **Contenu encadré** par une balise ouvrante <BOOK> et une fermante </BOOK>
- La **casse** des balises doit être **respectée** (BOOK est différent de book).
- Les éléments doivent **s'imbriquer** les uns dans les autres (sans chevauchement) pour former un arbre.

Les attributs d'éléments

- Définissent des **données** à l'intérieur des balises. <BOOK LANG="fr" SUBJECT="Informatique">
- Les valeurs des attributs sont **encadrées** par des apostrophes ou des **guillemets**.

Choix de description d'une donnée: attribut d'élément ou contenu d'élément

En tant qu'attribut d'élément

```
<LIVRE LANGUE="fr"
        SUJET="Informatique">
</LIVRE>
```

En tant que contenu d'élément

```
<LIVRE>
    <LANGUE> fr </LANGUE>
    <SUJET> Informatique </sujet>
</LIVRE>
```

Recommandation

- Utiliser les attributs avec parcimonie: les attributs brisent **la structuration**.
- Utiliser les éléments à chaque fois qu'il s'agit d'une **donnée applicative.**
- **Réserver les attributs pour les aspects externes**.

Cas typique : définition d'une URL dans un lien <LIEN ref="URL"> texte </LIEN>

Documents bien formés

Un document XML a deux facettes logique et physique qui doivent être toutes les deux bien formées.

Si un document XML respecte les règles de la grammaire XML on dit qu'il est **bien formé.**

Les règles d'un document bien formé

- Toute balise **ouverte** doit être **fermée**,
  Ex: <livre> ….. </livre>
- L'ensemble des balises est **correctement imbriqué**. Exemple mal formé

- Les valeurs d'attributs sont entre **guillemets**
- Les noms des attributs sont en **minuscules**.
- Les **balises uniques correspondent à des documents vides** et sont notées:
  Ex : <IMG SRC="image.gif"/>
- Les caractères < & sont notés < &
- Un document commence par une **déclaration**

<?xml version="1.0" standalone="yes"?>

III Le prologue d'un document XML

III.1) La déclaration XML

```
<? xml version="1.0"
                    encoding="iso-8859-1" ?>
```

- Un **numéro de version obligatoire** (une seule version 1.0 à ce jour).

- **encoding=** Choix d'un codage des caractères utilisé pour coder le document. XML utilise les jeux de caractères de la norme ISO 10646 ou Unicode, voir http://www.unicode.org
  Codes fréquemment utilisés:
  UTF-16 : codage sur 16 bits
  UTF-8 : codage sur 8 bits
  ISO-8859-1

- **standalone=**'yes' ou 'no' permet de définir un document auto suffisant ou dépendant d'informations externes.

III.2) Les instructions de traitement ('Processing instructions')

- Permettent de passer **des directives** aux applications (pragma) sous la forme :

<?application_cible directive+ ?>

- **Application_cible**: nom d'une application de traitement (sauf xml qui est réservé).

- **Directive+**: plusieurs chaînes de caractères interprétables par l'application.

Exemple type: Définition d'un ensemble de feuilles de styles en XSL pour permettre l'affichage du document à l'écran.

```
<?xml:stylesheet type="text/xsl"
                 href="simple.xsl" ?>
```

III.3) Déclaration de type de document DTD 'Document Type Definition'

DTD : une notion héritée de SGML permettant la définition formelle de la structure d'arbre d'un document XML

- C'est une information de niveau méta pour qu'un analyseur syntaxique (parseur) vérifie la **conformité** du document réel qui suit avec une spécification de structure.
  . **intitulé** des balises,
  . **imbrication** des balises,
  . **caractère obligatoire ou facultatif** de la présence de certaines informations,
  . **enchaînement**.
- Une DTD est **optionnelle** en XML :

En son absence: on constate une structure.
En sa présence: on contraint la conformité d'un document XML.

- Lorsqu'un document XML **possède** une DTD associée et **la respecte**, on dit qu'il est **valide**.

Usages de la notion de DTD

Les usages habituels du typage

- Des **groupes** d'utilisateurs **indépendants** peuvent se mettre d'accord sur la structure commune des informations qu'ils partagent.

- **Un utilisateur** peut vérifier la validité de ses propres données.

- Une **application** informatique peut vérifier qu'une structure de données reçue est **valide**.

Z De nombreux forums sont apparus pour définir des **DTD standards** dans différents métiers => remplacement en vue de l'EDI ('Electronic Data Interchange')

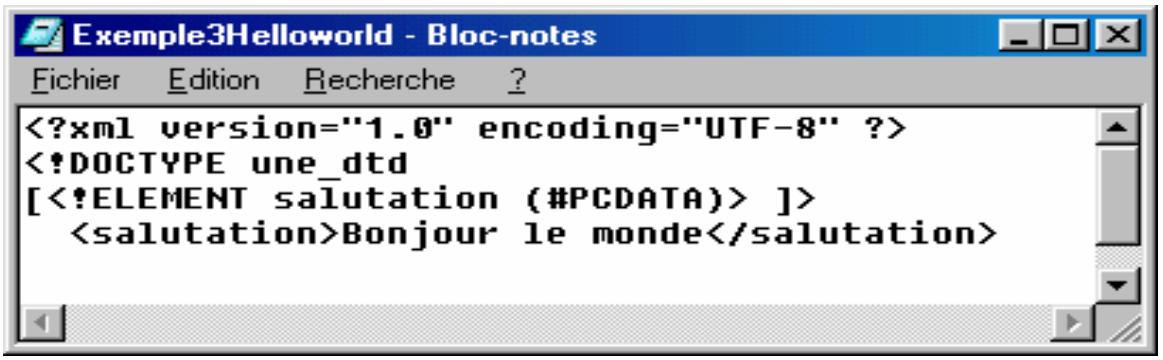
Exemples:
CommerceNet's http://www.xmlx.com/.
Ou http://www.schema.net/.

