Le Projet 1

1.1 L'exposé du projet

Nous avons un robot géré par un IPC à notre disposition. Un peu plus loin nous expliquons comment le robot et l'IPC fonctionnent et en même temps en quelles pièces le constitue. Au début de notre stage l'objectif du projet consistait de 3 composants.

Visualisation du robot sans IPC et sans robot

Il y a deux ordinateurs dans un réseau. Le premier a la fonction de télécommande et le deuxième simule le robot. Tous les programmes sont écrits en Delphi 7.0. La communication se passe avec TCP qui a été réalisée en Delphi par l'aide d'FPiette. Nous n'utilisons que deux composants d'FPiette [(1)] c'est-à-dire le TWSocketServer et le TWSocket. Pour simuler le robot en 3D nous utilisons le GLScene [(2)].

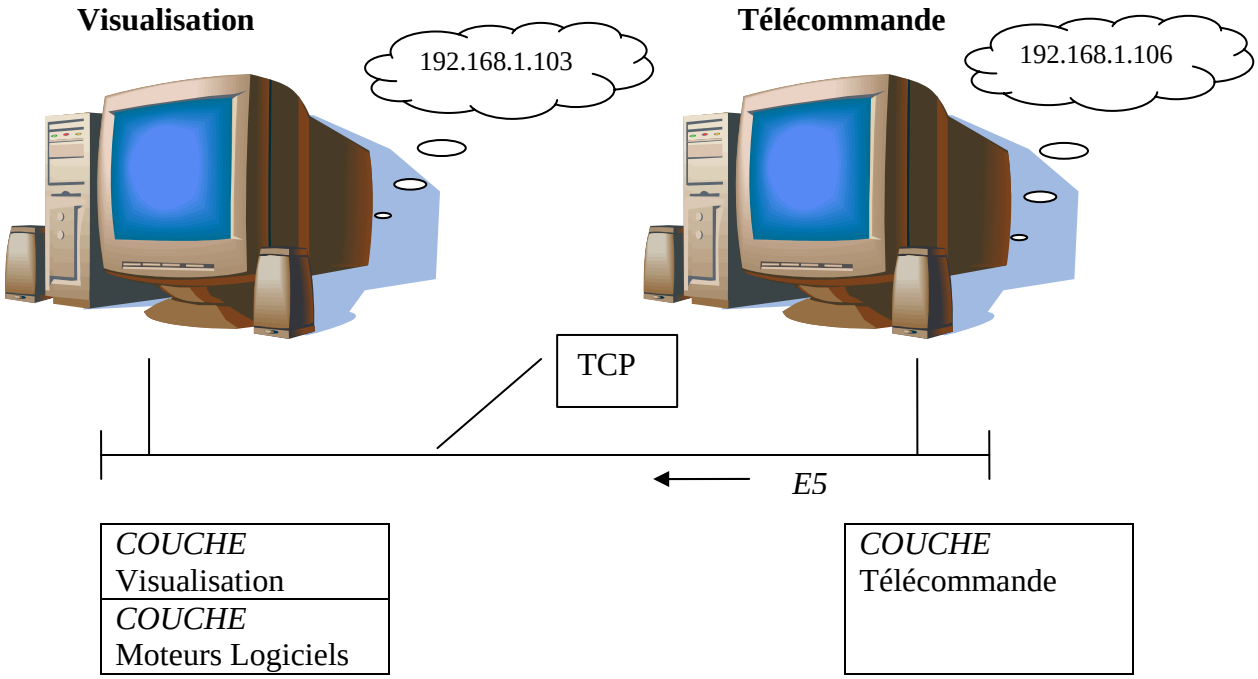


Fig. 1.1

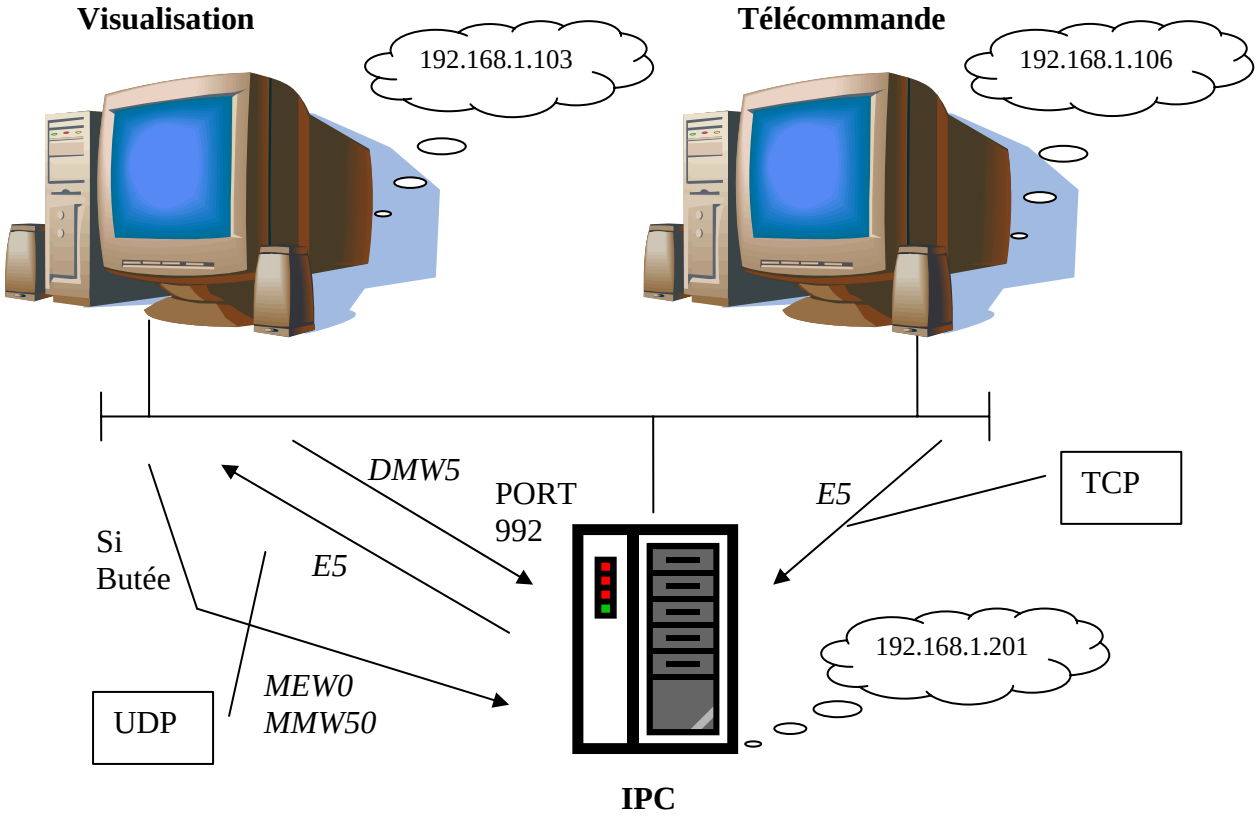
[(1)] FPIETTE : Collection des composants pour réaliser des connexions à travers un réseau.
[(2)] GLSCENE : Collection des composants pour créer un environnement 3D. ››› 1.3.2 p 8

L'ordinateur qui est responsable pour la visualisation attend le moment où l'autre ordinateur se connecte. Quand il y a une connexion la télécommande envoie un commandement pour faire bouger la simulation. L'ordinateur de visualisation traite l'ordre donné et fait bouger la visualisation.

Vous trouvez la liste des commandements expliquée dans le point 3.3 .2 de ce mémoire.

Visualisation du robot avec IPC et sans robot

Il y a toujours deux ordinateurs dans le réseau, mais maintenant aussi l'IPC est branché. L'ordinateur de visualisation demande des flag words [1] à l'IPC chaque 250 millisecondes et avec ces flag words la simulation est corrigée. La télécommande se connecte maintenant avec l'IPC et pas avec l'autre ordinateur et elle envoie des commandements. La télécommande n'est pas changée.



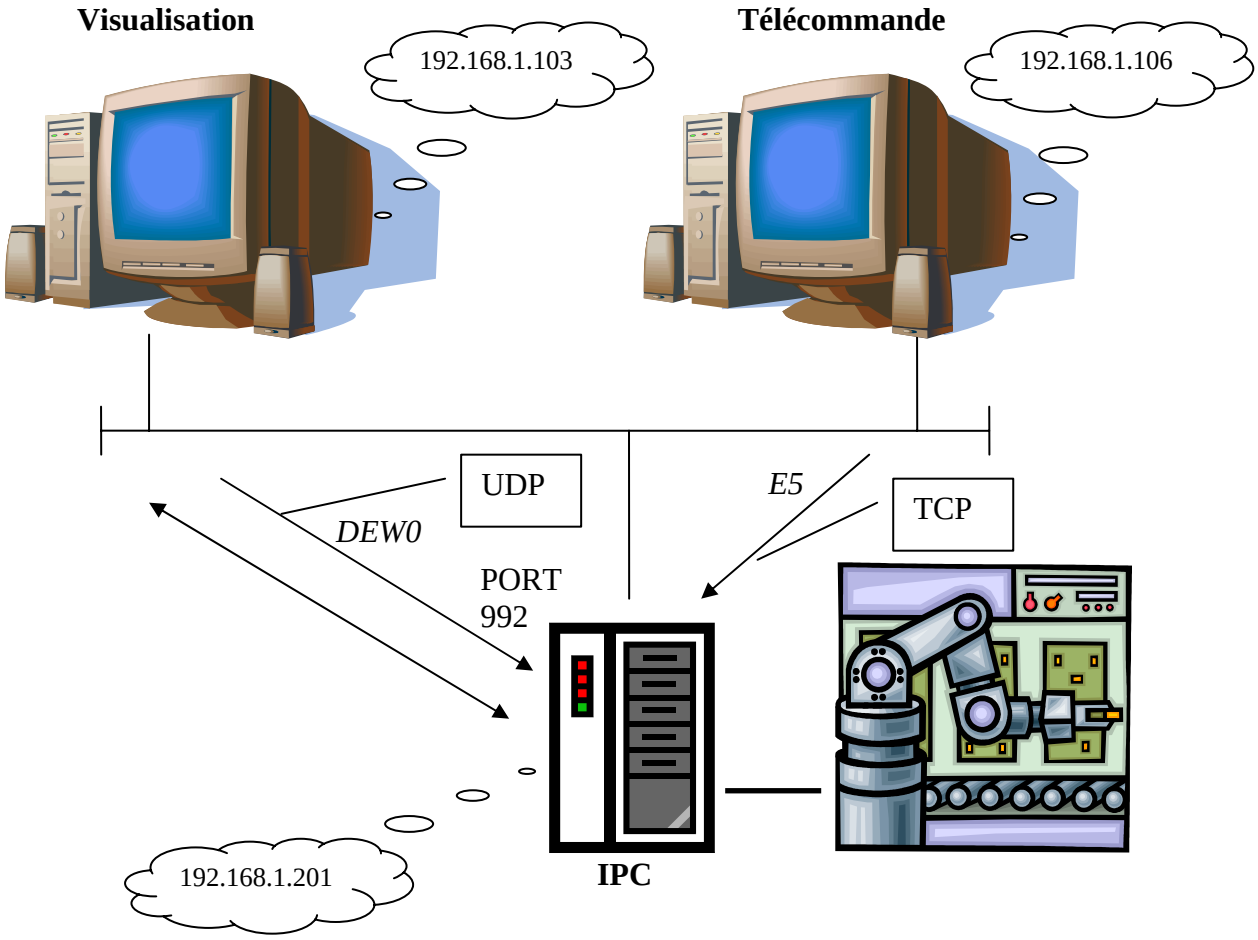
COUCHE Visualisation		FST RBTCPSIM	*COUCHE* Télécommande
Moteurs	Butées	Butées IW0	

Fig. 1.2

[1] FLAG WORD : Un mot dans l'IPC qui nous peut dire si un moteur est en train de bouger et si cela est le cas dans quelle direction.

Visualisation du robot avec IPC et avec robot

Maintenant sauf les deux ordinateurs et l'IPC le vrai robot sont branchés sur le réseau. Comme dans le cas précédent la télécommande envoie des commandements à l'IPC par TCP. L'ordinateur de la visualisation surveille les valeurs des butées qui sont enregistrées dans l'IPC. Le coude et le poignet du robot ont chacun un potentiomètre [1], dont les valeurs sont aussi enregistrées dans l'IPC. Avec les valeurs des butées et des potentiomètres la visualisation est synchronisée avec le robot en temps réel.



COUCHE Visualisation		FST RBTCPIO	*COUCHE* Télécommande
	Butée Logicielle	IW0	

Fig. 1.3

[1] POTENTIOMETRE : Un élément électronique qui peut nous donner la position exacte du moteur.

1.2 Matériel utilisé

1.2.1 Le Robot

1.2.1.1 Général

Le robot est en fait un bras mécanique constitué de 5 parties : le socle, l'épaule, le coude, le poignet et la pince. Le robot a 5 moteurs qui font bouger chacun une partie du robot, tournant autour d'une articulation. Le robot a aussi des capteurs pour déterminer la position des différentes parties, mais ils ne sont pas les mêmes pour chaque élément.

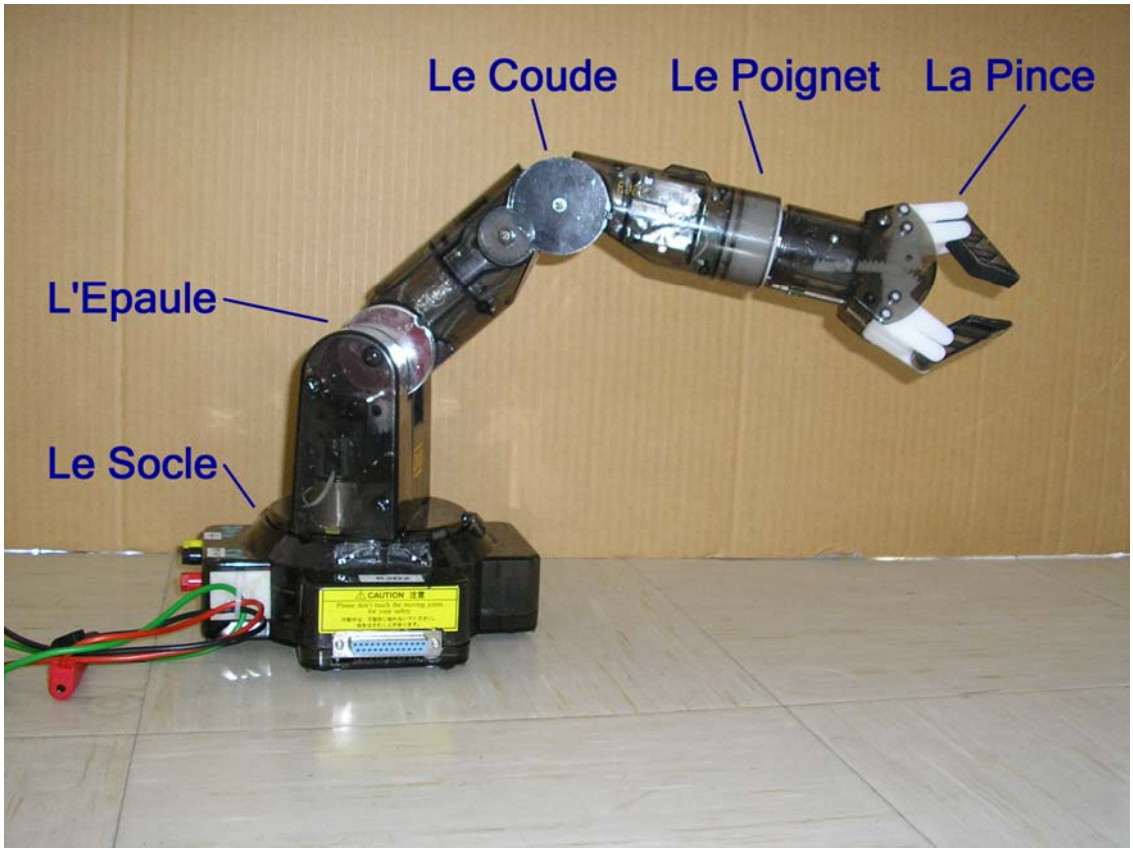


Fig. 1.4

1.2.1.2 Les moteurs

- **Le socle** fait tourner le robot entier autour d'un axe de trois cent cinquante (350) degrés. Il a trois capteurs pour déterminer sa position, à savoir si le socle préconise entièrement la droite ou entièrement la gauche et si le socle se trouve au milieu. Les capteurs sont des boutons qui sont pressés par le socle lui-même.
- **L'épaule** fait bouger verticalement tout ce qui se trouve au dessus du socle à un angle de cent dix (110) degrés. L'épaule a un capteur infrarouge qui compte les passages des dents d'une roue dentée et avec ces données on peut calculer la position de l'épaule.

- **Le coude** peut faire monter et faire descendre l'avant-bras ensemble avec le poignet et la pince. Le coude a un angle vertical avec un maximum de cent trente-cinq (135) degrés. Le coude a comme capteur un potentiomètre qui peut calculer l'angle du coude.
- **Le poignet** peut tourner, pas comme un poignet humain qui peut bouger verticalement. La rotation du poignet est limitée à trois cent quarante (340) degrés. Il a aussi un potentiomètre comme capteur.
- **La pince** a deux mouvements, soit fermer soit ouvrir. L'ouverture maximum de la pince est de cinquante (50) millimètre. La pince n'a pas de capteur, mais avec les signaux du moteur l'IPC peut savoir si la pince est ouverte où fermée.

1.2.2 L'IPC

1.2.2.1 Général

Grâce à la conception modulaire du FESTO IPC, des différents composants peuvent être convenus pour des domaines d'application très différents. La modularité du système garantit des conditions logistiques minimales parce que tous les systèmes emploient les mêmes blocs fonctionnels de base.

