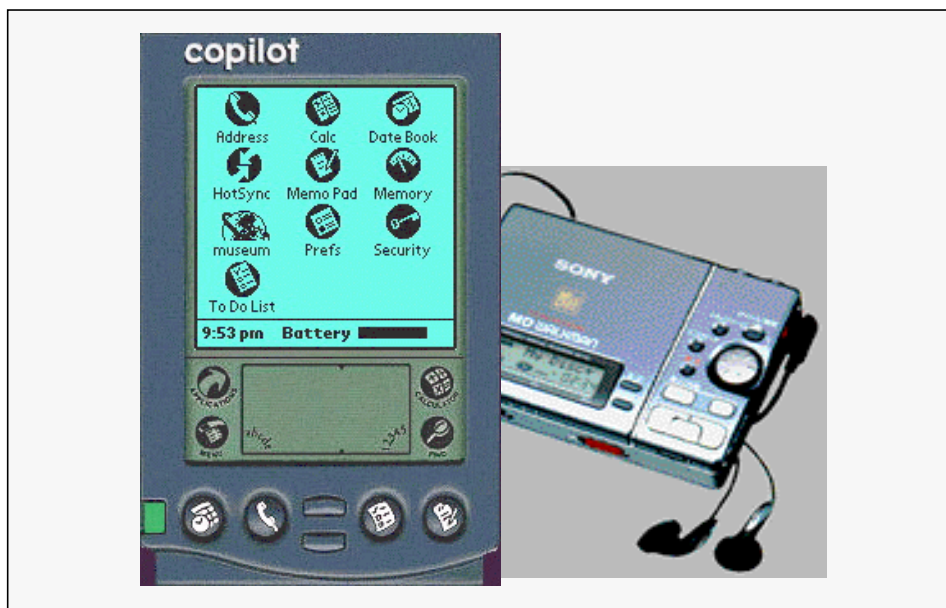

Guide interactif pour visite de musées et expositions sur pen computer



Niklaus Hirt, Informatique

Travail de 8ème semestre

Table des matières

1. Introduction	5
1.1. Buts du Projet	5
1.2. Solutions envisagées	5
2. Solution choisie	6
3. MiniDisc	7
3.1. Télécommande	7
4. Interface	8
5. PalmPilot	9
5.1. Caractéristiques	9
5.2. Possibilités	9
5.3. Programmation	9
5.3.1. Jump	11
5.3.2. Pila	11
5.3.3. PilRC	11
5.3.4. PilRCUI	11
5.3.5. Copilot	12
5.3.6. Le Pilot Studio	13
5.4. Les différents fichiers	14

Table des matières

5.4.1. le fichier .java	14
5.4.2. le fichier .rcp	15
5.4.3. le fichier .res	15
5.4.4. le fichier .asm	15
5.4.5. le fichier .prc	15
5.5. Structure générale d'un programme Pilot	16
5.6. Schéma du programme	17
5.6.1. Structure des données	18
6. Conclusion	20
7. Annexe	21
7.1. Créer de nouveaux musées	21
7.2. Schémas électroniques	22
7.3. Listings des programmes	26
7.3.1. le programme principal pour Pilot (musee.java)	26
7.3.2. le fichier musee.rcp	34
7.3.3. le fichier musee.res	34
7.3.4. le fichier de description du musée	34
7.3.5. le fichier du programme script.java	36

1.0 Introduction

Le but de ce projet de semestre était de concevoir et réaliser un dispositif de guide électronique, faisant les présentations des oeuvres choisies par le visiteur. Ceci demande une interaction avec l'utilisateur afin de connaître ses choix et pour délivrer les informations en question. Ce guide sera destiné à des expositions dans des musées qui ont une structure fixe.

Ce rapport de projet de semestre est fait de manière à pouvoir servir comme document de base pour des développements futurs et il donne les notions de base pour la programmation du Pilot de USRobotics.

1.1 Buts du Projet

On a fixé des buts dans trois domaines différents:

- commodité: la solution doit être légère et compacte pour assurer la portabilité.
L'utilisateur veut pouvoir disposer d'un guide de musée qu'il peut facilement porter sur lui et qui ne le gêne pas dans ses mouvements lors de la visite.
- facilité: le dispositif doit être facile à utiliser.
On ne peut pas supposer qu'un visiteur de musée moyen sait utiliser des dispositifs informatiques. Il doit être capable d'utiliser le guide sans connaissances préalables.
- économie: le dispositif doit coûter le moins possible
Pour qu'une solution soit intéressante pour un musée il faut réduire les coûts au strict minimum par guide, car on considère que le musée en procure plusieurs, afin de satisfaire les besoins des visiteurs.

On a plusieurs possibilités tout en essayant d'assurer le mieux les trois contraintes.

1.2 Solutions envisagées

Au cours du projet on a examiné plusieurs possibilités. En principe on a le choix entre deux approches différentes:

- L'utilisation d'un ordinateur portable capable de reproduire du son enregistré auparavant (sampling).
- L'utilisation d'un ordinateur portable pour télécommander un dispositif audio comme un lecteur CD ou MiniDisc.

Pour la première solution on avait envisagé d'utiliser un Newton de Apple. Mais la limitation de la taille des samples à 32k bytes et le prix très élevé des extensions de mémoire nous on vite fait abandonner la recherche dans cette direction.

2.0 Solution choisie

Pour la solution qui a finalement été retenue on a choisi de télécommander un lecteur MiniDisc à l'aide d'un PalmPilot de USRobotics. Pour pouvoir faire cela, il fallait concevoir une interface capable de commander le MiniDisc.

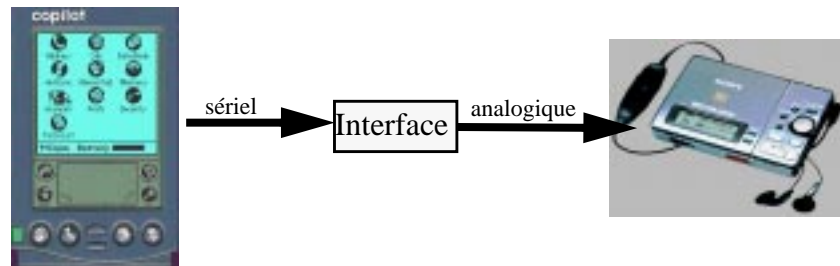


FIGURE 1.

Le principe de fonctionnement

Par la suite on présentera les trois composantes (MiniDisc, Interface et Pilot) de la solution qui a été choisie.

Lecteur MiniDisc commandé par pen computer

- + - Coût modéré
- Echange de disque pour différentes langues
- Facile à enregistrer
- Résistant aux vibrations

- - Pas compact (pas en une pièce)

3.0 MiniDisc

La technique du MiniDisc est très récente. Elle offre à l'utilisateur la qualité d'un lecteur CD conventionnel avec accès direct à des pistes, tout en donnant la possibilité d'enregistrement. En plus de cela, les lecteurs sont équipés d'une mémoire tampon pour éviter les interruptions du son (si la tête de lecture perd la synchronisation sur la piste actuelle lors de vibrations fortes). Avec les disques qui ont considérablement été réduites en taille, on peut dire qu'on a un dispositif qui réunit les avantages des Walkman (à cassettes audio) et des Discman (à CD).



Les lecteurs MiniDisc sont équipés d'une prise de télécommande dont le fonctionnement sera décrit par la suite.

3.1 Télécommande

La télécommande est constituée de quatre pins.

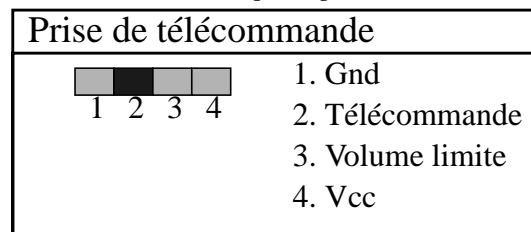


FIGURE 2.

Schéma de la prise de télécommande

La pin numéro deux est celle qui est intéressante pour nous, car elle permet d'accéder à toutes les commandes à partir de différents niveaux de tension.

Niveaux pour les commandes courantes					
Play/Fwd	Rew	Stop	Vol.Up	Vol.Down	Pause
2.2 V	3.0 V	2.9 V	1.8 V	1.9 V	2.1 V

FIGURE 3.

Tensions caractéristiques

Il suffit donc de générer des suites de tensions pour accéder une piste bien précise. Le seul problème qui reste est qu'on n'est pas capable de connaître la piste actuelle du lecteur.

4.0 Interface

Tout le problème de l'interface consiste à décoder un signal provenant d'une ligne série RS232 standard et faire correspondre de manière déterministe des niveaux analogiques ajustables aux informations reçues.

Bien que des circuits spécialisés existent pour la communication RS232, on a choisi de faire appel aux composants bien connus comme des portes ou des compteurs, pour réaliser un prototype d'interface.

La solution consiste à compter les flancs à l'entrée sérielle pendant un certain temps. Lors du premier flanc on lance un timer (ajustable) qui limite le temps d'accumulation (sommation) de flancs (T_{acc}). A la fin de ce temps, le compteur est mis à zéro et on attend le prochain signal. La sortie du compteur (3 bits de poids faible) est passée à un démultiplexeur qui réalise un codage un parmi m (8 lignes). Avec ces lignes on peut commander des coupleurs optiques qui agissent comme des passeurs exclusifs de tensions préajustés (V_{comm}).

