Rapport de projet de fin d'Etude

Retwine HP4155

e3 option TIC

Jérôme UZEL

ENSERB IXL

19 juin 1998

Résumé

Ce rapport présente le travail que j'ai réalisé dans le cadre d'un projet de fin d'étude, à l'IXL.

Il s'agit de permettre l'accès d'un Analyseur de Paramètres pour Semiconducteurs (HP4155A) à travers internet.

Le rapport expose l'architecture du projet et les solutions technologiques retenues.

Abstract

This report is a presentation of a end-school project conducted at the IXL.

It aims at giving an access to a Semiconductor Parameter Analyser (HP4155A) through the internet.

The report mainly explains the architecture of the project and technical solutions retained.

Sommaire

Table des matières

Table des figures

Liste des tableaux

1 Introduction

1.1 Présentation du projet Retwine

Le laboratoire de micro-électronique bordelais de l'IXL mène un projet dont le nom est Retwine (*http://www.ixl.u-bordeaux.fr/retwine.html*).

Il vise à permettre l'utilisation de matériel électronique couteux via un réseau et ainsi de mieux rentabiliser des investissements lourds. Nottament, en mettant à profit le décalage horaire, on peut espérer utiliser potentiellement 24h/24 ces appareils. Des accords de partenariat permettront en retour d'accéder à des instruments non locaux.

1.2 Architecture

Le dispositif repose sur une architecture client-serveur. Une applet java [1] communique avec un serveur HTTP :

L'applet ouvre une socket sur le serveur pour y ordonner l'éxécution d'un programme. Ce programme commande, via un bus GBIP, le dispositif électronique et fournit en retour un résultat.

Le serveur HTTP doit donc à la fois gérer les scripts cgi pour l'éxécution de commandes locales et reconnaitre les applets java qui seront transmises aux clients pour l'interfaçage avec le serveur. Il doit être connecté au Bus GBIP (au moins indirectement [2]).

De plus il est très grandement souhaitable que le serveur dispose de moyens de protection d'accès pour contrôler et limiter l'utilisation des instruments, celle-ci n'étant pas pris en considération par l'applet elle-même.

On gardera constamment à l'esprit que le travail effectué devra être le plus générique possible afin de faciliter le développement ultérieur d'autres drivers pour d'autres instruments.

1. En fait, cette applet permet également une éxécution en tant que programme autonome. Néanmoins, pour plus de commodité, nous continuerons par la suite à utiliser le terme d'Applet

2. à l'IXL, le serveur HTTP invoque l'éxécution distante du driver GPIB via un *rsh*

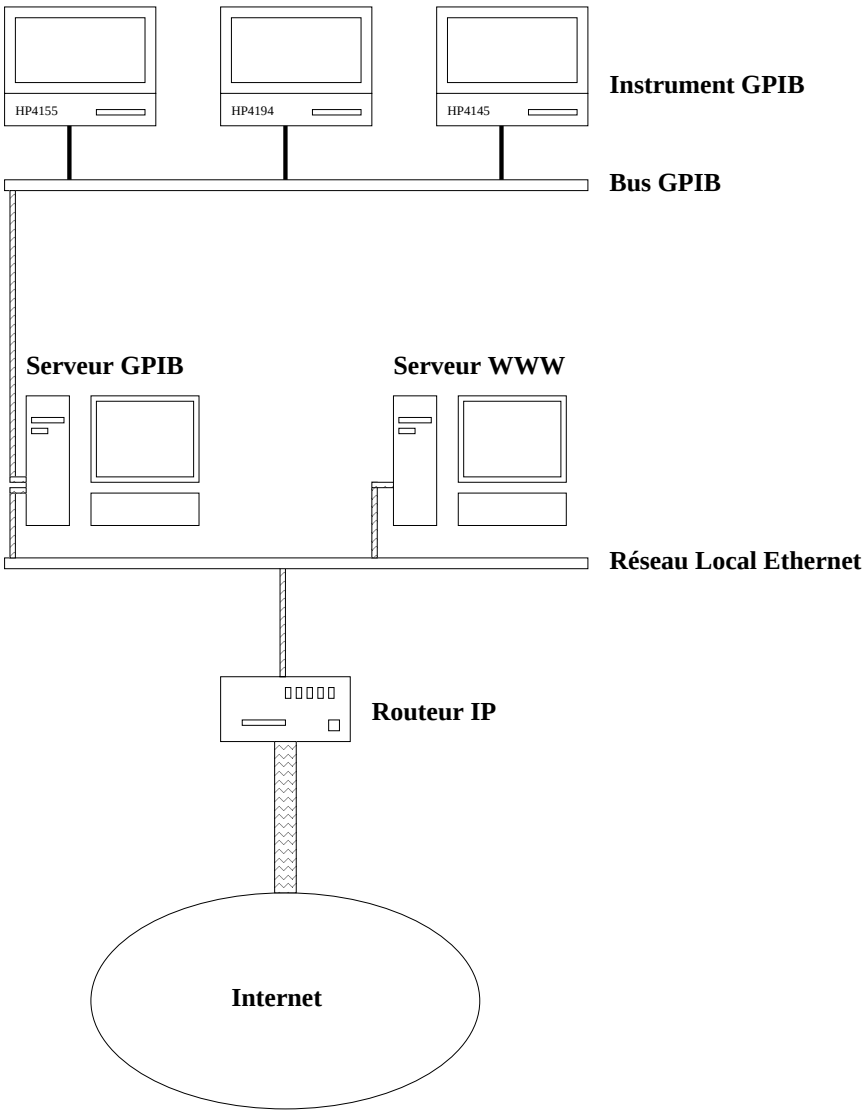

FIG. 1 - *Architecture*

2 Solutions technologiques

2.1 Serveur

La principale alternative réside dans le choix entre script cgi et servlet pour commander l'appareil GPIB.

2.1.1 Cgi-bin

Les scripts cgi sont une solution simple et relativement ancienne pour effectuer des actions sur un serveur et répondre de façon dynamique à des requêtes.

Prenons l'exemple suivant :

$(GET|POST)$ `/cgi-bin/mon.programme.cgi?param1?param2`

Le serveur configuré de facon adéquat reconnaît, par exemple au chemin `/cgi-bin` qu'il s'agit d'un cgi ; il éxécute le programme `mon.programme.cgi` correspondant en récupérant sa sortie standard pour la retourner au client[3]. Le programme ainsi invoqué peut être n'importe quel programme éxécutable : shell Script, Perl, binaire, etc... Des paramètres sont passés par les variables d'environnement[4].

Dans le cadre d'un développement de *Retwine* , ils présentent l'avantage d'être connus de beaucoup. En revanche, de nombreux problèmes de sécurité existent, en particulier dans l'exploitation des paramètres passés à un script shell. De plus, il résulte de chaque requête la création par le serveur de process, tâches relativement coûteuses.

2.1.2 Servlet

Les servlets proposent une solution plus récente, puisque basée sur Java. Schématiquement, le serveur HTTP inclue une machine virtuelle qui éxécutent les servlets lors de requêtes. De nombreux avantages existent :

- Bien évidemment, les servlets bénéficient de tout ce qui fait l'attrait du langage Java dans un cadre standard :
  - Langage de haut niveau;

3. Notons qu'il est nottament de la charge du client de générer le Header HTTP.

4. Citons principalement `QUERY_STRING` qui contient `param1?param2`, et `CONTENT_TYPE` et `CONTENT_LENGTH` qui correpondent respectivement aux type et longueur du flot de donnée fournie par l'entrée standard (lors d'un `POST`).

  - Portabilité[5];
  - etc.

- Rapidité d'éxécution : Si la lenteur est un des reproches souvent adressés à Java, la situation est ici très différente. Il est rare que les traitements à éxécuter soient très complexes (ce n'est en tout cas pas notre cas). En revanche, si on les compare aux cgi, un point crucial fait pencher la balance en faveur des servlets : Les scripts cgi procèdent à chaque requête à la création de processus, ce qui constitue une charge importante tant pour le serveur HTTP que pour la machine elle-même. Les servlets ne connaîssent pas ce problème;
- Meilleure interaction applet//servlet que applet//cgi;
- Sécurité accrue par l'utilisation de *Security Manager*;
- Possibilité de gérer les cookies au niveau du servlet et donc de gérer très finement les accès clients et les conflits potentiels;
- Possibilités étendues d'accès à des SGBD[6], etc.

2.1.3 Conclusion

Etant donné la faible durée du stage, il m'a semblé préférable d'adopter la solution qui avait déjà été employée dans un projet similaire. Celle-ci reposait sur l'emploi d'un script cgi et d'un driver C. Je les ai adapté au HP4155.

Néanmoins, à terme, il serait certainement interressant de migrer vers les servlets. Ainsi, le serveur serait plus robuste, plus efficace et plus facilement installable sur une autre plateforme. De plus, l'évolutivité serait bien meilleure.

5. Il suffit de disposer sur la machine d'un serveur HTTP qui gère les servlets (Java Web Server, Apache, Netscape, IIS, etc.).
6. JDBC, ODBC

2.2 Client

2.2.1 Java

Le language java s'est rapidement imposé comme la solution la plus adéquate à la réalisation du projet.