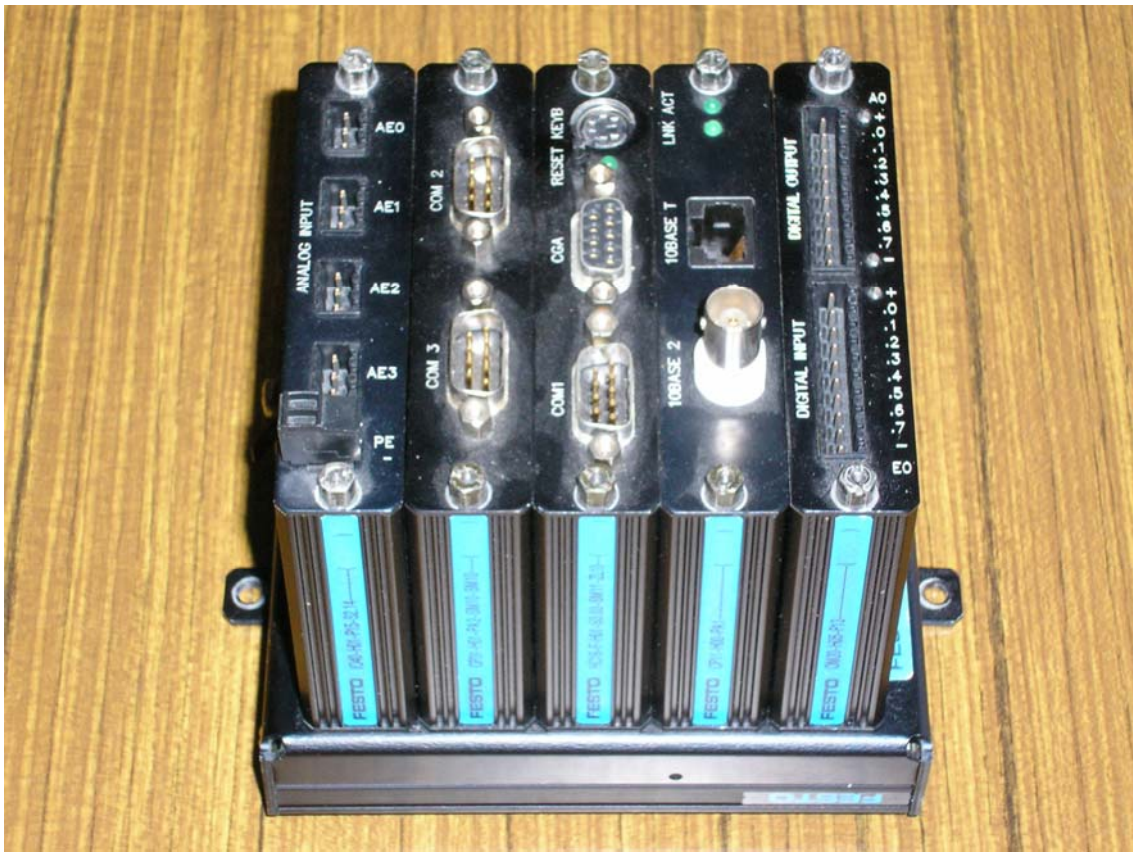


Fig. 1.5

Notre IPC consiste de cinq modules qui sont insérées dans la base (E.IPC-BP50). Le système d'exploitation qui est installé sur l'IPC est MSDOS 6.0, mais cela ne suffit par pour faire fonctionner des automates programmables industriels, donc l'IPC avait besoin d'un autre noyau qui est du temps réel au contraire de MSDOS qui n'est pas du temps réel → FSTPCR22.

1.2.2.2 Les modules

- **HC16-F-H01** : ce module est le seul obligatoire pour concevoir un IPC. On peut comparer se module avec la carte mère d'un ordinateur. Le HC16 comprend un CPU [(1)].
  Au niveau externe la carte a une entrée pour un clavier, une sortie pour un écran CGA [(2)] et une connexion sérielle pour la communication entre l'IPC et un ordinateur (par exemple pour échanger des fichiers entre l'IPC et un ordinateur).
- **CP31-H01-PA2-SM10** : ce module contient deux connexions sérielles pour la communication avec des autres ordinateurs.
- **CP11-H00-PA1** : cette carte contient une partie Ethernet qui permet d'avoir une communication TCP où UDP. Il y a un liaison UDP [(3)] et COAX [(4)]
- **OM20-H05** : avec l'OM20 c'est possible d'échanger des informations avec une machine automatisée, dans notre cas naturellement le Robot. La carte contient des sorties et des entrées. Les sorties sont utilisées quand l'IPC donne une commande pour bouger au robot et dans les entrées les capteurs du robot écrivent leurs valeurs.
- **IO40-H01** : ceci est la carte des entrées analogiques. Les valeurs des entrées analogiques doivent être compris entre zéro et dix volts et l'IPC peut comprendre ces tensions comme des valeurs logiques, respectivement entre zéro et 4095.

1.3 Logiciel utilisé

1.3.1 Festo 4.02

La partie logicielle qui entretient l'interaction avec l'IPC. Il y a la possibilité d'écrire des programmes dans le langage de FST4 et de les charger dans l'IPC. Les programmes chargés servent à l'automatisation des matériaux branchés à l'IPC.

[(1)] CPU : Central Processing Unit.
[(2)] CGA : Color Graphics Adapter.
[(3)] UDP : User Datagram Protocol, un protocole qui est utilisé sûr l'Internet à côté de TCP.
[(4)] COAX : Un câble de réseau qui consiste d'un fil clé et une couverture conductible. Le COAX est maintenant souvent utilisé pour la connexion de télévision.

Le Festo est fourni d'un amplement caractère fonctionnel, comme l'outil « Online Display » qu'on peut utiliser pour visualiser les entrées, les sorties, les Flag Words, et cetera.

L'IPC est un système modulaire et il y a plusieurs de modules de sortie et d'entrée disponible. Normalement ils se sont introduits par vos programmes de contrôle à travers des variables de sortie et d'entrée du FST. Si on veut utiliser un des modules IO dans un projet c'est nécessaire de les charger et configurer dans « l' IO Configuration ». Dans notre cas nous devions charger les IO modules « OM20 » et « IO40 ».
Si on veut faire une connexion avec un réseau on a besoin d'un TCP/IP driver et on doit l'insérer dans « le Driver Configuration ».

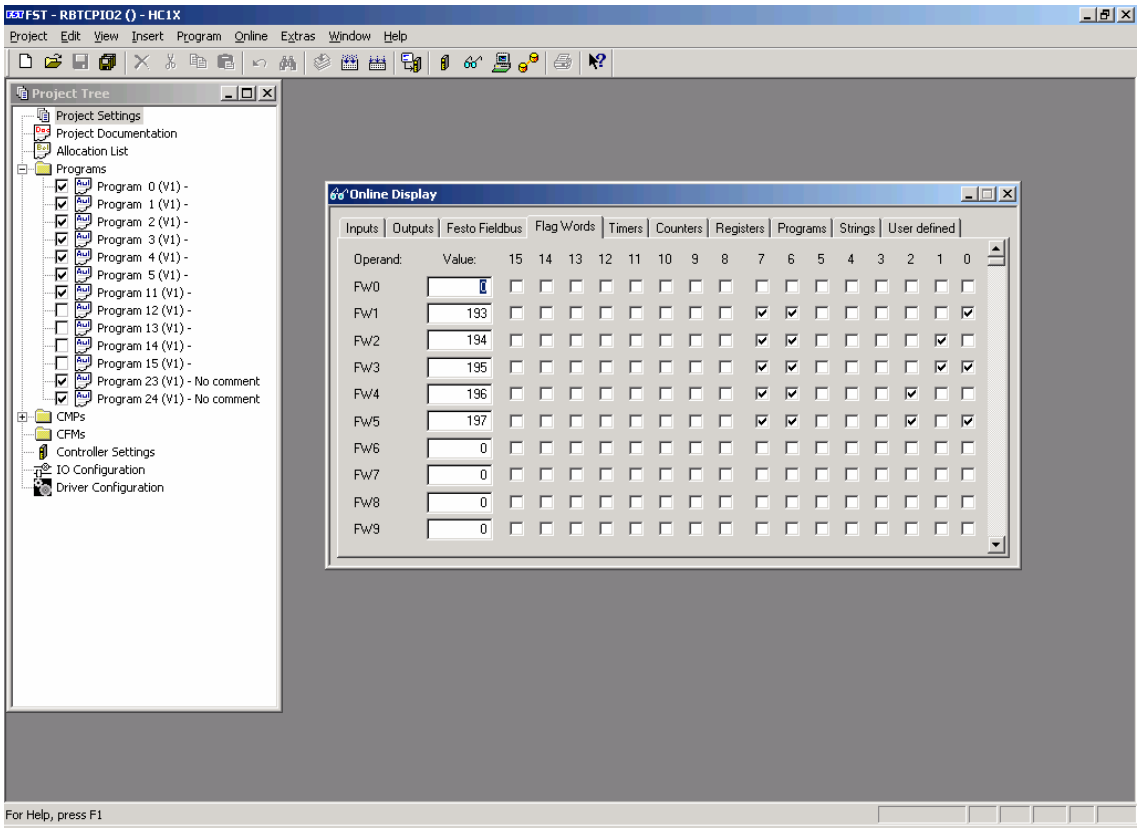


Fig. 1.6

1.3.2 Borland Delphi 7.0 et composants

Delphi 7.0 de Borland est un environnent de travail complet. Il nous donne la possibilité de créer un projet dons sa totalité. Des outils graphiques sont prévus pour réaliser le GUI[(1)]. Le Delphi nous offre des moyens pour développer une application orientée objet. C'est possible de coder, compiler et déboguer de façon professionnelle.

[(1)] GUI : Graphical User Interface

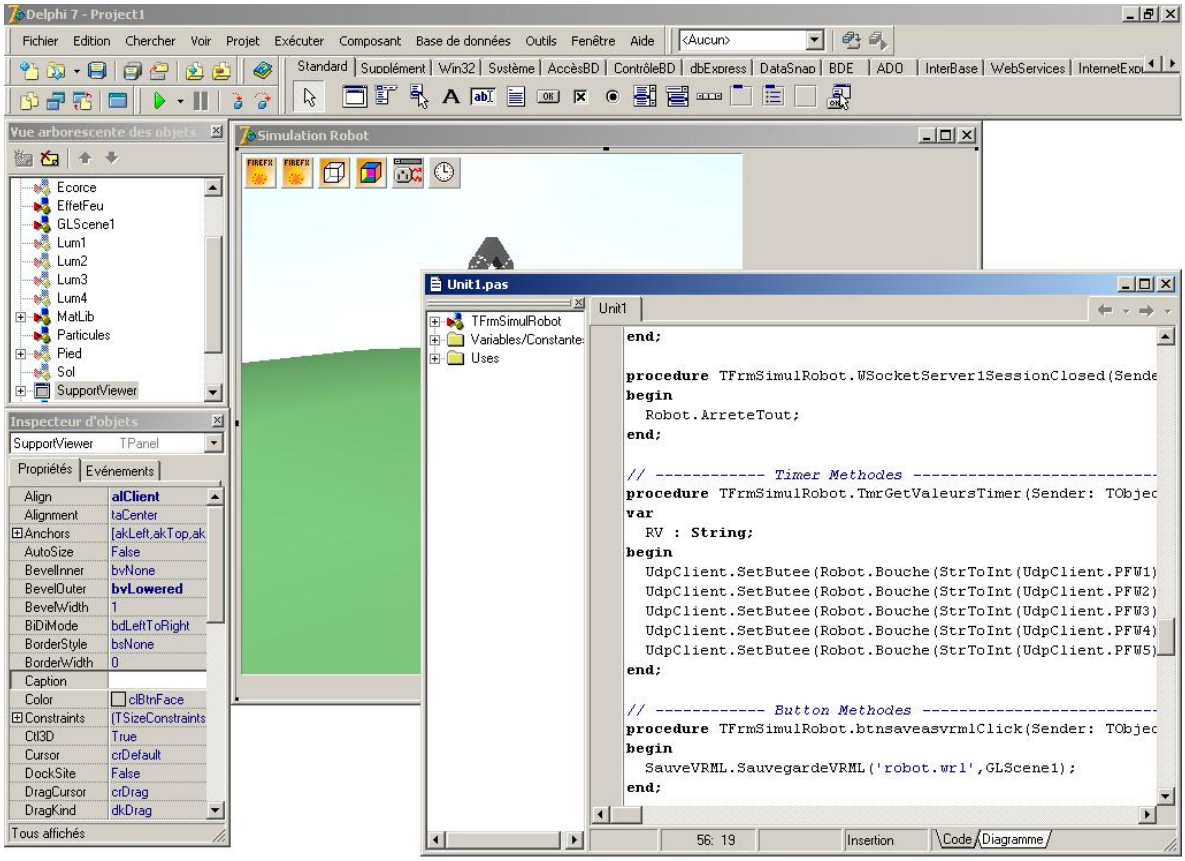


Fig. 1.7

GLScene Snapshot

Le GLScene est une bibliothèque 3D fondée sur OpenGL[1] créé pour Delphi. Il nous donne des composants et objet visuel qui nous permettent de décrire et de traduire facilement des scènes en 3D.

Tout d'abord on doit faire l'extraction du fichier `GLScene_SnapShot_150305.zip` dans un dossier. Après, en Borland Delphi, intercale tous les dossiers qui se trouvent dans le dossier `\source` à la bibliothèque qu'on peut trouver dans « les option d'environnement » au-dessous du menu Outils. Ensuite on ouvre le fichier `/Delphi7/GLScene7.dpk` on le compile et après on doit l'installer. C'est possible qu'on ait encore besoin d'un fichier da la dernière version. Quand cela arrive on doit en plus intercaler `\source\plugIn` à la bibliothèque.

FPiette

Les composants de FPiette nous donnent la possibilité de faire des connexions UDP et TCP sur un réseau.

Même manipulation que précédemment sauf que le nom du paquet à parcourir est `ics.zip`.

---

[1] OpenGL : Le premier environnement pour développer 2D et 3D applications portables et interactives.

1.3.3 Installation des logiciels nécessaires

1.3.3.1 Apache Serveur 2.0.53

Le projet d'Apache est un effort de collaboration de développement de logiciel visant à créer une exécution robuste, de catégorie courante, de promesse, et libre disponibilité de code source d'un serveur de HTTP (Web). Le projet est conjointement contrôlé par un groupe de volontaires situés autour du monde, en utilisant l'Internet et le Web pour communiquer, projeter, et développer le serveur et sa documentation relative. Ces volontaires sont connus en tant que groupe d'Apache. En outre, les centaines d'utilisateurs ont contribué aux idées, du code, et de la documentation au projet.

La première partie dont tu as besoin quand tu veux construire un site web est un serveur. C'est possible de louer un serveur chez une firme spécialisée, mais tu peux aussi en installer un sur ton ordinateur. Le site web que nous avons construit contient du code PHP, donc nous avions besoin d'un serveur capable de comprendre le code et de transformer ce code en HTML. Apres la transformation le serveur envoie la page HTML vers le client qui est en train de visiter notre site web. Le serveur adéquat est l'APACHE, de plus il est gratuit.

L'installation en elle même est très facile, il suffit de suivre les étapes. Si le sorcier te demande où tu veux installer le serveur tu remplis `C:\webserver\`. L'Apache commence maintenant avec la vraie installation. Si l'installation est finie c'est nécessaire de la tester. Ouvre un browser et remplis l'adresse `http://127.0.0.1 ou http://localhost`. Quand l'installation a été un succès tu obtiens une indication positive.

Maintenant nous devons changer quelques choses dans la configuration de l'Apache. Ouvre le fichier `httpd.conf` et vais vers ligne 228. Là-bas tu cherche la ligne `DocumentRoot "C:/webserver/Apache2/htdocs"` et tu la changes avec `DocumentRoot "C:/webserver/www"`. Apres tu vas vers ligne 253 et tu changes `<Directory "C:/webserver/Apache2/htdocs">` dans `<Directory "C:/webserver/www">`.

Le serveur Apache sait maintenant que votre dossier source est `c:/webserver/www/`, mais c'est encore nécessaire de le créer. Dans ce dossier tous les fichiers PHP et HTML seraient sauvegardés.