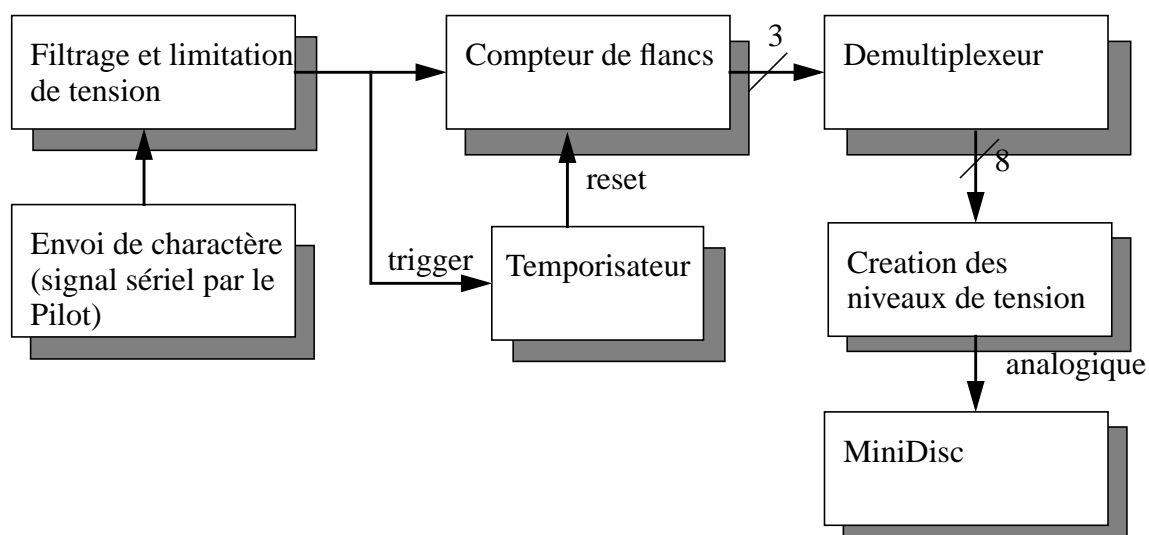


FIGURE 4.

Schéma de fonctionnement de l'interface

La solution est très facile et offre tout de même la flexibilité de décoder huit commandes différentes.

Pour plus d'information veuillez vous référer aux schémas dans l'annexe

5.0 PalmPilot

5.1 Caractéristiques

Le Pilot est un petit ordinateur portable de la famille des PenComputers, fabriqué par l'entreprise USRobotics. Il a les caractéristiques suivantes.

- Display: tactile, 160 x 160 pixels avec éclairage¹
- Programmable en C/C++ ou Java
- Mémoire: 1 MO
- Autonomie: 8-12 semaines avec 2 piles AAA
- Connection série: RS-232C 9-Pin
- Compatible Windows et Macintosh



5.2 Possibilités

L'écran tactile du Pilot se prête parfaitement au développement d'applications demandant une interaction précise qui consiste à faire un choix parmi plusieurs objets représentés graphiquement.

La possibilité de communication par ligne série standard RS232, facilite la réalisation de dispositifs externes plus complexes ainsi que leur programmation.

De plus le prix modéré permet d'obtenir un prix final avantageux pour le dispositif.

5.3 Programmation

La programmation du Pilot peut se faire en C/C++ à l'aide d'un kit de développement: le CodeWarrior de Metrowerks. Malheureusement il n'est que disponible pour Macintosh et il est coûteux.

L'autre possibilité consiste à programmer en Java sous Windows95/NT et utiliser des utilitaires disponibles en Shareware sur l'internet.

Une documentation des routines propres au Pilot peut être obtenue par Internet, sur le site de USRobotics (<http://www.usr.com/>). Il est fortement recommandé de se la procurer, car elle contient la description de la syntaxe des routines ainsi que des informations supplémentaire sur la programmation (événements, graphisme, etc.).

1. Seulement Pilot Personal et Pilot Professional. Le Pilot 5000 n'a pas d'éclairage.

Les pas de la programmation:

- On cree le programme en Java en utilisant une classe supplémentaire contenant toutes les primitives pour les appels à des fonctions internes et spécifiques au Pilot (ouvrir des fenêtres, envoyer des bytes en sériel, etc.).
- On obtient un fichier .class à l'aide d'un compilateur Java courant (javac ou jvc). Ce fichier est ensuite traduit en Assembleur (.asm) en utilisant le compilateur Jump.
- Parallèlement, on cree le fichier .rcp qui contient une description de l'interface de l'utilisateur, faite à l'aide d'un petit pseudo langage. On utilise Pilrc pour en obtenir les fichiers de ressource (.bin et .res).
- Finalement, on fait appel à Pila qui génère le code exécutable à partir du fichier .asm et des fichiers faits par Pilrc.
- Le programme final peut être testé à l'aide de l'emulateur Pilot qui s'appelle Copilot.

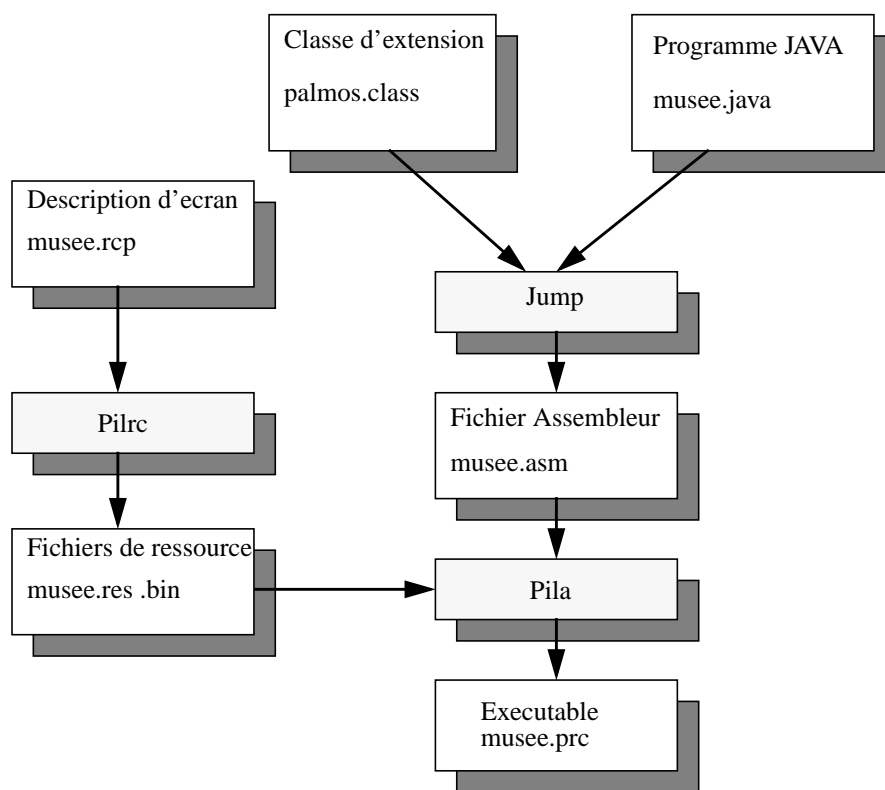


FIGURE 5.

Cycle de réalisation d'un programme pour le Pilot

Comme mentionné, tous ces outils sont disponibles sur Internet et sont en Shareware, donc on peut les utiliser gratuitement. Depuis le début des ventes il y a un petit groupe de programmeurs du Pilote qui se sont mis à réaliser plein d'outils pour la programmation et des informations peuvent être obtenus dans des newsgroups.

Pour la suite on aura besoin des archives suivantes:

- ASDK
- Copilot
- Pilot Studio

Ces fichiers peuvent être trouvés par exemple sur un des trois sites suivantes:

```
http://www.massena.com/darrin/pilot/tanda.htm  
http://www.sls.lcs.mit.edu/~raylau/pilot/  
http://www.inforamp.net/~adam/pilot
```

5.3.1 Jump

Jump est un programme qui génère du code assembleur 68000 à partir du fichier .class du programme Java compilé (avec compilateur Java standard comme le javac de Sun ou le jvc de Microsoft).

Jump demande quelques connaissances pour l'installation:

- Il faut bien mettre la variable CLASSPATH à jour. (dans autoexec.bat: SET CLASSPATH = C:\Myath\palmos.class)
- Il faut avoir une version récente de machine virtuelle. (Dernière version du SDK de Microsoft)

Suivez bien chaque pas de la description d'installation!

Lors de la compilation il se peut que l'utilisateur obtient un message de manque d'une fonction native. Il peut être ignoré, car il s'agit d'une option qui n'est pas encore implémenté.

5.3.2 Pila

Pila est un assembleur qui prend du code assembleur 68000 en entrée et produit du code exécutable pour des processeurs Motorola 68328 "Dragon Ball". La nouvelle version de pila cree automatiquement le fichier de ressources (.res).

Avec Pila, aucun problème n'a été rencontré.

5.3.3 PiIRC

PiIRC est le compilateur de ressources pour le Pilot. A l'aide d'un langage spécial (cf 3.5.8) et avec PiIRC on cree (compile) les ressources de l'interface utilisateur.

5.3.4 PiIRCUI

PiIRCUI permet de prévisualiser l'allure de l'interface utilisateur.

Les quatre outils présentés jusqu'ici s'utilisent sur la ligne de commande, c'est à dire dans une fenêtre DOS. Les programme suivants ont tous une interface graphique et sont donc plus facile à utiliser.

5.3.5 Copilot

Le programme copilot est un emulateur Pilot sous Windows95. Il a besoin d'une copie de la ROM du Pilot qu'il exécute afin d'obtenir une emulation réaliste.

