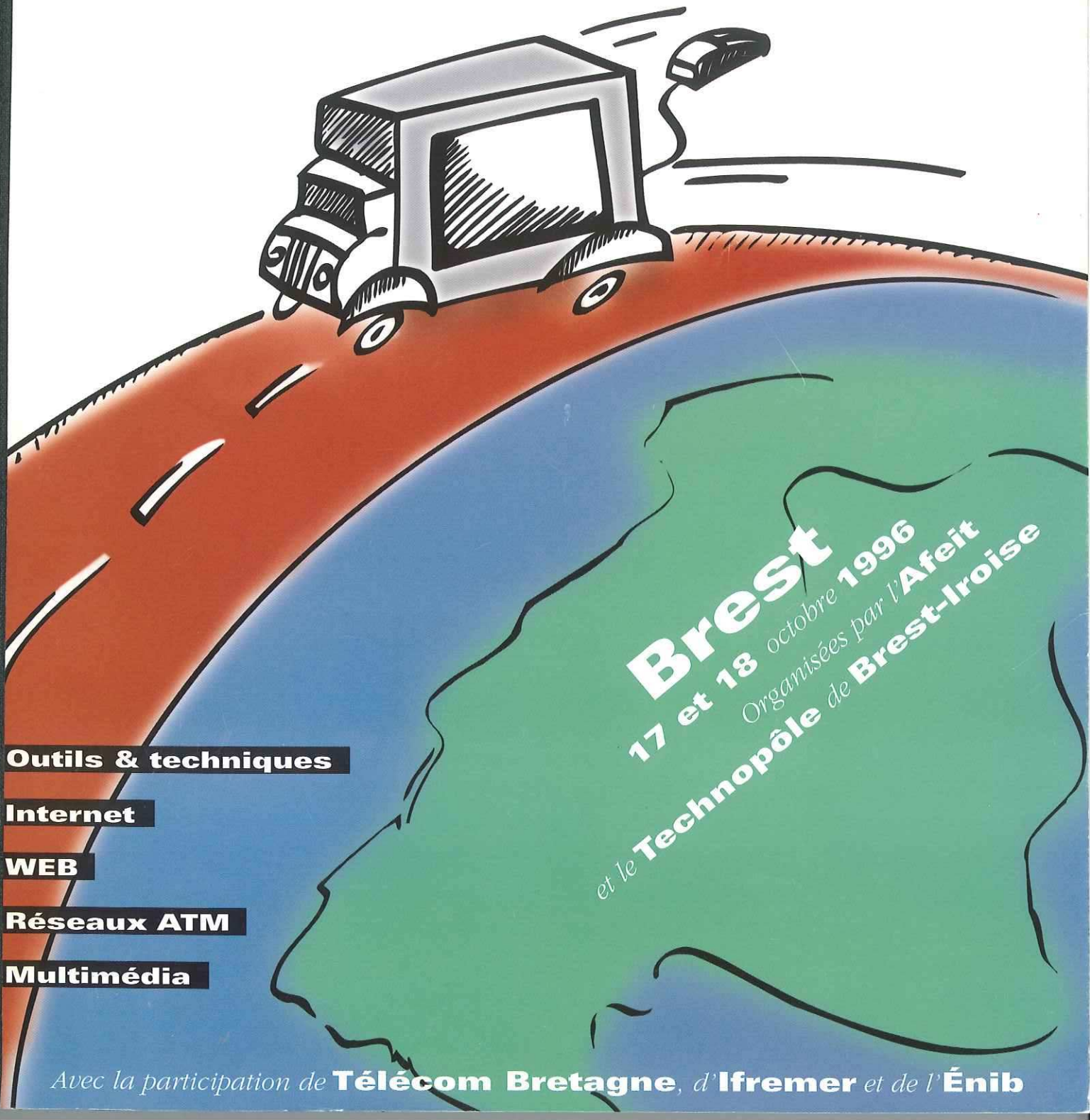
5èmes

JOURNÉES THÉMATIQUES

L'ingénierie des autoroutes de l'information



Brest
17 et 18 *octobre* **1996**
*Organisées par l'***Afeit**
et le **Technopôle** *de* **Brest-Iroise**