Le résultat final

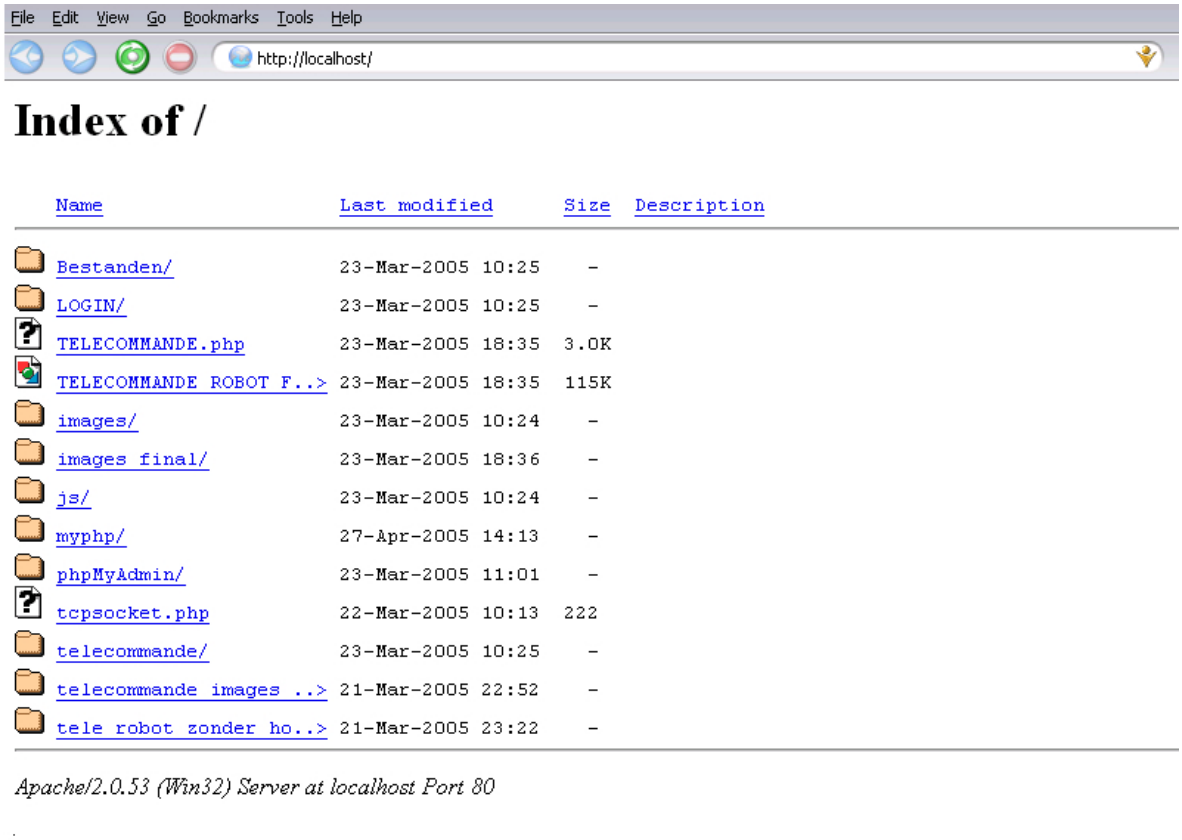


Fig. 1.8

1.3.3.2 PHP 4.3.10

Si les démarches à suivre sont bien respectées, l'Apache est maintenant installé et configuré sur ton ordinateur, c'est déjà un vrai serveur web. Ton serveur Apache n'est pas encore capable d'interpréter du code PHP, donc c'est nécessaire d'installer du PHP.

Tu dois faire l'extraction du fichier `php-4.3.10-Win32.zip` vers `C:\webserver\php\`. Ouvre ce dossier et cherche pour le fichier `php.ini-dist` et débaptise le vers `php.ini`. Ce fichier tu dois transférer vers le dossier « Windows ». Si cela est fait tu transfères les deux fichiers `Msvcrt.dll` et `php4ts.dll` vers `C:\Windows\System\`.

La seule chose qui nous reste à faire est de modifier quelques lignes de configurations dans le serveur Apache. Ouvre encore une fois le fichier `httpd.conf` et vais vers ligne 320. Là-bas tu trouveras le texte suivant `DirectoryIndex index.html index.html.var` et change cela vers `DirectoryIndex index.html index.php`. Vas vers la ligne dernière et annexe du texte.

```
ScriptAlias /php/ "c:/Webserver/php/"
AddType application/x-httpd-php .php
Action application/x-httpd-php "/php/php.exe"
```

Pour tester si le serveur et le PHP sont installés d'une façon correcte tu peux écrire une petite page PHP dans un éditeur texte.

```
<?php
echo "Hello world!";
?>
```

Sauvegarde le fichier comme `test.php` et place le dans le dossier `C:/webserver/www/`

Le résultat final

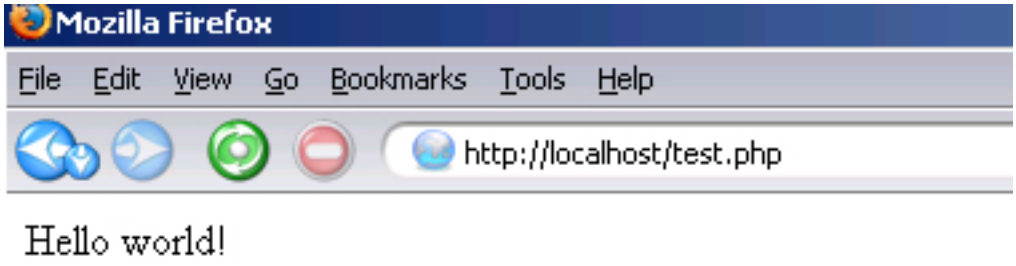


Fig. 1.9

1.3.4 Borland Together 6.2

Together Control Center contient les possibilités dont tu as besoin pour manufacturer des analyses professionnelles. Avec tous les instruments qui sont à notre disposition, on peut rédiger des diagrammes de classe, des diagramme de succession, et cetera. Le programme nous donne une bonne vue général et rend clair l'interaction entre les différents diagrammes.

1.3.5 Adobe Photoshop 7.0

C'est certainement l'un des meilleur logiciels pour éditer des photos. Un grand éventail de possibilités est à votre disposition. Créer des ombres, masquer des couleurs, dessiner des illusions 3D et travailler avec plusieurs couches sont seulement quelques exemples de la fonctionnalité de Photoshop. Nous avons utilisé Photoshop pour notre télécommande du site web.

Partie programmation en Festo 4.02 2

2.1 L'application d'initialisation

Application zéro [P0] est pour l'initialisation des applications de gestion. Donc elle lance programme [P1], [P2], [P3], [P4], [P5].

```
STEP INIT
 IF                  NOP
 THEN  SET           P1      // P1 est pour la gestion du moteur
Pince
       LOAD          V$C1
         TO          FWPince
       RESET         T31
```

2.2 Les applications de gestion

Pour chaque moteur il y a une application pour la gestion.
[P1] La Pince - [P2] Le Poignet - [P3] Le Coude - [P4] L'Epaule - [P5] Le Socle

Cette méthode doit toujours être répétée. Si FWCoude est différent d'IMCoude on doit récupérer la valeur V$80 (hexadécimal 80 – binaire 1000 0000) dans la variable « sortie » et mettre le chronométreur du programme 3 sur 0.1 secondes. Après on récupère la valeur de FWPince dans moteur (OW0) aussi dans IMPince.

```
STEP E1
 IF                  IMCoude
        <>           FWCoude
 THEN
       LOAD          V$80
         TO          sortie  //OW0 = output word 0
       SET           T3
         WITH        0.1s
STEP E11
 IF            N     T3
 THEN  LOAD          FWCoude
         TO          Sortie
         TO          IMCoude
       SET           X31
       SET           T3
         WITH        0.1s
```

2.3 Le fonctionnement du moteur pince

Ce programme est pour savoir si la pince est ouverte ou fermée. Quand le moteur commence à bouger il génère une pulsation. Si la pince est tout ouverte ou fermée le moteur est bloqué et il va donner une deuxième pulsation. Avec les deux pulsations nous pouvons découvrir l'état de la pince.

```
STEP INIT
 IF                  NOP
 THEN  RESET         T11

STEP E2
 IF                  SPince
 THEN  SET           T11
         WITH        3s
STEP OU
 IF            N     T11
 THEN  JMP TO fin
 IF            N     Spince
 THEN  LOAD          V0
         TO          ButeeP
STEP E3
 IF                  Spince
 THEN  LOAD          FWPince
       AND           V$30
         TO          ButeeP

STEP fin
 IF                  NOP
 THEN  LOAD          V$C1
         TO          FWPince
```

2.4 Le traitement de la trame IP

Ce logiciel met la trame IP reçue dans le flagword correct. Il interprète la trame et il la place dans le flagword correspondant au moteur.

```
STEP choix
 IF                  moteur
         =           V1
 THEN  LOAD          recu
         TO          FWPince
 IF                  moteur
         =           V2
 THEN  LOAD          recu
         TO          fwPoigne
```

2.5 Les entrées, les sorties et les flagwords

Les entrées [inputs]

Les entrées sont des valeurs qui l'IPC reçoit du robot. Le socle a trois butées : droite, milieu et gauche, mais nous ne l'utilisons pas. L'épaule a aussi des butées, à savoir haute et basse. Ces cinq butées sont modifiées dans l'entrée zéro [IW0].

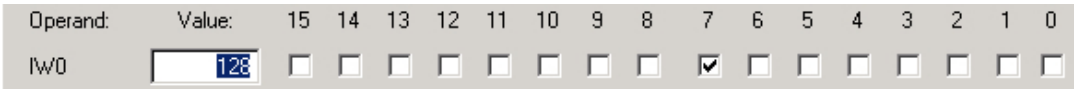


Fig. 2.1

MOTEUR	DIRECTION	BINAIR	HEXA	DECIMAL
Socle	Droite	1000 0000	80	128
	Milieu	0100 0000	40	64
	Gauche	0010 0000	20	32
Epaule	Haute	0001 0000	10	16
	Basse	0000 1000	8	8

Fig. 2.2

Des entrées douze [IW12] et treize [IW13] contiennent les valeurs des potentiomètres du coude et du poignet. Un potentiomètre est une résistance variable, donc quand le coude ou la poignet changent leurs positions la valeur du potentiomètre va être modifiée.

Fig. 2.3

Les sorties [outputs]

Pour faire bouges le robot on doit écrire dans des sorties. La valeur qu'on écrit dans sorties zéro [OW0] va déterminer quel moteur bougera dans quelle direction.

Fig. 2.4

MOTEUR	DIRECTION	BINAIR	HEXA	DECIMAL
Pince	Fermer	1101 0001	D1	209
	Arrêter	1100 0001	C1	193
	Ouvrir	1110 0001	E1	225
Poignet	Gauche	1101 0010	D2	210
	Arrêter	1100 0010	C2	194
	Droite	1110 0010	E2	226
Coude	Lever	1110 0011	E3	227
	Arrêter	1100 0011	C3	195
	Baisser	1101 0011	D3	210
Epaule	Lever	1101 0100	D4	211
	Arrêter	1100 0100	C4	196
	Baisser	1110 0100	E4	228
Socle	Gauche	1101 0101	D5	212
	Arrêter	1100 0101	C5	197
	Droite	1110 0101	E5	229

Fig. 2.5

Les flagwords

Des flagwords sont des références aux des sorties. Chaque moteur a reçu un flagword. Maintenant on peut faire bouger le robot d'une autre manière, c'est-à-dire par changer des flagwords.

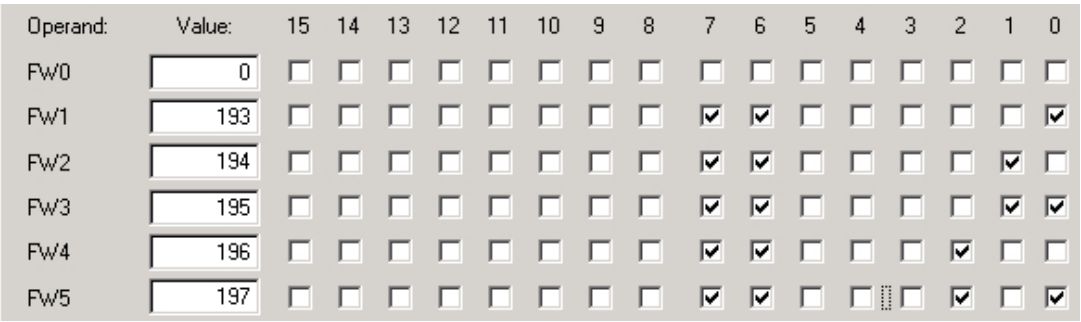


Fig. 2.6

Moteur	Pince	Poignet	Coude	Epaule	Socle
FlagWord	FW1	FW2	FW3	FW4	FW5

Fig. 2.7

Le flagword 50 [FW50] contient les butées de la pince et du poignet.

Fig. 2.8

MOTEUR	BUTEE	BINAIR	HEXA	DECIMAL
Poignet	Droite	1000 0000	80	128
	Gauche	0100 0000	40	64
Pince	Ouvert	0010 0000	20	32
	Fermé	0001 0000	10	16

Fig. 2.9

Partie programmation en Delphi 7 3

3.1 L'analyse

L'analyse se fait en plusieurs étapes. Le plus important est de comprendre la problématique du projet et d'obtenir une image complète de ce que le commettant veut. La deuxième facette est de faciliter l'entretien du logiciel. La troisième raison de faire une analyse est que votre logiciel serait plus de bonne qualité.

3.1.1 Use Cases

Un Use Case donne une récapitulation générale des différents processus et de la fonctionnalité du système. Il décrit l'attitude du système du point de vue du monde extérieur. Nous pourrions dire que le contexte du système est décrit. Le monde extérieur peut se résumer en des utilisateurs qui communiquaient avec le système, mais aussi d'autres systèmes, des appareils ou des périphériques qui sont branchés au système.

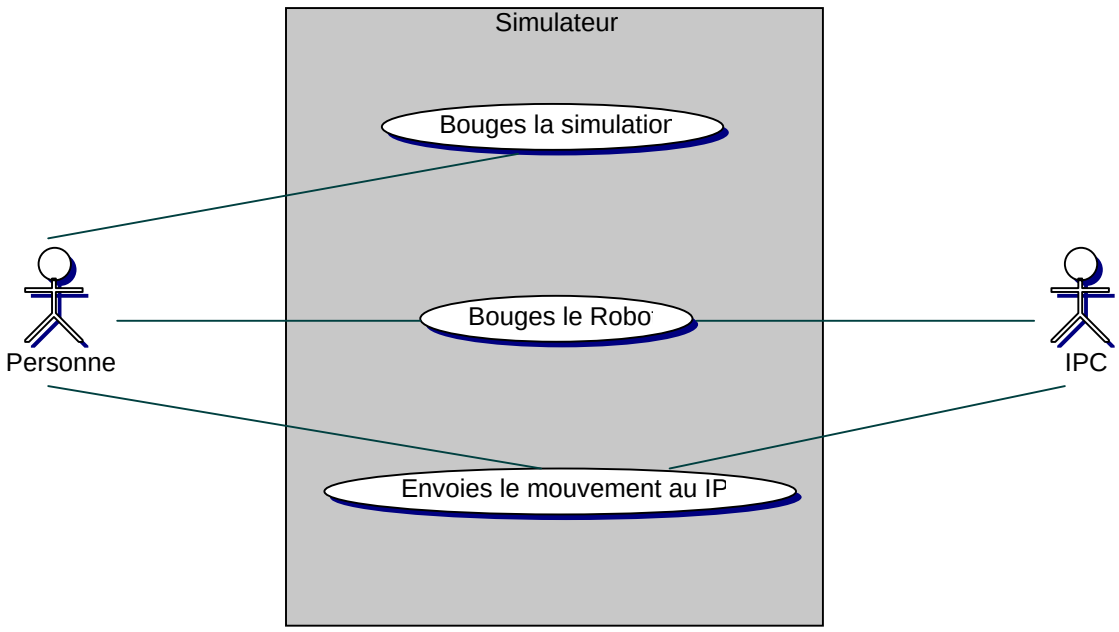


Fig. 3.1

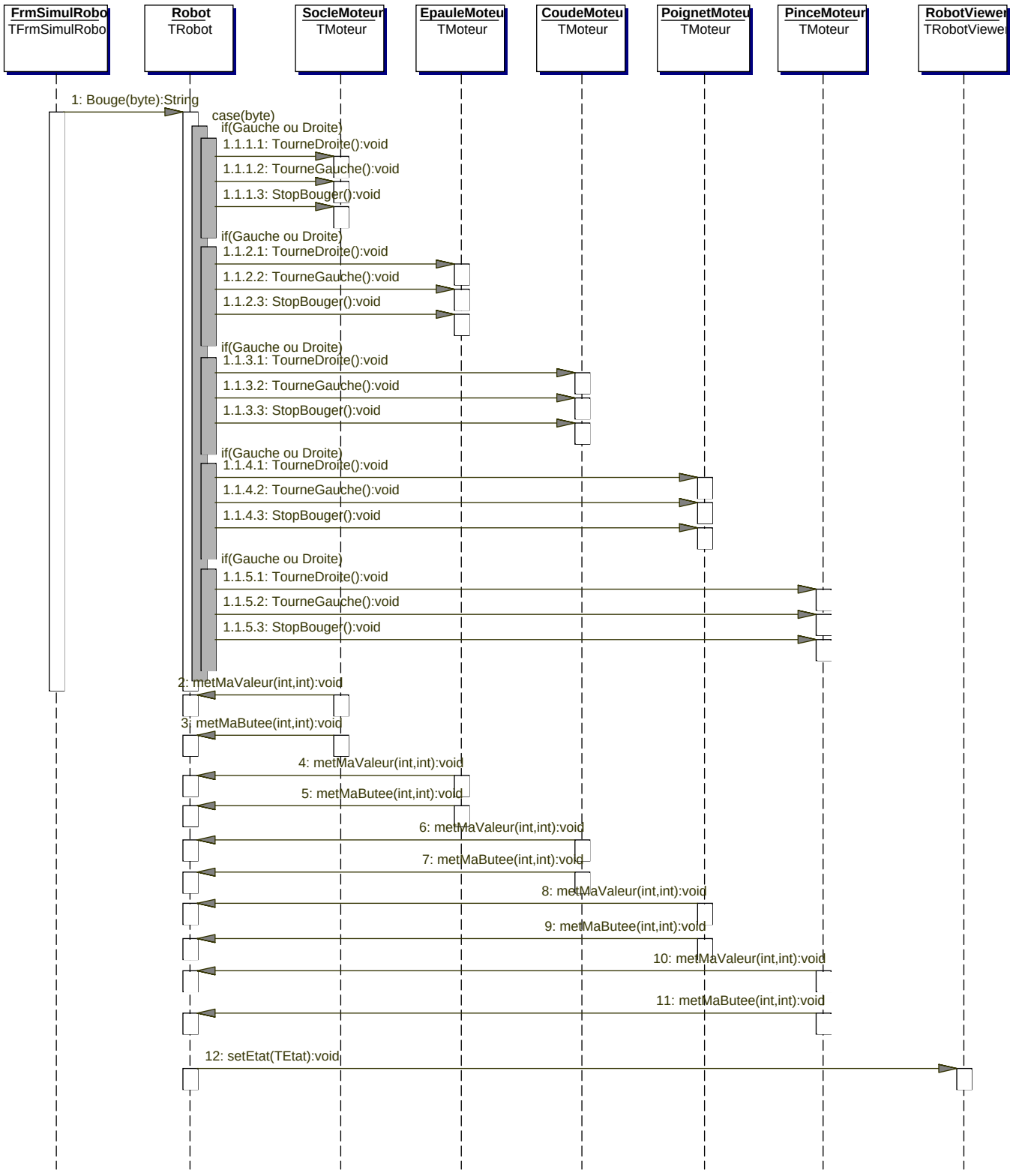

FrmSimulRobo
TFrmSimulRobo
Robot
TRobot
SocleMoteur
TMoteur
EpauleMoteu
TMoteur
CoudeMoteu
TMoteur
PoignetMoteu
TMoteur
PinceMoteur
TMoteur
RobotViewe
TRobotViewe
1: Bouge(byte):String
case(byte)
if(Gauche ou Droite)
1.1.1.1: TourneDroite():void
1.1.1.2: TourneGauche():void
1.1.1.3: StopBouger():void
if(Gauche ou Droite)
1.1.2.1: TourneDroite():void
1.1.2.2: TourneGauche():void
1.1.2.3: StopBouger():void
if(Gauche ou Droite)
1.1.3.1: TourneDroite():void
1.1.3.2: TourneGauche():void
1.1.3.3: StopBouger():void
if(Gauche ou Droite)
1.1.4.1: TourneDroite():void
1.1.4.2: TourneGauche():void
1.1.4.3: StopBouger():void
if(Gauche ou Droite)
1.1.5.1: TourneDroite():void
1.1.5.2: TourneGauche():void
1.1.5.3: StopBouger():void
2: metMaValeur(int,int):void
3: metMaButee(int,int):void
4: metMaValeur(int,int):void
5: metMaButee(int,int):void
6: metMaValeur(int,int):void
7: metMaButee(int,int):void
8: metMaValeur(int,int):void
9: metMaButee(int,int):void
10: metMaValeur(int,int):void
11: metMaButee(int,int):void
12: setEtat(TEtat):void

Fig. 3.2

- **Bouges la simulation** : Une personne a la possibilité de faire bouger la simulation visuel.
- **Bouges le Robot** : Une personne a la possibilité de faire bouger le vrai Robot.
- **Envoies le mouvement au IPC** : Une personne a la possibilité d'envoyer un mouvement désiré au IPC.