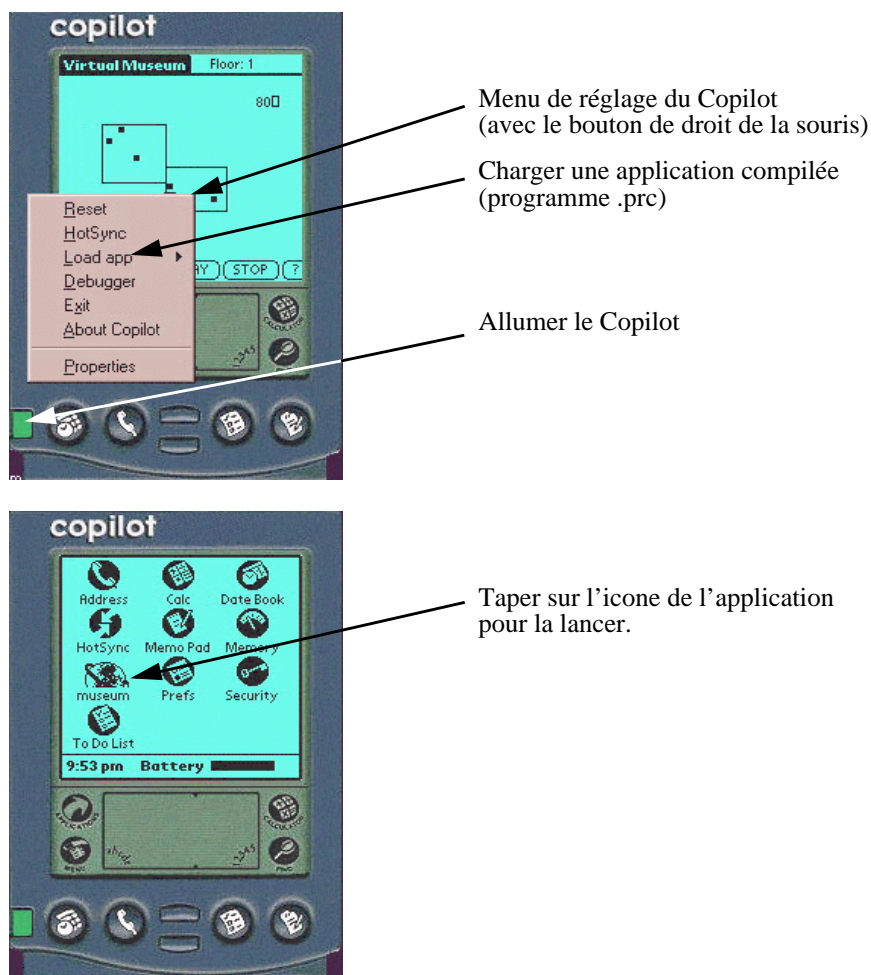


FIGURE 6.

Exemples d'écran du Copilot

Le copilot est un outil très puissant pour tester des applications directement sur PC.

5.3.6 Le Pilot Studio

Le Pilot Studio est un outil de développement graphique qui se présente dans le style des langages Visual de Microsoft (Visual C++, Visual J++, etc.). Le Pilot Studio fait appel à tous les outils présentés avant et enlève donc la tâche fastidieuse de taper sur la ligne de commande.

Cet outil présente le plus grand avantage de cette méthode de programmation car il facilite considérablement le procès de développement.

On dispose d'une fenêtre montrant le programme en édition. De plus on a une fenêtre montrant tous les fichiers constituant le projet actuel.

Toute l'édition et la compilation se font par des boutons.

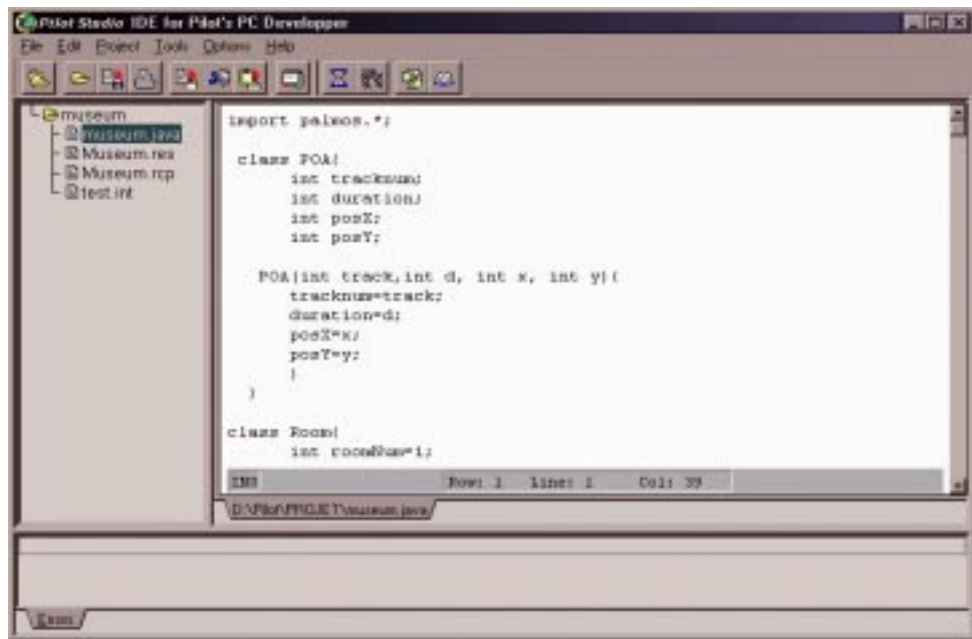


FIGURE 7.

Exemple d'écran du Pilot Studio

5.4 Les différents fichiers

On va présenter un programme d'exemple très facile pour montrer le rôle des différents fichiers.

5.4.1 le fichier .java

Ce fichier contient le programme de base en Java.

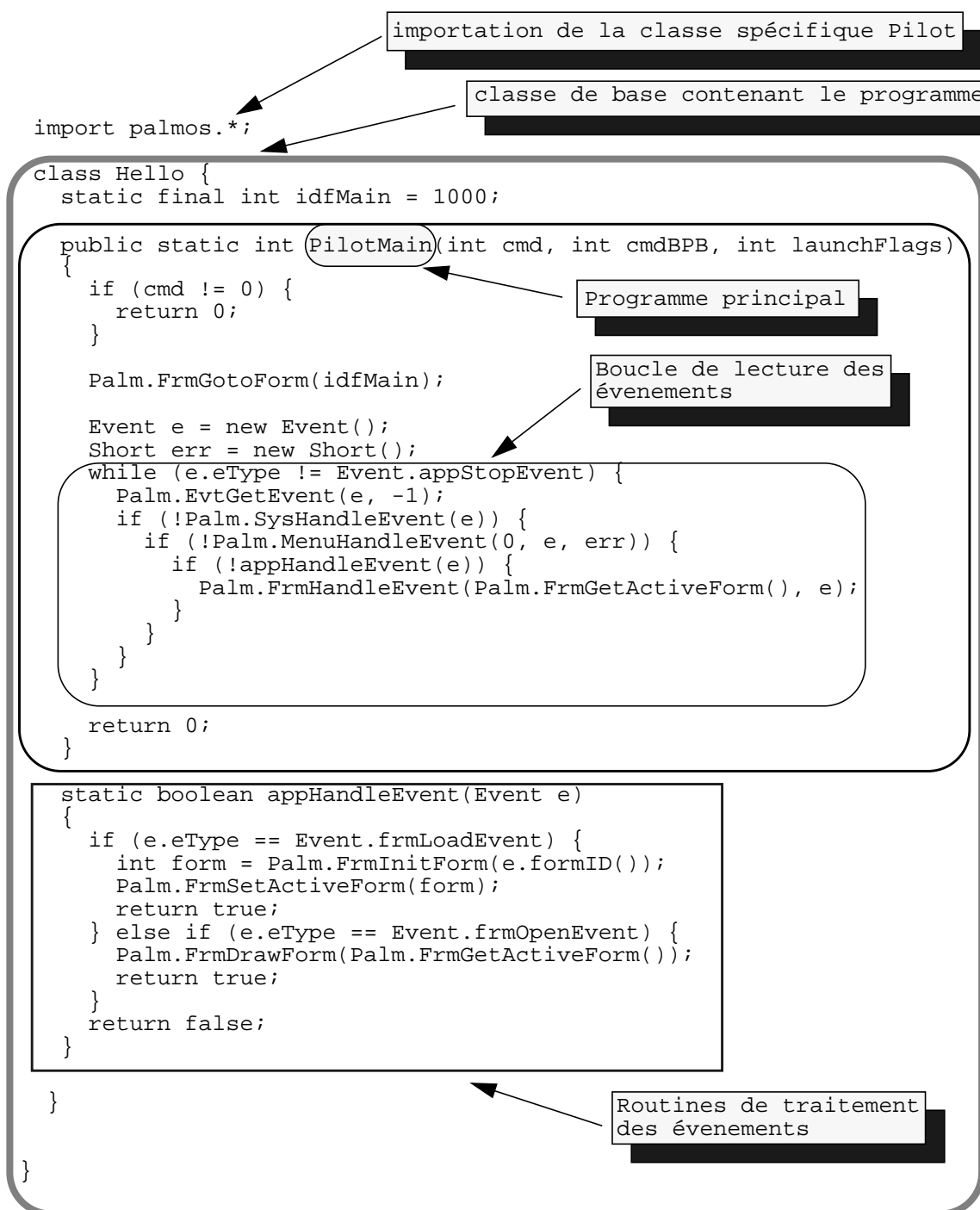


FIGURE 8.

Squelette d'un programme pour Pilot

5.4.2 le fichier .rcp

Ce fichier contient la description.

```
FORM (1000) 0 0 160 160
USABLE
NOFRAME
BEGIN
    TITLE "Hello World"
    LABEL "Hello, world!" 1001 60 70
END
```

Numéro d'identification pour l'affichage

5.4.3 le fichier .res

Ce fichier contient les ressources nécessaires.

```
res 'WBMP', $7ffe, "Hello.bmp"
res 'tFRM', 1000, "tFRM03e8.bin"
```

Nom de l'icone pour la representation Pilot

5.4.4 le fichier .asm

Le fichier produit par Jump. Il contient le code assembleur Motorola 68000.

5.4.5 le fichier .prc

Le fichier final. Contient le code Motorola 68328 "Dragon Ball". Pour l'exécuter il faut le charger, ou bien dans le Copilot pour faire des tests ou dans le Pilot par Hot-Sync².