Tout d'abord, la portabilité du code est un atout majeur : Il aurait été inenvisageable de développer une solution pour chacune des différentes plateformes du marché.

De nombreuses classes Java sont disponibles sur Internet. Elles permettent souvent d'éviter de re-développer des solutions « classiques » (graphiques, etc.).

Java, langage objet, facilite l'écriture de code réutilisable. Ainsi, lorque l'on souhaitera ajouter un instrument GPIB au parc disponible sur Retwine, on pourra espérer n'avoir qu'à modifier « en surface » un autre appareil, la structure elle restant identique. C'est dans cette optique que le code a été écrit.

2.2.2 Non utilisation de Java WorkShop

Il avait été un temps envisagé d'utiliser Java Workshop pour générer l'interface de l'Applet.

Plusieurs raisons m'ont ammené à y renoncer :

La principale est surement que la version de JWS installée à l'IXL, très ancienne, n'autorisait que l'écriture de code 1.0[7]. Or, les dernières versions du JDK[8] disponibles sont incompatibles avec la précédente sur bien des points... et Sun annonce la sortie du JDK 1.2[9] pour le mois d'Août.

De plus, l'intérêt du code généré reste très limité et celui-ci utilise des classes spécifiques à JWS.

Enfin, il ne m'a pas été possible d'obtenir une license pour JWS 2.0.

2.2.3 JFC (Java Fundation Classes)

L'applet réalisée repose sur l'utilisation des classes Swing (ou JFC) introduites par Sun, Netscape et IBM sur la base des IFC de netscape. Disponibles

7. JDK 1.0
8. JDK 1.1.x
9. déjà disponible en Beta 3

séparément du JDK1.1.2, elles seront intégrées dans les versions du JDK postérieures à la 1.2.

Les avantages que l'ont peu escompter sont nombreux. Citons succintement une portabilité accrue, des performances améliorées tant au chargement qu'à l'éxécution, l'introduction de concepts comme le look & feel, le drag & drop, les possibilités d'impressions...

En revanche, il faut noter que les VM supportant des versions du JDK en mesure d'utiliser les JFC ne sont pas encore très répanduees...

3 Serveur

3.1 Shell Script

Côté serveur, lorqu'une requête (`POST`) est effectuée pour un script cgi, le serveur HTTP éxécute le shell script et lui passe par l'entrée standard les arguments du client.

Un mécanisme de protection est alors mis en œuvre pour réserver les ressources matérielles à un utilisateur (client). Celui-ci est identifié par l'adresse de la machine qu'il utilise. Pratiquement, un fichier est créé dont le nom comporte un préfixe déterminant l'appareil (par exemple HP4155) et un suffixe identifiant l'utilisateur. Il est alors simple de tester l'existence d'un fichier préfixé du nom de l'appareil pour déterminer si celui-ci est employé.

Un « chien de garde »[10] est utilisé pour effacer régulièrement ces verrous : un processus est lancé en tâche de fond, attends une période fixe déterminant le temps alloué au client avant libération de ressources pour finalement réinitialiser le verrou[11]. L'intérêt majeur du chien de garde est que quoiqu'il arrive par ailleurs (arrêt brutal du programme, etc.), on est assuré de son bon comportement. On ne risque donc pas de réserver inutilement un appareil, ce qui bloquerait son utilisation pour un autre client.

Un message (`USED` ou `NOT_USED`) permet au client de savoir si sa requête a été acceptée ou non.

Ensuite, dans le cas de ressources libres, le shell script invoque par rsh l'éxécution du driver GPIB sur une autre machine. Les arguments sont passés par l'entrée standard.

3.2 Driver C

Le driver utilisé est une surcouche de celui de National Instruments Corporation[12].

L'appareil GPIB à utiliser est défini par une adresse sur le bus GPIB (`#define HP4155A_ADDR 17`).

Après initialisation, il est possible d'envoyer des chaînes de caractère à

10. Watchdog

11. bien évidemment, lorqu'un client effectue une nouvelle requête alors qu'il a réservé l'appareil précédemment, son « crédit temps » est remis à jour.

12. NI-488.2M Driver for the Sun Sparcstation, ©1992

l'appareil grâce à la fonction C :

```
int ibwrt(int handle, char *buffer, u_long cnt)
```

où `handle` détermine l'appareil cible du bus, `buffer` les données à envoyer et `cnt` la longueur de buffer.

Les données à envoyer sont lues, les unes après les autres, sur l'entrée standard par le programme C. Chaque commande est située sur une ligne et le driver les envoie toutes. Eventuellement, si une commande se termine par '?' [13], le driver lit la réponse de l'appareil.

13. ce qui signifie qu'il s'agit d'une question.

4 Client

Il s'agit ici de définir le fonctionnement général de l'applet : Les détails propres à l'implémentation sont volontairement écartés de ce rapport pour en alléger la lecture. Le code source, très documenté, permettra une lecture plus en profondeur.

4.1 Structure de l'applet

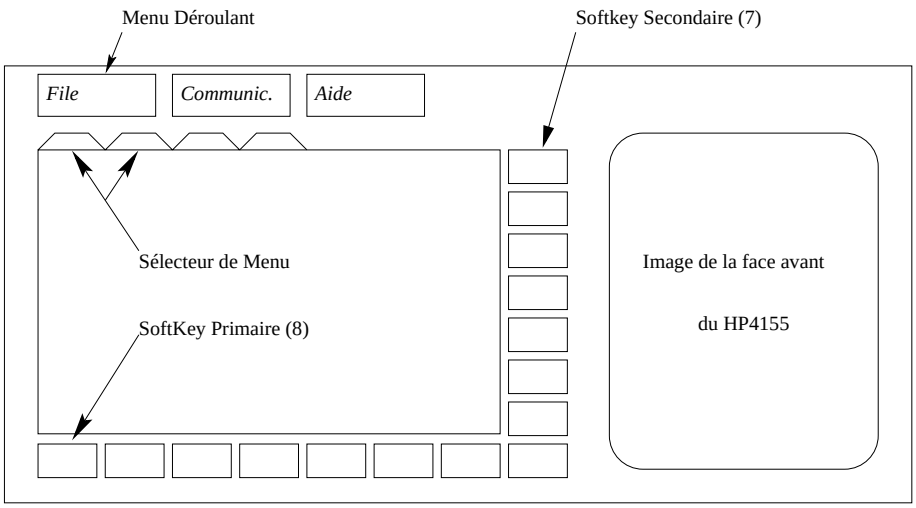

FIG. 2 - *Aspect de l'applet*

L'applet comprend un ensemble de panels, une photo de la face avant de l'appareil, deux séries de SoftKey et une barre de menus.

Les MenuPanels ont en charge la simulation de l'affichage de l'écran. Ils doivent présenter une interface proche de ce que l'appareil affiche effectivement ; Un seul est actif et donc affiché à un instant donné. Ils acceptent des ordres des softkey, de la souris (photo façade de l'appareil) et du clavier. Néanmoins, cette liste n'est pas limitative (Cf. Sous-Section 4.3 : Les évènements).

L'image représentée sur la face avant de l'appareil est active et permet un fonctionnement similaire à celui de l'instrument physique. Ainsi, la façade de l'appareil présente à l'utilisateur les différentes touches qui lui seraient accessibles si il était physiquement en train de manipuler l'instrument.

Ces touches se décomposent en 5 blocs principaux :

- PageControl :
  touches de fonctions permettant de choisir le MenuPanel Courant;
- MarkerCursor :
  permet de déplacer le curseur sur lecran;
- Measurement :
  lancement de mesure;
- IBasic;
- Entry :
  Canal d'entrée divers de données.

Les softkeys se décomposent en deux séries, primaires et secondaires. Leur rôle est en fait celui de touches de fonctions contextuelles c'est à dire que l'action qu'elles engendrent dépend du contexte (le MenuPanel courant, etc.).

Des menus déroulant permettent nottament l'accès aux fonctions de communication et d'impressions.

4.2 Les différentes classes utilisées

4.2.1 Schéma : Relations d'Héritage et d'Utilisation

FIG. 3 - *Principales Relations d'Héritage*

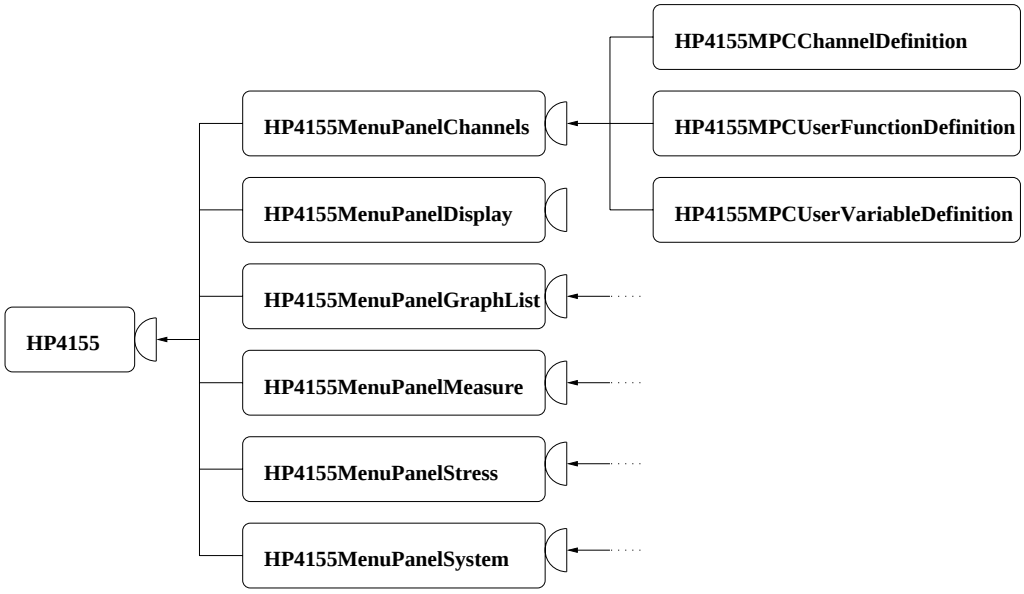

FIG. 4 - *Relations d'Utilisation entre classes*

4.2.2 HP4155

La classe HP4155 (sous-classe de JApplet) constitue le cœur de l'applet.