3.1.2 Sequence Diagrams

Un « Sequence Diagram » est rédigé parce que dans une application orientée objet il peut arriver qu'au bout du temps on commence à perdre la clarté et ordonné des objets et leurs interaction. La possibilité est réelle qu'on commence à faire des travails superflus.

Un « Sequence Diagram » est une représentation schématique où on place les objets vis-à-vis du temps. Chacun à son propre ligne de vie qui est dessinée comme pointillé jusqu'au-dessous l'objet. Un message d'un objet à l'autre peut être dessiné comme une flèche entre deux lignes de vie. L'envoi d'un message en OO est accompli par un appel d'une méthode. En plus on entretient un arrangement de ces messages qui est présenté par un numérotage devant les noms des méthodes. Ensuite il y a la possibilité de spécifier des structures logiques comme « if, for, switch, et cetera ». Ce qui est accomplit par un cadre gris avec au-dessus le nom de la structure.

3.1.2.1 Le diagramme de la simulation

Tout commence avec l'envoi d'un commandement par la télécommande vers le serveur avec TCP. Le serveur reçoit ce commandement et le renvoie au `FrmSimulRobot` qui appelle la méthode `bouge(byte)` du robot pour lui dire de faire bouger un de ces moteurs à une direction déterminée par le Flag Word. Le robot peut savoir de quel moteur et de quelle direction il s'agît et comme ça il peut faire tourner le propre moteur dans la direction correcte. Cela est fait avec les méthodes `TourneDroite()` et `TourneGauche()`. Après un mouvement d'un moteur, le moteur renvoie une valeur, pour indiquer s'il y a une butée, et sa nouvelle position. Avec ces valeurs le robot met son état à jour et il envoie le nouvel état au `RobotViewer`. Le `RobotViewer` adapte avec le nouvel état la visualisation en 3D.

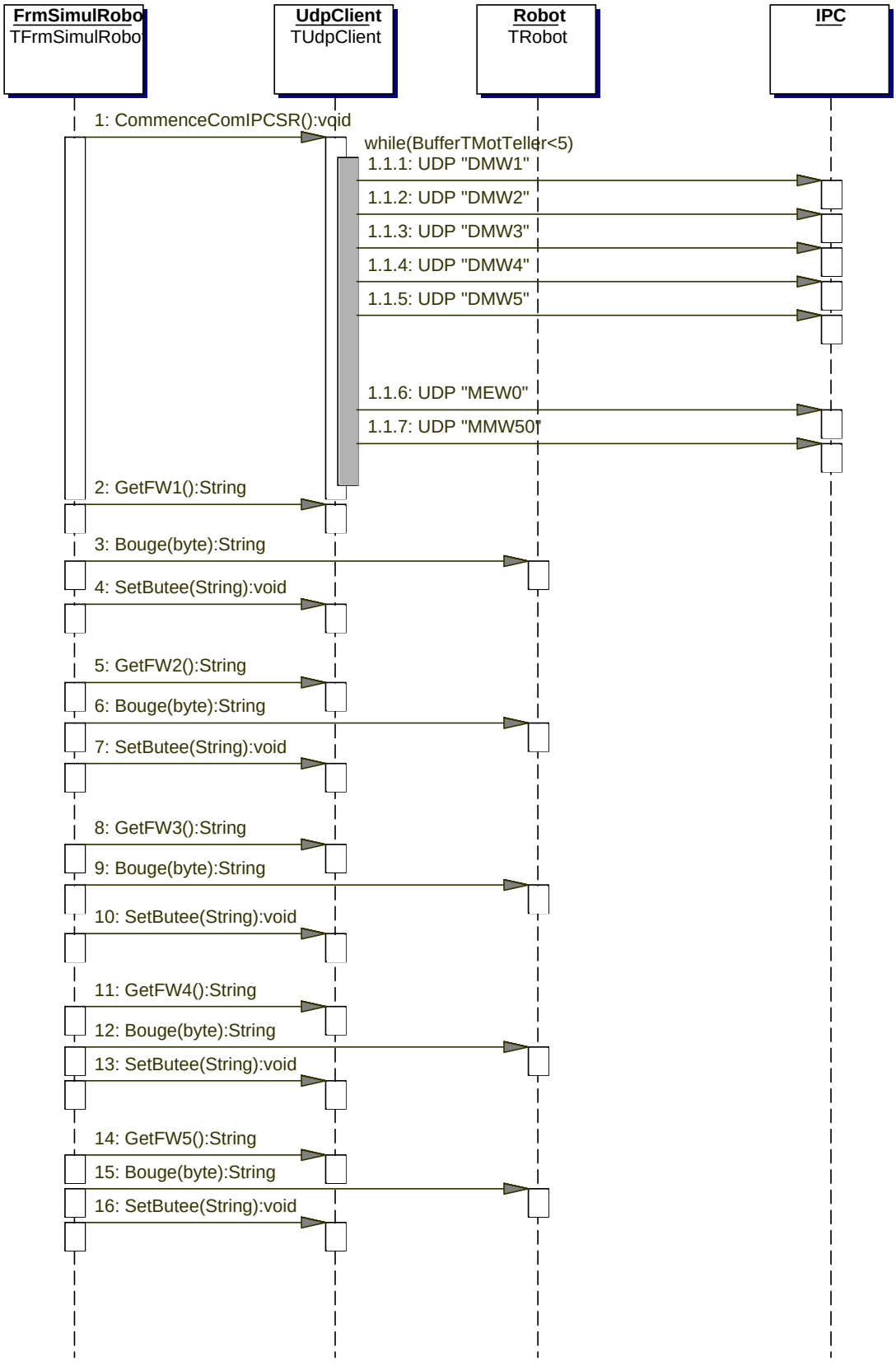

FrmSimulRobo
TFrmSimulRobo
UdpClient
TUdpClient
Robot
TRobot
IPC
1: CommenceComIPCSR():void
while(BufferTMotTeller<5)
1.1.1: UDP "DMW1"
1.1.2: UDP "DMW2"
1.1.3: UDP "DMW3"
1.1.4: UDP "DMW4"
1.1.5: UDP "DMW5"
1.1.6: UDP "MEW0"
1.1.7: UDP "MMW50"
2: GetFW1():String
3: Bouge(byte):String
4: SetButee(String):void
5: GetFW2():String
6: Bouge(byte):String
7: SetButee(String):void
8: GetFW3():String
9: Bouge(byte):String
10: SetButee(String):void
11: GetFW4():String
12: Bouge(byte):String
13: SetButee(String):void
14: GetFW5():String
15: Bouge(byte):String
16: SetButee(String):void

Fig. 3.3

3.1.2.2 Le diagramme du traitement des entrées

Le but d'ici est de simuler le robot avec les valeurs dans l'IPC, mises par la télécommande. L'Objet `UdpClient` demande les valeurs des Flag Words à L'IPC. Il fait ça en envoyant ces mots « DMW1 », « DMW2 », « DMW3 », « DMW4 » et « DMW5 ». Dans ces Flag Words se trouve la valeur pour indiquer quel mouvement un moteur conditionné est en train de faire. Puisque `l'UdpClient` sait quand un moteur a une butée, ensuite il indique qu'il y a des butées dans l'IPC avec les mots « MEW0 », « MMW50 ». Chaque unité de temps l'objet `FrmSimulRobot` demande toutes les valeurs des Flag Words à `l'UdpClient`. Avec ces valeurs `FrmSimulRobot` dit au robot de bouger ces moteurs dans la direction correcte. Le robot renvoie une valeur avec `FrmSimulRobot` et peut se distraire s'il y a une butée d'un moteur. L'objet `FrmSimulRobot` envoie lui-même cette valeur au `UdpClient` de façon à ce que `l'UdpClient` peut savoir quels moteurs ont une butée.

3.1.2.3 Le diagramme du mouvement du robot

Si le robot est branché et la télécommande envoie les commandements à l'IPC c'est possible de demander les valeurs des butées et les valeurs des potentiomètres. `L'UdpClient` commence par demander les valeurs de « DEW0 », « DMW50 », »DEW12 », « DEW13 ». Dans « DEW0 » et dans « DMW50 » se trouvent les valeurs des butées, et « DEW12 » et « DEW13 » contiennent les valeurs des potentiomètres. Après que `l'UpdClient` a reçu ces valeurs il les donne au `FrmPosEcran` pour la visualisation. Chaque unité de temps l'objet `FrmSimulRobot` demande les valeurs au `UdpClient` et avec ces valeurs il modifie la visualisation en 3D.

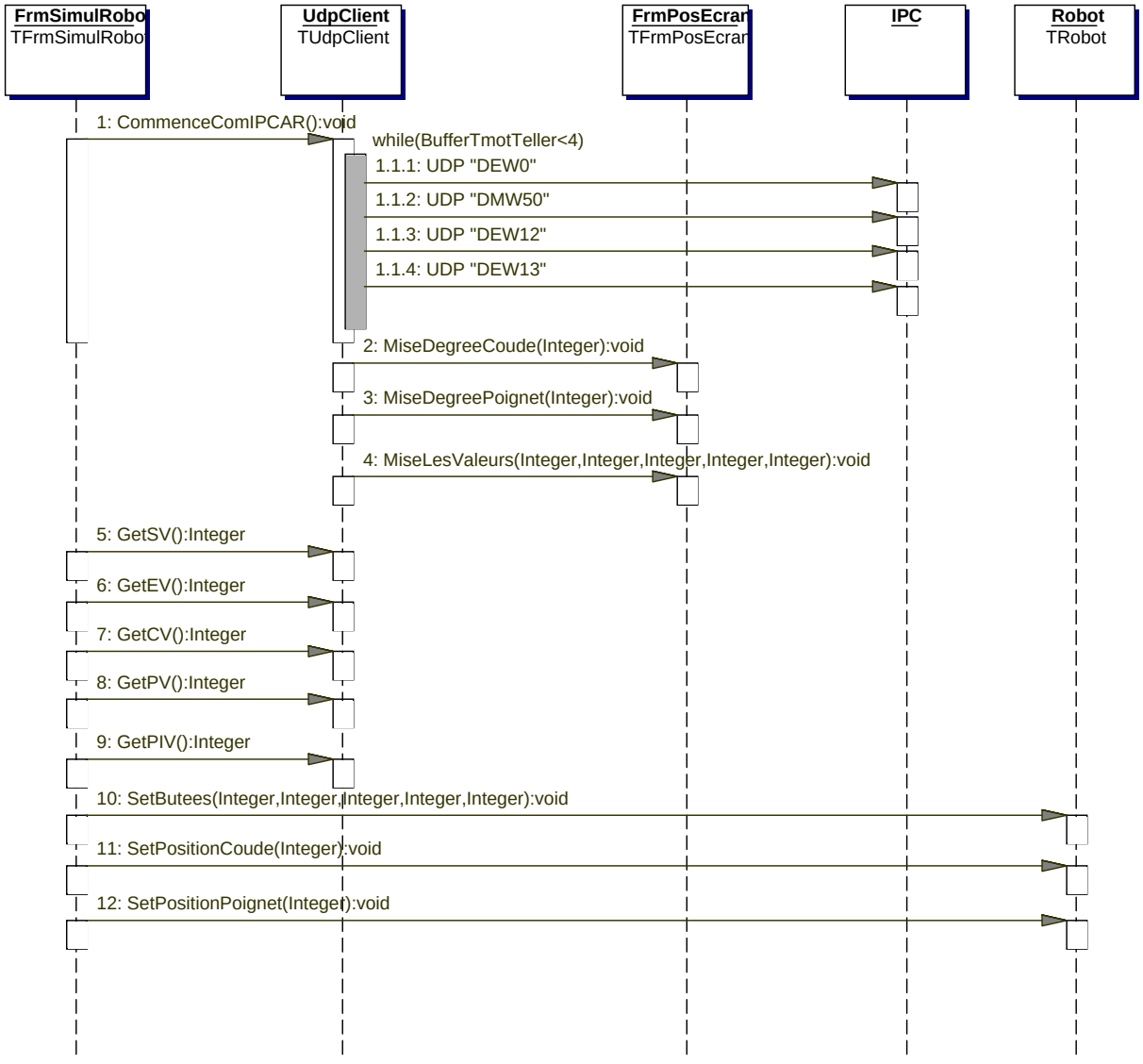


Fig. 3.4

3.1.3 Class Diagrams

Un « Class Diagram » est une collection des éléments statiques comme des classes, des interfaces et leurs liaisons, qui sont reliés à les un les autres.

3.1.3.1 Le diagramme

Domaine

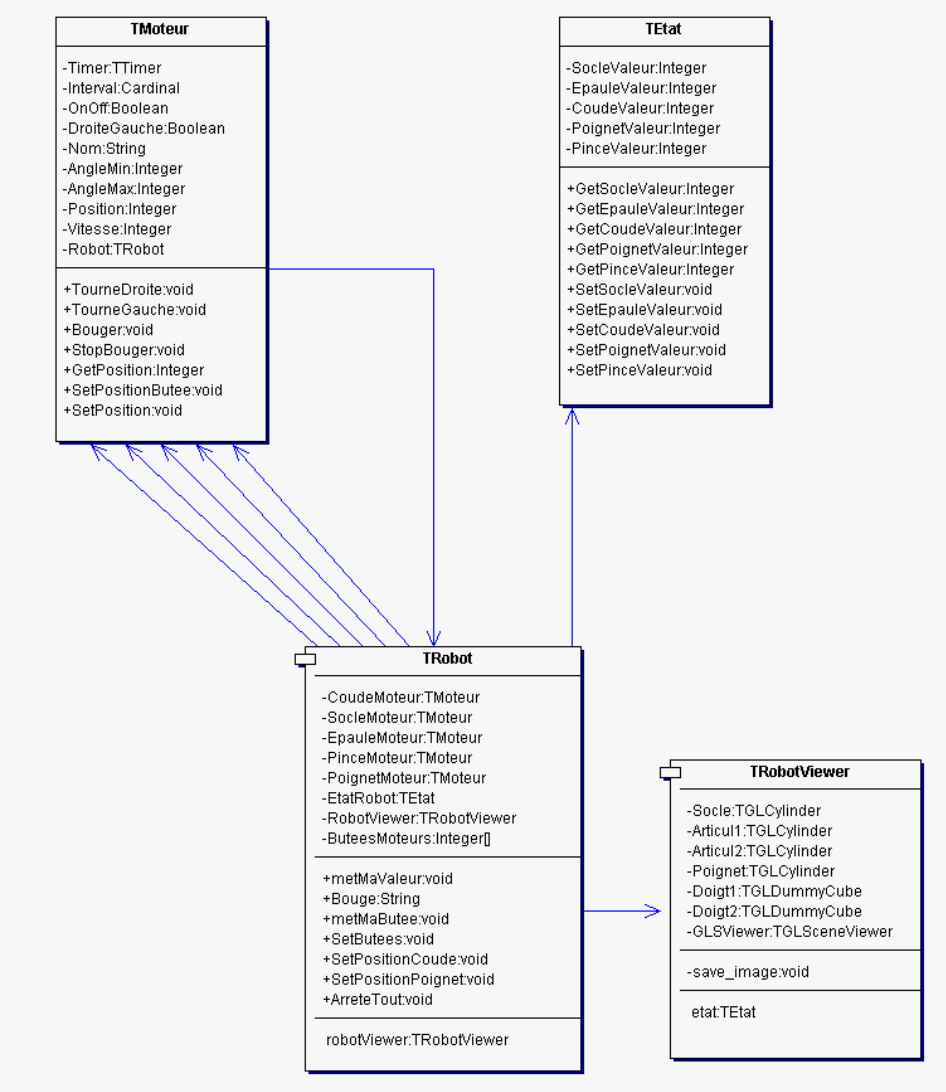


Fig. 3.5

L'objet `TRobot` connaît ses cinq moteurs et ses cinq moteurs savent qu'ils font partie du robot. Le `TRobot` connaît son état actuel qui contient les positions des moteurs. Finalement le `TRobot` a une liaison avec `TRobotViewer` pour être capable de modifier la visualisation en 3D.