2. Le programme HotSync est fourni avec le Pilot. Pour plus d'information veuillez consulter le manuel du Pilot.

5.5 Structure générale d'un programme Pilot

Toute action du Pilot est déclenchée par des événements. Donc la structure principale est constituée par une boucle de traitement d'événements.

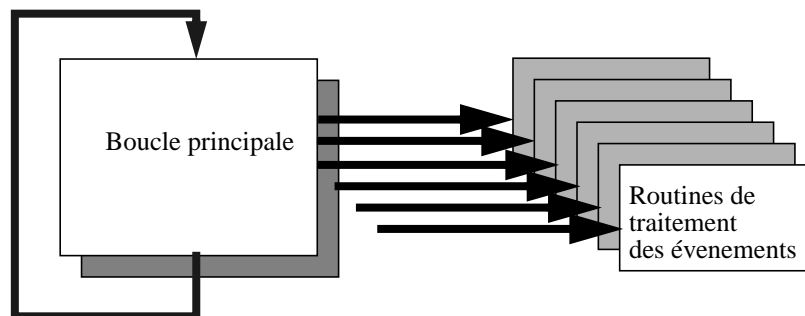


FIGURE 9.

Boucle de traitement d'événements

Les événements sont alors traités dans des routines séparées de la boucle principale.

Les objets actifs de l'interface utilisateur sont identifiés à l'aide de nombres déterministes lors de la définition dans le fichier .rcp (cf. 5.4.2). Si on actionne un tel objet, un événement est déclenché. Le programme Java peut récupérer ce nombre qui identifie l'objet ayant produit l'événement et agir en fonction.

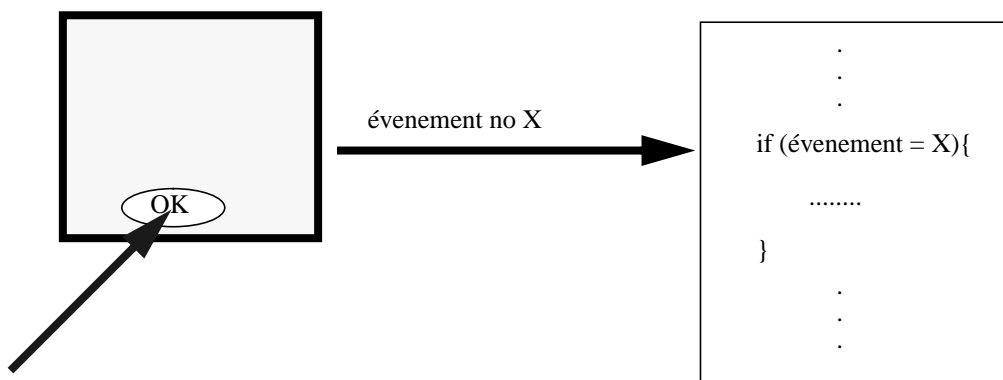


FIGURE 10.

Interaction bouton - programme

5.6 Schéma du programme

L'interface contient cinq boutons:

- Up Monter d'un étage
- Down Descendre d'un étage
- Play Donner l'explication pour l'oeuvre sélectionné
- Stop Interrompre l'explication au cours
- ? Informations sur le programme

Une carte du musée est affichée, un étage à la fois.

Une oeuvre peut être sélectionné en tapant à proximité et sera marquée.

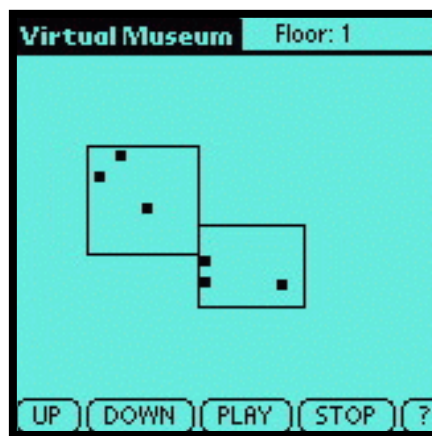


FIGURE 11.

Copie d'écran du programme

La boucle principale appelle la routine de traitement propre au programme, si elle reçoit un événement qui provient d'un appui sur l'écran.

Cette routine vérifie s'il s'agit d'un bouton appuyé ou d'une sélection sur la carte du musée.

- Si le bouton UP est appuyé:
On incrémente le compteur interne de l'étage actuel et on redessine l'étage.
- Si le bouton DOWN est appuyé:
On décrémente le compteur interne de l'étage actuel et on redessine l'étage.
- Si le bouton PLAY est appuyé:
On cherche le numéro de la piste à jouer correspondant à l'oeuvre actuel.
On va à la piste 1 du MiniDisc, en envoyant un nombre suffisant de signaux au niveau de tension REW (cf. chapitre 3.1).
On va à la piste désiré en envoyant N signaux FWD.
- Si le bouton STOP est appuyé:
On envoie le signal STOP au MiniDisc.

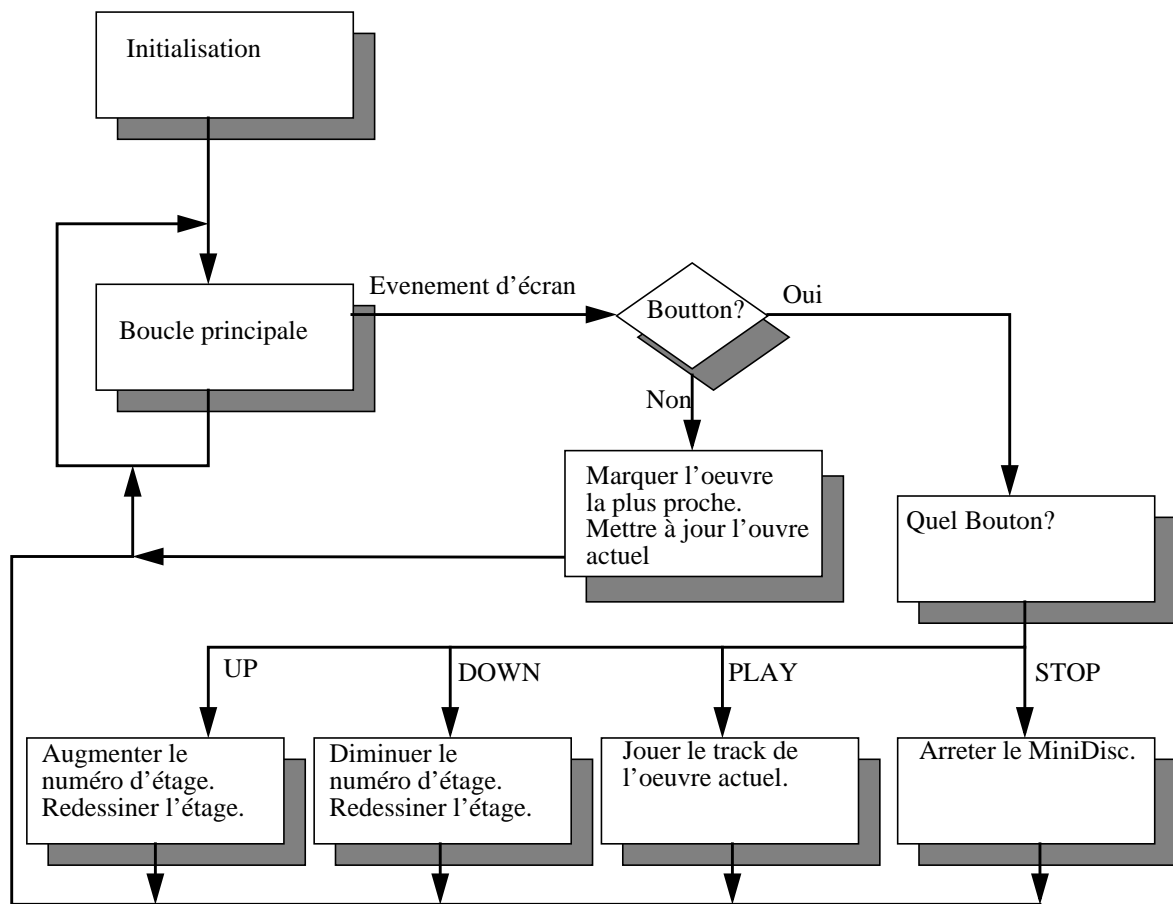


FIGURE 12. Schéma du déroulement du programme

5.6.1 Structure des données

Comme Java est un langage de programmation objet, on a implémenté toute la structure de données en utilisant des objets.

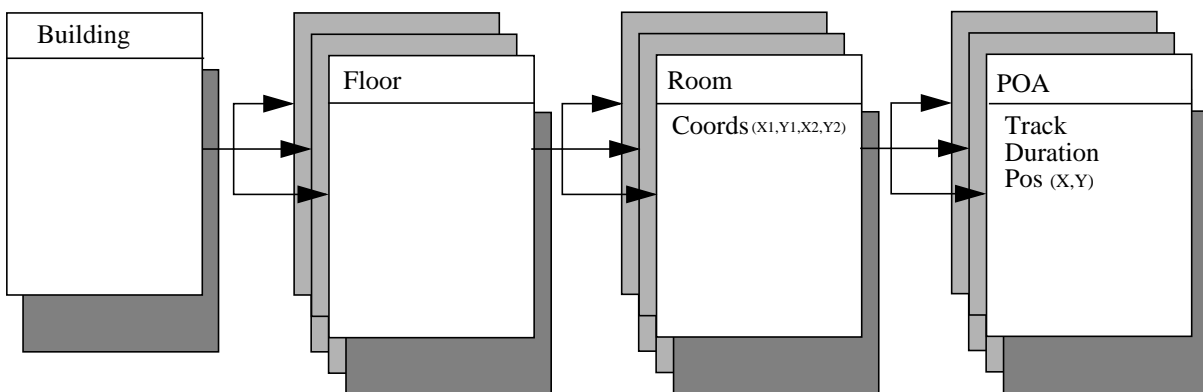


FIGURE 13. Structure des objets

On a quatre types d'objets. On a l'objet *Building* (bâtiment), qui engendre toute la structure et qui représente tout le musée. Le *Building* contient plusieurs objets *Floor* (étage) qui eux sont constitués d'objets *Room* (chambre). Un objet *Room* contient l'information sur sa taille et sa position donnée par deux points opposés. Finalement chaque objet *Room* peut contenir plusieurs objets *POA* (Pièce of Art; oeuvre) qui sont représentés par leur position, le numéro de la piste correspondante et de la durée de la piste (pour pouvoir arrêter le MiniDisc après qu'il ait donné toutes les explications).