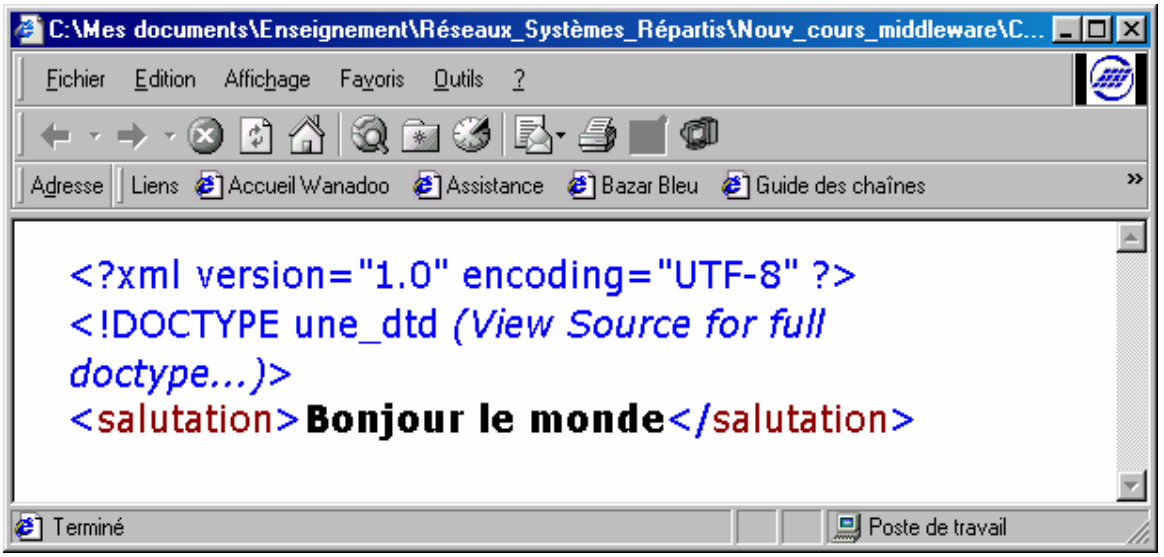
Définition interne d'une DTD

- Déclarée dans le document XML elle apparaît dans le prologue **après la déclaration xml**.

- Forme <!**DOCTYPE** nom_de_la_dtd **[**contenu_de_la_dtd**]**>

Exemple avec dtd interne





Définition externe d'une DTD

- Comme pour toute partie d'un document XML, la DTD peut être définie sous la forme d'une **entité externe**.
- On peut ainsi réutiliser des dtd, partager des dtd entre usagers.
- On déclare l'URI (ou le fichier) ou se trouve la DTD avec l'attribut
  SYSTEM pour une dtd privée
  PUBLIC pour une dtd publique.

Exemple avec DTD externe

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE exdtd SYSTEM dtdsalut.txt>
<salutation>Bonjour le monde</salutation>
```

Dans le fichier dtdsalut.txt on trouve:

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

Exemple : dtd complexe W3C (format de spec)http:://www.w3.org/XML/1998/06/xmlspec-v21.dtd

Les déclarations de types d'éléments

Déclaration d'un élément

Les éléments XML sont déclarés par:

<!**ELEMENT** nom_élément
(contenu_élément**)**>

- Nom : **le nom de l'élément** soit le nom de la balise dans un document.

- Contenu: **un type de base** (chaîne de caractères avec variantes) ou un type construit (en fait une séquence d'éléments).

Types de base

EMPTY : Chaîne vide

<! ELEMENT vide (EMPTY)>

Eléments vides: support d'attributs

Notation développée d'un élément vide

<vide></vide>

Notation compactée d'un élément vide

<vide/>

#CDATA : Chaîne non analysée

La chaîne est supposée contenir des données qui n'ont pas à être parsées.

#PCDATA : ('Parsed Character DATA')

Chaîne de caractères devant être parsée.

- Elle ne doit pas contenir de caractères XML spéciaux <,>,& …. => utilisation d'un mode transparent < > & …..
- Si une valeur PCDATA contient des éléments, ils doivent aussi être déclarés.

ANY : Une chaîne quelconque.

Elément avec fils : séquence

Types d'éléments construits à partir d'autres types d'éléments (types article).

- Les types d'éléments utilisés (les fils) sont à décrire par ailleurs.

- On peut préciser la structure (séquence) des fils au moyen **de constructeurs.**

Nom_élément	une seule occurrence
Nom_élément+	1 ou n occurrences
Nom_élément*	0, 1 ou n occurrences
Nom_élément?	0 ou 1 occurrence (valeur optionnelle)
Nom1|Nom2	alternative
,	séquence
( )	parenthèses

Types éléments 'mixtes'

<!ELEMENT mémoire (titre, résumé, remerciements?, introduction, chapitre+, conclusion, bibliographie,#PCDATA) >

Les déclarations d'attributs.

- Les listes d'attributs 'attlist' définissent pour un élément la liste des attributs de cet élément avec leurs propriétés.

Nom de l'attribut,
Type de l'attribut,
Valeur par défaut.

- Forme d'une liste d'attributs

<!ATTLIST nom_element
nom_attribut1 type1 défaut1
…
nom_attributn typen défautn>

Le paramètre défaut

#REQUIRED:attribut obligatoire à fournir
#IMPLIED : attribut optionnel.
#FIXED 'value': attribut dont la valeur est invariable et égale à 'value'.
'value': une valeur pour l'attribut quand celle ci n'est pas donnée.

Typage des attributs (1)

Trois catégories de types: les chaînes, les listes de chaînes, les valeurs énumérées. Selon des modalités variées.

CDATA : Les valeurs possibles sont des chaînes de caractères non analysées par le parseur **('Character DATA')**.

Ex: <!ATTLIST fichier
cheminaccès CDATA #REQUIRED>

ID : Les valeurs possibles sont des identifiants uniques.

```
Ex:     <!ATTLIST dossier
            ident   ID      #REQUIRED
            nom   CDATA   #IMPLIED>
```

IDREF, IDREFS : Les valeurs possibles sont des noms d'identifiants uniques déclarés par ailleurs (resp des listes).

Typage des attributs (2)

ENTITY, ENTITIES : la valeur est un nom d'entité déclaré et existants par ailleurs resp une liste de noms).

NMTOKEN, NMTOKENS : la valeur est obligatoirement un mot clé valide de xml (resp une liste de mots clés).
xml: Les valeurs qui commencent par xml: sont prédéfinies.
Ex: xml:lang désigne une langue dans laquelle un document est rédigé.

NOTATION: La valeur est le nom d'une notation.

Valeurs énumérées : Soit des notations de valeurs déjà définies soit des valeurs.
Séparation par des barres (val1|val2|…)
Déclaration : <!ATTLIST paiement type (chèque|CB|liquide) "liquide">
Utilisation : <paiement type="chèque">

Les déclarations d'entités.

Les entités supportent l'organisation physique d'un document.

- Les entités sont des textes sources constituant des parties d'un document
Ex: Un texte xml, un fichier image ….

Entités analysables (parsed) et entités non analysables (unparsed)

- Les entités sont dites **analysables** s'il s'agit de **texte xml** insérable tel que comme une partie d'un document xml.

- Les entités **non analysables** ("unparsed') sont des textes non xml (pour une autre application qu'un processeur xml).

Z Les entités non analysables doivent faire l'objet d'une définition de notation (voir plus loin) qui définit leur format et une application pour les prendre en charge.

Entités internes et externes

Entité interne: Il s'agit d'une entité définie dans le cadre d'un document.

Syntaxe de déclaration:
<!ENTITY nom_entité "valeur_entité">
Exemple: <!ENTITY gf "Gérard Florin">

Syntaxe d'utilisation:
<author>&gf;</author>

Entité externe: Il s'agit d'une entité définie dans un fichier connu par son URI.

Syntaxe de déclaration:
<!ENTITY nom SYSTEM "URI/URL">
Ex: <!ENTITY gf SYSTEM
"http://deptinfo.cnam.fr:8080/ ……
/mon_nom.xml">

Syntaxe d'utilisation:
<author>&gf;</author>

Entités générales et entités paramètres

- Les entités analysées (‘parsées’) sont dites **paramètres** s'il s'agit d'un texte à insérer dans une dtd.