GUI

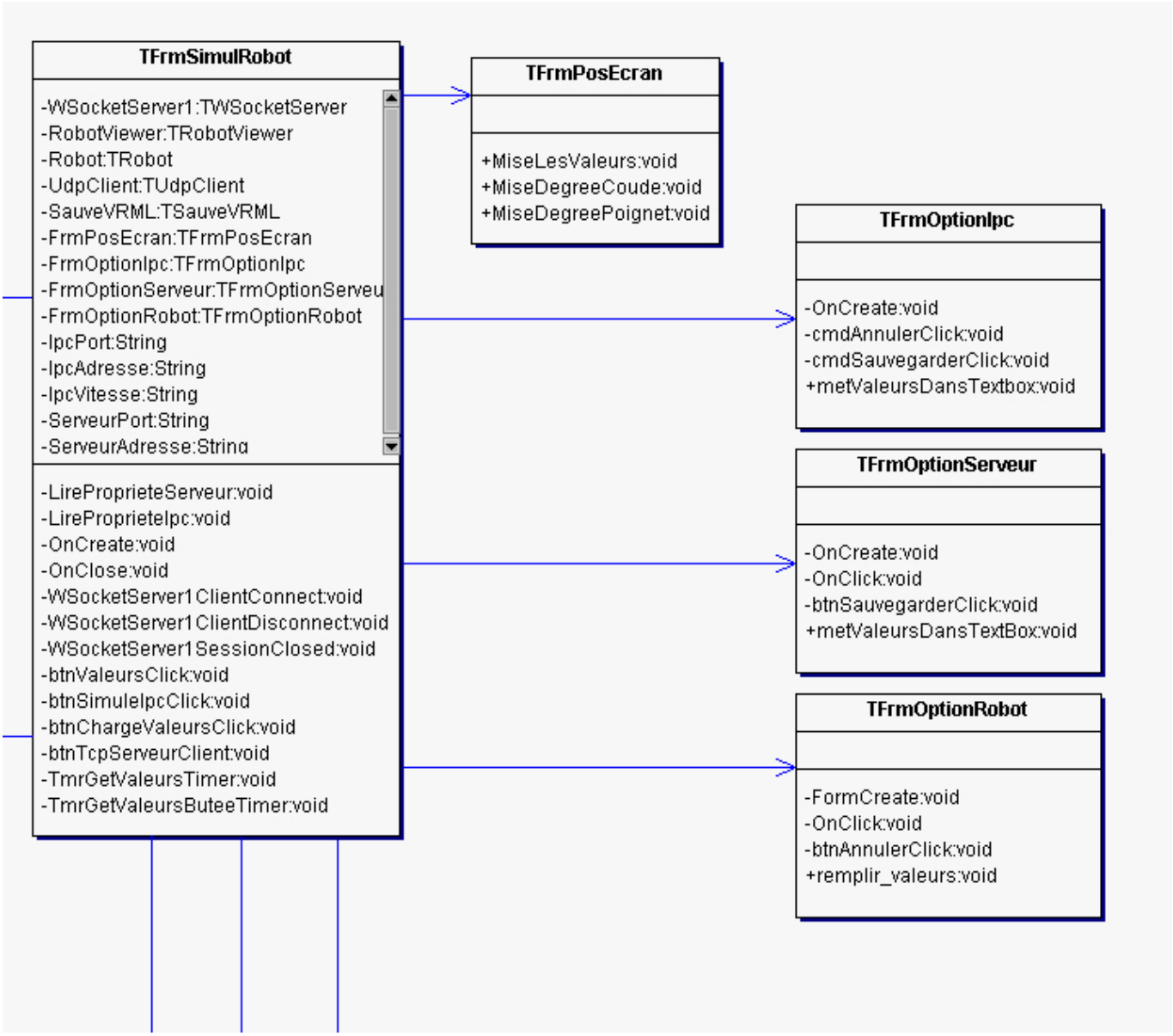


Fig. 3.6

`TFrmSimulRobot` est notre écran principal. Il s'y trouvent les possibilités de modifier les propriétés de l'IPC et du serveur. On peut faire cela avec les écrans : `TFrmOptionIpc` et `TFrmOptionServeur`. TFrmPosEcran sers a présenter les valeurs des butées et les valeurs de la positions du coude en de l'épaule.

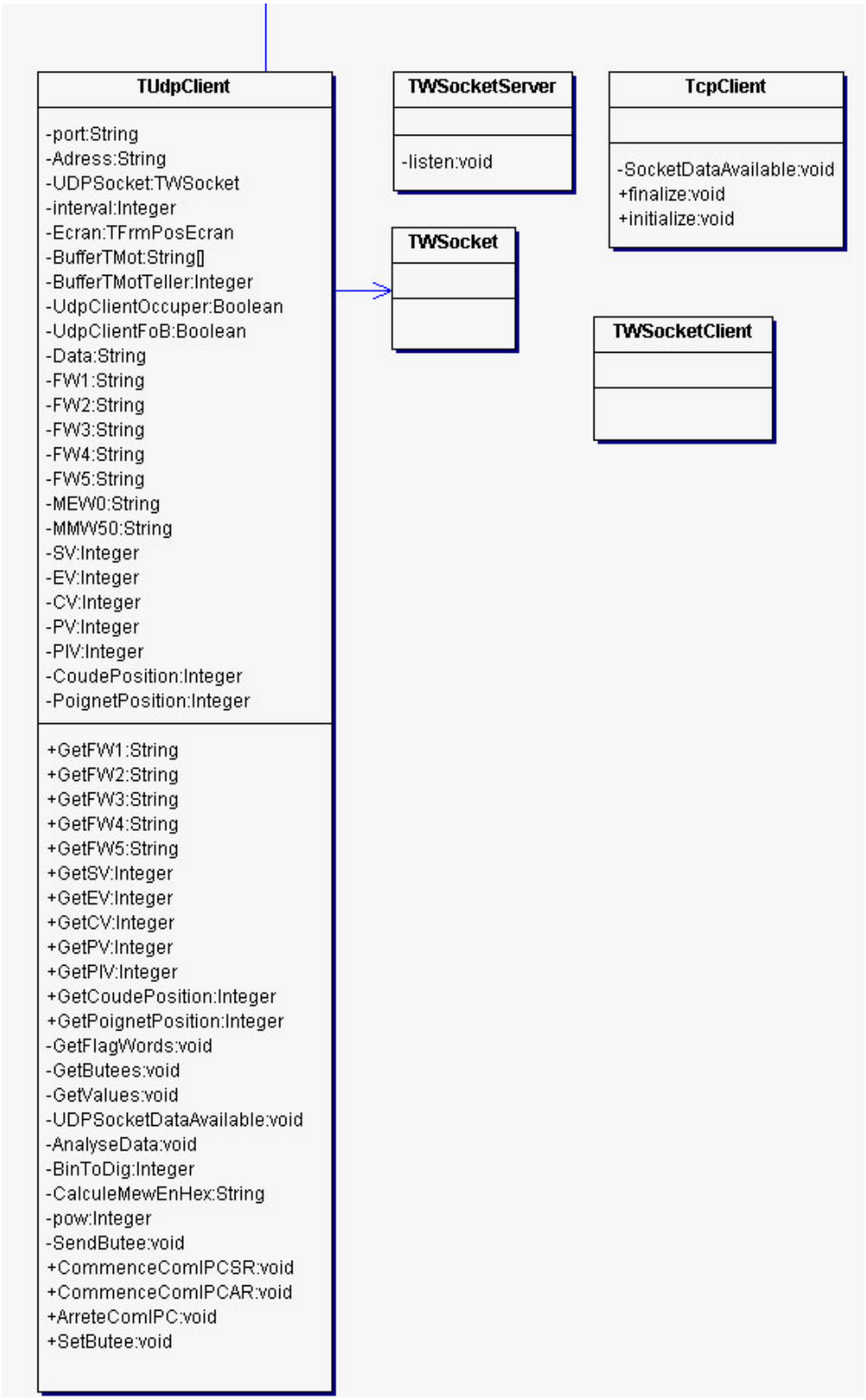

TUdpClient
-port:String
-Adress:String
-UDPSocket:TWSocket
-interval:Integer
-Ecran:TFrmPosEcran
-BufferTMot:String[]
-BufferTMotTeller:Integer
-UdpClientOccuper:Boolean
-UdpClientFoB:Boolean
-Data:String
-FW1:String
-FW2:String
-FW3:String
-FW4:String
-FW5:String
-MEW0:String
-MMW50:String
-SV:Integer
-EV:Integer
-CV:Integer
-PV:Integer
-PIV:Integer
-CoudePosition:Integer
-PoignetPosition:Integer
+GetFW1:String
+GetFW2:String
+GetFW3:String
+GetFW4:String
+GetFW5:String
+GetSV:Integer
+GetEV:Integer
+GetCV:Integer
+GetPV:Integer
+GetPIV:Integer
+GetCoudePosition:Integer
+GetPoignetPosition:Integer
-GetFlagWords:void
-GetButees:void
-GetValues:void
-UDPSocketDataAvailable:void
-AnalyseData:void
-BinToDig:Integer
-CalculeMewEnHex:String
-pow:Integer
-SendButee:void
+CommenceComIPCSR:void
+CommenceComIPCAR:void
+ArreteComIPC:void
+SetButee:void
TWSocketServer
-listen:void
TWSocket
TcpClient
-SocketDataAvailable:void
+finalize:void
+initialize:void
TWSocketClient

Fig. 3.7

3.1.3.2 La classe TRobot

La classe robot représente l'ensemble des tous les moteurs. Il est composé d'un socle, une épaule, un coude, un poignet et une pince qui sont tous des objets de la classe `TMoteur`. Ensuite il connaît son état actuel sauvegardé dans un objet de la classe `TEtat`. Pour modifier la visualisation en 3D, il doit d'abord reconnaître un `TRobotViewer`, qu'il l'utilise si un de ses moteurs bougent. Avec la méthode `TRobot.Bouge(FW :Byte) :String` on peut lui dire qu'un de ses moteurs doit bouger dans une direction. Quel moteur et de quelle direction s'agit-il, sont-ils passés par un Flag Word. La méthode `TRobot.Arretout()` laisse tous les moteurs arrêter. Quand un moteur se déplace, il dit sa nouvelle position au robot au moyen de la méthode `TRobot.metMaValeur (MType:String; Valeur:Integer)`. Avec la variable `MType` c'est possible pour le Robot de savoir de quel moteur il s'agit.

3.1.3.3 La classe TMoteur

Celui-ci fait la simulation d'un moteur. A l'aide d'un « timer » la valeur de la position est ajustée à une direction donnée. Les méthodes `TMoteur.TourneDroite` et `TMoteur.TourneGauche` font que le moteur commence à bouger lui-même dans une direction définie. Cette direction est sauvegardée dans la variable `DroiteGauche`, une des caractéristiques d'un moteur. En plus il connaît son nom, sa position, les angles min et max, un booléen variable et sa vitesse. La vitesse indique un intervalle pour le chronométreur, plus le moins l'intervalle le plus vite le moteur va se bouger.

3.1.3.4 La classe TRobotViewer

Celui-ci s'occupe avec l'ajustement de la visualisation en 3D. En utilisant les références qui sont données par l'écran principal la classe modifie les angles et cela se passe avec l'aide de la méthode `setEtat(Etat :Tetat)`. Cette méthode cherche les valeurs des angles dans un `Etat` objet et ensuite modifie la position de la visualisation.

3.1.3.5 La classe TEtat

Cette classe-ci définit l'état du robot à un moment déterminé et elle contient pour chaque moteur une variable qui représente la valeur de la position du moteur. On a implanté les méthodes nécessaires pour être capable d'écrire et de recevoir ces valeurs. En générale on utilise cette classe comme un objet d'interaction entre les classes de notre projet.

3.1.3.6 La classe TcpClient

Cette classe est dérivée de la classe `TWSocketClient`. Si un client fait une connexion avec le serveur, le serveur créera un nouvel objet de la classe `TcpClient` et il le sauvegarder dans une liste. Après l'envoi d'un message du client la méthode `SocketDataAvailable(Sender: TObject; Error: Word)` est évoquée et traitée dans le `TcpClient`. Ensuite l'objet `FrmSimulRobot` peut requérir ce message et l'utiliser pour faire bouger le robot.

3.1.3.7 La classe UdpClient

Cette classe prend soin de la communication avec l'IPC par UDP. Sa logique peut être divisée en deux possibilités. La première chose qu'on peut lui demander est de commencer sa communication avec l'IPC sans robot et on utilise `CommenceComIPCSR()` pour le réaliser. Ce qui veut dire qu'après chaque intervalle du Timer la classe demande les valeurs des Flag Words d'un jusqu'à cinq. Ces valeurs veulent dire qu'un moteur est en train de bouger et dans quelle direction. A la réception de ces valeurs la méthode `UDPSocketDataAvailable(Sender: TObject; Error: Word)` est provoqué. Dans cette procédure elle appelle la méthode `AnalyseData()` pour l'analyse des données. Pendant que l'analyse met ces valeurs dans des variables globales. Ces variables-là peuvent être demander par des autres classes, parce qu'elles sont définies comme des propriétés. Après le traitement des Flag Words la classe envoie encore les butées. Plus ou moins la même chose est fait quand on lui a demandé, avec `CommenceComIPCAR()`, de commencer sa communication avec l'IPC, qui a un robot branché. Seulement maintenant elle demande et traite les valeurs des butées et potentiomètres. Le traitement est aussi fait dans la procédure `AnalyseData()`. La classe finit avec l'envoi de ces valeurs à `TFrmPosEcran`.

3.2 Apprentissage de Delphi 7.0

3.2.1 TCP/IP serveur et client

3.2.1.1 GUI

Le serveur et le client sont la même application, donc l'utilisateur doit choisir s'il veut jouer le rôle du serveur ou du client. Dépensant du choix l'utilisateur doit mettre le serveur en écoute ou il doit se connecter sur le serveur qui est en train d'écouter jusqu'au moment un client se connecte.

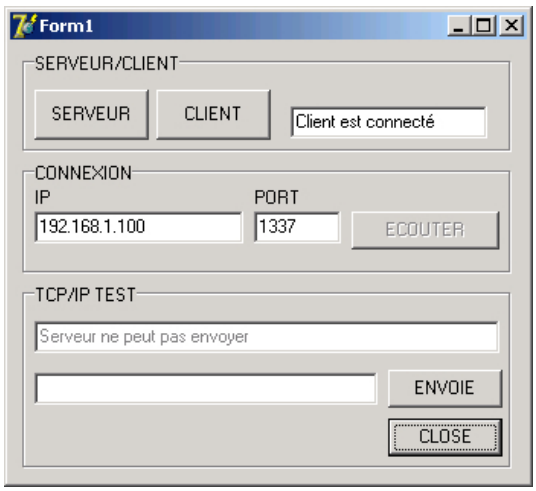


Fig. 3.8

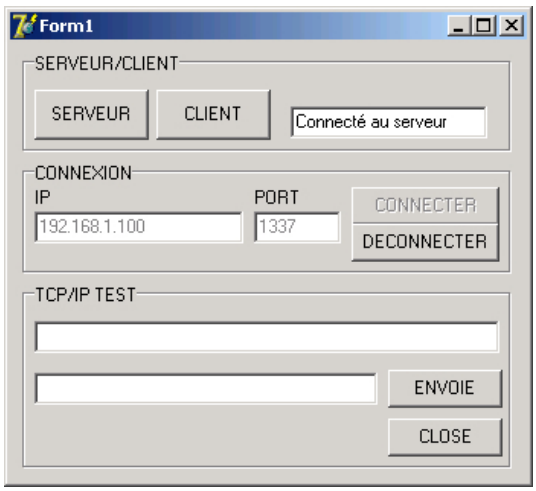


Fig. 3.9

3.2.1.2 Le code

Functions du WSocketServer

```
procedure TForm1.WSocketServerClientConnect(Sender: TObject;
  Client: TWSocketClient; Error: Word);
var
  Iclient: TcpClient;
begin
  // création d'un objet TcpClient si un client se connecte
  Iclient := TcpClient(Client);
  Iclient.OnReceive := ReceiveMessage;
  Iclient.Initialize;
end;

procedure TForm1.WSocketServerClientDisconnect(Sender: TObject;
  Client: TWSocketClient; Error: Word);
var
  Iclient: TcpClient;
begin
  // création d'un objet TcpClient
  Iclient := TcpClient(Client);
end;
```

La classe TcpClass

On peut seulement utiliser un `SocketServer` s'il y a une classe qui est dérivée de la classe `TWSocketClient`.

```
unit TcpClass;
interface
uses Windows, Messages, SysUtils, Variants, Classes, Dialogs,
Forms,
 WSocket, WSockets;
type
  TReceiveMessage = procedure(Sender: TObject; Action: String)
  of Object;

//---------------------------------------------------------------
TcpClient = class(TWSocketClient)
private
  //Variables privées de la classe TcpClient
  FReceiveMessage: TReceiveMessage;
  //Méthodes privées de la classe TcpClient
  procedure SocketDataAvailable(Sender: TObject; Error: Word);
public
  //Méthodes publiques de la classe TcpClient
  procedure Initialize;
  //Propriété de la classe TcpClient
   property OnReceive: TReceiveMessage read FReceiveMessage write
    FreceiveMessage;
end;

//---------------------------------------------------------------
implementation
uses Unit1;
//====================== TcpClient ============================
procedure TcpClient.Initialize;
begin
```

```
  OnDataAvailable := SocketDataAvailable;
end;
procedure TcpClient.SocketDataAvailable(Sender: TObject; Error:
Word);
var
  Data : String;
begin
  Data := TWSocket(Sender).ReceiveStr;
  FReceiveMessage(Self, Data);
end;

end.
```

Functions du WSocket

```
procedure TForm1.btn_connecterClick(Sender: TObject);
begin
  with WSocket do
    begin
        if State <> wsConnected then
        begin
          // le client se connecte sur le serveur
          Connect;
          while State in [WsConnecting] do
          begin
            Application.ProcessMessages;
            if Application.Terminated then
              Exit;
          end;
        end;
    end;
```

3.2.2 UDP client

3.2.2.1 GUI

Tu peut commencer la communication en UDP quand tu pousses le bouton « commence ». Au fond l'UdpClient commence à demander une valeur à l'IPC. Pour voir cette valeur tu dois pousser le bouton « montre ».