Afin d'obtenir cette base de données de manière facile, on a développé un programme qui génère le code nécessaire à partir d'un script descriptif du musée (cf 8.1).

Pour une vue plus détaillée veuillez consulter le listing documenté dans l'annexe (cf. 8.2)

6.0 Conclusion

Le projet a été très varié, car il nécessitait du développement hardware ainsi que du développement logiciel.

Du côté hardware, il fallait concevoir et réaliser une interface à partir de très peu de connaissances. On a du faire du reverse developpement (développement inverse), pour trouver comment la télécommande du MiniDisc est mise en oeuvre et en plus il fallait trouver une solution facile à réaliser.

Du côté logiciel on a bien pu découvrir les bases de la programmation du Pilot qui est encore très récente. C'est aussi à cause de cela que parfois, il était presque impossible de trouver des réponses à des questions concernant l'utilisation de certaines routines. Même dans les newsgroups qui existent au sujet du Pilot, on remarque que le niveau de connaissances est encore assez faible.

De toute façon c'était très intéressant de pouvoir participer à ces développements et de voir de quelle manière le Pilot a commencé à se placer dans le domaine de l'informatique. Dans le futur, le Pilot jouera sûrement un rôle important comme ordinateur pour des applications demandant un dispositif bon marché, portable et facilement maniable.

Lausanne, le **23 juin 1997**

7.0 Annexe

7.1 Créer de nouveaux musées

On a créé un programme appelé script, qui lit un fichier de description en entrée et génère le code nécessaire en sortie. Le code créé devra remplacer la partie marqué dans le listing du fichier musee.java.

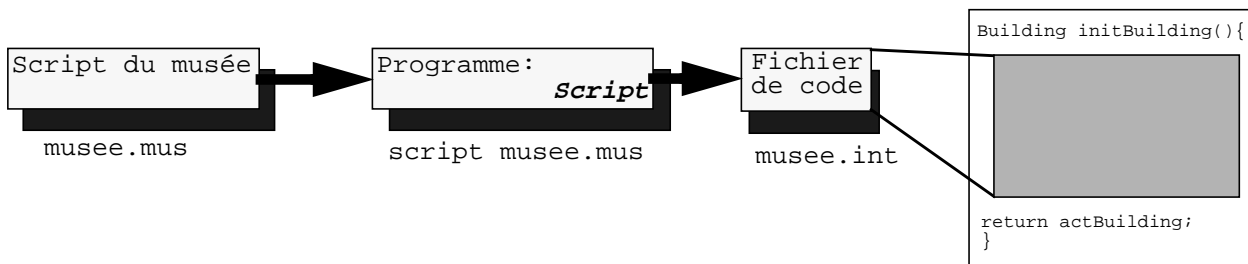


FIGURE 14.

Procès de création de nouveaux musées

Ensuite le programme doit être recompilé et rechargé dans le Pilot.

7.2 Schémas électroniques

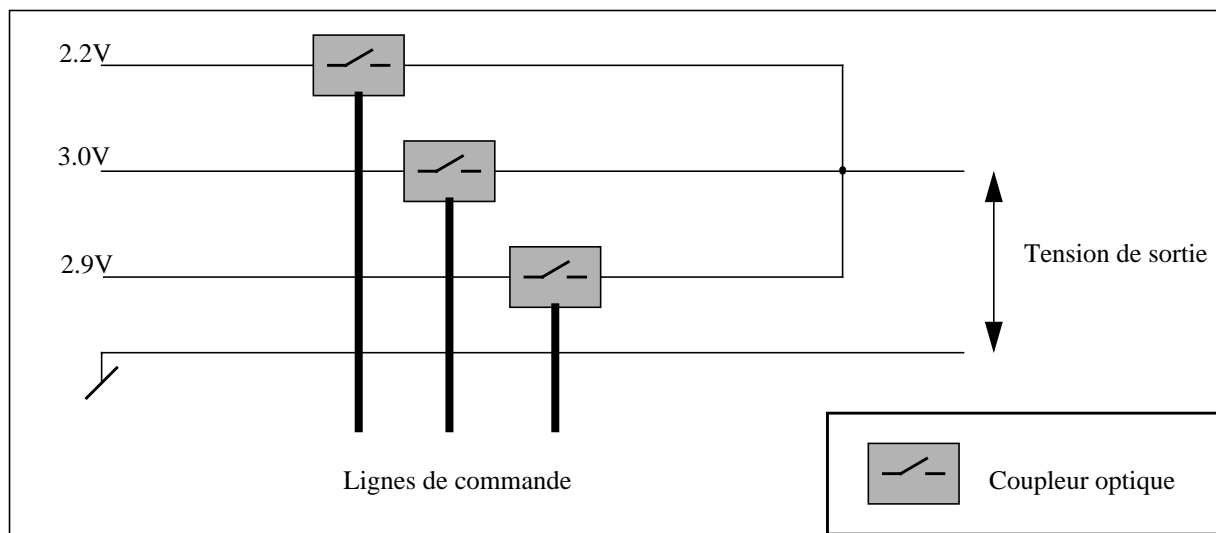


FIGURE 15.

Etage de sortie

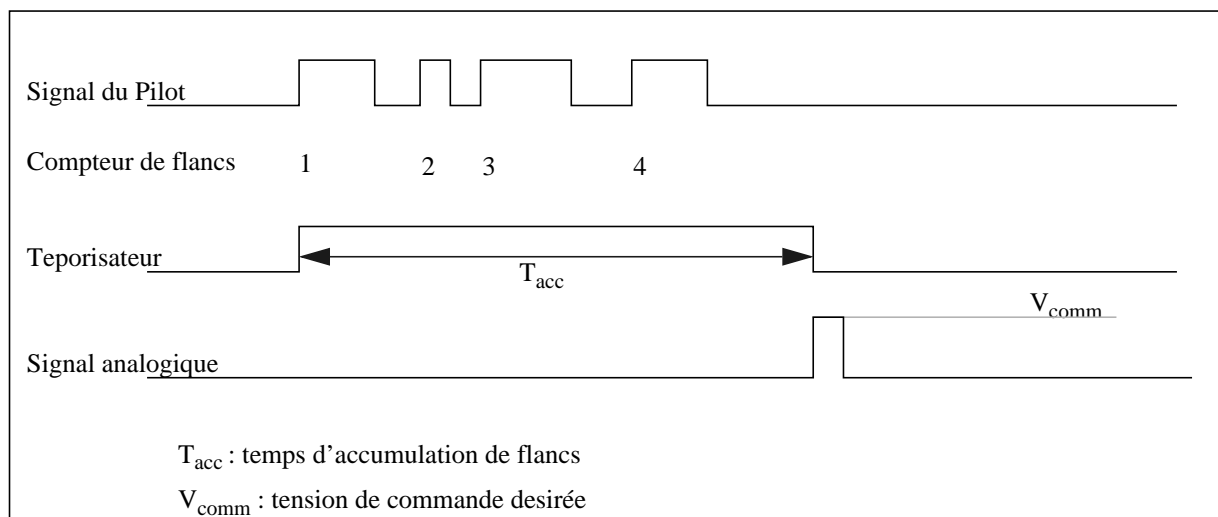


FIGURE 16.

Diagramme de timing

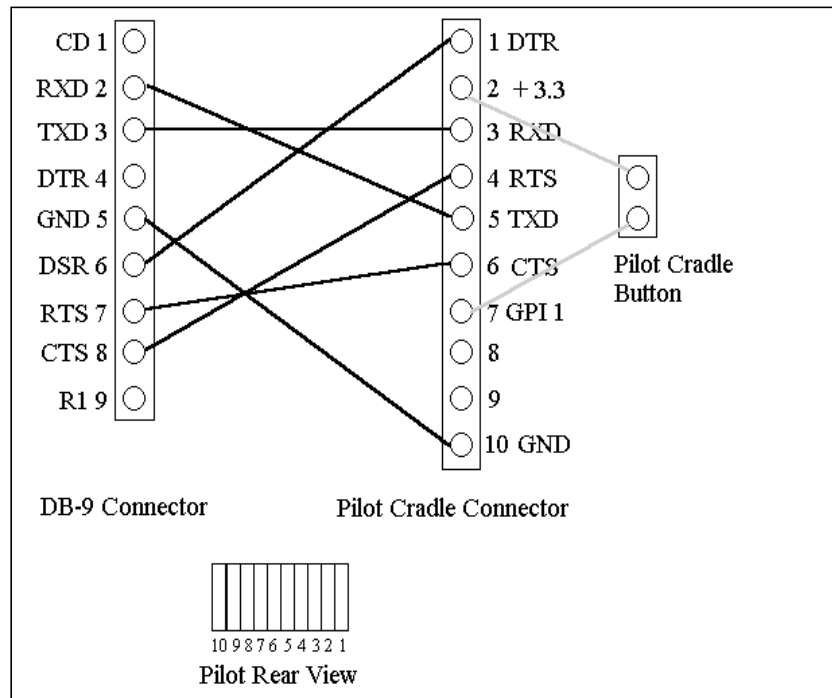


FIGURE 17.