```
<!--Déclaration entité param "ISOLat2"-->
<!ENTITY % ISOLat2 SYSTEM
"http://www.xml.com/iso/isolat2-
                                xml.entities" >

 <!-- Une référence pour l'importer -->
 %ISOLat2;
```

- Les entités analysées (‘parsées’) sont dites **générales** s'il s'agit d'un texte à insérer dans un document.

Z Les entités paramètres et générales sont utilisées dans des contextes différents.
Il ne peut y avoir d'ambiguïté.

Les déclarations de notations.

- Pour définir **le format des entités non analysables** (en fait non xml).
  . identifient par un nom le format.
  . l'application qui traitent les données.

Exemple: Une image en format gif
<!NOTATION gif SYSTEM "sh_gif.dll">

Exemple: Un texte compressé format zip
<!NOTATION zip SYSTEM 'zip.exe' >
gif, zip : nom du format.
SYSTEM : notation dans une application (PUBLIC pour les normes)
zip.exe : une application pour ce format.

- Des valeurs de notation sont de type NDATA (Ex : NDATA zip).

Utilisation des notations

- Des éléments peuvent **avoir des attributs dont la valeur est une notation**.

```
<!NOTATION zip SYSTEM 'zip.exe' >
<!ELEMENT un_document ……
<!ATTLIST un_document
compression NOTATION (zip|gzip)
                              #REQUIRED
                        ….. >
```

- Autre exemple, des entités non xml peuvent être typées :

```
<!NOTATION jpeg SYSTEM
              "/usr/local/bin/afficheur_jpeg">
<!ENTITY animal SYSTEM
         "./photos/cheval.jpg" NDATA jpeg>
```

Le traitement des caractères

Les références à des entités caractères

Pour utiliser un caractère au moyen de son code interne, définition d'entités caractères (conforme codage UCS ISO/CEI 10646)

Ex: é Caractère é codé en décimal
é Caractère é codé en hexadécimal

Les entités caractères prédéfinies

- Les caractères spéciaux xml (<, >, & …) ont besoin d'un mode transparent dans les chaînes analysées (PCDATA).
- Les déclarations standards suivantes (a priori toujours disponibles) montrent que le caractère d'échappement est & #38.

```
<!ENTITY lt     "&amp;#38;#60;">
<!ENTITY gt     "&amp;#62;">
<!ENTITY amp    "&amp;#38;#38;">
<!ENTITY apos   "&amp;#39;">
<!ENTITY quot   "&amp;#34;">
```

Exemple d'utilisation des entités caractères

Exemple de la norme xml:

```
<!ENTITY %ed "&#xc9;ditions Gallimard" >
<!ENTITY droits "Tous droits
                    r&#233;serv&#233;s" >
<!ENTITY livre "La Peste : Albert Camus,
                    &#xA9; 1947 %ed;. &droits;" >
```

La Peste : Albert Camus, © 1947 Éditions Gallimard . Tous droits réservés

Commentaires:

- ed est considérée comme une entité paramètre interne. Elle est définie en interne et elle sert dans une dtd…
- droits est considérée comme une entité générale interne. Elle sert dans une valeur.
- Ces deux entités comportent des caractères considérés comme spéciaux (des accents français dans une bibliographie en anglais).

Un exemple récapitulatif de dtd : Une structure de rapport (1)

Auteur: Richard Erlander. Site Web: http://pdbeam.uwaterloo.ca/~rlander/

```
<!DOCTYPE REPORT [
<!ELEMENT REPORT (TITLE,(SECTION|SHORTSECT)+)>
<!ELEMENT SECTION (TITLE,%BODY;,SUBSECTION*)>
<!ELEMENT SUBSECTION
                    (TITLE,%BODY;,SUBSECTION*)>
<!ELEMENT SHORTSECT (TITLE,%BODY;)>
<!ELEMENT TITLE %TEXT;>
<!ELEMENT PARA %TEXT;>
<!ELEMENT LIST (ITEM)+>
<!ELEMENT ITEM (%BLOCK;)>
<!ELEMENT CODE (#PCDATA)>
<!ELEMENT KEYWORD (#PCDATA)>
<!ELEMENT EXAMPLE (TITLE?,%BLOCK;)>
<!ELEMENT GRAPHIC EMPTY>
```

Une structure de rapport (2)