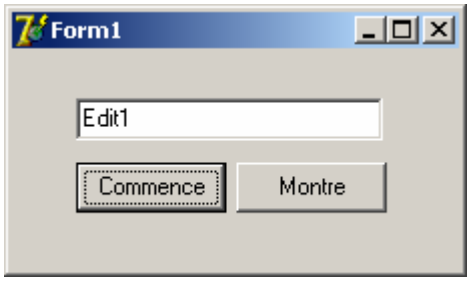


Fig. 3.10

3.2.2.2 Le code

```
procedure TUdpClient.GetValue(Sender:TObject);
begin
  TMot:='DEW0';
  UdpSocket.Proto:='udp';
  UdpSocket.Addr:=Adress;
  UdpSocket.Port:=Port;
  UdpSocket.Connect;
  UdpSocket.SendStr('DEW0');
end;
procedure TUdpClient.UDPSocketDataAvailable(Sender: TObject;
 Error: Word);
begin
  TestMot := UdpSocket.ReceiveStr;
  UdpSocket.Close;
end;
```

3.2.3 Sauvegarder d'un fichier

3.2.3.1 GUI

Si tu pousses sur le bouton, un fichier est créé avec dedans deux lignes de texte.

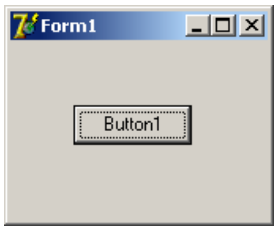


Fig 3.11

3.2.3.2 Le code

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Fichier:TextFile;
  testtekst:TextFile;
begin
  assignfile(Fichier,'test.ini');
  try
    Rewrite(Fichier);
    writeln(Fichier,'La première ligne');
    writeln(Fichier,'La deuxième ligne');
  finally
    CloseFile(Fichier);
  end;
end;
```

3.2.4 Lecture d'un fichier

3.2.4.1 GUI

Cette application lit le contenu d'un fichier et le mémorise dans un « buffer ». Nous parcourrons le buffer et nous remplissons le « StringGrid ».

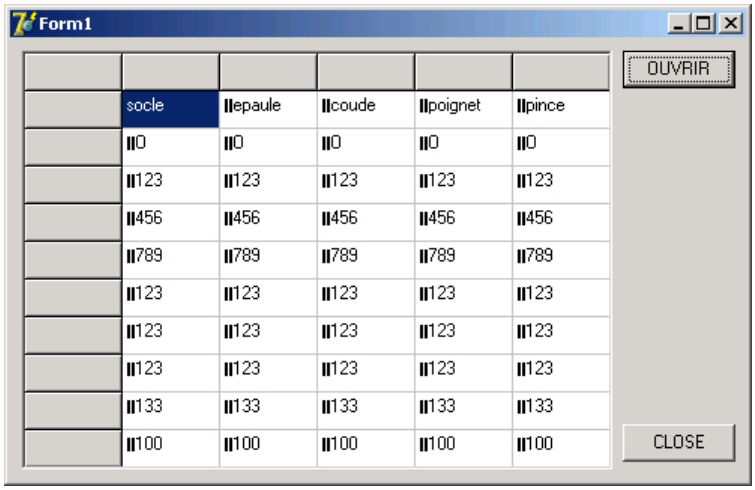


Fig. 3.12

3.2.4.2 Le code

```
procedure TForm1.btn_ouvrirClick(Sender:TObject);
begin
  if OpenDialog1.Execute then
  begin
    value:='';
    j:=1;
    try
    // ------ LIRE DES DONNEES DU FICHIER DANS UN BUFFER -------
      // FileOpen donne 0 ou supérieur chez réussite, sinon -1
      iFileHandle :=FileOpen(OpenDialog1.FileName,fmOpenRead);

      // FileSeek donne la nouvelle position du pointeur (fin du
      // fichier), sinon -1
      iFileLength :=FileSeek(iFileHandle,0,2);
      FileSeek(iFileHandle,0,0);

      // creation du buffer et assigner la mémoire
      Buffer :=PChar(AllocMem(iFileLength +1));

      // FileRead donne le nombre des bytes lus et lis les dans
      // le buffer
      iBytesRead :=FileRead(iFileHandle,Buffer^,iFileLength);

      // fermeture du fichier
      FileClose(iFileHandle);
      end;
    finally
      // Action de vider du buffer
      FreeMem(Buffer);
    end;
  end;
end;
```

3.3 La télécommande

La télécommande peut se connecter avec un autre ordinateur qui simule la visualisation et aussi avec l'IPC . Elle envoie toujours des commandements représentés en hexadécimal qui sont envoyés comme un « byte » de huit bits. Pour la télécommande c'est égal avec quel SocketServeur elle se connecte et la communication se passe avec TCP.

3.3.1 GUI

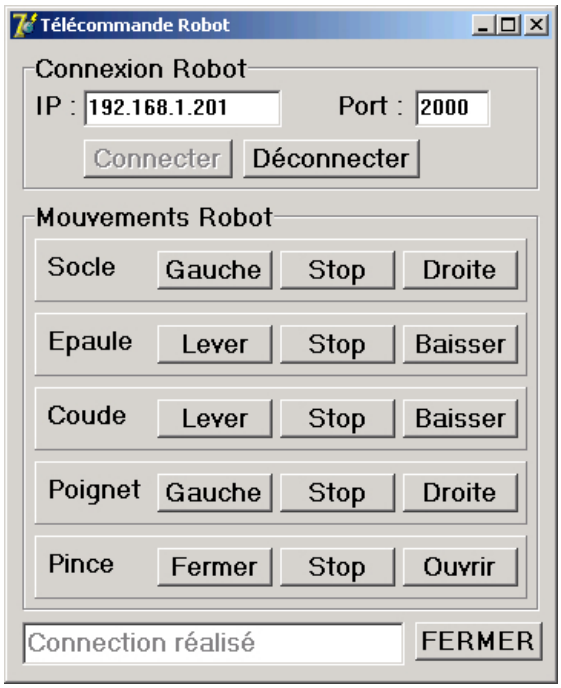


Fig. 3.13

3.3.2 Des commandements envoyés

MOTEUR	DIRECTION	BINAIR	HEXA	DECIMAL
Pince	Fermer	1101 0001	D1	209
	Arrêter	1100 0001	C1	193
	Ouvrir	1110 0001	E1	225
Poignet	Gauche	1101 0010	D2	210
	Arrêter	1100 0010	C2	194
	Droite	1110 0010	E2	226
Coude	Lever	1110 0011	E3	227
	Arrêter	1100 0011	C3	195
	Baisser	1101 0011	D3	210
Epaule	Lever	1101 0100	D4	211
	Arrêter	1100 0100	C4	196
	Baisser	1110 0100	E4	228
Socle	Gauche	1101 0101	D5	212
	Arrêter	1100 0101	C5	197
	Droite	1110 0101	E5	229

Fig. 3.14

3.3.3 Le code

Le « socket » de la télécommande doit se connecter sur le TCP serveur avant elle peut envoyer des commandements. Quand le socket n'est pas encore connecté on met les valeurs dans les variables du socket et on se connecte. S'il y a une erreur pendant que le socket est en train de se connecter, l'application échoue.

```
procedure TForm1.btn_connectClick(Sender: TObject);
begin
  with WSocket do
    begin
        if State <> wsConnected then
        begin
          Addr := edit_ip.Text;
          Port := edit_port.Text;
          Proto := 'tcp';
          Connect;
          while State in [WsConnecting] do
          begin
            Application.ProcessMessages;
            if Application.Terminated then
              Exit;
          end;
        if State = WsConnected then
        begin
          box_movement.Enabled := True;
        end;
        end;
    end;
end;
```

La méthode pour envoyer des commandements. En premier lieu on doit vérifier si le socket est encore connecté avec le serveur et si est le cas on utilise la méthode `SendStr()`. Parce que cette méthode doit avoir un « string » comme argument on doit convertir le string dans un byte avec `chr()`.

```
procedure TForm1.send_fw(FW:byte);
begin
  if WSocket.State = WsConnected then
  begin
    WSocket.SendStr(chr(FW));
  end;
end;
```

Quand tu pousses sur un bouton la méthode `send_fw(FW :Byte)` avec comme un argument le commandement que tu veux envoyer, est évoquée.

```
procedure TForm1.btn_poignet_gaucheClick(Sender: TObject);
begin
  send_fw($D2);
end;
```

3.4 Le simulateur

3.4.1 GUI

L'écran principal

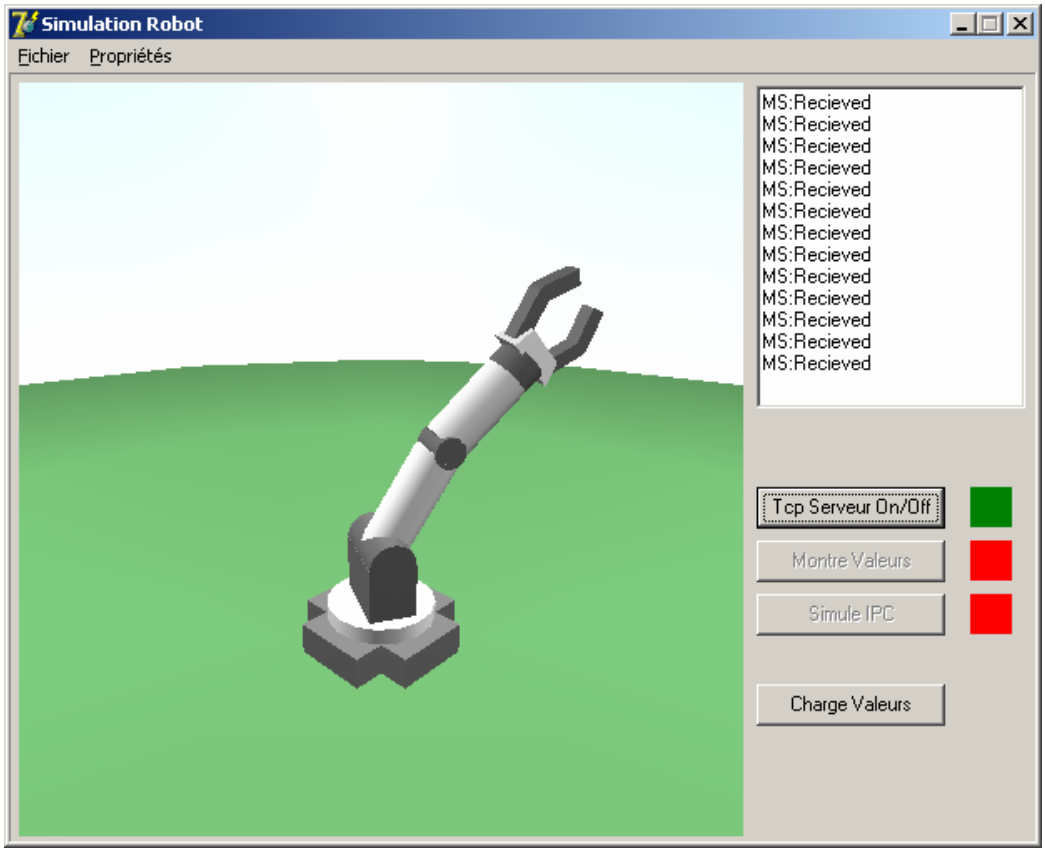


Fig. 3.15

Cet écran nous donne les moyens pour la gestion de la simulation. On peut choisir entre le TCP Serveur, Montre Valeurs et Simule IPC. Au moyen du bouton « TcpServeur ON/OFF » on peut démarrer et arrêter le Serveur qu'on utilise pour la reçu des commandements. On montre un peu de l'information des choses qui se passent avec le serveur. Si on appuie le bouton « Montre Valeurs » on commence la communication avec l'IPC. On demande les valeurs des angles et des butées qui se trouve sur l'IPC et on les montre sur un nouvel écran. Alors on vois le nom du moteur avec à côté la position en degré et à la fin ces butées qui sont indiqués avec les couleurs rouge et verte.

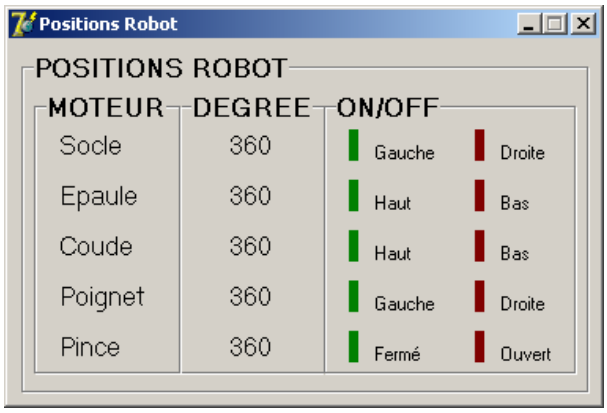


Fig. 3.16

Il y a trois écrans à votre disposition pour modifier les préférences de l'IPC, du serveur et du robot.
Dans l'écran de la modification des préférences de l'IPC. On peut enregistrer des nouvelles valeurs pour la porte, l'adresse du réseau et la vitesse du « refresh ».

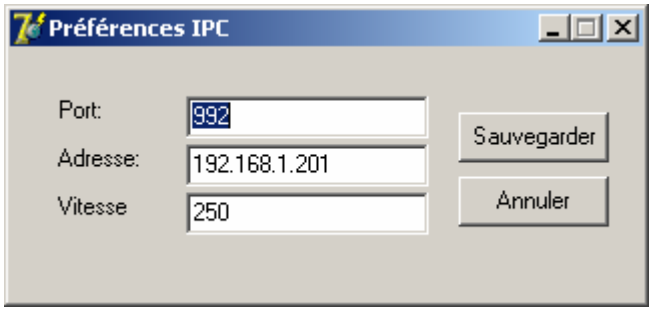


Fig. 3.17

La porte et l'adresse du serveur peut être aussi modifié.

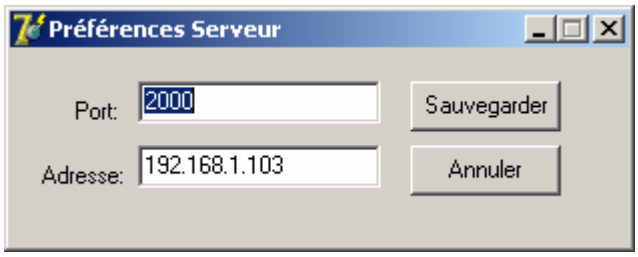


Fig. 3.18

Le dernière écran nous donne la possibilité d'adapter toutes les préférences du robot. La vitesse, l'angle max l'angle min et la position de chaque moteur peut être réglé.
Après chaque modification d'une de les préférences on doit encore appuyer le bouton « Charge Valeurs » que tous les valeurs vont être charger.

Fig. 3.19

3.4.2 Clarification du code

3.4.2.1 La partie simulation

A la création d'un robot le constructor est provoqué.

```
Constructor TRobot.Create();
var
  t1 : integer;
begin
  iFileHandle :=FileOpen('PropRobot.ini',fmOpenRead);
  iBytesRead :=FileRead(iFileHandle,Buffer^,iFileLength);
  …
  analyse du buffer, met les propres valeurs dans les variables
  …
  SocleMoteur := TMoteur.Create(TComponent(Self),vitesse,nom
                                ,anglemax,anglemin,position,self);
  ButeesMoteurs[1] := butee;
  …
  EtatRobot :=
TEtat.CreateAV(SocleMoteur.Position,EpauleMoteur.Position,CoudeM
oteur.Position,PoignetMoteur.Position,PinceMoteur.Position);

end;
```

D'abord le constructeur lit le fichier `PropRobot.ini` qui consiste des propriétés des moteurs. Avec ces valeurs il crée cinq moteurs. Ensuite il crée l'état du robot dans l'objet `EtatRobot` avec les positions initiales. Parce que le robot doit

connaître le robot viewer qu'on ne peut pas l'envoyer avec le constructeur on lui donne avec la méthode `setRobotViewer`.

```
procedure TRobot.setRobotViewer(RV:TRobotViewer);
begin
  RobotViewer := RV;
  RobotViewer.setEtat(EtatRobot);
end;
```

`SetRobotViewer` termine avec l'envoi d'état initial de façon que le `RobotViewer` puisse modifier la visualisation en 3D.
Quand un robot est construit et le `RobotViewer` est envoyé, tu as la possibilité de faire bouger un de ces moteurs.

```
function TRobot.Bouche(FW:byte):string;
begin
  if FW = $E1 then
    begin
      PinceMoteur.TourneGauche;
      Bouche := '5' + IntToStr(ButeesMoteurs[5]);
    end;
…
end;
```

Cette fonction analyse le Flag Word pour savoir quel moteur et quelle direction. Il renvoie un string qui comprend deux chiffres, le premier indique quel moteur et le deuxième indique s'il est en butée et dons quel sens. Un moteur en a deux, droite et gauche, qui peuvent être évoqué avec les méthodes `TourneDroite` et `TourneGauche`.

