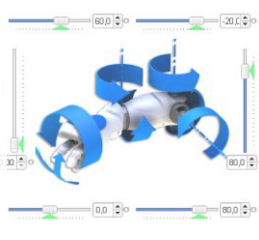
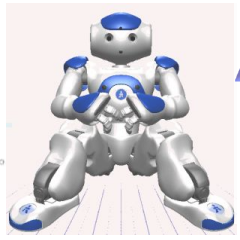




NAO communique avec des sockets

Comment faire communiquer le robot NAO avec des sockets…

Ce court tutoriel montre comment faire communiquer le robot NAO avec des applications externes via un réseau et l'utilisation de socket.

1 Ressources et remerciements

- Le livre Apprendre à programmer en python[1] de Gérard Swinnen en particulier le chapitre n°18 « Communications à travers un réseau. ».

- Le support de la société Aldebaran qui a développé et testé le module de mise en œuvre des sockets utilisé dans l'exemple de Choregraphe.

- Pour le serveur en Delphi avec la mise en œuvre des sockets http://delphipage.free.fr/sockets.html

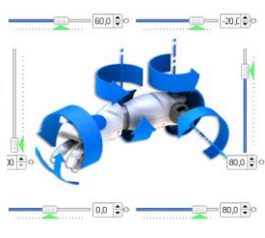
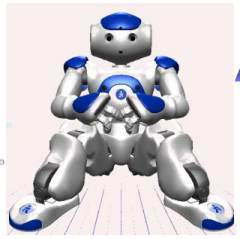
2 Objectif

Nous souhaitons que le robot NAO envoie des messages à une application externe via le réseau. Cette communication réseau utilise des sockets et une application serveur qui reçoit des informations d'une application client toutes les deux présentes sur le réseau.

La mise en œuvre expliquée dans ce document utilise une application serveur écrite en python et qui tourne sur un RaspberryPI. Elle a aussi été testée avec un serveur écrit en Delphi. Les deux fonctionnent de la même manière.

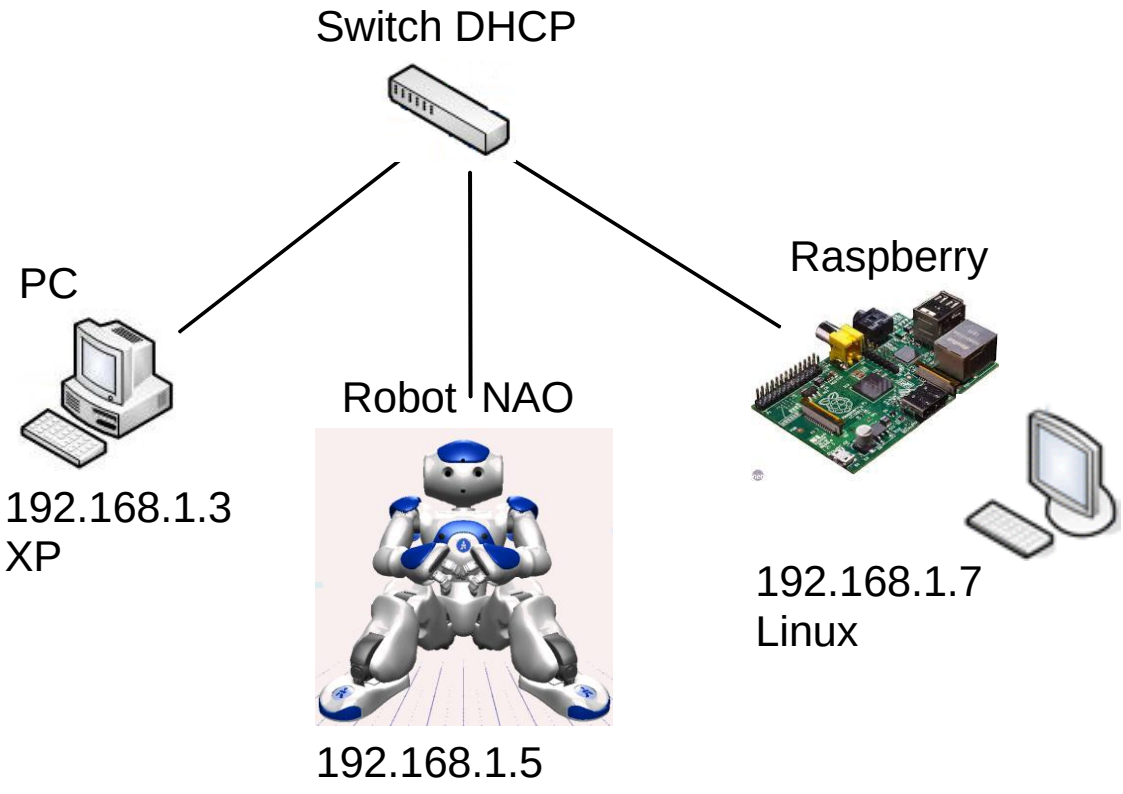
[1] Téléchargeable en effectuant une recherche avec google sur le titre et l'auteur du livre.

STI2D SIN

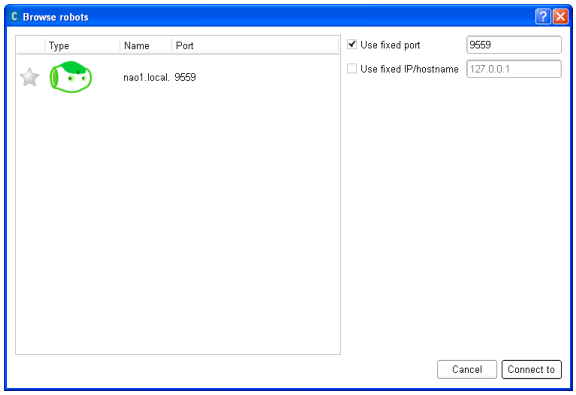

3 Le réseau

Le réseau est présenté ci-dessous, un serveur d'adresse DHCP attribue des adresses dans la plage 192.168.1.0/24.