```
<!ATTLIST REPORT security (high | medium | low ) "low">
<!ATTLIST CODE type CDATA #IMPLIED>
<!ATTLIST GRAPHIC file ENTITY #REQUIRED>
<!ENTITY xml "Extensible Markup Language">
<!ENTITY sgml "Standard Generalized Markup Language">
<!ENTITY pxa "Professional XML Authoring">
<!ENTITY % TEXT
            "(#PCDATA|CODE|KEYWORD|QUOTATION)*">
<!ENTITY % BLOCK "(PARA|LIST)+">
<!ENTITY % BODY "(%BLOCK;|EXAMPLE|NOTE)+">
<!NOTATION GIF SYSTEM "">
<!NOTATION JPG SYSTEM "">
<!NOTATION BMP SYSTEM "">  ]>
```

Conclusion XML (1)

Multiples avantages

- Un langage **effectivement assez simple**.
- XML permet la description précise et universelle des contenus de données, de sorte que la recherche et l'accès sont plus faciles pour des **plateformes hétérogènes**.
- XML rend possible l'arrivée d'une nouvelle génération **d'outils logiciels** :
  . de manipulation,
  . de transmission,
  . de visualisation de données distribuées.

Les inconvénients

- La simplicité de base conduit à **énormément de compléments de toutes natures**.
- **Ces lacunes** sont assez importantes pour voir apparaître **des améliorations xml successives** et des DTD très nombreuses qui ajoutent des détails.

Conclusion XML (2)

L'avenir de XML

XML pourrait permettre pendant un certain temps (combien?) et pour de nombreuses applications (celles qui ne posent pas de problèmes cruciaux de performances) de supporter dans un cadre unifié aussi bien:

La définition des documents.

Outils de bureautique, de documentation ..

La définition des données.

SGBD, logiciels de gestion, Échanges de Données Informatisé (EDI), …

La présentation des réseaux

Format des données échangées par messages ou par invocations de méthodes.

Bibliographie

- J.C. Bernadac, J.F. Knab 'Construire une application xml', Eyrolles.

- Extensible Markup Language (XML) 1.0 (Second Edition) W3C Recommendation 6 October 2000
http://www.w3.org/TR/2000/REC-xml-20001006
Versions antérieures
http://www.w3.org/TR/2000/WD-xml-2e-20000814
http://www.w3.org/TR/1998/REC-xml-19980210

- La page XML pour débutants de Emmanuel Lazinier
http://www.chez.com/xml/initiation/

- Les très nombreux cours XML sur le WEB.
http://www.xmlephant.com/

La communication dans le WEB

Le protocole HTTP

Introduction

- HTTP 'HyperText Transfer Protocol' est un protocole de communication client serveur défini à partir de 1989 pour supporter les échanges de données du 'World Wide Web'.

- Donc il est conçu à l'origine pour **transporter des textes HTML**.

- HTTP a **évolué considérablement** sous la pression d'utilisateurs très nombreux d'une version simple et rustique 0.9 à une version assez puissante 1.1

- Les versions successives de HTTP:
  HTTP/0.9
  HTTP/1.0
  HTTP/1.1

- Présentation des principales caractéristiques des versions 1.0 et 1.1

Quelques caractéristiques de base

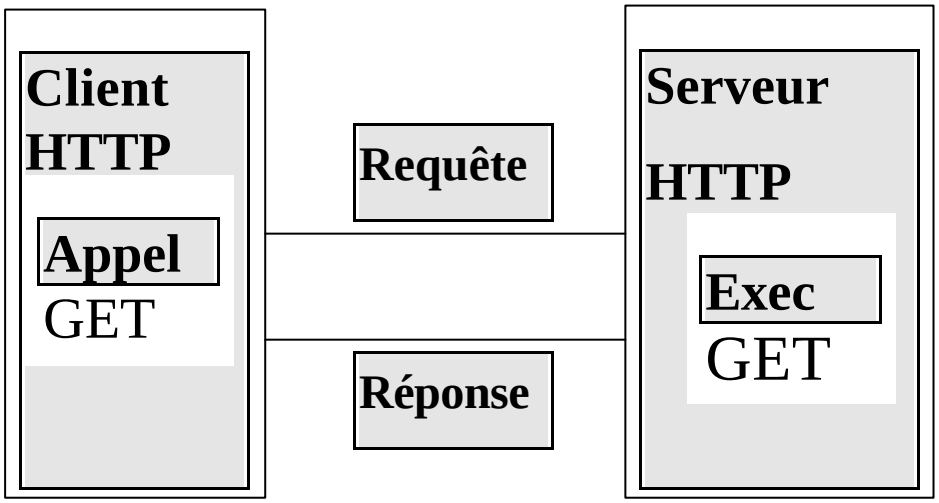
- HTTP se définit dans la RFC principale comme un protocole
  - de niveau **application**
  - **sans état**
  - orienté **objet ?**
  - dédié aux applications **distribuées**
  - **hypermédia**

- HTTP permet à des systèmes conçus indépendamment de dialoguer :
  - **typage** des données échangées.
  - **négociation de la représentation** des données.

- Les **applications visées** par les auteurs **sont très générales**: toutes les applications objets répartis au moyen de l'extensibilité des **méthodes et du typage** des données : (en fait il reste beaucoup à faire)

HTTP : un protocole client serveur basé sur le mode message

- Tous les échanges sont basés sur un couple de **messages requête réponse**.
- A chaque couple requête réponse est associé une '**méthode**'.



- Des méthodes **de base** sont définies.
- **Approche asynchrone** : un client n'est pas bloqué par une requête (il peut émettre plusieurs requêtes en parallèle).
- HTTP utilise le **transport fiable TCP** (d'autres réseaux sont possibles).

La notion de ressource

- HTTP :un protocole de niveau application utilisé pour transporter des données baptisées **ressources**.

- Une ressource est un **ensemble d'informations** identifié par une URL ou plus généralement une URI.

- Le plus souvent une ressource est un **fichier**.

- Dans les autres cas une ressource est un **ensemble de données** généré par un programme (un script, une question d’accès base de données …).

La version HTTP/1.0

RFC 1945, mai 1996, 60 pages

I Introduction
Structure des échanges HTTP/1.0

- Le client HTTP/1.0 ouvre une **connexion TCP** (le numéro de port par défaut de http est 80, un autre numéro est possible).

- Le client envoie **un message de requête** (exemple type : obtenir une ressource en fonction de différents paramètres).

- Le serveur répond par **un message de réponse** (la ressource demandée).

- Le client **ferme la connexion TCP**

HTTP/1.0 est un protocole sans état ('stateless'): pas de relation sur le serveur entre requêtes successives.

En HTTP/1.0 chaque ressource nécessite sa propre connexion TCP.

La liste des méthodes HTTP/1.0

Les trois méthodes principales

GET : Pour lire la valeur d'une ressource.

HEAD : Pour lire seulement l'entête.

POST: Pour faire exécuter un programme distant.

Les méthodes annexes

PUT : Pour envoyer une ressource sur un site distant.

DELETE : Pour détruire une ressource distante.

LINK : Pour établir un lien entre deux ressources.

UNLINK : Pour supprimer un lien.

II Le format des messages

- Les informations protocolaires échangées le sont sous forme de **lignes de textes**.

- Un message comporte:
    . une **ligne initiale** (structure différente dans les requêtes et les réponses),
    . **zéro ou plusieurs lignes entêtes**,
    . une ligne vide (un retour ligne LF ou un CRLF soit '13' '10' en ASCII),
    . **un corps de message** pas toujours présent (un document HTML,).

Soit encore

    Ligne initialeCRLF
    Entête1: valeur1CRLF
    Entête2: valeur2CRLF
    ….
    Entêten: valeurnCRLF
    CRLF (la ligne vide)
    Corps de message (pouvant être vide, texte ou binaire).

Un exemple

La requête

GET /index.html HTTP/1.0

La réponse

HTTP/1.1 200 OK
Date: Wed, 25 Oct 2000 10:02:12 GMT
Server: Apache/1.3.6 (Unix) PHP/3.0.7
Last-Modified: Wed, 18 Oct 2000 11:07:00 GMT
ETag: "ac-221e-39ed8454"
Accept-Ranges: bytes
Content-Length: 8734
Connection: close
Content-Type: text/html

<html>
<head>
<link rel="stylesheet" …….
……

II.1 La ligne initiale

Le format de la ligne initiale des requêtes

Trois parties séparées par des espaces

- Un **nom de méthode**,
- Le **chemin d'accès à la ressource**,
- La **version du protocole HTTP**.

Exemple : GET /chemin HTTP/1.0

Compléments

- Les noms de méthodes sont en **majuscule**. Exemple : GET, POST, HEAD, etc ….).
- /chemin est la partie de l'URL après le nom du site serveur qui **permet de définir localement le chemin** d'accès à la ressource.
- La version HTTP est de la forme **HTTP/x.x** (en majuscule x.x vaut 0.9, 1.0 , 1.1).

Le format de la ligne initiale des réponses

- **Trois parties** séparées par des espaces

. La **version** du protocole HTTP,
. Le **statut** de la réponse (code réponse),
. Un **texte** expliquant le code réponse.

- **Exemples caractéristiques** de réponses:
  HTTP/1.0 200 OK
  HTTP/1.0 404 Not Found

Compléments

- La version HTTP de la réponse est **identique** à celle de la requête HTTP/x.x.
- Le code réponse est **numérique sur trois chiffres décimaux**. Il est destiné au calculateur client.
- L'explication est **un texte en clair** pour être compris par l'usager client (la forme dépend du serveur HTTP).

Les codes réponses

Le premier chiffre définit la catégorie:

1xx un message informatif.
2xx une opération réussie.
3xx une redirection vers une autre URL
4xx une erreur du client
5xx une erreur du serveur

Les codes réponses les plus importants

"200" ; OK
"201" ; Created
"202" ; Accepted
"204" ; No Content
"301" ; Moved Permanently
"302" ; Moved Temporarily
"304" ; Not Modified
"400" ; Bad Request
"401" ; Unauthorized
"403" ; Forbidden
"404" ; Not Found
"500" ; Internal Server Error
"501" ; Not Implemented
"502" ; Bad Gateway
"503" ; Service Unavailable

II.2 Les lignes d'entête ('Header Lines)

- Fournissent des informations concernant le message de requête ou de réponse qui les contient (en particulier au sujet de la charge utile du message).

- Le format des lignes d'entête est le même que celui des courriers électroniques Internet définis par la RFC 822.

Une ligne par entête de la forme:

Nom-entête: valeur CRLF (ou LF).

Détails concernant le format:

- Le nom de **l'entête est indépendant de la casse** (non case-sensitive).
- Un **nombre quelconque d'espaces** et de tabulations peut séparer le : et la valeur.

- HTTP 1.0 **définit 16 entêtes principales** et **10 entêtes annexes**.
- Leur présence **obligatoire ou optionnelle dépend du type du message**.

Les entêtes principales normalisées en HTTP/1.0 (1)

1 Allow : donne la liste des méthodes supportées par le serveur**.**
Exemple : **Allow: GET, HEAD**

2 Authorization : définit les paramètres d'authentification d'un usager (lettres de créances 'credentials' principalement nom :mot_de_passe)
Authorization: credentials

3 Content-Encoding : définit le type de codage appliqué au contenu pour sa transmission. La principale utilisation concerne la compression des données.
Exemple : **Content-Encoding: x-gzip**

4 Content-Length : définit en décimal le nombre d'octets du corps du message. Quand le corps de message est présent cet entête est indispensable.
Exemple : **Content-Length: 3495**

Les entêtes normalisées en HTTP/1.0 (2)

5 Content-Type : indique le type des données contenues dans le corps du message comme en **MIME**.
Cinq catégories + type contenu:
text/html ou image/gif.
Entête obligatoire si le corps de message est non vide (application/octet-stream est la définition pour un type inconnu).
Exemple : **Content-Type: text/html**

6 Date: donne la date et l'heure GMT de traitement de la requête.
Exemple : **Date: Wed, 25 Oct 2000 10:02:12 GMT**

7 Expires : définis la date de validité du contenu d'un message.
Exemple : **Expires: Wed, 25 Oct 2000 10:02:12 GMT**

Les entêtes normalisées en HTTP/1.0 (3)

8 From: adresse mail de l'administrateur de l'usager émetteur de la requête ou demandeur de l'exécution d'un programme.
Exemple : **FROM webmaster@cnam.fr**

9 If-Modified-Since : utilisé dans une requête GET pour la rendre conditionnelle
Exemple : **If-Modified-Since: Wed, 25 Oct 2000 10:02:12 GMT**

10 Last-Modified: donne la date et l'heure GMT 'Greenwich Mean Time', de la dernière modification de la page.
Exemple : **Last-Modified: Wed, 18 Oct 2000 11:07:00 GMT**

11 Location : donne l'URL ou a été obtenue la ressource en cas de redirection.
Exemple :**http://www.w3.org/hypertext/WWW/NewLocation.html**

Les entêtes normalisées en HTTP/1.0 (3)

12 Pragma : pour transmettre des directives spécifiques de l'implantation du service.
Exemple : **Pragma : no-cache**

13 Referer : le client transmet au serveur l'URI à partir de laquelle l'URI demandée a été obtenue pour améliorer la gestion des caches, générer des liens arrières, détecter de mauvais liens …
Exemple: **Referer: http://www.w3.org/hypertext/DataSources/Overview.html**

14 Server: identification du serveur sous la forme "Program-name/x.xx" avec possibilité d'un commentaire en plus.
Exemple d'un serveur Apache :
Server: Apache/1.3.6 (Unix) PHP/3.0.7

Les entêtes normalisées en HTTP/1.0 (4)

15 User-Agent: définit le programme client émetteur de la requête sous la forme Program-name/x.xx (x.xx est le numéro de version de l'émetteur).
Exemple : Netscape 3.0 se définit par
User-agent: Mozilla/3.0Gold

16 WWW-Authenticate : du serveur vers le client pour transmettre des nonces (challenge d'authentification).
Contenu complexe : dépendant de la méthode d'authentification et du serveur (adresse IP client, estampille, … chiffrées en MD5).

Les entêtes annexes normalisées en HTTP/1.0 (1)

1 Accept: Utilisé pour indiquer une liste de types de données acceptées.
Ex:"*/*" indique qu'on accepte tout type.

2 Accept-Charset: Pour définir les jeux de caractères acceptés (en dehors des jeux de caractères standards)

3 Accept-Encoding: Similaire à l'entête accept : restreint le type des données acceptables dans le message de réponse.

4 Accept-Language: Pour définir les langues naturelles acceptées dans une réponse.

5 Content-Language: Pour définir le langage naturel utilisé dans les données transmises.

Les entêtes annexes normalisées en HTTP/1.0 (2)

6 Link: Méta information pour établir un lien entre la ressource décrite et une autre.

7 MIME-Version: Pour définir la version du format MIME utilisée.

8 Retry-After: Dans le cas d'une réponse 503 (service indisponible) cette entête permet de définir à quel délai la ressource pourrait redevenir disponible.

9 Title: Définit un titre pour la donnée.

10 URI: Définit une ou plusieurs URI (s'il elles existent) identifiant la ressource.

Remarque

- Des types non normalisés peuvent être ajoutés par les clients ou les serveurs (les entêtes non reconnues sont ignorées).

Résumé: Les entêtes courantes d'un message requête

Objectif des entêtes clients

- Définir le corps du message (s'il y en a un)
- Aider les administrateurs des serveurs à identifier les problèmes.

Les entêtes requêtes les plus fréquentes

2 Authorization
3 Content-Encoding
4 Content-Length
5 Content-Type
8 From
9 If-Modified-Since
12 Pragma
13 Referer
15 User-Agent

Résumé: les entêtes courantes d'un message réponse

Objectif des entêtes réponses

- Informer le client du contenu du corps du message.
- Informer le client des conditions de traitement de la requête.

Les entêtes réponse les plus fréquentes

1 Allow
3 Content-Encoding
4 Content-Length
5 Content-Type
6 Date
7 Expires
10 Last-Modified:
11 Location
14 Server:
16 WWW-Authenticate

II.3 Le corps de message

Les utilisations

- Contient une **ressource** demandée.

- Contient **l'explication d'une erreur**.

- Contient des **données envoyées du client au serveur** dans un message de requête (par exemple un fichier émis).

Les entêtes associées au corps

L'entête **Content-Type** doit définir le type MIME du contenu.
Content-Type: image/gif

L'entête **Content-Length** doit définir le nombre d'octets du corps de message.

L'entête **Content-Encoding** définit le format de codage du contenu.
Content-Encoding: Base64

II.4 Les différentes méthodes

La méthode GET

Pour obtenir le contenu d'une ressource.

tulipe::/users/ensinf/gerard _1 telnet www.cnam.fr 80
Trying 163.173.128.28...
Connected to bose.cnam.fr.
Escape character is '^]'.
GET /index.html HTTP/1.0

HTTP/1.1 200 OK
Date: Wed, 25 Oct 2000 10:02:12 GMT
Server: Apache/1.3.6 (Unix) PHP/3.0.7
Last-Modified: Wed, 18 Oct 2000 11:07:00 GMT
ETag: "ac-221e-39ed8454"
Accept-Ranges: bytes
Content-Length: 8734
Connection: close
Content-Type: text/html

<html>
<head>
<link rel="stylesheet"
href="http://www.cnam.fr/styles/cnam_steel …..>

La méthode HEAD

- La requête est exactement **la même** que pour la méthode GET.
- La signification est **différente**: le client demande le transfert des entêtes de la ressource sans la partie données.
- Pour obtenir des **renseignements sur une ressource sans la transférer** (pour économiser de la bande passante)

tulipe::/users/ensinf/gerard _7 telnet www.cnam.fr 80
Trying 163.173.128.28...
Connected to bose.cnam.fr.
Escape character is '^]'.
HEAD /index.html HTTP/1.0

HTTP/1.1 200 OK
Date: Tue, 31 Oct 2000 16:36:03 GMT
Server: Apache/1.3.6 (Unix) PHP/3.0.7
Last-Modified: Wed, 18 Oct 2000 11:07:00 GMT
ETag: "ac-221e-39ed8454"
Accept-Ranges: bytes
Content-Length: 8734
Connection: close
Content-Type: text/html
Connection closed by foreign host.

La méthode POST

- Une requête POST permet d'envoyer des données à un serveur pour **exécuter un programme** (c'est une sorte de RPC).
- On ajoute des entêtes pour décrire l**es données envoyées** du client au serveur (Content-Type:, Content-Length:)
- L'URI décrit le **programme à exécuter**.
- La réponse est constituée par les **données résultats** de l'exécution du programme.
- Exemple: exécution d'un script CGI.