```
procedure TMoteur.TourneDroite;
begin
  OnOff := True;
  DroiteGauche := False;
  Timer.Enabled := True;
end;

procedure TMoteur.TourneGauche;
begin
  OnOff := True;
  DroiteGauche := True;
  Timer.Enabled := True;
end;
```

Chaque unité de temps est provoquée par un chronométreur, il survole la méthode `Bouger`.

```
procedure TMoteur.Bouger;
begin
  if OnOff = True then
  begin
    if DroiteGauche = true then    //Gauche
      begin
        if Position < AngleMax then
        begin
          Position := Position + 1;
          robot.metMaValeur(nom,Position);
```

```
            robot.metMaButee(nom,2);
          end
          else
          begin
            Timer.Enabled := False;
            robot.metMaButee(nom,1);
          end
        end
      else
        begin
          if Position > AngleMin then
          begin
            Position := Position - 1;
            robot.metMaValeur(nom,Position);
            robot.metMaButee(nom,2);
          end
          else
          begin
            Timer.Enabled := False;
            robot.metMaButee(nom,3);
          end
        end
    end
end;
```

La position du moteur peut être seulement corrigé quand le moteur est en marche ce qui est indiqué avec la variable `OnOff`. Quand il a répondu à cette condition on doit encore savoir s'il ne traverse pas l'angle max ou l'angle min. Si cela se passe il envoie la valeur «un» ou «trois» pour dire au robot que le moteur est en butée et il fini avec la terminaison du « timer ». Sinon il modifie la position, envoie la nouvelle position au robot et la valeur «deux» qui représente un moteur ne pas en butée.

```
procedure TRobot.metMaValeur(MType:String; Valeur:Integer);
begin
  case MType[1] of
    'S': EtatRobot.SetSocleValeur(Valeur);
    …
  end;
  RobotViewer.setEtat(EtatRobot)
end;
```

De ce fait le robot peut modifier sa position actuelle. D'après la valeur `MType`, qui représente le nom d'un moteur, il a la possibilité d'enregistrer la bonne valeur dans l'état du robot. Ensuite elle envoie le nouvel état au `RobotViewer`, qui peut modifier la visualisation en 3D.

```
procedure TRobot.metMaButee(MType:String;BGoD:Integer);
begin
  case MType[1] of
    'S' : ButeesMoteurs[1] := BGoD;
    …
  end;
end;
procedure TRobotViewer.setEtat(Etat:TEtat);
begin
  Doigt1.PitchAngle:= - Etat.GetPinceValeur;
  Doigt2.PitchAngle:= Etat.GetPinceValeur;
```

```
      socle.TurnAngle := Etat.GetSocleValeur;
      Articul1.TurnAngle :=  - Etat.GetEpauleValeur;
      Articul2.TurnAngle := - Etat.GetCoudeValeur;
      Poignet.TurnAngle := Etat.GetPoignetValeur;
    end;
```

Le `RobotViewer` reçoit un état déterminé et modifie avec ces valeurs les angles des objets de GLSCENE.

3.4.2.2 La partie serveur TCP

On a utilisé le composant `TWSocketServer` pour la gestion du serveur « TCP ». Les propriétés du serveur sont enregistrées dans un fichier `PropServ.ini`. c'est celui qu'on doit lire avant d'initialiser le serveur.

```
procedure TFrmSimulRobot.LireProprieteServeur();
begin
  try
    iFileHandle :=FileOpen('PropServ.ini',fmOpenRead);
    iBytesRead :=FileRead(iFileHandle,Buffer^,iFileLength);
    FileClose(iFileHandle);

    for i := 0 to iBytesRead-1 do
    begin
      if((Buffer[i]<>'=')And(Buffer[i]<>';')And(place=1)) then
        KindValue := KindValue + Buffer[i];
      if (Buffer[i] = '=') then
        place := 2;
      if((Buffer[i]<>'=')And(Buffer[i]<>';')And(place = 2)) then
        Value := Value + Buffer[i];
      if (Buffer[i] = ';') then
      begin
        place := 3;
        if (KindValue = 'ip') then
          ServeurAdresse := Value;
        if (KindValue = 'port') then
          ServeurPort := Value;
        KindValue := '';
        Value := '';
      end;
      if (Buffer[i] = #$A) then
        place := 1;
    end;
  finally
    FreeMem(Buffer);
  end;
end;
```

On ouvre le fichier avec l'option `fmOpenRead`, comme ça on peut seulement lire dans le fichier et ne peut pas le modifier. On lit le contenu dans un `Buffer` qu'ensuite on analyse.

```
    WSocketServer1.Port := ServeurPort;
    WSocketServer1.Addr := '0.0.0.0';
    WSocketServer1.ClientClass := TcpClient;
    WSocketServer1.Listen;
```

On initialise le serveur et on lui dit de commencer à écouter pour des clients sur un port et une adresse déterminée. On lui dit que les clients qui vont se connecter sont de la classe TcpClient.

```
unit TcpClass;

type

  TReceiveAction = procedure(Sender: TObject; Action: byte) of Object;

//-------------------------------------------------------
TcpClient = class(TWSocketClient)

private
  //Var
  FReceiveAction: TReceiveAction;

  //Methodes
  procedure SocketDataAvailable(Sender: TObject; Error: Word);

public
  //Methodes
  procedure Initialize;
  procedure Finalize;

  //Properties
  property OnReceive: TReceiveAction read FReceiveAction
                                     write FreceiveAction;

end;
//-------------------------------------------------------
//============================TcpClient=================
procedure TcpClient.Initialize;
begin
  OnDataAvailable := SocketDataAvailable;
end;

procedure TcpClient.SocketDataAvailable(Sender: TObject; Error: Word);
var
  Data : String;
  BData : byte;
begin
  Data := TWSocket(Sender).ReceiveStr;
  if Data <> '' then
    BData :=  ord(Data[1]);
  FReceiveAction(Self, BData);
end;
end.
```

Quand un client se connecte sur le serveur, il le sauvegarde dans une liste des toutes les clients. Cela comme une référence aux objets de la classe `TcpClient` qui est dérivé de la classe `TWSocketClient`. Ce client reçoit les messages qui contiennent les Flag Words, envoyer d'un télécommande sur TCP. Si un message arrive le méthode `SocketDataAvailable` est provoqué. Dans cette méthode on transforme le flag word, reçu comme un string, dans un byte. Dans l'écran principal on utilise la propriété `OnReceive` pour recevoir la donnée.

```
Iclient.OnReceive := ReceiveAction;
```

```
procedure TFrmSimulRobot.ReceiveAction(Sender: TObject; Action: byte);
var
  RV : String;
begin
  if Action <> 1 then
  begin
    RV := Robot.Bouche(Action);
  end;
  mmoServerMessages.Lines.Add('MS:Recieved');
end;
```

A l'aide du flag word reçu dans la variable Action on laisse le robot bouger.

3.4.2.3 La partie client UDP

Il y a deux situations concrets pourquoi on utilise l'UDP client. Si quelqu'un fait le vrai robot bouger, on peut demander au UDP client d'aller chercher les valeurs des butées et des angles des certains moteurs. On commence cette communication avec la méthode CommenceComIPCAR. Dans la deuxième cas si quelqu'un envoie les flag words au IPC mais il n'y a pas un robot. On peut demander au UDP client d'aller chercher les valeurs des flag words sur l'IPC et de cette façon faire le simulation et visualisation en 3D bouger.

Communication avec l'IPC qui a un robot branché

Ici l'IPC contient les valeurs des butées et des angles de deux moteurs, le coude et le poignet. On va montrer ces valeurs dans un écran et modifier la position de la visualisation aux valeurs des angles.

```
procedure TFrmSimulRobot.btnValeursClick(Sender: TObject);
begin
  if (TmrGetValeurs.Enabled = False) then
  begin
    FrmPosEcran.Visible := true;
    UdpClient.CommenceComIPCAR;
    TmrGetValeurs.OnTimer := TmrGetValeursButeeTimer;
    TmrGetValeurs.Enabled := true;
  end
  else
  begin
    FrmPosEcran.Visible := False;
    UdpClient.ArreteComIPC;
    TmrGetValeurs.Enabled := False;
  end;
end;
```

FrmPosEcran est l'écran qu'on utilise pour la présentation des valeurs. Le Timer qui est mis en état du fonctionnement sers au ramassement des valeurs pour la modification de la visualisation.

```
procedure TUdpClient.CommenceComIPCAR();
begin
  Timer.OnTimer := GetButees;
  UdpClientFoB := True;
  Timer.Enabled := True;
end;
```

L'UDP client commence sa communication. Chaque unité de temps il demande les valeurs des butées et des angles.

```
procedure TUdpClient.GetButees(Sender:TObject);
begin
  if UdpClientOccuper = False then
  begin
    BufferTmotTeller := 1;
    BufferTMot[1] := 'DEW0';
    BufferTMot[2] := 'DMW50';
    BufferTMot[3] := 'DEW12';
    BufferTMot[4] := 'DEW13';
    UdpClientOccuper := True;
    GetValues();
  end;
end;
```

Les valeurs des butées se trouve dans l'output word numéro zero qu'on peut retrouver avec l'envoie du mot « DEW0 » et dans le flag word 50 qu'on peut retrouver avec l'envoie du mot « DMW50 ». La valeur d'angle de la coude se trouve dans l'output word 12 « DEW12 » et l'output word 13 « DEW13 » présente la valeur d'angle du poignet.

```
procedure TUdpClient.GetValues();
begin
  UdpSocket.Proto:='udp';
  UdpSocket.Addr:=Adress;
  UdpSocket.Port:=Port;
  UdpSocket.Connect;
  UdpSocket.SendStr(BufferTMot[BufferTMotTeller]);
end;
```

On met tous les mots qu'on doit demander au IPC dans une chaîne `BufferTMot`. A l'aide d'un compteur `BufferTMotTeller` on survole ce chaîne pour les envoie l'un après l'autre.

```
procedure TUdpClient.UDPSocketDataAvailable(Sender: TObject;
Error: Word);
begin
  Data := UdpSocket.ReceiveStr;
  UdpSocket.Close;
  AnalyseData();
  if UdpClientFoB = False then
    … // Pour les FlagWords
  if UdpClientFoB = True then
  begin
    if BufferTMotTeller < 4 then
    begin
      BufferTMotTeller := BufferTMotTeller + 1;
      GetValues();
    end;
  end;
end;
```

On attende jusqu'à l'UdpClient reçoit une réponse pour envoyer le mot suivant. Alors on reçoit la réponse, on l'analyse et après on augmente le valeur du compteur. De cette façon on peut envoyer la prochaine valeur.

```
procedure TUdpClient.AnalyseData();
begin
  //cherche dans le résultat le nom du demande
  reqmot := '';
  for t1 := 1 to 3 do
    reqmot := reqmot + Data[t1];

  //cherche dan le resultat le valeur du demande
  t1 := pos('=',Data);
  valeur := '';
  for t2 := t1+1 to length(Data) do
    valeur := valeur + Data[t2];

  if UdpClientFoB = False then
    … // Pour les FlagWords
  if UdpClientFoB = True then
  begin
    chiffre := StrToInt(valeur);

    if (reqmot = 'DEW') And (BufferTMotTeller = 1) then
    begin
      SV:=2; EV:=2;
      if (chiffre And $08) = $08 then //Butee Epaule basse
        EV := 3;
      if (chiffre And $10) = $10 then //Butee Epaule haute
        EV := 1;
      if (chiffre And $20) = $20 then //Butee Socle gauche
        SV := 1;
      if (chiffre And $40) = $40 then //Butee Socle milieu
        SV := 2;
      if (chiffre And $80) = $80 then //Butee Socle droite
        SV := 3;
    end;
    if (reqmot = 'DMW') And (BufferTMotTeller = 2) then
    begin
      PV:=2; PIV:=2;
      if (chiffre And $10) = $10 then //Butee Pince fermée
        PIV := 1;
      if (chiffre And $20) = $20 then //Butee Pince ouverte
        PIV := 3;
      if (chiffre And $40) = $40 then //Butee Poignet gauche
        PV := 1;
      if (chiffre And $80) = $80 then //Butee Poignet droite
        PV := 3;
      UdpClientOccuper := False;
    end;
    if (reqmot = 'DEW') And (BufferTMotTeller = 3) then
    begin
      chiffre := -((chiffre div 9)-237);
      CoudePosition := chiffre;
      Ecran.MiseDegreeCoude(chiffre);
    end;
    if (reqmot = 'DEW') And (BufferTMotTeller = 4) then
    begin
      PoignetPosition := chiffre;
      Ecran.MiseDegreePoignet(chiffre);
    end;

    ecran.MiseLesValeurs(SV,EV,CV,PV,PIV);
  end;
end;
```

Dans l'analyse on commence avec la division de la réponse en le mot d'envoie « `reqmot` » et la solution « `valeur` ». Ensuite si c'est une valeur d'une butée on fait un masque pour savoir de quel moteur et de quel butée il s'agit et on met la valeur dans le propre variable. A la fin de la méthode on envoie toutes les valeurs au écran. De cette façon l'écran peut modifier sa présentation des valeurs.

```
property PSV : integer read SV;
…
property PCoudePosition : integer read CoudePosition;
…
```

Maintenant l'écran principal doit encore modifier la visualisation, pour faire cela il a besoin de ces valeurs. A cause de cela on a déclaré les variables, qu'on utilise pour enregistrer les valeurs, comme des propriétés.

```
procedure TFrmSimulRobot.TmrGetValeursButeeTimer(Sender: TObject);
begin
  Robot.SetButees(UdpClient.PSV,UdpClient.PEV,UdpClient.PCV,UdpClient.PPV,
                                                        UdpClient.PPIV);
  Robot.SetPositionCoude(UdpClient.PCoudePosition);
  Robot.SetPositionPoignet(UdpClient.PPoignetPosition);
end;
```

Communication avec l'IPC sans Robot

Dans ce cas, il y a quelqu'un qui pilote l'IPC mais sans robot. Alors les valeurs sont mises au Flag words. Maintenant c'est à notre logiciel de simuler les mouvements qui sont indiquées par les Flag words.

```
procedure TUdpClient.CommenceComIPCSR();
begin
  Timer.OnTimer := GetFlagWords;
  UdpClientFoB := False;
  Timer.Enabled := True;
end;
```

A l'aide de la méthode CommenceComIPCSR on mettre la communication en route. Chaque unité de temps le client demande les flag words au IPC.

```
procedure TUdpClient.GetFlagWords(Sender:TObject);
begin
  if UdpClientOccuper = False then
  begin
    BufferTmotTeller := 1;
    BufferTMot[1] := 'DMW1'; //Pince
    …
    BufferTMot[5] := 'DMW5'; //Socle
    UdpClientOccuper := True;
    GetValues();
  end;
end;
```

Le client demande les flag words 1 jusqu'à 5 qui contiennent un valeur qui indique quelle mouvement un moteur doit faire.

```
procedure TUdpClient.GetValues();
begin
  UdpSocket.Proto:='udp';
  UdpSocket.Addr:=Adress;
  UdpSocket.Port:=Port;
  UdpSocket.Connect;
  UdpSocket.SendStr(BufferTMot[BufferTMotTeller]);
end;
```

Aussi ici on met tous les mots dans une chaîne et la survole avec un compteur. Chaque mot est envoyé l'un après l'autre. Mais entre deux mots le client attende à l'IPC pour une réponse qu'il analyse avant d'envoyer le prochain mot.

```
procedure TUdpClient.UDPSocketDataAvailable(Sender: TObject;
Error: Word);
begin

  Data := UdpSocket.ReceiveStr;
  UdpSocket.Close;
  AnalyseData();

  if UdpClientFoB = False then
  begin
    if BufferTMotTeller < 5 then
    begin
      BufferTMotTeller := BufferTMotTeller + 1;
      GetValues();
    end
    else
      SendButee();
  end;
  if UdpClientFoB = True then
  … //Les butées en valeurs des angles
  end;
end;
```

```
procedure TUdpClient.AnalyseData();
begin

… //La même qu'avant, la division

  if UdpClientFoB = False then
  begin
    if reqmot = 'DMW' then
    begin
      case Data[4] of
        '1':FW1 := valeur;
        '2':FW2 := valeur;
        '3':FW3 := valeur;
        '4':FW4 := valeur;
        '5':FW5 := valeur;
      end;
    end;
  end;

  if UdpClientFoB = True then
  begin
  …
  end;
end;
```

Dans l'analyse de la réponse, il voit de quel flag word il s'agit et met le valeur dans la variable correcte.

```
property PFW1 : string read FW1; //FW1: fonctionnement de pince
```

Ensuite l'écran déplace la visualisation au moyen du flag word qu'il demande au UdpClient. Le robot renvoie une valeur qu'indique si il est en butée ou pas. C'est ce valeur que l'écran fait suivre au UdpClient.

```
procedure TFrmSimulRobot.TmrGetValeursTimer(Sender: TObject);
begin
  UdpClient.SetButee(Robot.Bouche(StrToInt(UdpClient.PFW1)));
  UdpClient.SetButee(Robot.Bouche(StrToInt(UdpClient.PFW2)));
  UdpClient.SetButee(Robot.Bouche(StrToInt(UdpClient.PFW3)));
  UdpClient.SetButee(Robot.Bouche(StrToInt(UdpClient.PFW4)));
  UdpClient.SetButee(Robot.Bouche(StrToInt(UdpClient.PFW5)));
end;
```

L'UdpClient connaît deux chaînes qui représente le output words « 0 » et « 50 » en binaire. C'est là-dedans que la méthode SetButee met la propre valeur.