A l'initialisation, elle met en place les différents composants, charge l'image de fond et les MenuPanel.

Ensuite, elle gère la communication avec le serveur HTTP. Elle est également responsable des évènements du clavier, de la souris ou des SoftKeys et de la centralisation et du traitement des évènements (Cf. Sous-Section 4.3 : Les évènements).

Voici les principales méthodes qu'elle implémente:

- `init()`
  Au chargement de l'applet, init crée et positionne les différents éléments de l'IHM;
- `actionPerformed(ActionEvent)`
  Gestion des actions générées dans la barre Menu et par les softKeys;
- `ChangeSK(String[], String[])`
  Appelée par les MenuPanels pour effectuer un changement de Soft Key;
- `SendData(String)`
  Envoie des données à l'appareil sur le bus GPIB via http;

- `doPrint()`
  Gère l'impression.

Des constantes sont définies dans HP4155Infce.

4.2.3 HP4155MenuPanel

La classe abstraite HP4155MenuPanel est un JPanel qui constitue la classe mère de tous les MenuPanels que l'applet est susceptible de charger.

HP4155MenuPanel définit les méthodes à implémenter pour les classes filles. Nottament, toGPIB() traduit l'état du MenuPanel en une chaine de caractères qui envoyée au HP4155 permettra de mettre en cohérence l'appareil avec l'applet et fireSoftKeysChanged et cette méthode est appelée pas les MenuPanels lorsqu'il faut changer les SoftKey Affichées.

4.2.4 HP4155MenuPanelChannel

Cette classe est un MenuPanel qui propose des choix de réglage pour les différentes entrées sorties du HP4155. En fait, elle offre le choix entre trois sous panneaux qui sont affichés à leur tour.

- Channel Definition, implémenté dans HP4155MPCChannelDefinition, pour les définitions des entrées sorties.
- Users Function, implémenté dans HP4155MPCUserFunctionDefinition, pour définir et utiliser des fonctions.
- Users Variable, implémenté dans HP4155MPCUserVariableDefinition, pour des variables.

Ces classes utilisent intensivement les JTable qui constituent un package des JFC.

4.3 Les évènements

4.3.1 Mécanisme

Le mécanisme de gestion des évènements utilisé est celui préconisé par Sun à partir du JDK1.1.

Les méthodes du 1.0 sont certes un peu plus faciles à mettre en oeuvre mais sont moins performantes et offrent moins de souplesse. Le traitement des évènements est basé sur le principe d'héritage : une classe qui veut effectuer une action lors d'un évènement doit hériter d'un composant de l'interface graphique et redéfinir action() ou handleEvent(). Ces méthodes retourne un booléen. Si c'est true, l'évènement est réputé traité. Sinon, l'évènement remonte en quelque sorte la hiérarchie de l'interface graphique. Pour un Button qui est affiché dans un Panel, si Button.handleEvent() retourne false lors d'un click de souris, l'évènement est transmis au Panel, etc.

Ceci ammène à deux types de traitement: Soit créer des sous classes pour tous les composants utilisés de façon à ce qu'il soient à même de traiter eux même les évènements dont il sont la cible ; Soit faire traiter ces évènements globalement, par exemple par le panel qui contient tous les composants. Il s'en suit alors une pléthore de classes qui ne diffèrent souvent que par leur action en réponse aux évènements ou bien une classe hyper complexe. Ces méthodes sont in-esthétiques et particulièrement mal adaptées à la réalisation de vastes interfaces : le code devient vite brouillon, la maintenance peu aisée et le coût de traitement élevé pour la MV ralentit la vitesse d'éxécution !

```
public boolean handleEvent(Event e)
  {
    if (e.target == Button1)
      {
        System.out.println("Hit");
        return true;
      }
    return super.handleEvent(e);
  }
```

Dans les version 1.1.x, une nouveau mode de gestion est introduit. Le principe de base réside dans l'écriture de gestionnaires d'évènements (Listener) auxquels les différents composants doivent s'abonner. En celà, elle s'apparente aux mécanismes X11.

Les listener implémentent l'interface java.util.EventListener. En pratique, ils implémentent des sous interfaces comme java.awt.event.KeyListener,

java.awt.event.MouseListener, etc. Les méthodes de ces interfaces sont celles qui sont invoquées lorsqu'un évènement leur est destiné :

```
public interface interface java.awt.event.MouseListener
  extends java.lang.Object
  implements java.util.EventListener
    {
      public abstract void mouseClicked(java.awt.event.MouseEvent);
      public abstract void mousePressed(java.awt.event.MouseEvent);
      public abstract void mouseReleased(java.awt.event.MouseEvent);
      public abstract void mouseEntered(java.awt.event.MouseEvent);
      public abstract void mouseExited(java.awt.event.MouseEvent);
    }
```

Il existe par ailleurs des classes (les *adapters*) qui implémentent chacune de ces interfaces lorqu'elles possèdent plusieurs méthodes. Elles évitent d'avoir à écrire toutes les méthodes lorsqu'une seule vous intéresse.

La source des évènements est un objet quelconque qui possèdent les méthodes set<ClasseEvenement>Listener ou add<ClasseEvenement>Listener. En invoquant ces méthodes avec pour argument un listener, la source « s'abonne » au listener.

A priori, toutes les cardinalités de la relation d'abonnement sont envisageables : un listener peut écouter une ou plusieurs sources et une source peut s'abonner à un (set∗) ou plusieurs (add∗) listener.

```
public class MyClass implements ActionListener
{
  public void actionPerformed(ActionEvent e)
  { System.out.println("Action !"); }
}

aButton.addActionListener(new MyClass());
```

Notons que le Listener et la source peuvent en fait constituer en fait le même objet physique.

Les inner class ont été introduites pour permettre d'éviter de déclarer trop de classes pour les listener, si bien que l'on peut également écrire :

```
aButton.addActionListener(new ActionListener
  {
    public void actionPerformed(ActionEvent e)
      { System.out.println("Action !"); }
  }
```

Ce qui se lit : « Ajouter à aButton un listener d'action qui implémente l'interface ActionListener en imprimant Action à l'écran »

4.3.2 Principe Général

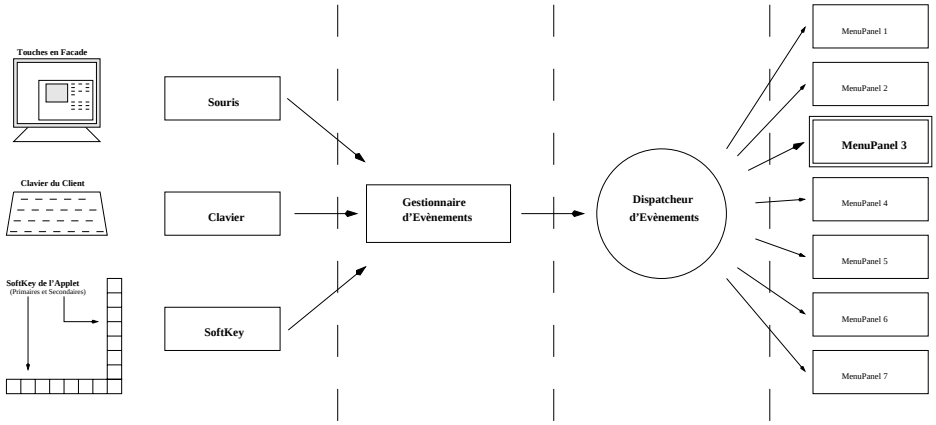

FIG. 5 - *Evénements : Principe Général*

Différents évènements sont susceptibles de survenir. Principalement les sources sont :

- Les softkeys
- Les Menus Déroulants
- L'image de la façade

Ils peuvent être générés par les MenuPanel ou par des composants plus simples tels les softKey. Ces évènements sont « remontés » à travers la hiérarchie de l'interface jusqu'à l'Applet. Ensuite, celle ci va chercher à les interpréter (`HP4155.actionPerformed()`). Si elle n'y parvient pas, elle les transmet au MenuPanel courant (`processAction`), etc.

Ainsi, la priorité est toujours laissée aux composants fondamentaux (on évite que un MenuPanel ne vienne perturber l'applet) tout en conservant un schéma de fonctionnement très générique qui permet de rajouter très facilement un nouveau périphérique d'entrée (Touches de fonctions du clavier, etc.).