Outils & techniques

Internet

WEB

Réseaux ATM

Multimédia

Avec la participation de **Télécom Bretagne**, *d'***Ifremer** *et de l'***Énib**

Conseils pratiques pour la réalisation d'un serveur Web

Vincent Rodin, Ecole Nationale d'Ingénieurs de Brest
Laboratoire d'Informatique Industrielle
Technopôle Brest-Iroise, C.P. 15 , 29608 Brest Cedex
e-mail: rodin@doelan-gw.univ-brest.fr

Résumé

Le World-Wide Web (W3) est un système d'information hypermédia qui constitue, pour les utilisateurs d'Internet, un outil simple et efficace permettant d'accéder à une grande variété de documents (textes, images,...). Ce système utilise le modèle client/serveur où le serveur peut envoyer, en réponse à la requête d'un client, le contenu d'un fichier ou le résultat de l'exécution d'un programme. Cette propriété, qui consiste à faire exécuter un programme au serveur, est particulièrement importante. Il nous a donc semblé intéressant d'en présenter ici les concepts fondamentaux.

1. Introduction

Le World-Wide Web (W3) est un système d'information hypermédia créé en 1989 à l'initiative de Tim Berners-Lee afin de faciliter la diffusion de l'information au sein du CERN (Centre d'Etude et de Recherche Nucléaire). En quelques années, ce système est devenu l'Outil indispensable, simple et efficace permettant aux utilisateurs d'Internet d'accéder à une grande variété de documents (textes, images, sons, univers 3D,...). Il utilise le modèle client/serveur où le serveur peut envoyer, en réponse à la requête d'un client, le contenu d'un fichier ou le résultat de l'exécution d'un programme.
Cette propriété, qui consiste à faire exécuter un programme par le serveur, est particulièrement importante car elle va permettre d'utiliser pratiquement n'importe quel logiciel en fournissant des résultats de n'importe quel type.
Il nous a donc semblé intéressant de présenter ici les concepts fondamentaux nécessaires à la réalisation de programmes s'exécutant sur un serveur Web: architecture client/serveur, URL (Uniform Ressource Locator), en-tête MIME (Multipurpose Internet Mail Extensions), variables d'environnement et transmission de paramètres d'exécution entre un client et un serveur Web (utilisation de CGI (Common Gateway Interface)).
Signalons que le présent document doit être perçu, non pas comme un document complet et "universel" traitant de l'exécution de programmes par un serveur Web, mais plutôt comme le fruit d'une expérience personnelle de mise en place d'un serveur utilisant cette technique (`http://www-isis.enst.fr/`, serveur du Groupement De Recherche Information Signal et ImageS).

2. Architecture client/serveur W3

Le système W3 utilise le modèle client/serveur comprenant:

- des clients W3 (les navigateurs): programmes qui permettent aux utilisateurs de soumettre des requêtes à un serveur W3 et de visualiser le résultat. Signalons que si la réponse à une requête contient une image, du son ou tout autre type de données non "visualisables" par le client, celui-ci peut faire appel à un programme externe de "visualisation" (par exemple **xv** sous Unix pour afficher une image).

- un serveur W3: programme qui s'exécute sur un ordinateur dans le seul but de répondre à des requêtes de clients W3. Rappelons qu'en réponse à la requête d'un client, le serveur peut envoyer le contenu d'un fichier ou le résultat de l'exécution d'un programme.

Le dialogue entre un client W3 et un serveur W3 (voir figure 1) utilise le protocole HTTP (HyperText Transfer Protocol) qui est très simple: établissement de la connexion, envoi de la ressource (contenu d'un fichier ou résultat de l'exécution d'un programme) demandée par le client, fermeture de la connexion.