Schéma des pins du Pilot

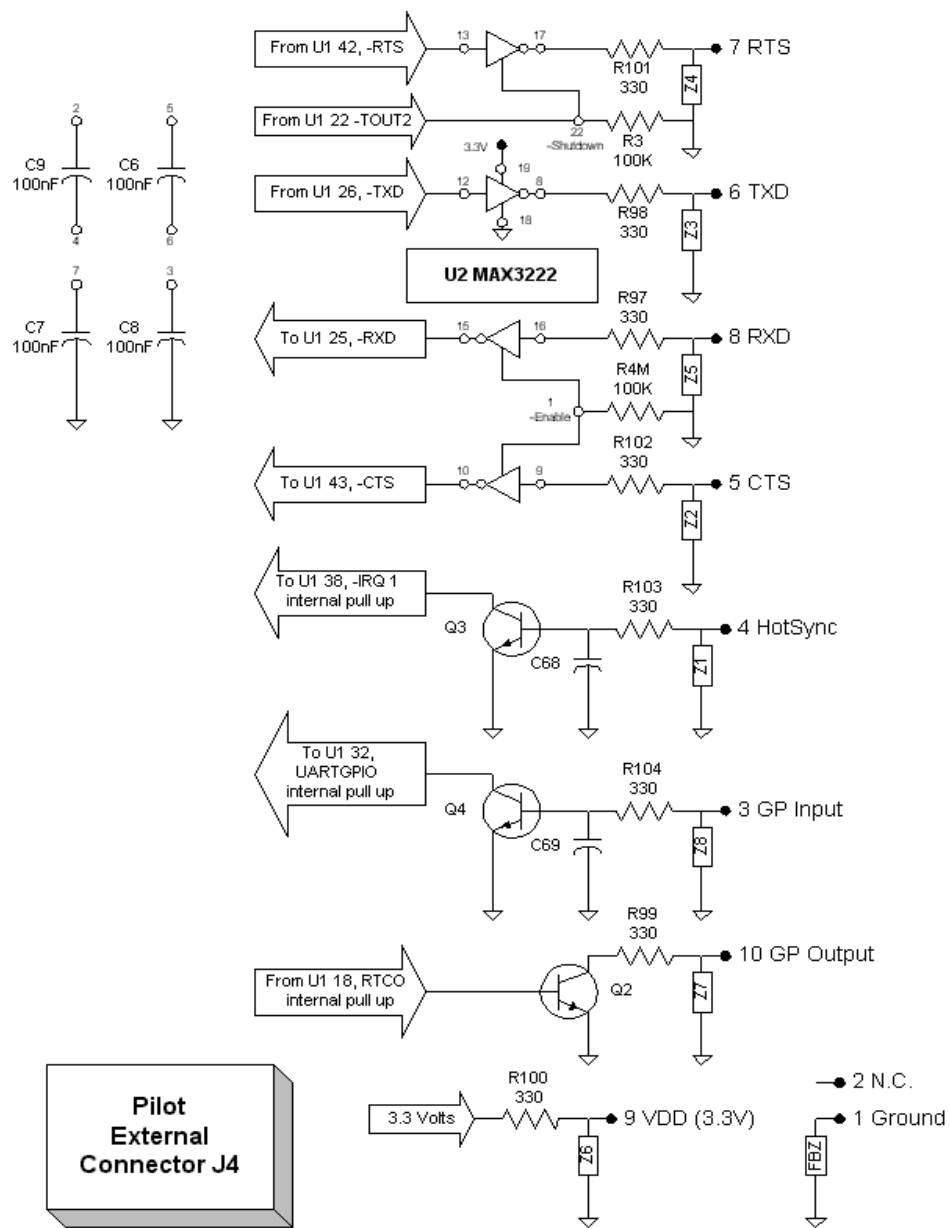
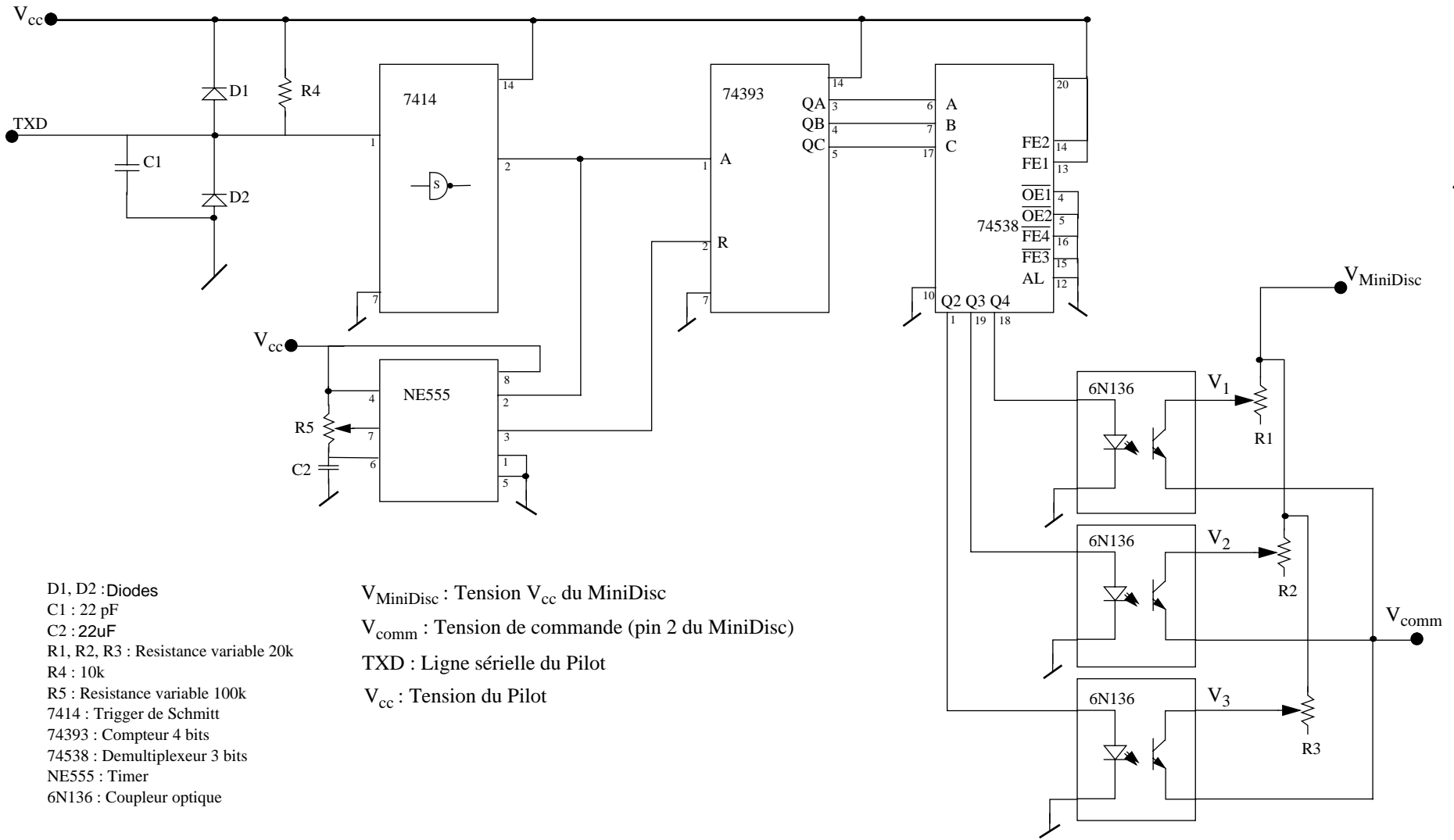


FIGURE 18.

Schéma électronique de l'étage de sortie du Pilot



- D1, D2 : Diodes
- C1 : 22 pF
- C2 : 22uF
- R1, R2, R3 : Resistance variable 20k
- R4 : 10k
- R5 : Resistance variable 100k
- 7414 : Trigger de Schmitt
- 74393 : Compteur 4 bits
- 74538 : Demultiplexeur 3 bits
- NE555 : Timer
- 6N136 : Coupleur optique

- $V_{MiniDisc}$: Tension V_{cc} du MiniDisc
- V_{comm} : Tension de commande (pin 2 du MiniDisc)
- TXD : Ligne sérielle du Pilot
- V_{cc} : Tension du Pilot

7.3 Listings des programmes

7.3.1 le programme principal pour Pilot (musee.java)

```
import palmos.*;

class POA{           //class for one PieceOfArt
    int tracknum;    //number of the track to play
    int duration;    //duration of the track
    int posX;       //position of the PieceOfArt
    int posY;

    POA(int track,int d, int x, int y){
        tracknum=track;
        duration=d;
        posX=x;
        posY=y;
    }
}

class Room{         //class for a Room
    int roomNum=1;  //id number of the room
    POA POAs [];   //POAs in the room
    int POACount=1;
    int upperX;    //coords of the room
```

```
    int upperY;
    int lowerX;
    int lowerY;

    Room(POA RoomPOAs [], int ux,int uy, int lx, int ly, int POANum){
        POAs=RoomPOAs;
        upperX=ux;
        upperY=uy;
        lowerX=lx;
        lowerY=ly;
        POACount=POANum;
    }

    int GetPOANum(){ //returns the number of POAs in a room
        return POACount;
    }

    POA GetPOA(int i){ //returns a POA with a specific number
        if (i<=POACount){
            return POAs[i];
        }
        else return POAs[0];
    }
}
```

```

class Floor {           //class for a floor
    int floorNum=1;    //floor number
    Room Rooms [];    //rooms on a floor
    int roomCount=0;  //number of rooms

    Floor( Room actRooms [], int roomNum, int actFloorNum){
        Rooms=actRooms;
        roomCount=roomNum;
        floorNum=actFloorNum;
    }

    int GetFloorNum(){ //returns the number of the floor
        return floorNum;
    }

    int GetRoomNum(){ //returns the number of rooms on a floor
        return roomCount;
    }

    Room GetRoom(int i){ //returns the room with a specific number
        if (i<=roomCount){
            return Rooms[i];
        }
        else return Rooms[0];
    }
}