4.3.3 Les SoftKey

Les SoftKey ne sont connues que de HP4155. Les MenusPanels ou toute entité qui désire changer la valeur des MenuPanels doit in-fine invoquer HP4155.ChangeSK(). Pour ce faire, les éléments de HP4155 implémentent souvent `fireSoftKeysChanged()` (en particulier, tous les MenuPanels.) Le MenuPanel courant est alors susceptible de recevoir le texte de ces SoftKey en paramètre de processAction().

4.3.4 Les Menus Déroulants

Ils sont directement gérés par l'applet qui est leur Listener. En effet, leurs actions sont par nature très couplées au cœur même de l'Applet. En l'état actuel, les évènements qu'ils génèrent ne sont pas susceptibles d'être transmis à d'autre composants de l'Applet.

5 Communication

Le client communique avec le serveur par l'intermédiaire de sockets. Il se connecte à un serveur HTTP et doit donc se conformer à ce protocole.

Nous décrirons ici le principe de fonctionnement de HTTP 1.0 (et supérieurs), la version précédente, HTTP 0.9 étant légèrement différente.

5.1 HTTP

Celui-ci restera relativement simple, du moins dans la limite de notre implémentation. Les données transmises se décomposent de deux parties: une en-tête (ou header) et le corps proprement dit, l'ensemble constituant un message[14]. L'entête est séparée du corps du message par deux sauts de lignes successifs, un saut de ligne étant définit par les caractères CR LF :

```
CR  = [CR ASCII-US, retour chariot (13)]
LF  = [LF ASCII-US, saut de ligne (10)]
```

5.2 Requête HTTP

L'en-tête contient des informations sur la transaction. Nottament, elle définit le type de transaction, la longueur et le type mime[15] du corps du message.

- GET est la méthode la plus courante. Elle permet de demander le contenu d'une URL au serveur. Notons ici qu'il possible de passer des paramètres dans l'URL, séparés par des '?', comme indiqué page 10.
- HEAD permet le même type de question mais ne demande en retour que l'en-tête. On peut ainsi tester la validité de liens, d'actualiser des données, etc.
- POST est utilisée pour transmettre des données au serveur dans le corps du message. Ces données peuvent être volumineuses et de n'importe quel type.

14. En fait, depuis la version 1.0, il est possible d'ouvrir une seule connexion entre client et serveur et de transmettre plusieurs messages. Ce ne sera pas le cas ici.

15. Multipurpose Internet Mail Extensions: identificateur de type de fichier, principalement utilisé pour le mail (smtp), le web (http) et les news (nntp). Il est défini dans le rfc 2048 (*http://src.doc.ic.ac.uk/computing/internet/rfc/rfc2048.txt*)

Content-Length: xx fera alors partie de l'entête du message et permettra de spécifier la taille des données (xx est un nombre) transmises [16].

Deux cas peuvent alors se produire:

Le client peut se connecter directement à la machine à laquelle il veut faire une requête. C'est le cas le plus simple.

Il se peut aussi que le client ne soit relié au reste de l'internet que par l'intermédiaire d'une machine tierce, le proxy, pour des raisons de sécurité.

5.2.1 Sans Proxy

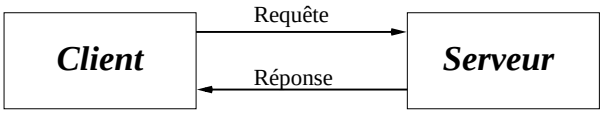

FIG. 6 - *Requête sans Proxy*

Par exemple : Le client ouvre une socket sur le serveur (www.server.dom) sur le port 8001 (par défault 80) [17] et envoie sa requête.

```
GET /dira/dirb/file.html HTTP/1.0
```

ou

```
POST /cgi-bin/hp4155.cgi HTTP/1.0
Content-length: 28

OPTREC=-R&OPTLONG=-l&OPTA=-a
```

Ce POST peut avoir été généré par une page html contenant:

```
<FORM ACTION="/cgi-bin/lsopt.sh" METHOD=POST>
  <INPUT TYPE=RADIO NAME=OPTREC  VALUE="-R">  Recursif
  <INPUT TYPE=RADIO NAME=OPTLONG VALUE="-l">  Long
  <INPUT TYPE=RADIO NAME=OPTA    VALUE="-a">  All      <P>
  <INPUT TYPE=SUBMIT VALUE="Envoyer">
  <INPUT TYPE=RESET VALUE="Annuler">
</FORM>
```

16. Ce champ est particulièrement important dans le cas de connections longues, sur plusieurs requêtes, pour permettre au serveur de séparer ces requêtes.

17. sur unix `socket www.server.dom 8001`

5.2.2 Avec Proxy

Ici, le client se connecte sur le Proxy (puisque par définition, il ne se connecter directement sur la machine voulue !). Il lui demande alors d'effectuer la requête à sa place et de lui renvoyer le résultat.

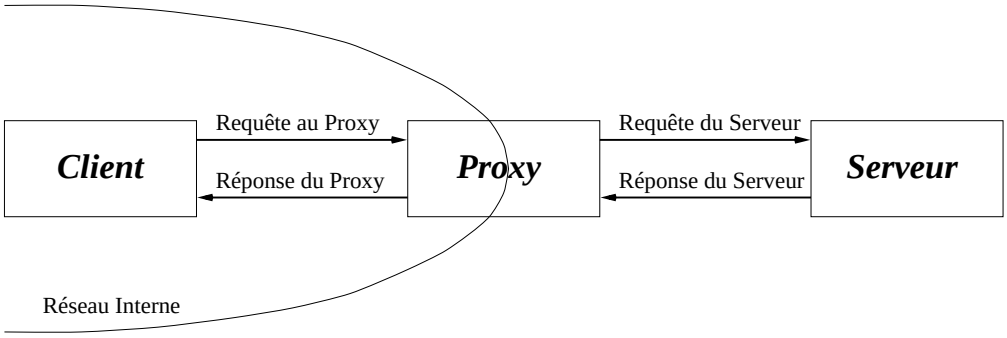

FIG. 7 - *Requête avec Proxy*

La requête du client serait dans ce cas:

```
GET http://www.server.dom:8001/dira/dirb/file.html HTTP/1.0
```

ou

```
POST http://www.server.dom:8001/cgi-bin/hp4155.cgi HTTP/1.0
Content-length: 28

OPTREC=-R&OPTLONG=-l&OPTA=-a
```

Le proxy n'aura aucun mal à interpréter ces demandes ! Ensuite, il se comportera, dans le principe [18], comme un client qui se connecte directement au serveur requis... à moins qu'il ne doivent à son tour passer par un autre proxy ! Il ne lui restera qu'à retourner au client originel le fruit de la requête.

18. des filtres et/ou un cache peuvent ici être mis en place.

5.3 Réponse HTTP

L'entête de la réponse que reçoit le client contient les informations suivantes: Numéro de protocole HTTP utilisé, Code d'Etat (2xx pas d'erreur, 4xx erreur survenue) et un message littéral explicitant le code d'Etat. Ensuite, on trouve une indication sur le type mime du corps qui suit pour permettre au client de l'interpréter et sa longueur.

```
HTTP/1.0 200 OK
Content-type: text/html
Content-length: 83

<HTML>
<HEAD><TITLE>Preface</TITLE></HEAD>
<BODY>Voila votre reponse</BODY>
</HTML>
```

ou

```
HTTP/1.0 200 OK
Content-type: text/html
Content-length:127

<HTML>
<HEAD><TITLE>Sortie Correcte</TITLE></HEAD>
<BODY><HR>28 octets: OPTREC=-R&OPTLONG=-l&OPTA=-a</P></BODY>
</HTML>
```

6 Conclusion

Il reste à terminer l'implémentation des différents `MenuPanel`.

De plus, plusieurs développements sont possibles et souhaitables à l'avenir :

- Mettre à jour le programme pour qu'il utilise les gestionnaires de sécurité de netscape, au niveau des sockets;
- Remplacer les scripts cgi par des servlets.
- Effectuer une remontée fine des erreurs GPIB.

A Java : Principes généraux

Java est un langage Objet récent développé par Sun Microsystems©. Il devait répondre à plusieurs objectifs :

A.1 Portabilité

Les langages compilés traditionnels aboutissent à un code qui est constitué de la suite des instructions natives que le processeur doit éxécuter. Ces instructions sont dédiées à l'architecture cible[19]. Pour pouvoir éxécuter le programme sur une autre architecture, il faut passer par une phase de portage qui consiste à transcrire les parties du source qui sont spécifiques à l'architecture.

L'approche du langage Java est différente. Elle consiste à passer par une étape intermédiaire : le code source est « compilé » en Byte Code c'est à dire en un langage assembleur virtuel. Assembleur car les concepts manipulés sont homologues à ceux des langages assembleurs traditionels (Registres, instructions élémentaires,...) et virtuel car il n'existe pas de processeur capable d'éxécuter ces instructions[20] ! Aussi, le code ne peut être éxécuté tel quel sur une une machine. Il faut passer par une seconde étape. C'est le rôle de la machine virtuelle. Elle constitue en quelque sorte un émulateur pour le pseudo-code. Ce programme est chargé de compiler le pseudo-code en code natif (Cf. Annexe B).