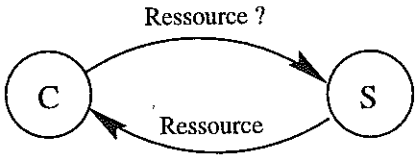


Figure 1: Dialogue client/serveur W3

Ainsi, il n'y a pas de session permanente entre un client et un serveur Web. Ce système est donc sans mémoire c'est-à-dire qu'un serveur ne conserve, à priori, pas d'informations relatives à un client donné.
Lorsqu'un client demande une ressource sur le réseau, il le fait en spécifiant un URL qui doit indiquer de manière non ambiguë où et comment atteindre cette ressource.

3. URL (Uniform Ressource Locator)

Un URL permet d'identifier une ressource dans le monde du World-Wide Web. Les URLs peuvent être perçus comme des extensions au réseau du concept de nom de fichier.
Ces URLs définissent le protocole d'échange entre les clients et les serveurs, l'adresse Internet de l'ordinateur, le catalogue contenant la ressource, le nom de la ressource et éventuellement des paramètres d'exécution. La syntaxe complète d'un URL est de la forme:

`méthode://nom-du-serveur[:port]/catalogues/fichier[#ancre][?paramètres]`

Où:

- `méthode`: http et ftp (File Transfer Protocol) sont les méthodes les plus utilisées,
- `#ancre`: une ancre permet de référencer un endroit précis dans un document,
- `?paramètres`: un ensemble de paramètres d'exécution peut être transmis au serveur Web par le navigateur.

Exemples: `ftp://ftp.urec.fr/pub/reseaux/services_infos/WWW/docs/WWW.ps.Z`
`http://www-isis.enst.fr/bin/BDImages/AccesBase.cgi?choix=requete`

Sur le Web, un URL peut référencer une ressource de n'importe quel type (texte, image, son,...). Il doit donc exister un mécanisme permettant au client de savoir quel type de données il va recevoir quand il a demandé un URL particulier. Ce mécanisme est mis en oeuvre à l'aide des en-têtes MIME.

4. En-têtes MIME (Multipurpose Internet Mail Extensions)

Lorsqu'un serveur envoie des données à un client, il les fait précéder par un en-tête MIME décrivant la nature de ces données. Ainsi, quand un client reçoit un en-tête, il sait si les données qui vont suivre doivent être affichées dans sa fenêtre principale ou si il doit appeler un "visualiseur" externe.

Lorsqu'un en-tête MIME est envoyé à un client, il doit être suivi d'une ligne vide et de la forme:

```
Content-type:  type/sous-type
```

Signalons qu'il existe un grand nombre de couples type/sous-type prédéfinis (image/tif, text/html, text/plain,...) et qu'il est possible d'en définir de nouveaux pour des applications particulières.
Un en-tête MIME peut également être porteur d'informations sur une éventuelle compression des données envoyées au client. Pour cela, le serveur envoie au client la ligne suivante:

```
Content-encoding:  x-compress
ou Content-encoding:  x-gzip
```

Ceci permet au client de savoir quel décompresseur (uncompress ou gunzip) il doit utiliser dès que toutes les données sont arrivées.
Notons que, lors d'un simple transfert de fichier, l'en-tête MIME est **automatiquement** envoyé au client par le serveur W3. Ceci n'est plus vrai lors de l'exécution d'un programme par le serveur: le serveur doit alors **impérativement** envoyer l'en-tête MIME au client avant de lui transmettre quoi que ce soit.

5. Exécution de programmes par un serveur Web

Cette section est consacrée à la création de programmes pouvant être exécutés par un serveur web. Nous supposerons ici que le langage HTML (HyperText Markup Language) et le langage de commandes Shell (Unix) sont connus. Sur le web, l'exécution d'un programme peut être déclenchée:

- soit directement par URL:

  ```
  http://machine/catalogues/test.cgi?parametres
  ```

  Un URL peut être spécifié soit au niveau d'un **lien** dans un fichier HTML, soit au niveau du navigateur (Open URL).
- soit par formulaire HTML:

  ```
  <form action="http://machine/catalogues/test.cgi" ...>...</form>
  ```

  Rappelons qu'un formulaire HTML permet aux utilisateurs d'entrer des paramètres dans des champs d'entrée simple, des menus déroulants,... et de lancer la requête.