```

```

    }
}

class Building{        //class for a building
    Floor Floors [];   //floors in the building
    int floorCount=1;  //number of floors in the building

    Building(Floor actFloors [],int floorNum){
        Floors=actFloors;
        floorCount=floorNum;
    }

    int GetFloorNum(){ //returns the number of floors in a building
        return floorCount;
    }

    Floor GetFloor(int i){ //returns a floor with a specific number
        if (i<=floorCount){
            return Floors[i-1];
        }
        else return Floors[0];
    }
}

```

```

class Museum {
    static final int idfMain    = 1000; //id numbers of the forms
    static final int idfAbout  = 1100;

    static final int idcUP    = 1001; //id numbers of the buttons
    static final int idcDOWN  = 1002;
    static final int idcHEAR  = 1003;
    static final int idcABOUT = 1004;
    static final int idcSTOP  = 1005;

    static final int XSIZE = 150; //size of the drawable area
    static final int YSIZE = 120;
    static final int maxTrackNumber = 100; //number of tracks allowed

    Building MyMuseum; //create the museum

    int floor = 1; // initial floor
    POA selectedPOA; // initial PieceOfArt

    boolean playing = false; //is the MiniDisc playing?
    int actTime = 0; //playing time
    static final char CSTOP = "3"; //character to forward a track
    static final char CFWD = "4"; //character to go back a track
    static final char CRWD = "5"; //character to go back a track

```

```

    Rectangle r = new Rectangle();

    public static int PilotMain(int cmd, int cmdBPB, int launchFlags)
    { //Main programm
        if (cmd == 0) {
            new Museum().run();
        }
        return 0;
    }

    public void run() {
        Event e = new Event();
        Short err = new Short();
        int win;

        Floor tmpFloor;
        MyMuseum=initBuilding(); //initialise the museum

        Palm.FrmGotoForm(idfMain); //opens the FORM
        //event loop
        while (e.eType != Event.appStopEvent) {

            Palm.EvtGetEvent(e, 1);

```

```

if (playing){actTime++;}
if (actTime>selectedPOA.duration){
    stopTrack(); //if the track has finished stop the MiniDisc
    playing=false;
    actTime=0;
}
if (!Palm.SysHandleEvent(e)) {
    if (!Palm.MenuHandleEvent(0, e, err)) {
        if (!appHandleEvent(e)) {
            Palm.FrmHandleEvent(Palm.FrmGetActiveForm(), e);
        }
    }
}
}

boolean appHandleEvent(Event e)
{ //handle the events for the program
if (e.eType == Event.frmLoadEvent) {
    int form = Palm.FrmInitForm(e.formID());
    Palm.FrmSetActiveForm(form);
    return true;
}
else if (e.eType == Event.frmOpenEvent) {

```

```

    Palm.FrmDrawForm(Palm.FrmGetActiveForm());
    drawFloor(MyMuseum.GetFloor(floor));//drawFloor(initRooms());
    return true;
}
else if (e.eType == Event.penDownEvent) {
    //POA selected
    int x = e.screenX;
    int y = e.screenY;
    if (x >= 10 && x <= XSIZE && y > 15 && y <= YSIZE) {
        selected(x,y); // pen down SELECTION
    }
}
else
if (e.eType == Event.ctlSelectEvent) {
    //handle the buttons
    if (e.controlID() == idcUP) { // UP selected
        up();
        return true;
    }
    else if (e.controlID() == idcDOWN) { // DOWN selected
        down();
        return true;
    }
    else if (e.controlID() == idcHEAR) { // HEAR selected
        hear(selectedPOA);

```

```

return true;
}
else if (e.controlID() == idcSTOP) { // HEAR selected
    stopTrack();
    playing=false;
    actTime=0;
return true;
}
else if (e.controlID() == idcABOUT) {
    Palm.FrmAlert(idfAbout);
}
}
else if (e.eType == Event.nilEvent) {
    return true;
}
return false;
}

void gotoTrackOne(){ //rewinds to first track
    for (int i = 0; i < maxTrackNumber; i++){
        Palm.SerSend(1,CREW,1); // sends maxTrackNumber RWD
        Palm.SysTaskDelay(50);
    }
}

```

```

void playTrack(int trackNumber){ // plays the track number trackNumber
    for (int i = 0; i < trackNumber; i++){
        Palm.SerSend(1,CFWD,1); // sends trackNumber Forwards
        Palm.SysTaskDelay(50);
    }
}

void stopTrack(){
    Palm.SerSend(1,CSTOP,1);
}

void selected(int x, int y){ // fixes the selection of one POA
    eraseWin();
    drawFloor(MyMuseum.GetFloor(floor));
    selectedPOA=getNearestPOA(MyMuseum.GetFloor(floor),x,y);
}

void up(){ //goes up one floor
    if (floor<MyMuseum.GetFloorNum()){
        floor=floor+1;
        eraseWin();
    }
}

void down(){ //goes down one floor

```

```

if (floor>1){
    floor=floor-1;
    eraseWin();
    drawFloor(MyMuseum.GetFloor(floor));//drawFloor(MyMuseum.GetFloor(floor));
}
}

```

```

void hear(POA actualPOA){ //plays a track
    playing=true;
    gotoTrackOne();
    playTrack(actualPOA.tracknum);
    Palm.SysTaskDelay(50);
}

```

```

Building initBuilding(){ //initialises the museum
    Floor MoreFloors[]; // this part has to be replaced if a new
                        // museum is generated
    MoreFloors = new Floor[2];
    Room MoreRooms0 [];

```

```

MoreRooms0 =new Room [2];

POA MorePOAs0 [];
MorePOAs0 =new POA [3];
POA actPOA0 = new POA(1,40,100,100);
MorePOAs0[0]=actPOA0;
POA actPOA1 = new POA(2,50,31,59);
MorePOAs0[1]=actPOA1;
POA actPOA2 = new POA(3,60,39,51);
MorePOAs0[2]=actPOA2;
Room actRoom0 = new Room(MorePOAs0,29,50,70,90,3);
MoreRooms0[0]= actRoom0;

POA MorePOAs1 [];
MorePOAs1 =new POA [3];
POA actPOA3 = new POA(4,70,71,91);
MorePOAs1[0]=actPOA3;
POA actPOA4 = new POA(5,80,71,99);
MorePOAs1[1]=actPOA4;
POA actPOA5 = new POA(6,90,49,71);
MorePOAs1[2]=actPOA5;
Room actRoom1 = new Room(MorePOAs1,71,80,110,110,3);
MoreRooms0[1]= actRoom1;
Floor actFloor0 = new Floor(MoreRooms0,2,0);
MoreFloors[0]=actFloor0;

```

```

Room MoreRooms1 [];
MoreRooms1 =new Room [2];

POA MorePOAs2 [];
MorePOAs2 =new POA [3];
POA actPOA6 = new POA(1,100,16,11);
MorePOAs2[0]=actPOA6;
POA actPOA7 = new POA(2,110,13,19);
MorePOAs2[1]=actPOA7;
POA actPOA8 = new POA(3,120,19,19);
MorePOAs2[2]=actPOA8;
Room actRoom2 = new Room(MorePOAs2,10,10,40,40,3);
MoreRooms1[0]= actRoom2;
POA MorePOAs3 [];

MorePOAs3 =new POA [3];
POA actPOA9 = new POA(4,130,51,51);
MorePOAs3[0]=actPOA9;
POA actPOA10 = new POA(5,140,51,59);
MorePOAs3[1]=actPOA10;
POA actPOA11 = new POA(6,150,79,51);
MorePOAs3[2]=actPOA11;
Room actRoom3 = new Room(MorePOAs3,50,50,90,80,3);
MoreRooms1[1]= actRoom3;
Floor actFloor1 = new Floor(MoreRooms1,2,1);

```

```

MoreFloors[1]=actFloor1;
Building actBuilding = new Building(MoreFloors,2);

return actBuilding;
}

void drawPOA(POA actPOA, boolean type){
    if (type){ //draws a marked or unmarked POA
        r.topLeft_x = (short)(actPOA.posX);
        r.topLeft_y = (short)(actPOA.posY);
        r.extent_x = (short)(4);
        r.extent_y = (short)(4);
        Palm.WinDrawRectangle(r,0);
    }else
        {r.topLeft_x = (short)(actPOA.posX-1);
        r.topLeft_y = (short)(actPOA.posY-1);
        r.extent_x = (short)(6);
        r.extent_y = (short)(6);
        Palm.WinDrawRectangleFrame(1,r);
        }
}

void drawRoom(Room actRoom){
    r.topLeft_x = (short)(actRoom.upperX);
    r.topLeft_y = (short)(actRoom.upperY);
    r.extent_x = (short)(actRoom.lowerX-actRoom.upperX);

```

```

    r.extent_y = (short)(actRoom.lowerY-actRoom.upperY);
    Palm.WinDrawRectangleFrame(1,r);
    for (int j = 0; j <= actRoom.GetPOANum()-1; j++){
        drawPOA(actRoom.GetPOA(j),true); //draw all POAs in the room
    }

void drawFloor(Floor actFloor){
    String OutStr = new String();

    for (int i = 0; i <= actFloor.GetRoomNum()-1; i++){
        drawRoom(actFloor.GetRoom(i)); //draw all rooms on a floor
    }
    OutStr="Floor: "+Integer.toString(floor)+" ";
    Palm.WinDrawChars(OutStr,OutStr.length(),100,1);
    Palm.SysTaskDelay(50);
    Palm.EvtFlushPenQueue(); //supress button presses while drawing
    Palm.EvtFlushKeyQueue();

}

int distance(int x1, int y1, int x2, int y2){
    return (int)((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)); //returns the cube of the
        distance of a point and a POA
}