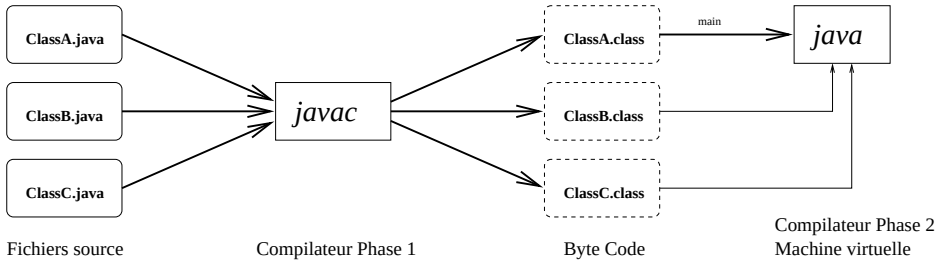

FIG. 8 - *Java : Compilation et éxécution*

De plus, des classes génériques permettent de manipuler les ressources du système avec une interface commune. On peut nottament citer ici les accès

19. celle pour laquelle le source à été compilé.

20. En fait, des constructeurs ont développé dans un second temps des puces destinées à éxécuter ce code intermédiaire.

disques, réseau et surtout l'interface graphique AWT[21].

Sun utilise le sigle WORA[22] pour décrire cet aspect de java.

A.2 Réseaux

Un autre attrait de java est qu'il permet de faire voyager aisément le byte code à travers internet : les différentes parties sont chargées à la demande et de façon transparente pour l'utilisateur, dans un contexte HTTP.

Dans la mesure où c'est le byte code qui est transmis, l'architecture de la machine client n'importe pas et le code source n'a pas à être diffusé[23]. Coté client, le navigateur (Cf. Annexe C) dispose d'une machine virtuelle qui éxécute le programme.

Notons que des considérations de sécurité sont à l'évidence ici à prendre en compte : Les ressources auxquelles peuvent accéder les applets doivent être limitées par la machine virtuelle du browser car l'utilisateur doit pouvoir maitriser les actions du programme que le browser a chargé (en pratique automatiquement) et qu'il ne connaît pas. Il faut par exemple éviter qu'un programmeur malveillant efface ou modifie des fichiers du système, lise et transmette des fichiers de mot de passe,...

Les limites les plus courantes imposées aux applets sont de leur empêcher les accès disques et les connexions à des machines tierces (différentes de celle d'où les classes java ont été chargées.)

A.3 Java, des concepts évolués

Java manie les tâches : il est possible d'effectuer des traitements parallèles. Les machines virtuelles utilisent souvent cette possibilité pour gérer les entrées-sorties, les interactions avec l'utilisateur, le Garbage Collector,...

Java connaît l'Introspection. A partir du byte code, il est possible de connaître la structure exacte d'une classe :

Quelles sont les différents atributs et méthodes ?
Quel est leur visibilité ?
Que retourne une méthode ?

21. Abstract Window Toolkit : « Boîte à outils Abstraite pour le fenêtrage »
22. Write Once, Run Anywhere : écrit une seule fois, éxécuté partout
23. Comme pour un langage assembleur, il est possible de « désassembler » , le byte code pour étudier et/ou modifier le fonctionnement de l'applet. Néanmoins il existe des outils qui sont en mesure de singulièrement compliquer la tâche.

Quelle est la super classe ?
etc.

Ces possibilités peuvent être utilisées dans le programme lui-même. Une application peut être par exemple la création d'un browser de classes, qui à partir du byte code présenterait les différentes caractéristiques d'une classe.

Java utilise le chargement dynamique. En mettant à profit l'Introspection, java charge dynamiquement les classes dont il a besoin. Ainsi, au moment du lancement du programme, les classes qui ne seront employées que plus tard dans le déroulement du programme peuvent ne pas être encore disponibles. Elle peuvent même être créées dynamiquement et chargées par un class loader.

Java employe un Garbage Collector (GC). Le programmeur n'a plus à se préoccuper de désallouer la mémoire utilisée. Tous les objets sont référencés en interne par la machine virtuelle de façon indirecte ce qui permet au GC de les deplacer en mémoire. Un compteur de nombre de reférences peut permettre de savoir quand un objet n'est plus utilisé (C'est à dire quand plus rien ne le référence !).

A.4 Java, langage ouvert

Enfin, Java est un langage très ouvert. Outre le fait qu'il soit particulièrement adapté à l'internet, il est possible de l'interfacer de façon standard avec des bases de données supportant odbc ; gràce au JNI[24], on peut lui faire éxécuter du code natif, écrit par exemple en C ou C++, Il peut également prendre Part à une architecture CORBA pour distribuer des objets entre diverses applications et plateformes mais les RMI[25] offrent une solution plus simple lorsque la distribution se limite à un contexte purement Java.

24. Java Native Interface
25. Remote Method Invocation

B Machines Virtuelles

Cette annexe traite des différentes générations de machines virtuelles qui se sont succédées jusqu'à présent. Il est donc principalement question ici de la façon dont les MV exécutent le code. Les aspects de synchronization des tâches et les méchanismes d'allocation et de garbage Collection ne sont pas directement concernés. De même, nous écartons de la présentation l'éxécution de méthodes natives[26].

B.1 Interpréteur de Byte Code

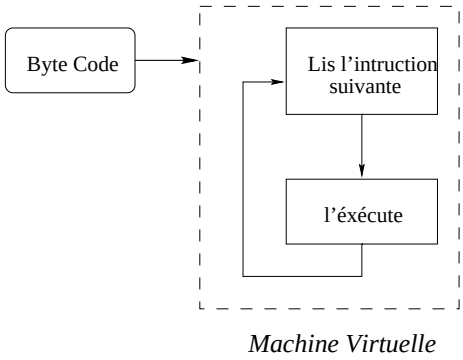

FIG. 9 - *Machine Virtuelle Java : Interpréteur*

Les premières machines virtuelles était en fait de simple interpréteurs de codes. Comme le schéma ci-dessus le montre, la MV parcours le byte code et pour chaque instruction, elle le traduit en une instruction native, c'est à dire une instruction du microprocesseur au cœur de la machine, et l'éxécute.

L'inconvénient majeur qu'il en résulte est une grande déchet dans le cas de boucles par exemple : dans l'exemple suivant, la MV doit traduire 1000 fois le byte code dans la boucle !

```
int a = 0;
for (int i = 0 ; i < 1000; i++)
   a += i;
```

26. qui permet à Java d'interagir avec les entrées sorties (AWT pour l'écran, Fichiers, clavier, souris, etc.)

B.2 Just In Time (JIT)

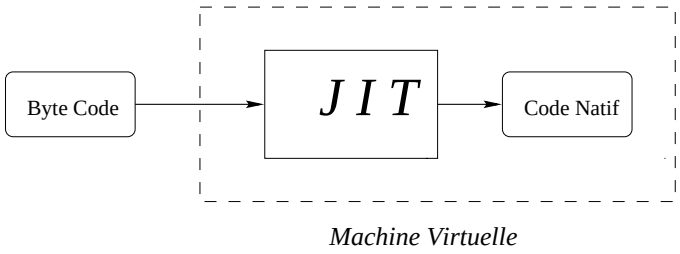

FIG. 10 - *Machine Virtuelle Java: Just In Time*

Les JIT MV opèrent différement: Elle commencent dans une première passe à traduire l'intégralité du byte code en code natif. Ensuite, bien évidemment, c'est ce code natif qu'elles éxécutent.

On voit ici que le gain est particulièrement grand dans le cas évoqué ci dessus. De plus, la Machine Virtuelle considérant les instructions du byte code dans leur globalité, elle est à même d'optimiser la traduction. Elle peut nottament exploiter finement les registres du processeur.

En revanche, la machine virtuelle ne sait pas lors de la traduction et de l'optimisation si le code sera souvent éxécuté ou non. Si elle cherche trop à optimiser le code, elle ne parviendra pas à rentabiliser cet investissement pour les parties qui ne sont que peu éxécutées.

B.3 HotSpot

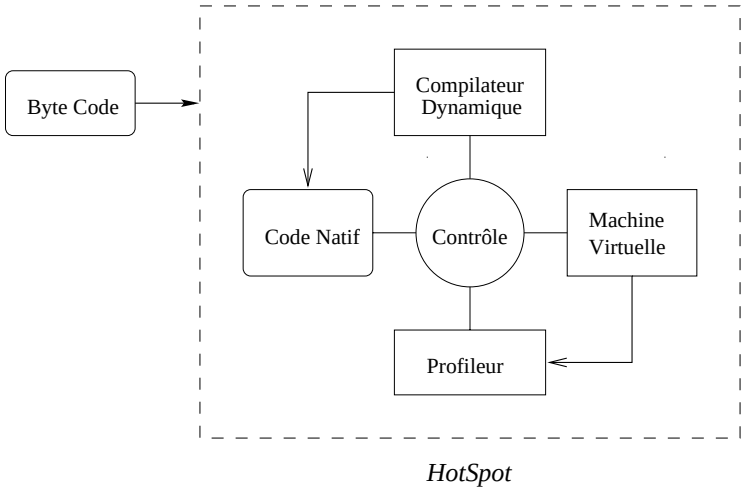

FIG. 11 - *Machine Virtuelle Java : HotSpot*

HotSpot est le nom d'une nouvelle technologie que Sun Microsystems devrait commercialiser à partir du mois d'Août. Elle constitue en quelque sorte une synthèse des deux générations précédentes.