```
POST /chemin/script.cgi HTTP/1.0
From: user@site_émetteur
User-Agent: HTTPTool/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: nnnn
```

Zone données : Le programme CGI reçoit les données sur son unité standard d'entrée STDIN.

Les autres méthodes de HTTP/1.0

1 PUT

- Permet **d'envoyer une ressource** dans une URI distante.

Remarque: Différence entre PUT et POST: POST fournit des données à faire traiter par un programme distant. Dans un PUT l'URL distante n'a pas à être un programme.

2 DELETE

- Permet **de détruire une ressource** définie par son URI.

3 LINK

- Pour **établir un lien** entre la ressource émettrice de la requête et la ressource définie par l'URI.

4 UNLINK

- Pour **supprimer un lien**.

La version HTTP/1.1

RFC 2068, janvier 1997, 162 pages

Introduction
Principes directeurs de HTTP/1.1

HTTP/1.1 est compatible avec HTTP/1.0 et apporte les améliorations suivantes:

Temps de réponse

- **Plusieurs échanges** client serveur successifs peuvent être réalisés **sur la même connexion TCP**.
- Gestion des **caches** par le protocole.
- Un message peut être émis **sans connaître sa longueur totale**.

Gestion de l'adressage

- De multiples **noms de domaines** peuvent être servis par la même adresse IP.

Remarques

- HTTP/1.1 impose le support de fonctionnalités de façon **obligatoire**.
- Des fonctionnalités **optionnelles** sont également définies.

Les 4 fonctionnalités obligatoires des clients HTTP 1.1

Entête Host:

- Plusieurs noms de domaines doivent pouvoir être servis par le même site pour éviter le gaspillage des adresses IP.

Exemple: deux noms de domaines peuvent être servis par la même machine. Exemple: www.cnam.fr et deptinfo.cnam.fr.

- En HTTP/1.1 chaque requête doit comporter l'indication du domaine concerné => L'entête Host est obligatoire puisque l'on désigne les ressources en relatif.