```

```

POA getNearestPOA(Floor actFloor,int x, int y){ //looks for the nearest POA
    Room actRoom;
    POA actPOA, nearestPOA;
    int actDist=1000;
    nearestPOA=actFloor.Rooms[1].POAs[1];

    for (int j = 0; j <= actFloor.GetRoomNum()-1; j++){
        actRoom=actFloor.GetRoom(j);
        for (int i = 0; i <= actRoom.GetPOANum()-1; i++){
            actPOA=actRoom.POAs[i];
            if (distance(x,y,actPOA.posX,actPOA.posY)< actDist){
                actDist=distance(x,y,actPOA.posX,actPOA.posY);
                nearestPOA=actPOA;
            }
        }
        drawPOA(nearestPOA,false);
        return nearestPOA;
    }
}

void eraseWin(){ //erases the card
    r.topLeft_x = (short)(0);
    r.topLeft_y = (short)(15);
    r.extent_x = (short)(XSIZE);
    r.extent_y = (short)(YSIZE);
    Palm.WinEraseRectangle(r,0);
}
}

```

7.3.2le fichier musee.rcp

```
FORM 1000 0 0 160 160
USABLE
FRAME
BEGIN
TITLE "Virtual Museum"
    BUTTON "UP" 1001 2 146 AUTO AUTO
    BUTTON "DOWN" 1002 prevright+4 146 AUTO AUTO
    BUTTON "PLAY" 1003 prevright+4 146 AUTO AUTO
    BUTTON "STOP" 1005 prevright+4 146 AUTO AUTO
    BUTTON "?" 1004 prevright+4 146 AUTO AUTO
END

ALERT 1100
INFORMATION
BEGIN
    TITLE "Virtual Museum"
    MESSAGE "This is an implementation of a virtual museum guide. \n
    Niklaus Hirt (C) 1997  "
    BUTTONS "Dismiss"
END
```

7.3.3le fichier musee.res

```
res 'WBMP', $7ffe, "Museum.bmp"
res 'tFRM', 1000, ".\tFRM03e8.bin"
res 'tFRM', 1200, ".\tFRM04b0.bin"
res "Talt", 1100, ".\Talt044c.bin"
```

7.3.4Le fichier de description du musée

Building

2

Floor

2

Room

10

10

40

40

3

POA

11

11

1

POA

11

19
2
POA
19
11
3

Room
50
50
90
90
3
POA
51
51
4
POA
51
79
5
POA
79
51
6

Floor
2

Room
10
10
40
40
3
POA
16
11
1
POA
13
19
2
POA
19
19
3

Room
50

50
90
80
3
POA
51
51
4
POA
51
59
5
POA
79
51
6

End.

7.3.5le fichier du programme script.java

```
/*script*/  
import java.io.*;  
    class POA{  
        int tracknum;  
        int duration;  
        int posX;  
        int posY;  
  
        POA(int track,int d, int x, int y){  
            tracknum=track;  
            duration=d;  
            posX=x;  
            posY=y;  
        }  
    }  
  
class Room{  
    int roomNum=1;  
    POA POAs [];  
    int POACount=1;  
    int upperX;  
    int upperY;  
    int lowerX;
```

```

int lowerY;

Room(POA RoomPOAs [], int ux,int uy, int lx, int ly, int POANum){
    POAs=RoomPOAs;
    upperX=ux;
    upperY=uy;
    lowerX=lx;
    lowerY=ly;
    POACount=POANum;
}

int GetPOANum(){
    return POACount;
}

POA GetPOA(int i){
    if (i<=POACount){
        return POAs[i];
    }
    else return POAs[0];
}

}

class Floor {

```

```

int floorNum=1;
Room Rooms [];
int roomCount=0;

Floor( Room actRooms [], int roomNum, int actFloorNum){
    Rooms=actRooms;
    roomCount=roomNum;
    floorNum=actFloorNum;
}

int GetFloorNum(){
    return floorNum;
}

int GetRoomNum(){
    return roomCount;
}

Room GetRoom(int i){
    if (i<=roomCount){
        return Rooms[i];
    }
    else return Rooms[0];
}

}

```

```

class Building{
    Floor Floors [];
    int floorCount=1;

    Building(Floor actFloors [],int floorNum){
        Floors=actFloors;
        floorCount=floorNum;
    }

    int GetFloorNum(){
        return floorCount;
    }

    Floor GetFloor(int i){
        if (i<=floorCount){
            return Floors[i-1];
        }
        else return Floors[0];
    }
}

class Script{

```

```

public static void main(String args[]){

    String name = args[1];
    Building actBuilding =null;

    try{
        String actLine = null;
        int POACount = -1;
        int EachPOA = -1;
        int actTrack =0;
        int actDur =0;
        int actposX =0;
        int actposY =0;

        int RoomCount =-1;
        int EachRoom =-1;
        int x1=0;
        int x2=0;
        int y1=0;
        int y2=0;

        int FloorCount=0;
        POA MorePOAs ]= null;
        Room MoreRooms ] =null;
        Floor MoreFloors [ =null;

```

```

FileInputStream intmp = new FileInputStream(name+".mus");
DataInputStream inf = new DataInputStream(intmp);
FileOutputStream outtmp = new FileOutputStream(name+".int");
PrintStream outf = new PrintStream(outtmp);

actLine=inf.readLine();
if (actLine.compareTo("Building")==0) {
    while (actLine.compareTo("End.")!=0){
        FloorCount=0;
        POACount=0;
        RoomCount=0;

        actLine=inf.readLine();
        outf.print("Floor MoreFloors[];");
        outf.println();
        outf.print("MoreFloors = new Floor["+Integer.parseInt(act-
            Line)+"];");
        outf.println();
        System.out.println("MoreFloors = new Floor["+Integer.par-
            seInt(actLine)+"];");

        while (actLine.compareTo("Floor")!=0){
            actLine=inf.readLine();}
        while(actLine.compareTo("Floor")==0){

```

```

actLine=inf.readLine();
outf.print("Room MoreRooms"+FloorCount+" [];");
outf.println();
outf.print("MoreRooms"+FloorCount+" =new Room
["+Integer.parseInt(actLine)+"];");
outf.println();
System.out.println("MoreRooms"+FloorCount+" =new
Room["+Integer.parseInt(actLine)+"];");
EachRoom=0;
while (actLine.compareTo("Room")!=0){
    actLine=inf.readLine();}
while(actLine.compareTo("Room")==0){
    actLine=inf.readLine();
    x1 = Integer.parseInt(actLine);
    actLine=inf.readLine();
    y1 = Integer.parseInt(actLine);
    actLine=inf.readLine();
    x2 = Integer.parseInt(actLine);
    actLine=inf.readLine();
    y2 = Integer.parseInt(actLine);

    actLine=inf.readLine();
    outf.print("POA MorePOAs"+RoomCount+" [];");
    outf.println();

```

```

outf.print("MorePOAs"+RoomCount+" =new POA
["+Integer.parseInt(actLine)+"];");
outf.println();
System.out.println("MorePOAs"+RoomCount+"
=new POA["+Integer.parseInt(actLine)+"];");
EachPOA=0;

while (actLine.compareTo("POA")!=0){
actLine=inf.readLine();}
while(actLine.compareTo("POA")==0){

actLine=inf.readLine();
actTrack = Integer.parseInt(actLine);
actLine=inf.readLine();
actDur = Integer.parseInt(actLine);
actLine=inf.readLine();
actposX = Integer.parseInt(actLine);
actLine=inf.readLine();
actposY = Integer.parseInt(actLine);

outf.print("POA actPOA "+POACount+" = new
POA("+actTrack+", "+actDur+", "+act-
posX+", "+actposY+");");
outf.println();
System.out.println("POA actPOA "+POACount+" =
new POA("+actTrack+", "+actDur+", "+act-
posX+", "+actposY+");");

```

```

outf.print("MorePOAs"+RoomCount+"["+Each-
POA+"]=actPOA"+POACount+");");
outf.println();
System.out.println("MorePOAs"+Room-
Count+"["+EachPOA+"]=actPOA"+POACount+");");
actLine=inf.readLine();
POACount=POACount+1;
EachPOA=EachPOA+1;

}

outf.print("Room actRoom"+RoomCount+" = new
Room(MorePOAs"+Room-
Count+", "+x1+", "+y1+", "+x2+", "+y2+", "+Each-
POA+");");
outf.println();
System.out.println("Room actRoom"+RoomCount+"
= new Room(MorePOAs"+Room-
Count+", "+x1+", "+y1+", "+x2+", "+y2+", "+Each-
POA+");");
outf.print("MoreRooms"+Floor-
Count+"["+EachRoom+"]= actRoom"+Room-
Count+");");
outf.println();
System.out.println("MoreRooms"+Floor-
Count+"["+EachRoom+"]= actRoom"+Room-
Count+");");

```

```

        actLine=inf.readLine();
        RoomCount=RoomCount+1;
        EachRoom=EachRoom+1;
    }

    outf.print("Floor actFloor"+FloorCount+" = new
        Floor(MoreRooms"+Floor-
            Count+", "+EachRoom+", "+FloorCount+"");
    outf.println();
    System.out.println("Floor actFloor"+FloorCount+" = new
        Floor(MoreRooms"+Floor-
            Count+", "+EachRoom+", "+FloorCount+"");

    outf.print("MoreFloors["+FloorCount+"]=act-
        Floor"+FloorCount+"");
    outf.println();
    System.out.println("MoreFloors["+FloorCount+"]=act-
        Floor"+FloorCount+"");
    //actLine=inf.readLine();
    FloorCount=FloorCount+1;

    }
}

outf.print("Building actBuilding = new Building(MoreFloors,"+Floor-
    Count+" );");
outf.println();

```

```

    }
}
catch(IOException e){
}
}
}

```