Le point central est le Contrôleur qui décide pour chaque « bout de code » si il doit être interprété ou traduit et optimisé selon les méthodes évoquées plus haut. Pour celà, il dispose d'informations fournies par le « Profileur » qui maintient à jour un journal des différentes éxécutions. Le code compilé est stocké en mémoire pour pouvoir être éxécuté telquel plus tard.

Ainsi, pour éxécuter une partie de byte code trois cas peuvent se produire:

- Cette partie n'a jamais été éxécutée : elle est interprétée.
- Elle a déjà été éxéxutée, compilée et éxécutée. C'est bien sûr la version compilée qui est utilisée.
- Elle a déjà été éxécutée mais pas compilée. HotSpot demande au profiler d'effectuer un choix. Selon le cas, on se retrouve soit dans le fonctionnement interpréteur, soit dans celui du compilateur JIT. Dans ce dernier cas, le code compilé est stocké en mémoire pour pouvoir être réutilisé.

La complexité principale du système réside dans l'algorithme qui permet de décider si on doit chercher ou non à compiler une portion du byte code.

Il doit effectué un compromis entre l'estimation du temps qu'il va passer à optimiser, le gain escompté et le nombre d'appels prévu pour la partie traduite ! Néanmoins, il est capable de se rendre compte qu'une optimisation à échoué et alors d'utiliser systématiquement le byte code pour les futurs passages[27].

HotSpot est également capable de décider de traduire « in-line » certaines fonctions courtes et fréquemment appelées.

Bien sûr, les tentatives de HotSpot peuvent échouer. Par exemple, si HotSpot compile une méthode alors qu'il s'avèrera que c'est sa dernière invocation.

La règle des 80/20[28] lui assure, selon Sun, des performances proches de celle du C++[29] compilé. Cette affirmation étonnante, au moins à priori, est expliquée par les gains que procurrent les informations obtenues au cours de l'éxécution, informations dont ne dispose pas les compilateurs dits « statiques » ou traditionnels, en particulier pour le choix de compilation in-line de certaines méthodes.

27. Ce cas n'est pas décrit dans l'algorithme ci-dessus.

28. 80 % du temps d'éxécution d'un programme est passé dans 20 % du code.

29. On fait référence à du C++ utilisé en tant que langage objet et non comme un avatard du C.

B.4 Générateur de code natif

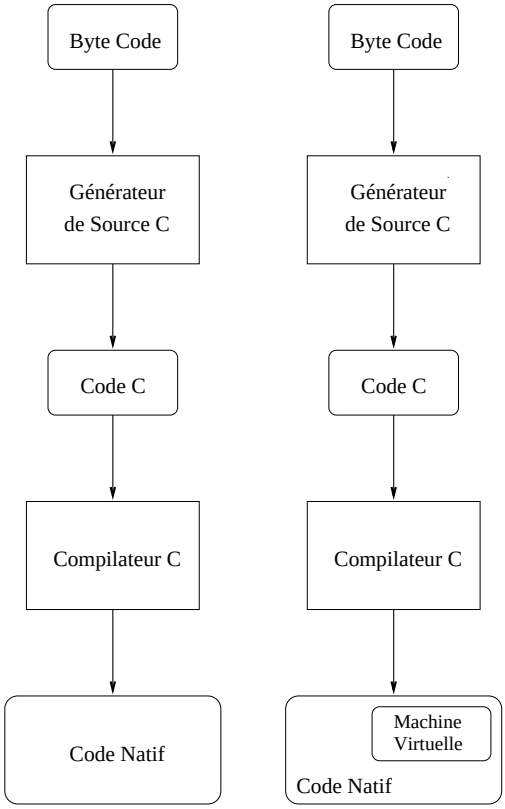

FIG. 12 - *Compilateur natif*

Devant la lenteur des permieres machines virtuelles, des développeurs ont entrepris de produire des générateurs de code natif.

Deux possibilités existent alors: traduire directement et complétement en C le programme Java ou conserver une machine virtuelle qui permet de continuer à utiliser l'introspection.

C Navigateur

Les applets java doivent être éxécutées par une machine virtuelle qui interprète[30] le code semi-compilé pour éxécuter du code natif. Dans le cas d'applet, il revient au navigateur de fournir cette machine virtuelle.

Trois concurrents principaux se font face.

C.1 Sun

Sun, en tant qu'instigateur et principal acteur du développement de ce nouveau langage qu'est Java, propose comme l'on pourrait s'y attendre les solutions les plus à jour avec les spécifications. Sun propose nottament un kit de développement grand public, le JDK, accessible gratuitement au téléchargement. Ce kit comprend nottament un compilateur, une machine virtuelle, un debugger et un browser. Il est disponible sur les principales plateformes du marché (PC/Win95, PC/WinNT, PC/Solaris, Sparc/Solaris).

C.2 Microsoft

Microsoft quand à lui semble hésiter sur l'attitude à adopter face à la menace que peut constituer pour lui le développement à grande échelle d'un langage multiplateforme. Après une phase d'adhésion au projet, Microsoft développe maintenant des solutions propriétaires dans son navigateur Internet Explorer dont la principale conséquence est de limiter l'éxécution des applets conçues autour de ce système aux seules PC équipés de Windows et Internet Explorer. De plus, il n'est même pas assuré que java figure dans la configuration par défault du prochain Windows 98.

C.3 Netscape

Netscape, enfin, propose lui aussi une VM dans ses navigateurs depuis la version 2. Cependant, le support n'est souvent que partiel. Ainsi, Navigator 3.x reste limité au JDK 1.02, Navigator 4.0x ($x \leq 4$) n'implémente qu'une partie limitée du JDK1.1.2)[31]. Navigator 4.05 existe en fait en deux versions.

30. les machines virtuelles les plus récentes(entre autre JDK1.1.x; $x \geq 5$) utilisent des *compilateurs temps réel (Just In Time)*, d'autres même une nouvelle technique nommée *HotSpot*

31. un patch existe pour upgrader la MV au niveau du JDK1.1.2

La première supporte le JDK1.1.2. Netscape vient dernièrement de présenter une nouvelle version 4.05 beta (preview release 4.05 AWT 1.1.5) qui présente l'avantage de pouvoir éxécuter les class Swing.

A titre d'illustration de la complexité de l'offre de navigateurs, voici les différentes versions de Netscape Navigator et les versions du JDK qu'elles supportent :

TAB. 1 - *Netscape Navigator et Java*

Version de Netscape	Version supportée du JDK
2.0	1.021
3.0x	1.0.2
4.03	1.1.2
4.05	1.1.2
4.05 preview release awt	1.1.5

Voila qui semble bien compliqué pour le simple utilisateur !

C.4 Conclusion

Pour l'heure, la solution conseillée pour l'éxécution du projet est l'utilisation du plugin réalisé par sun et qui présente les avantages d'être à la fois disponible pour un large éventail de machines et de permettre à l'utilisateur de continuer à employer son navigateur préféré !

Dans le future, il n'est pas à douter que Netscape et Microsoft fourniront un navigateur qui permettra d'éxécuter correctement l'applet.

D Servlets

Les servlets permettent de remplir les memes fonctionnalitées que les scripts cgi mais elles leur sont supérieures par bien des points (Cf. 2.1.2).

Leur utilisation est relativement simple. Outre quelques options de configuration permettant nottament au serveur de lier une URL à l'éxécution d'une servlet, il suffit d'écrire une class qui implémente l'interface `javax.servlet.Servlet`.

```
javap javax.servlet.Servlet
public abstract interface Servlet extends java.lang.Object {
    public abstract void destroy();
    public abstract javax.servlet.ServletConfig getServletConfig();
    public abstract java.lang.String getServletInfo();
    public abstract void init(javax.servlet.ServletConfig);
    public abstract void service(javax.servlet.ServletRequest,
javax.servlet.ServletResponse);
}
```

La méthode `service` prend en argument la requête et doit informer la réponse. Elle est invoquée par le serveur lors d'une requête sur la servlet.

`init` et `destroy` permettent quand à elles d'effectuer certaines opérations au chargement et au déchargement de la classe par la MV (par exemple, connexion à une base de donnée ...).

ServletRequest permet d'obtenir les informations relatives à la requête reçue par le serveur (Type de la requête, longueur du corps, date, Protocole utilisé, adresse du Client, du serveur, etc.).

ServletResponse offre un ensemble de méthode pour faciliter la génération du header http.

Il est à remarqué que dans le cas de plusieurs requêtes arrivant dans un même temps, les capacités multitâches de Java s'appliquent pleinement.