```
procedure TUdpClient.SetButee(RV:String);
begin
  if RV = '11' then //butee socle gauche
    MEW0[3] := '1';
  if RV = '13' then //butee socle droite
    MEW0[1] := '1';
  if RV = '12' then //Socle ne pas butee
    MEW0[3]:='0';
    MEW0[1]:='0';
  if RV = '21' then // Epaule haute
    MEW0[4] := '1';
  if RV = '23' then // Epaule basse
    MEW0[5] := '1';
  if RV = '22' then // Epaule ne pas butee
  begin
    MEW0[4] := '0';
    MEW0[5] := '0';
  end;
  if RV = '41' then //butee Poignet gauche
    MMW50[2] := '1';
  if RV = '43' then //butee Poignet droite
    MMW50[1] := '1';
  if RV = '42' then //Poignet ne pas butee
  begin
    MMW50[1] := '0';
    MMW50[2] := '0';
  end;
  if RV = '51' then // butee pince fermée
    MMW50[3] := '1';
  if RV = '53' then
    MMW50[4] := '1';
  if RV = '52' then
  begin
    MMW50[3] := '0';
    MMW50[4] := '0';
  end;
end;
```

Après que tous les flag words sont demandés le client envoie les butées au l'IPC. De cette façon il peut les mettre dans la flag word et output word correct. Il y en a deux « MEW0 » et « MMW50 ».

```
procedure TUdpClient.SendButee();
begin
  UdpSocket.Proto:='udp';
  UdpSocket.Addr:=Adress;
  UdpSocket.Port:=Port;
  UdpSocket.Connect;
  SS := 'MEW0=' + IntToStr(BinToDig(MEW0));
  UdpSocket.SendStr(SS);
  UdpSocket.Close;
  UdpSocket.Proto:='udp';
  UdpSocket.Addr:=Adress;
  UdpSocket.Port:=Port;
  UdpSocket.Connect;
  SS := 'MMW50=' + IntToStr(BinToDig(MMW50));
  UdpSocket.SendStr(SS);
  UdpSocket.Close;
  UdpClientOccuper := False;
end;
```

Partie programmation Internet 4

Le but était de créer un site web qui pouvait faire bouger le vrai robot et aussi la simulation. La télécommande écrite en Delphi ne devrait pas savoir s'elle était connectée avec l'IPC ou la simulation, donc le site web, qui est écrit en HTML et PHP ne doit pas le savoir aussi.
Le PHP est un langage dynamique qui nous donne la possibilité de nous connecter ou moyen des sockets sur un serveur et d'échanger des informations, quelque chose qu'on ne peut jamais réaliser avec seul l'HTML.

4.1 La côté client

4.1.1 La télécommande

La télécommande consiste en deux parties, c'est-à-dire un morceau statique, écrit en HTML et un morceau dynamique, écrit en PHP. Nous utilisons l'HTML pour montrer la télécommande elle-même, des boutons et des textes. Le PHP doit régler la connexion avec le serveur et l'envoi des commandements.

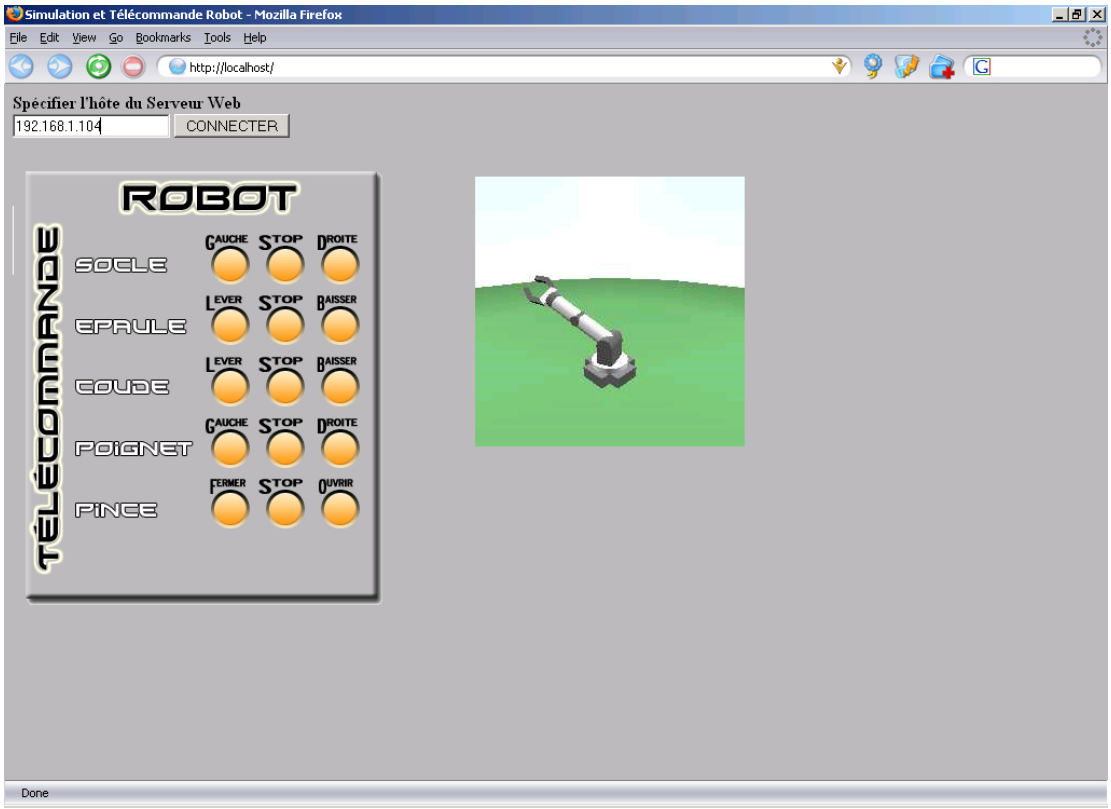


Fig. 4.1

La page principale, `index.html`, est divisée dans 3 « frames » ou dedans des pages PHP ou HTML sont chargées.

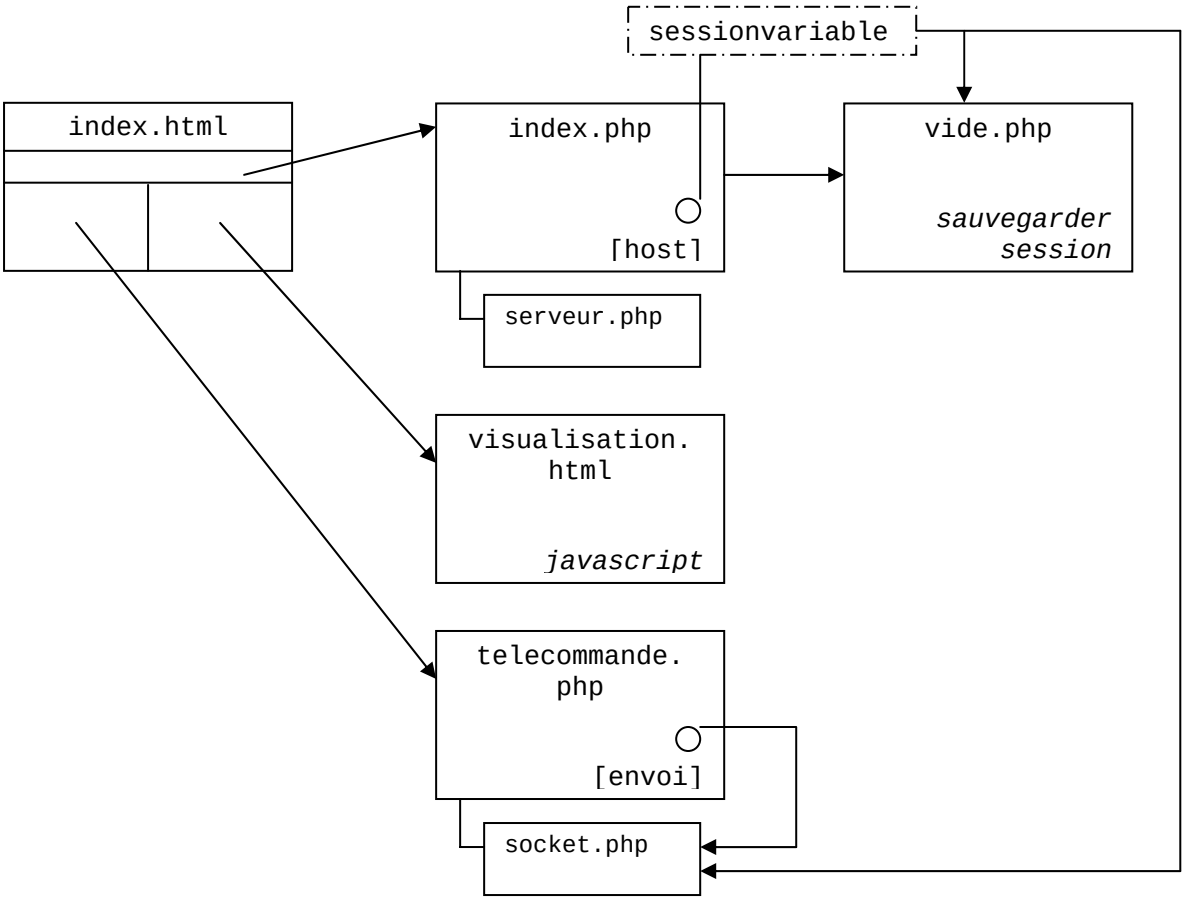


Fig. 4.2

Le processus du site web commence chez `index.html`. Quand l'apache serveur se situe dans un dossier avec un fichier index.html, il ne va pas montrer le contenu du dossier, mais ce fichier lui-même. Un fichier avec l'extension .html ne peut jamais contenir le code PHP, donc c'est pour cette raison que nous avons commencé avec index.html. Le premier cadre, `index.php`, te demandera pour l'hôte du serveur site web, le serveur qui va connecter avec l'IPC. En effet index.php n'est plus que la page `serveur.php` qui est incluse.

```
<?php
include ("serveur.php");
?>
```

Le code du `serveur.php` est maintenant transformé par l'apache serveur et renvoyé au demandeur du site web.

```
<?php
print ("<HTML><HEAD><TITLE>TELECOMMANDE
  ROBOT</TITLE></HEAD><BODY BGCOLOR=#BBBBBB TEXT=#000000><FORM
  METHOD=\"GET\" ACTION=\"vide.php\" NAME=HoteForm>");
print ("<B>Spécifier l'hôte du Serveur Web</B>");
print ("<br>");
```

```
print ("<B><input name=\"HOST\" type=\"text\"
  value=\"192.168.1.104\">");
print (" ");
print ("<input name=\"submit\" type=\"submit\"
  value=\"CONNECTER\"></FORM></BODY></HTML>");
?>
```

Un « input type » doit toujours être inclus dans un « form » sinon l'envoi au serveur ne va pas fonctionner. Le contenu de l'input type `HOST` deviendra envoyé à la page `vide.php` quand tu pousse sur le bouton `CONNECTER`.

Dans la page `vide.php` nous sauvegardons le string reçu dans une « session variable ». Une session variable peut être considérée comme une variable globale. Nous devons seulement la sauvegarder la première fois.

```
if (isset($_GET['HOST']))
{
      $session_host = $_GET['HOST'];
      session_start();
      $_SESSION['session_host']=$session_host;
}
```

Sur la page de la télécommande on peut faire bouger le robot. Des boutons sont en fait des hyper links. En PHP (et aussi en HTML) c'est possible de transmettre des valeurs dans un url.

```
<area shape="circle" coords="304,99, 16"
href="/TELECOMMANDE.php?FW=E5" alt="" >
```

Ce code peut comprendre : l'envoie de la variable `FW`, avec la valeur `E5`, à la page `TELECOMMANDE.php`. Dans cette page, la même ou nous étions, on doit traiter ce qui est envoyé. Pour faire ce traitement nous utilisons `socket.php`, qui est inclus dans `TELECOMMANDE.php`.

```
<?php
include ("socket.php");

if (isset($_GET['FW']))
{
      session_start();
      sendflagword($_GET['FW']);
}
?>
```

Si la page reçoit une valeur `FW` elle va évoquer la méthode `sendflagword` qui est une fonction du `socket.php`.

```
<?php
function sendflagword($FW)
{
      error_reporting(E_ALL);

      $address = $_SESSION['session_host'];
      $service_port = '2001';
      $socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
```

```
        $result = socket_connect($socket, $address, $service_port);

        if (isset($_GET['FW']))
              {
                    $in = $_GET['FW'];
                    socket_write($socket, $in, strlen($in));
              }
        socket_close($socket);
}
?>
```

La méthode fait une connexion avec le serveur. L'adresse est encore sauvegardée dans la session variable. Un socket est créé et connecté. Après l'envoi du commandement on ferme le socket.

4.1.2 Visualisation

4.1.2.1 Manière .JPG

La visualisation est générée par l'application de la simulation et la page `visualisation.html` va montrer un image en format jpg[1]. Chaque fois le robot change sa position l'application en Delphi va créer une nouvelle image du robot complet et la sauvegarder dans le fichier `C:/webserver/www`. La page HTML contient un JavaScript qui rafraîchit chaque seconde la page. Ainsi on peut voire 'bouger' le robot.

```
<html>
<head>
<META HTTP-EQUIV="REFRESH" CONTENT="1">
</head>
<body>
<br>
<img name=visu src="robot.jpg">
<script language=Javascript>
 visu.reload;
</script>
</body>
</html>
```

4.1.2.2 Manière VRML

On a essayé d'écrire une classe qui peut sauvegarder un GLScene dans un fichier du format VRML. Malheureusement on n'a pas eu assez de temps pour finir cette partie de la projet. On a arrivé à un fichier qui consiste tous les éléments en 3D et cela avec la propre hiérarchie, mais regrettable on n'était pas capable de les mettre à leurs propre place. La même chose avec le java applet qu'on a construit pour la visualisation d'une fichier en 3D. c'était écrit mais on n'a pas eu le temps pour retirer les erreurs qui sont encore dedans.

---

[1] JPEG : Joined Picture Expert Group

4.2 La côté serveur

Nous étions obligés de créer un serveur pour traiter des informations reçues du site web parce que PHP peut seulement envoyer des strings et l'IPC et la simulation veulent recevoir des bytes. En effet le code PHP envoie des commandements à un serveur qui nous appelons le « serveur site web ».

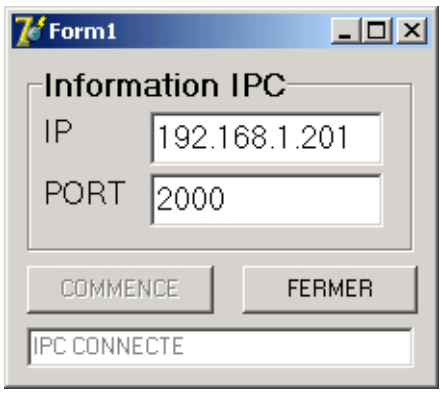


Fig. 4.3

Le serveur site web a besoin d'un « SocketServeur » ou le site web peut se connecter et un « Socket » qui peut se connecter sur l'IPC ou la simulation. Le fonctionnement de ces deux components est égal à ceux qui sont expliqués en haut.

```
procedure TForm1.choisir_message(flag:String);
begin
  if (flag = 'C1') then
    send_fw($C1);
end;
```

Le serveur site web reçoit des strings, il les convertit dans des bytes et il les renvoie au l'IPC ou la simulateur.

Références 5

Les sites web

APACHE

http://www.apache.org/	[Sauvegarde]
http://www.tweakzone.nl/tutorials/44/1	[Tutorial]

PHP

http://www.php.net/	[Sauvegarde]
http://aspn.activestate.com/ASPN/docs/PHP/class.orbitobject.html	[Manuel]
http://www.phphulp.nl/	[Manuel]
http://www.websitemaken.be/	[Manuel]
http://aspn.activestate.com/ASPN/	[Manuel]
http://www.php4ever.tk/	[Manuel]

VRML

http://vrmlworks.crispen.org/examples.html	[Exemples]
http://www.computertotaal.nl/cursussen/cst.cfm?id=56&sub=2167	[Manuel]
http://www.csv.ica.uni-stuttgart.de/vrml/dune/down.html	[Editor]
http://www.smeenk.com/	[Exemples]
http://home.snafu.de/hg/	[Viewer]
http://tecfa.unige.ch/guides/vrml/vrml2/spec/	[Manuel]
http://vrml.pagina.nl/	[Manuel]
http://www.lighthouse3d.com/vrml/tutorial/	[Tutorial]

JAVA

http://www.frontiernet.net/~imaging/java_vrml.html	[Java applets - VRML]
http://tecfa.unige.ch/guides/vrml/java/vrml_top.html	[Java - VRML]
http://www.vrmlsite.com/sep96/spotlight/javavrml/javavrml.html	[Java - VRML]
http://tecfa.unige.ch/guides/vrml/vrml2/spec/part1/vrmlscript.html	[Exemple]
http://www.j3d.org/faq/index.html	[Manuel]

DELPHI

http://www.blong.com/Conferences/DCon99/Corba/Corba.htm	[Corba]
http://www.mit.jyu.fi/~vesal/kurssit/winohj/delphi/	[Corba]
http://www.nldelphi.com/	[Forum]
http://www.overbyte.be/frame_index.html	[FPiette]
http://sheepdogguides.com/tut.htm	[Tutorials]
http://www.delphidabbler.com/	[Tutorials]
http://www.festra.com/introfr.htm	[Source]

GLSCENE

http://glscene.sourceforge.net/index.php	[Sauvegarde]
http://caperaven.co.za/	[Tutorials]

HTML

http://handleidinghtml.nl/	[Manuel]

PHOTOSHOP

http://www.absolutecross.com/	[Tutorials]
http://www.pixel2life.com/	[Tutorials]

Les livres

PHP

PHP5 SAMS Unleashed	ISBN 0-672-32511-X
PHP5 Superboek	ISBN 90-5940-137-9

DELPHI

Delphi 2 Secrets d'experts	ISBN 2-7440-0135-X
Delphi 2 PC Poche	ISBN 2-7429-0675-4
Delphi 2 Pour Windows 95/NT	ISBN 2-7361-2159-7
Delphi 7 programmeren	ISBN 90-430-0767-6

UML

Toepassing van UML	ISBN 90-395-1447-X

Liste des illustrations 6