Dans tous les cas, le programme `test.cgi` appelé peut être écrit dans n'importe quel langage de programmation (C, Perl, Shell,...) et doit être capable de traiter les paramètres d'exécution qu'il reçoit. De plus, les sorties "écran" du programme `test.cgi` exécuté sur le serveur sont envoyées vers le client (en réponse à la requête).

Exemples:

1- Le programme `test1.cgi` envoie le fichier image enib.tif vers le client:

```
#!/bin/sh
echo Content-type:  image/tif
echo
cat enib.tif
```

L'appel de ce programme doit être réalisé par l'URL:

```
http://machine/catalogues/test1.cgi
```

2- Le programme `test2.cgi` envoie un fichier image passé en paramètre après avoir compressé les données:

```
#!/bin/sh
echo Content-type:  image/tif
echo Content-encoding:  x-compress
echo
compress -c $1
```

L'appel de ce programme doit être réalisé par l'URL:

`http://machine/catalogues/test2.cgi?fichier.tif`

Maintenant, nous allons étudier comment un programme peut traiter les paramètres d'exécution qu'il reçoit.

5.1 Rappels sur les formulaires HTML

Un formulaire permet à un utilisateur de saisir des paramètres et de provoquer l'exécution d'un programme capable de les traiter. La syntaxe d'un formulaire est la suivante:

`<form action="URL" [method=GET|POST]> ... </form>`

Où:

- La méthode (`GET` ou `POST`) permet de fixer le mode de transmission des paramètres entre le client et le serveur:

  a) `GET`: les paramètres sont passés après l'URL (identique à ?)

  b) `POST`: les paramètres sont envoyés au serveur dans le corps du message qui lui est destiné (c'est-à-dire la requête).
- Entre `<form ...>` et `</form>` sont définis des champs d'entrée, des boutons de sélection, des zones d'entrée de texte, des boutons d'envoi de requête, ... et aussi des zones cachées.

Exemple:

```
<title>Un exemple de formulaire</title>
<form action="http://www.enib.fr/prog.cgi" method=GET>
nom:  <input name=nom size=25> <br>
prenom:  <input name=prenom size=25> <br>
<select name=choix size=4>
<option> choix un
<option> choix deux
<option> choix trois
</select>
<input name=variable Type=Hidden value="valeur cachee" >
<input value=Annuler Type=reset>
<input value=Envoyer Type=submit>
</form>
```

Signalons que certains caractères sont codés avant d'être envoyés vers le serveur:

- les espaces sont remplacés par des +
- certains caractères sont remplacés par: %xy (xy étant leurs codes ASCII)

Ainsi, si l'utilisateur entre **Rodin** (nom), **Vincent** (prenom) et **choix un** (choix), le serveur reçoit les paramètres via la chaîne de caractères:

`nom=Rodin&prenom=Vincent&choix=choix+un&variable=valeur+cachee`

Le serveur doit donc décoder cette chaîne afin d'utiliser un à un les champs du formulaire.

5.2 CGI (Common Gateway Interface) et les variables d'environnement

Le système W3 utilise CGI qui est un standard pour l'écriture de passerelles entre les serveurs W3 et les programmes exécutables par ces serveurs. Dans un programme, CGI permet d'utiliser les variables d'environnement suivantes:

- `$#`, `$*` (en C: `argc`, `argv`): nombre d'arguments et arguments eux-mêmes,
- `$REQUEST_METHOD`: méthode d'accès HTTP (`GET`,`POST`),
- `$SCRIPT_NAME`: nom du programme exécuté,
- `$QUERY_STRING`: chaîne contenant les paramètres codés (non valable si `$REQUEST_METHOD=POST`),
- `$REMOTE_HOST`: nom de la machine cliente,
- `$REMOTE_ADDR`: adresse IP de la machine cliente,
- `$REMOTE_USER`: nom de l'utilisateur client (valable uniquement si le programme est dans un catalogue protégé par mot de passe),
- `$CONTENT_LENGTH`: nombre de caractères à lire sur le flot d'entrée pour obtenir la chaîne contenant les paramètres codés (valable si `$REQUEST_METHOD=POST`),
- ...