```
GET /chemin/ressource HTTP/1.1
Host: deptinfo.cnam.fr:8080
```

Exemple de mauvaise invocation

GET /index.html HTTP/1.1

HTTP/1.1 400 Bad Request
Date: Tue, 31 Oct 2000 16:31:11 GMT
Server: Apache/1.3.6 (Unix) PHP/3.0.7
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html

138
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>400 Bad Request</TITLE>
</HEAD><BODY>
<H1>Bad Request</H1>
Your browser sent a request that this server could not understand.<P>
client sent HTTP/1.1 request without hostname (see RFC2068 section 9, and 14.23)
: /index.html<P>
</BODY></HTML>

0

Connection closed by foreign host.

2 Obligation d'accepter le transfert des réponses par morceaux

- En HTTP/1.0 si un script produit des résultats **volumineux ou lentement** on doit connaître la longueur totale des données pour commencer à envoyer.
- L'entête **Transfer-Encoding: chunked** permet de traiter ce problème**.**
- Elle doit être connue du client HTTP/1.1
- On transmet la réponse par morceaux.

Format d'un morceau

- Chaque morceau est défini par sa longueur en hexadécimal suivi d'un point-virgule (première ligne).
- Exemple de morceau

2C; Tout commentaire ici est ignoréCRLF
Ici un texte avec longueur de 32 caractères.

Format d'un message par morceaux

- A la fin de tous les morceaux on place un **morceau vide** (de longueur 0).
- On peut ensuite placer des lignes de **postfaces** (footers) analogues à des lignes d'entêtes (traités comme des entêtes). Les postfaces servent pour les codes de contrôle ou les signatures
- A la fin une **ligne blanche**.

Un exemple complet