En pratique, la plupart des servlets sont invoquées à travers le protocole http et un package spécifique javax.servlet.http propose des méthodes à même de simplifier le travail du développeur en le libérant de la gestion de http.

```
javap javax.servlet.http.HttpServlet
public abstract class HttpServlet
extends javax.servlet.GenericServlet
implements java.io.Serializable
{
    public javax.servlet.http.HttpServlet();
    protected void doDelete(javax.servlet.http.HttpServletRequest,
```

```
javax.servlet.http.HttpServletResponse);
    protected void doGet(javax.servlet.http.HttpServletRequest,
javax.servlet.http.HttpServletResponse);
    protected void doPost(javax.servlet.http.HttpServletRequest,
javax.servlet.http.HttpServletResponse);
    protected void doPut(javax.servlet.http.HttpServletRequest,
javax.servlet.http.HttpServletResponse);
    protected void doOptions(javax.servlet.http.HttpServletRequest,
javax.servlet.http.HttpServletResponse);
    protected void doTrace(javax.servlet.http.HttpServletRequest,
javax.servlet.http.HttpServletResponse);
    protected long getLastModified(javax.servlet.http.HttpServletRequest);
    public void service(javax.servlet.ServletRequest,
javax.servlet.ServletResponse);
    protected void service(javax.servlet.http.HttpServletRequest,
javax.servlet.http.HttpServletResponse);
}
```

Les méthodes do⋆ (`doGet`, `doPost`, `doPut`, `doDelete`, `doOptions` et `doTrace`) sont appelées par `service` lors de requêtes correspondantes (doGet doit implémenter à la fois les requêtes GET et HEAD).

Elles acceptent en paramètres en HttpServletRequest ou HttpServletResponse qui complètent les ServletRequest et ServletResponse de méthodes spécifique à http (coté requête : Méthode, Date, Cookies, Authentification, URI, Session, Corps du message, etc. ; coté réponse principalement les codes erreurs http).

Enfin, ici comme ailleurs avec Java, il est facile et très profitable d'écrire des composants réutilisables qui simplifieront le développement d'autres applets similaires.

E Références et Documents utilisés

TAB. 2 - *Références*

Retwine	IXL Retwine	http://www.ixl.u-bordeaux.fr http://www.ixl.u-bordeaux.fr/retwine.html http://aramis.ixl.u-bordeaux.fr:8080
JAVA	généralités spécification JDK (version courante) JDK (version beta) JFC (swing) [a] jikes [b]	http://java.sun.com http://java.sun.com/docs/books/jls/html/jTOC.doc.html http://java.sun.com/products/jdk/1.1/ http://java.sun.com/products/jdk/1.2/ http://java.sun.com/products/jfc/ http://www.alphaWorks.ibm.com/formula/jikes
Doc Ressources	Présentation Java Tutorial Interface Graphique Swing CGI Servlets Faq Java Developper Connection	http://java.sun.com/docs/white/langenv/ http://java.sun.com/docs/books/tutorial/ http://java.sun.com/docs/books/tutorial/java/index.html http://java.sun.com/docs/books/tutorial/ui/swing/ http://java.sun.com/products/jfc/swingdoc-current/ http://developer.javasoft.com/developer/onlineTraining/swing/ http://agora.leeds.ac.uk/nik/Cgi/start.html http://www.apl.jhu.edu/ hall/java/CGI-with-Java.html http://www.javasoft.com/products/jdk/1.2/docs/ ... ... /ext/servlet/servlet_tutorial.html http://jserv.javasoft.com/products/java-server/servlets/index.html http://webreview.com/97/10/10/feature/index.html http://www.afu.com/javafaq.html http://developer.javasoft.com http://www.javalobby.org/ http://www.javaworld.com/javasoft.index.html http://www.jars.com/ http://www.developer.com/directories/pages/dir.java.html http://www.sys-con.com/java/index2.html
BROWSER	NC4.05 (Prerelease) NC3.0 + Activator	ftp://ftp.netscape.com/pub/communicator/4.05/development/ ... ... /english/unix/4.05_plus_JDK1.1_pr1/ http://java.sun.com/products/plugin/
Divers	HTTP	http://www.eisti.fr/eistiweb/docs/normes/rfc1945/1945tm.htm
News		news:///fr.comp.lang.java news:///comp.lang.*

[a] Classes à charger en plus du JDK1.1.x ; Elles font partie du JDK1.2.

[b] Un compilateur java plus rapide que javac (JDK)

F Lexique

TAB. 3 - *Lexique*

Sigle	Description	Page
AWT	Abstract Window Toolkit	
CORBA	Common Object Request Broker Technologie autorisant l'utilisation d'objets distribués entre différentes applications (Java, C, C++, Smalltalk)	
CGI	Common Gateway Interface Norme permettant de générer dynamiquement des pages Web et d'effectuer des actions sur le serveur.	10
GC	Garbage Collector	32
GPIB	General Purpose Interface Bus Norme IEEE 488.2, extension de HPIB	
HPIB	Hewlett Packard Interface Bus Norme IEEE 488, proposée par HP	
HTTP	Hyper Text Transfer Protocol protocol de communication au dessus de TCP/IP. Il est à la base du World Wide Web.	
IFC	Internet Fundation Classes	
JDK	Java Developpment Kit.	
JFC	Java Fundation Classes.	
JIT	Just In Time. Type de VM.	34
JNI	Java Native Interface Interface permettant d'invoquer du code natif en Java	
MV	Machine Virtuelle. Elle est nécessaire à l'éxécution des programmes Java.	
RMI	Remote Method Invocation Technologie autorisant l'utilisation d'objets distribués entre programmes java.	
rsh	Remote SHell. permet l'éxécution distantes d'un programme entre deux Machines Unix.	
VM	Virtual Machine. Cf: MV	

G Tâches accomplies

TAB. 4 - *Tâches accomplies*

Tâche	Durée (jours)
Prise de contact avec l'existant	7
Etude AGL/GC JWS1.0	7
AWT (Abstract Window Toolkit) JFC (Java Fundation Classes) Navigateurs disponibles	5 10 8
Définition de l'architecture	7
Rédaction rapport	10

H Sources

H.1 Structure des classes Java

H.1.1 HP4155

```
Compiled from HP4155.java
public synchronized class Ixl.Retwine.HP4155.HP4155
extends com.sun.java.swing.JApplet
implements Ixl.Retwine.HP4155.HP4155Infce , java.awt.event.ActionListener
{
    public static void main(java.lang.String[]);
    public void isApplet(boolean);
    public void showStatus(java.lang.String);
    public void init();
    public void paint(java.awt.Graphics);
    public void loadImage();
    public void waitForAllImage();
    public void createMenuBar();
    public void actionPerformed(java.awt.event.ActionEvent);
    public void doPrint();
    public synchronized java.lang.String SendData(java.lang.String);
    public void ChangeSK(java.lang.String[], java.lang.String[]);
    com.sun.java.swing.JPanel access$0();
    com.sun.java.swing.JPanel access$1();
    com.sun.java.swing.JPanel access$2();
    com.sun.java.swing.JPanel access$3();
    java.awt.Image access$4();
    public Ixl.Retwine.HP4155.HP4155();
    public Ixl.Retwine.HP4155.HP4155(boolean);

synchronized class Ixl.Retwine.HP4155.HP4155.PictureMouseEventListener
extends java.awt.event.MouseAdapter
{
    public void mouseClicked(java.awt.event.MouseEvent);
    public Ixl.Retwine.HP4155.HP4155.PictureMouseEventListener
        (Ixl.Retwine.HP4155.HP4155,com.sun.java.swing.JApplet);
}
}
```

et une interface de constantes...

```
package Ixl.Retwine.HP4155;
import java.lang.String;

public interface HP4155Infce
{
  /** Le repertoire dans lequel chercher Sz_ImageURL. */
  String Sz_BasenameImageURL = "/net/e3/uzel/prog/java/Ixl/Retwine/HP4155/";
  /** L'image de fond qui sert de base a l'IHM. */
  String Sz_ImageURL         = "HP4155.gif";

  /** L'adresse du serveur Web. */
  String Sz_WebServerName    = "aramis.ixl.u-bordeaux.fr";
```

```
  /** Le port surlequel le serveur Web ecoute. */
  int    N_WebServerPort     = 8080;

  /** Le cgi-script a invoquer. */
  String Sz_CgiScriptName    = "/cgi-bin/HP4155A_perform.sh";

  /** Le message d'information à afficher. */
  String Sz_InfoString       = "HP4155 (0.1) Java Driver\n
        J\u00e9r\u00f4me UZEL (1998)\n
        http://www.ixl.u-bordeaux.fr";
}
```

H.1.2 HP4155MenuPanel

```
Compiled from HP4155MenuPanel.java
public abstract synchronized class Ixl.Retwine.HP4155.HP4155MenuPanel
extends com.sun.java.swing.JPanel
{
    public final void reInit();
    public java.lang.String toGPIB();
    public void fireSoftKeysChanged(java.lang.String[], java.lang.String[]);
    public void processAction(java.lang.String);
    Ixl.Retwine.HP4155.HP4155MenuPanel(Ixl.Retwine.HP4155.HP4155);
    protected Ixl.Retwine.HP4155.HP4155MenuPanel();
}
```