CGI permet, grâce aux variables d'environnement, de récupérer les paramètres transmis à un programme.

5.3 Récupération des paramètres d'exécution

Un programme peut récupérer les paramètres d'exécution en étudiant les valeurs de certaines variables d'environnement (`$#`, `$*`, `$REQUEST_METHOD`, `$CONTENT_LENGTH` et `$QUERY_STRING`). En effet, ces valeurs dépendent de la méthode employée pour provoquer l'exécution du programme (URL ou formulaire) et de la manière dont cette méthode est utilisée. Nous allons ainsi considérer quatre cas.

a) Exécution du programme provoquée par un formulaire avec `method=POST`

Afin de mieux interpréter les valeurs des variables d'environnement, prenons pour exemple le formulaire suivant:

```
<form action="http://www.enib.fr/prog.cgi" method=POST>
nom:  <input name=nom size=25> <br>
prenom:  <input name=prenom size=25> <br>
<input value=Envoyer Type=submit>
</form>
```

Supposons également que l'utilisateur entre rodin pour le nom et vincent pour le prenom. Dans ce cas, nous avons: `$# = 0` et `$* = ""`

```
$REQUEST_METHOD = POST
$QUERY_STRING = ""
$CONTENT_LENGTH = 24
```

Ici, il faut lire 24 caractères sur le flot d'entrée (par lecture clavier) car c'est la longueur de la chaîne `nom=rodin&prenom=vincent`

b) Exécution du programme provoquée par un formulaire avec `method=GET`

Supposons ici que nous ayons le même formulaire que précédemment et que nous y ayons remplacé `POST` par `GET`. Supposons également que l'utilisateur entre rodin pour le nom et vincent pour le prenom.

Dans ce cas, nous avons: $# = 0 et $* = ""
$REQUEST_METHOD = GET
$QUERY_STRING = "nom=rodin&prenom=vincent"
$CONTENT_LENGTH = ""

c) Exécution du programme provoquée par un URL du type:
`http://machine/catalogues/test.cgi[?mot1+mot2+...+moti]`

Lors de l'appel d'un URL de cette forme, nous avons:
$# = i et $* = "mot1 mot2 ... moti"
$REQUEST_METHOD = GET
$QUERY_STRING = "mot1+mot2+...+moti"
$CONTENT_LENGTH = ""

d) Exécution du programme provoquée par un URL du type:
`http://machine/catalogues/test.cgi?var1=val1[&var2=val2&...]`

Lors de l'appel d'un URL de cette forme, nous avons un comportement identique à un appel par formulaire avec `method=GET`:
$# = 0 et $* = ""
$REQUEST_METHOD = GET
$QUERY_STRING = "var1=val1[&var2=val2&...]
$CONTENT_LENGTH = ""

5.4 Exemple de programme

Le programme suivant, `script.cgi`, réalise l'exécution **par le serveur** du programme `xv` qui permet de visualiser le fichier image passé en paramètre. De plus, le DISPLAY est redirigé sur la machine appelante. Le programme doit être appelé directement via l'URL: `http://machine/catalogues/script.cgi?nomdefichier` .

```
#!/bin/sh
if test $# -ne 1
then echo Content-type:  text/html
     echo
     echo "Erreur sur le param&egrave;tre"
else echo Content-type:  text/plain
     echo
     DISPLAY=$REMOTE_HOST:0.0 ; export DISPLAY
     XV=/usr/local/bin/xv
     $XV $1
     echo "execution terminee"
fi
```

Références

[Dagorn 1994] Dagorn F., Gross C., "World-Wide Web", 11 avril 1994.
`ftp://ftp.urec.fr/pub/reseaux/services_infos/WWW/docs/WWW.ps.Z`

[CNRS 1995] CNRS & Université (ouvrage collectif), "L'Internet professionnel", CNRS Editions, 1995. A suivre sur: `http://www.urec.fr/internet.pro/`

[Péliks 1995] Péliks G., "Le World-Wide Web", Addison-Wesley, Avril 1995.

[Gardarin 1996] Gardarin G., Gardarin O., "Le Client-Serveur", Eyrolles, 1996.