```
HTTP/1.1 200 OK
Content-Type: text/plain
Transfer-Encoding: chunked

2C
Ici un texte avec longueur de 32 caractères.
10
Texte de 16 cars
0
postface: valeur
ligne blanche
```

3 Obligation de gérer les connexions TCP persistantes

- En HTTP/1.0 les **connexions** TCP ne servent que pour une ressource.

- Quand un échange porte **sur plusieurs** ressources comme des images les ouvertures/fermetures sont très **coûteuses.**

- En HTTP/1.1 les **connexions TCP persistantes** sont les connexions par défaut de sorte que l'on peut transmettre de multiples requêtes réponses.

- L'entête "**Connection: close**" dans une requête demande la fermeture par le serveur de la connexion après la réponse

- Si la réponse contient **Connection: close** la connexion est fermée après la réponse (si on ne veut pas de connexions persistantes mettre toujours l'entête).

4 Entretien de la communication: obligation d'accepter la réponse "100 Continue"

- Pendant la transmission d'une requête le serveur doit transmettre si tout va bien une réponse de **code 100** avec commentaire **Continue**. S'il y a un problèmes le code réponse est associé au problème.

- Cette réponse sert dans le cas ou les **communications sont lentes**.

- Le client est averti que le traitement progresse normalement. Il **doit pouvoir traiter la réponse** "100 Continue" (peut-être simplement en l'ignorant).

- Exemple: **Réponses successives** serveur.
  HTTP/1.1 100 Continue
  HTTP/1.1 200 OK

Les 8 fonctionnalités obligatoires des serveurs HTTP/1.1

1 Obligation d'utiliser l'entête Host:

- Si un serveur reçoit une requête client sans l'entête host il doit répondre:

```
HTTP/1.1 400 Bad Request
Content-Type: text/html
Content-Length: 111

<html><body>
<h2>No Host: header received</h2>
HTTP 1.1 requests must include the Host: header.
</body></html>
```

- En fait pour une URI absolue (avec nom de domaine) il n'est pas nécessaire de définir le domaine par l'entête host

⇨ on oblige cependant le client à le faire.

2 Obligation d'accepter les URL absolues

- La situation actuelle avec un entête **Host:** est considérée comme temporaire.

- Dans les futures versions de HTTP (par exemple HTTP/1.2) on devrait normaliser une définition URI absolue du chemin d'accès.

- Exemple des requêtes futures:

  GET http://www.cnam.fr/index.html HTTP/1.2

- Actuellement les clients HTTP/1.1 n'envoient pas ce type de requête mais les serveurs doivent les accepter si elles sont néanmoins émises (à titre de transition vers la nouvelle solution).

3 Obligation d'accepter les transferts de requêtes par morceaux

- Comme les clients doivent traiter les réponses par morceaux, les serveurs doivent également accepter les requêtes par morceaux.

4 Obligation de supporter les connexions TCP persistantes

- Les serveurs comme les clients doivent gérer les connexions persistantes et l'entête "Connection: close".

- Quand une suite de requêtes est transmise par un client les réponses doivent être générées dans le même ordre.

5 Obligation de générer les réponses "100 continue"

- La réponse "100 Continue" est générée par le serveur HTTP/1.1 au client HTTP/1.1 des qu'il a reçu une ou plusieurs lignes de requêtes pour traiter le cas des liaisons lentes.

6 Obligation de dater les réponses

- L'entête Date donne la date courante a laquelle une réponse a été transmise.

  Date: Tue, 31 Oct 2000 16:31:11 GMT

- Elle est obligatoire dans toutes les réponses sauf 100 continue.

- Elle permet un bon fonctionnement des caches (problème de vieillissement des informations dans les caches).

7 Obligation de gérer les entêtes de modification

- HTTP/1.1 prévoit que les clients peuvent émettre des requêtes avec conditions
  If-Modified-Since:
  If-Unmodified-Since:

- Objectif: éviter de répéter la transmission d'informations déjà présentes dans les caches (gain de bande passante).

- Trois formes (pour compatibilité)

If-Modified-Since: Tue, 31 Oct 2000 16:31:11 GMT
If-Modified-Since: Tuesday, 31 Oct 2000 16:31:11 GMT
If-Modified-Since: Tue, Oct 31 16:31:11 2000 GMT

- Si aucune modification code réponse 304
  HTTP/1.1 304 Not Modified
  Date: Fri, 2 Nov 2000 17:27:34 GMT

8 Obligation de support de certaines méthodes

- Pour un serveur HTTP/1.1 GET et HEAD doivent toujours être supportées.

- Il est certainement souhaitable de supporter la méthode POST.

- Quatre méthodes annexes sont définies: PUT, DELETE, OPTIONS, TRACE.

- Si elles ne sont pas implantées:
  "501 Not Implemented"

- Support des méthodes en mode HTTP/1.0 également obligatoire pour des raisons de compatibilité. Dans ce cas bien sur pas d'entête host ni de réponse 100 continue.

Les méthodes HTTP/1.1

En plus des méthodes HTTP/1.0 GET, HEAD et POST, qui ont le même sens HTTP/1.1 définit:

OPTIONS

Une demande d'informations sur les options de communication disponibles pour atteindre la ressource mentionnée.

PUT

Demande de transmission d'une entité placée dans la requête sur l'URI mentionnée.

DELETE

Destruction de la ressource définie par l'URI.

TRACE

Demande de retour immédiat du message requête par le serveur distant ('loop-back').

Approfondissement
Utilisation de mandataires (‘proxys’)

- Un **mandataire** (un **proxy**) est un programme qui agit en intermédiaire entre un client et un serveur.

  . Les requêtes à destination du serveur sont interceptées par le mandataire (proxy) qui peut leur faire subir un traitement avant de les retransmettre au serveur.
  . De même les réponses sont interceptées.

- Dans le WEB un proxy peut servir principalement à gérer des fonctions:
  . De **cache** de pages
  . De **sécurité** (mur pare-feu)

- Pour qu'un client utilise un proxy il faut filtrer les messages et les délivrer au proxy pour qu’il puisse effectuer son travail. On peut faire agir plusieurs proxys en cascade.

Approfondissement
Les caches WEB

Objectif : Améliorer les performances

- Diminuer les temps de chargement des documents (temps d'attente des usagers).

- Réduire la charge des serveurs (très sollicités)

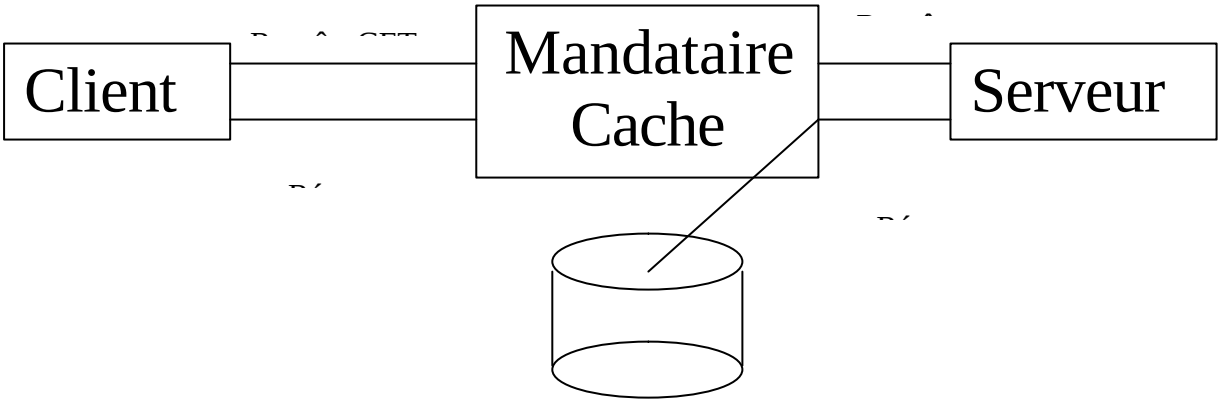
- Réduire la bande passante utilisée sur les réseaux.

Moyen : Déployer des caches

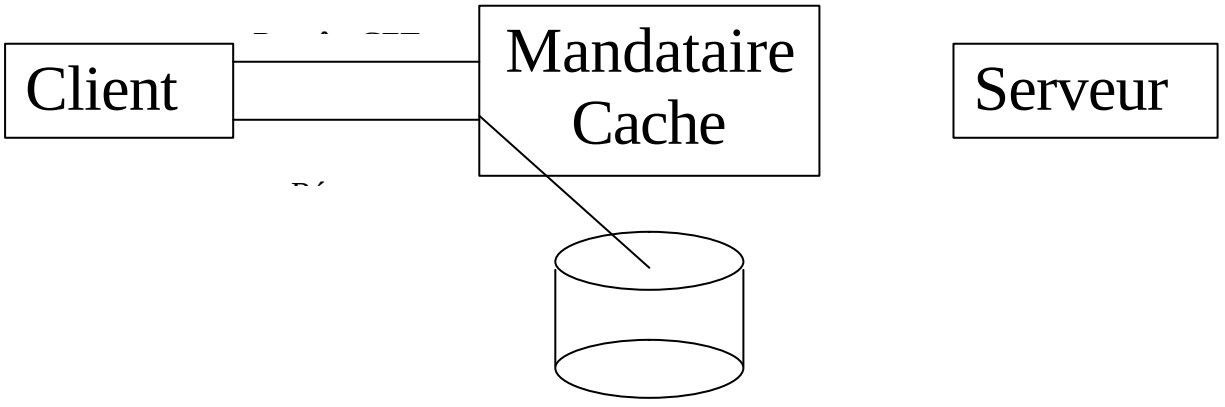
Notion de cache : une hiérarchie de mémoire telle que le temps apparent d'accès à la hiérarchie est inférieure au temps d'accès à la mémoire.

Fonctionnement d'un cache

Première requête à une page WEB



Requête ultérieure à la même page WEB



Problèmes à résoudre

- **Le remplacement des pages** dans la mémoire cache : quels documents sont à **supprimer** du cache quand il n'y a plus de place (algorithmes LRU, LFU , ….)

- **La cohérence des informations** : traiter le cas du changement d'un document sur le serveur par rapport à la valeur contenue dans la cache (invalidation).

- **Répartir la charge de travail** dans les caches (utiliser plusieurs caches qui coopèrent et qui possèdent des pages différentes).

Conclusion : Protocole HTTP

- HTTP le protocole d'application **le plus utilisé.**

- Evolutions maintenant **lentes** en raison de la diffusion

- Existence d'un version de HTTP **non adoptée** : HTTP/NG développée en parallèle de HTTP/1.1

- Existence **d'une norme** d'utilisation de HTTP pour transporter des **charges nouvelles** ('cadre pour l'ajoût de nouvelles entêtes')

Biliographie

HTTP Made Really Easy : A Practical Guide to Writing Clients and Servers

RFC 1945 Hypertext Transfer Protocol -- HTTP/1.0

RFC 2068 Hypertext Transfer Protocol -- HTTP/1.1

RFC 2069 An Extension to HTTP: Digest Access Authentication

RFC 2084 Considerations for Web Transaction Security

RFC 2109 HTTP State Management Mechanism

WEB : Conclusion générale

WEB : Un développement extraordinaire

- Le Web est **l'environnement client serveur majeur** des années 1990.

- Un foisonnement **de propositions, de standards, d'outils** … en évolution au jour le jour.

- Des secteurs multiples impliqués donc des points de vue assez divers selon les approches des différents métiers.
  Réseaux et client serveur.
  Gestion électronique de documents.
  Interfaces homme machine.
  Informatique de gestion.

Le mode client serveur incontournable des années 1990 et suivantes.
('Killing application')

Le World Wide WEB: des avantages majeurs

- Des logiciels très très largement **diffusé**.

- Assez **interopérables** entre les différents fournisseurs dans les mécanismes de base (difficultés entre fournisseurs de navigateurs pour les fonctions avancées).

- Un format de présentation des données (HTML+MIME) comprenant une **IHM conviviale** d'affichage des ressources.

- Protocole HTTP **très simple** d'utilisation et très répandu candidat au **support universel des échanges**.

- Les messages HTTP sont **toujours reconnus et transmis** (murs anti feu, …

- Des outils très **nombreux** et plutôt **bon marché**.

Le World Wide WEB: des limitations

- Dans le web il faut actuellement **plusieurs outils hétérogènes** avec divers formats de données et divers protocoles (**BD SQL** odbc/jdbc, **annuaires/sécurité** DNS/LDAP, **affichage** HTML/MIME, protocoles HTTP mode message socket).

- Les ressources non standards WEB sont accessibles uniquement via des programmes ou des codes mobiles **spécifiques** (cgi, applets, servlets scripts).
=> En fait une construction par étapes, verrues successives, sans unité…

- La représentation des données comme les protocoles sont orientés documents à afficher et **n'offre pas un cadre de typage** cohérent et complet pour des échanges entre programmes

Questions posées

Comment le WEB s'**adapte t'il** pour résoudre les problèmes précédents? (**intégrer les progrès** de l'informatique et continuer de supporter un large éventail d'applications)

Le principal problème: la nécessité d'une même interface objets répartis pour tous les types de programmes et de données.

Deux types d'évolutions possibles.

- Utilisation du WEB et en parallèle d'une approche **objets répartis** (types de candidats CORBA, RMI).

- Evolution du WEB à partir de ses outils actuels pour améliorer ses possibilités de support des objets répartis. Solution en développement: les **WEB services**