Partie 1
Internet : graphe de données

L'ensemble des données disponibles sur l'Internet peut être vu comme un graphe (*Web*) où :

- les noeuds = des données (texte, image, son, ...)
- les arcs = des *liens*

Plan de la présentation :

1. Adressage des données :
   URI, URL, URN
2. Construction des documents statiques :
   SGML, HTML, CSS, DTD, XML
3. Construction des documents dynamiques :
   architecture "3-tiers", CGI, PHP
4. Structure du graphe de Web :
   algorithmes, graphes "petit monde"
5. Recherche de l'information sur le Web :
   robots de recherche : extraire, transformer et retrouver

Adressage des données

Niveaux d'adressage (rappels réseaux) :

- adresse physique : adresse Ethernet
- adresse réseau : adresse IP
- adresse application : socket = IP + numéro de port

+ adressage des ressources sur le WEB =

URI (*Uniform Resource Identifiers*)

RFC 2396

Exemples :

```
ftp://ftp.is.co.za/rfc/rfc1808.txt
http://www.ufr-info-p7.jussieu.fr/FAQ/index.html
mailto:mduerst@ifi.unizh.ch
telnet://melvyl.ucop.edu/
```

URI : caractéristiques

Uniforme :

- le même adressage pour différents types de ressources
- ajout facile de nouveaux types de ressource

Ressource :

- = quelque chose qui a une identité
- exemples : documents, services, personnes, livres
- peut évoluer dans le temps

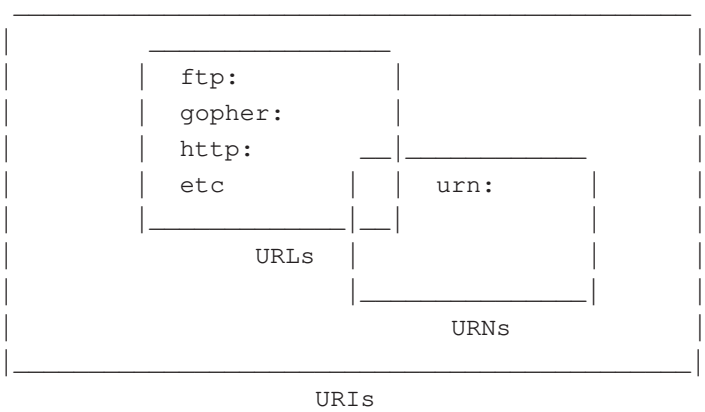
Identificateur :

- référence à quelque chose qui a une identité

Transcription :

- suite de caractères reproductible sur différents supports
- facile à mémoriser

URI $\supseteq$ URL $\cup$ URN



URL (*Uniform Resource Locator*) : terme informel (obsolète) pour les applications usuelles.

URN (*Uniform Resource Name*) : identifie de façon permanente des ressources.

Evolution : XML Pointer

URI absolus

Forme générale (voir grammaire BNF) :

```
scheme ":" scheme-specific-part
```

avec `scheme` le protocole d'accès à la ressource.

Forme usuelle :

```
scheme ":" "//" authority path "?" query
```

Attention : caractères réservés pour chaque composante !

Exemples :

```
mailto:sighirea@liafa.jussieu.fr

file:/home/verif/sighirea/cours/Internet/c_intro.ps

http://www.w3.org/Addressing/schemes.html#dcm1998

http://www.google.fr/search?q=citations&ie=UTF-8&oe=UTF-8&hl=fr&meta=

telnet://sighirea@18.29.1.34:80
```


URI relatifs

Constats :

- les données sont organisées en arbre
- une donnée peut être accessible sous différents schéma
- besoin de déplacer les données sans beaucoup de changements

Forme générale :

```
relativeURI = ( net_path | abs_path | rel_path )
              [ "?" query ]
```

avec `rel_path` contenant les chaînes de caractères :

- "." = "ce niveau de l'hiérarchie"
- ".." = "le niveau supérieur celui courant"

Exemples :

```
../g?y#s
g;x?y#s
../..
```

Déterminer la base d'un URI relatif

Sans une *base*, un URI relatif n'a pas de sens.

Algorithme pour déterminer la base d'un URI `R` :

```
if ( *R contient une base B explicite )
     return B;

else if ( *R est contenu dans une entité *B)
    return B;

else if ( R a été retrouvé à partir de B)
    return B;

else
    return default; /* dépendant d'application */
```

Exemple cas 1 : élément `BASE` en HTML.

```
<!doctype html public "-//IETF//DTD HTML//EN">
<HTML><HEAD>
<TITLE>An example HTML document</TITLE>
<BASE href="http://www.ics.uci.edu/Test/a/b/c">
</HEAD><BODY>
... <A href="../x">a hypertext anchor</A> ...
</BODY></HTML>
```


Résoudre un URI relatif

= obtenir un URI absolu à partir d'une base `B` et d'un URI relatif `R`.

Algorithme `resolve(URI B, URI R)` :

```
Données :
B = schemeB ":" "//" authority path "?" queryB
R = ( net_path | abs_path | rel_path ) [ "?" queryR ]

1. if (net_path, abs_path and rel_path sont vides)
     return B avec le fragmentR et le queryR;

2. if (R a un scheme non vide)
     /* interpreté comme référence absolue */
     return R;
   else
     schemeR = scheme;

3. if (R a un authority non vide)
     /* changement de la partie réseau */
     goto 6;
   else
     authorityR = authority;

4. if (R commence par un "/")
     /* référence absolue */
     goto 6;
```

Algorithme `resolve(URI B, URI R)` (cont.)

```
5. /* ici on est sûr d'avoir un URI relatif */
   /* on combine le rel_path de R avec path de B */
   buffer = copie de path(B) sans la dernière partie;
   buffer = append(buffer, rel_path de R);

   effacer de buffer les segments complets "./"

   effacer de buffer le segment final "." s'il existe

   effacer de buffer itérativement
           les segments "<segm>/../" avec
           <segm> différent de ".."

   effacer de buffer itérativement
           les segments finaux "<segm>/.." avec
           <segm> différent de ".."

   if (buffer débute avec un ou plusieurs "..")
      return ERREUR;
   else
      pathR = buffer;
```


Algorithme `resolve(URI B, URI R)` (cont.)

```
6. /* les composantes de l'URI sont combinées */
   result = "";

   if (schemeR est défini)
     result = append (result,  schemeR ":");

   if (authorityR est défini)
     result = append (result, "//" authorityR);

   result = append (result, pathR);

   if (queryR est défini)
     result = append (result, "?" queryR);

   if (fragmentR est défini)
     result = append (result, "#" queryR);

   retur result;
```

Exercice : Déterminer pour la base
`http://a/b/c/d;p?q`
les URI absolus des URI relatifs suivants :

```
g:h           =
./g           =
g/            =
/g            =
//g           =
?y            =
g?y           =
g?y#s         =
g;x?y#s       =
../../g       =

../../../g    =
../../../../g =

/./g          =
/../g         =
g.            =
g..           =

g;x=1/../y    =

g?y/../x      =
g#s/./x       =
```


URI et sécurité

Pas de problème en soi mais ...

Attention :

- problème de persistance : changement de location
- problème d'authentification : changement de propriétaire
- problème de confidentialité : y introduire un password
- problème de requête incorrecte : peut planter le serveur
- problème d'accès à un port réservé