H.1.3 HP4155MenuPanelChannels

```
Compiled from HP4155MenuPanelChannels.java
public synchronized class Ixl.Retwine.HP4155.HP4155MenuPanelChannels
extends Ixl.Retwine.HP4155.HP4155MenuPanel
{
    public void paint(java.awt.Graphics);
    public java.lang.String toGPIB();
    public void processAction(java.lang.String);
    Ixl.Retwine.HP4155.HP4155MenuPanelChannels(Ixl.Retwine.HP4155.HP4155);
}
```

H.1.4 HP4155MPCChannelDefinition

```
Compiled from HP4155MPCChannelDefinition.java
public synchronized class Ixl.Retwine.HP4155.HP4155MPCChannelDefinition
extends Ixl.Retwine.HP4155.HP4155MenuPanel
{
    Ixl.Retwine.HP4155.HP4155MenuPanel Cl_MP;
    public java.lang.String toGPIB();
    public final void fireSoftKeysChanged(java.lang.String[], java.lang.String[]);
    final void setCurrentEditedObject(java.lang.Object);
    public void processAction(java.lang.String);
    Ixl.Retwine.HP4155.HP4155MPCChannelDefinition(Ixl.Retwine.HP4155.HP4155MenuPanel);
}
```

H.2 Script Shell

```
#!/bin/sh

########################
# Configuration diverse
########################
unset $PATH
PATH=$HOME/cgi-bin:/usr/ucb:/usr/5bin/:/usr/bin;
export PATH
GPIBSERVER=treville                     # Nom de la machine controleur GPIB
SERVER=$SERVER_NAME:$SERVER_PORT        # Nom du serveur WWW

########################
# Formatage document stdout
########################
echo 'Content-type: application/octet-stream'
echo ''

########################
# Test si appareil deja utilise
########################
LOCKFILE=$HOME/cgi-bin/lock/HP4155A-$REMOTE_ADDR
CURRENTLOCKFILE=`ls $HOME/cgi-bin/lock/HP4155A-*  2>/dev/null`

if [ "${CURRENTLOCKFILE:=x}" = "x"  ]; then
        nohup watchdog.sh $LOCKFILE 1>$LOCKFILE 2>/dev/null  &          # Start watchdog
else
        if [ "$LOCKFILE" != "$CURRENTLOCKFILE" ]; then
                echo "USED"
                exit 1
        else
                kill -9 `cat $LOCKFILE` 1>/dev/null 2>/dev/null
                rm -f $LOCKFILE 1>/dev/null 2>/dev/null                 # Reset watchdog
                nohup watchdog.sh $LOCKFILE 1>$LOCKFILE 2>/dev/null  &  # restart watchdog
        fi
fi

# Appareil non utilise
echo "NOT_USED"

########################
# Lancement de la commande de l'appareil IEEE
########################
# DO NOT GET RIDE OF 'dd': it is necessary to assume you get an EOF
# WITHOUT dd, the C program never begin to read stdin !
dd ibs=1 count=$CONTENT_LENGTH 2>/dev/null | rsh $GPIBSERVER ~/cgi-bin/HP4155A_drive

BASENAME=measurements/HP4155A_mesure.txt
FILE=$HOME/pages_web/$BASENAME
mv $HOME/HP4155A_mesure.tab $FILE  1>/dev/null 2>/dev/null
```

H.3 Driver C

```
/*************************************************************************/
/*            CGI Command String Receiving and Processing                */
/*                         HP4155A DRIVER                                */
/*           february 1998              Laboratoire IXL                  */
/*************************************************************************/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

/* Bibliotheque des instructions pour le bus GPIB : */
#include "/users/ixl/retwine/src/GPIB_DRV/ugpib.h"

#define HP4155A_ADDR 17

/* Definition des fonctions utilisees              */
int   init();
char *getNextCommandfromfile();
char *start();

/* Ces fonctions ne font pas partie du listing      */
char *trait1();
char *trait2();
char *trait3();
char *start_g();
char *trait_list();
int   duree_trans();



void main( argc, argv)
int argc;
char ** argv;
{
    int ud,j=0,lenght,lenght1,b_read,k=0,b_graph=0,choix=0;
    int duree;
    char donn1[7000],donn2[7000],donn3[7000],var1[7000],*donn_rec,tab[20000];
    FILE *stream;
    char *LSzCommande = NULL;

    ud=init();

    lenght1=0;
    strcpy(tab,"\nTitleText:HP4155A\n");

    donn1[0]='\0';
    donn2[0]='\0';
    donn3[0]='\0';

    while ((LSzCommande = getNextCommandfromfile(stdin)) != NULL)
        {
            b_read=0;
            lenght=strlen(LSzCommande)+lenght1;
```

```
/* no ‘no question command‘ after ":PAGE:SCON:SING" */
if(b_graph==0)
    {
        /* Send the command. */
        ibwrt(ud,LSzCommande,strlen(LSzCommande));

        /* Test for an eventual error */
        /* if ok var1 == "+0,\"No error\"" */
        strcpy(var1,":SYST:ERR?");
        ibwrt(ud,var1,strlen(var1));
        donn_rec=start(&ud);
        strcpy(var1,donn_rec);
        printf("message:%s",var1);
    }


/* Test if command is a question (if it contains a ?)
   set b_read if yes (unset if not) */
for(j=lenght1;j<lenght+1;j++)
    {
        if(*(LSzCommande+j) == '?')
            {
                b_read=1;
                k++;
                break;
            }
        else
            b_read=0;
    }

lenght1=lenght; /* last significant char (? or unless last one) */


/* if command is a question */
if(b_read==1)
    {
        printf("\nstart acquisition: %d\n",k);
        if(k==1)
            {
                donn_rec=start(&ud);
                strcpy(donn1,donn_rec);
                printf("values=%s\n",donn1);
            }
        if(k==2)
            {
                donn_rec=start(&ud);
                strcpy(donn2,donn_rec);
                printf("values=%s\n",donn2);
            }
        if(k==3)
            {
                donn_rec=start(&ud);
                strcpy(donn3,donn_rec);
                printf("values=%s\n",donn3);
            }
    }
```

```
        else
            puts("\nWrite only");

        /* a question with no ? */
        if(!strcmp(LSzCommande,":PAGE:SCON:SING"))
            {
                donn_rec=start_g(ud,duree);
                strcat(tab,donn_rec);
                b_graph=1;
            }

        ibwait;

        free(LSzCommande);
        LSzCommande = NULL;
    }


/* if one of all command was ":PAGE:SCON:SING" */
if (b_graph == 1)
    {
        /* Sauvegarde des données dans HP4155A_mesure.tab */
        stream=fopen("HP4155A_mesure.tab","w");
        fprintf(stream,"%s",tab);
        fclose(stream);
        printf("\nData saved in file \"HP4155A_mesure.tab\"\n");

    }
else
    {
        strcat(tab,"\n");
        if(k==1)
            {
                donn_rec=trait1(donn1);
                if(!strcmp(argv[1],":PAGE:DISP:LIST?"))
                  donn_rec=trait_list(donn1,ud);
                strcat(tab,donn_rec);
            }
        if(k==2)
            {
                donn_rec=trait2(donn1,donn2);
                strcat(tab,donn_rec);
            }
        if(k==3)
            {
                donn_rec=trait3(donn1,donn2,donn3);
                strcat(tab,donn_rec);
            }

        /* Sauvegarde des données dans HP4155A_txt.tab */
        stream=fopen("HP4155A_txt.tab","w");
        fprintf(stream,"%s",tab);
        fclose(stream);
        printf("%s",tab);
        printf("\nData saved in file \"HP4155A_txt.tab\"\n");
    }
```

```
    printf("\nEnd of Command Prossessing\n");
}

/*---------------------------------------------------------------------*/

int init()
{
    int ud;

    SendIFC(0);                 /* Initialise le bus IEEE */
    if(ibsta & ERR)
        {
            printf("\n\tSendIFC Error\n");
            exit(1);
        }
    ibfind("gpib0");            /* Recherche de la liaison GPIB      */

    if(ibsta & ERR) printf("\n\tGPIB Error : ibfind gpib0\n");
    ud=ibfind("dev1");          /* Recherche de l'adresse du 4194    */
    if(ibsta & ERR) printf("\n\tGPIB Error : ibfind dev1\n");
    ibpad(ud,HP4155A_ADDR);     /* Changement d'adresse              */
    return ud;                  /* Envoi l'adresse definie           */

}

/*---------------------------------------------------------------------*/

char *start(ud)
int *ud;
{
    char donn[7000];

    /* otherwise pb with length of donn (not ended by 0) */
    char *Pt;
    Pt = donn + 7000;
    while(Pt-- > donn)
        *Pt = 0;

    ibrd(*ud,donn,7000L);

    return donn;
}

/*---------------------------------------------------------------------*/
/*
  retourne:
  un char* (a desallouer) sur la prochaine commande a executer.
  null sinon.

  Taille des commandes limitee a MAXSIZEBUFFER.
*/
char *getNextCommandfromfile(AF_FileIn)
FILE * AF_FileIn;
{
#define MAXSIZEBUFFER 256

    char *LSz_Return = (char *)malloc(MAXSIZEBUFFER * sizeof(char));
```

```
    if (fgets(LSz_Return, MAXSIZEBUFFER, AF_FileIn))
        {
            LSz_Return[MAXSIZEBUFFER] = '\000';
            return LSz_Return;
        }
    free(LSz_Return);
    return NULL;
}
```