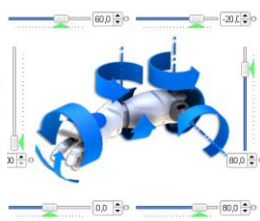
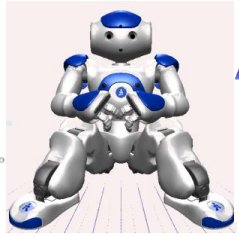
4 L'environnement de développement

Le serveur est écrit en python, il est très rudimentaire, il écoute sur le port 2000 le client qui se connecte et affiche les messages reçus. Pour simplifier le développement du code la carte Raspberry est télé opéré via Putty, les scripts python étant transférés via WinSCP. Le fonctionnement est synthétisé sur le schéma de la page suivante.



Le robot NAO est piloté via le logiciel Choregraphe depuis le PC.

STI2D SIN

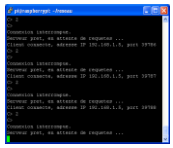




PC Base de développement

RaspberryPI modèle B

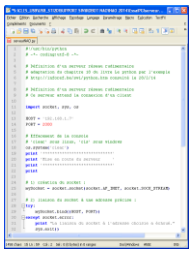


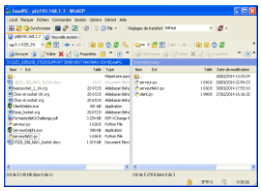
Putty



Exécution des scripts depuis le PC


Python


NotePad++
Edition des scripts


WinSCP
Transfert sur la carte SD
des scripts

Transfert des fichiers


Raspbian

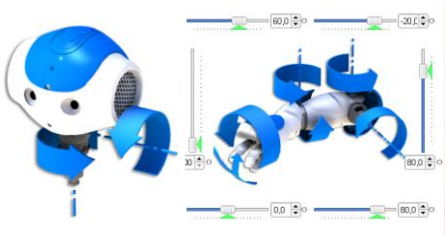
5 Le programme du serveur

Le programme est donné ci-dessous, une fois transféré sur le PI il est démarré par la commande :

sudo python serveurNAO.py (dans un sous-répertoire /reseau)

```
pi@raspberrypi: ~/reseau
login as: pi
pi@192.168.1.7's password:
Linux raspberrypi 3.10.25+ #622 PREEMPT Fri Jan 3 18:41:00 GMT 2014 armv6l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Mar 30 17:52:17 2014 from 192.168.1.3
pi@raspberrypi ~ $ cd reseau
pi@raspberrypi ~/reseau $ sudo python serveurNAO.py
```

STI2D SIN

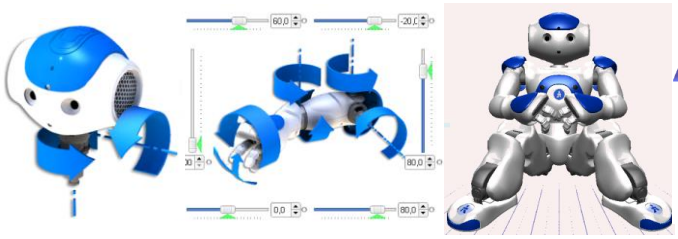


Le programme du serveur

serveurNAO.py

```
#!/usr/bin/python
# -*- coding:utf-8 -*-

# Définition d'un serveur réseau rudimentaire
# adaptation du chapitre 18 du livre Le python par l'exemple
# http://inforef.be/swi/python.htm consulté le 28/2/14

# Définition d'un serveur réseau rudimentaire
# Ce serveur attend la connexion d'un client

import socket, sys, os

HOST = '192.168.1.7'
PORT = 2000

# Effacement de la console
# 'clear' sous linux, 'cls' sous windows
os.system('clear')
print '*****************************'
print 'Mise en route du serveur     '
print '*****************************'
print

# 1) création du socket :
mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# 2) liaison du socket à une adresse précise :
try:
    mySocket.bind((HOST, PORT))
except socket.error:
    print "La liaison du socket à l'adresse choisie a échoué."
    sys.exit()
```

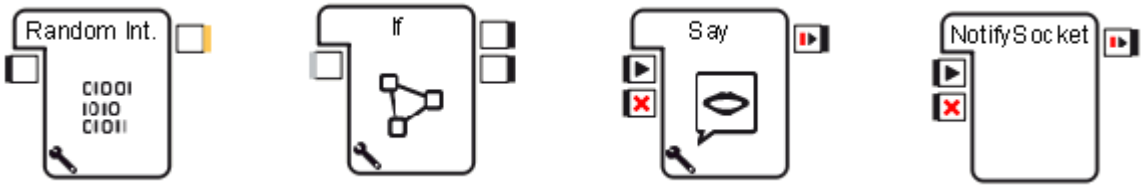
STI2D SIN



```
33
34  while 1:
35  # 3) Attente de la requête de connexion d'un client :
36      print "Serveur pret, en attente de requetes ..."
37      mySocket.listen(5)
38
39  # 4) Etablissement de la connexion :
40      connexion, adresse = mySocket.accept()
41      print "Client connecte, adresse IP %s, port %s" % (adresse[0], adresse[1])
42
43  # 5) Dialogue avec le client :
44      msgClient = connexion.recv(1024)
45      while 1:
46          print "C>", msgClient
47          if msgClient.upper() == "FIN" or msgClient =="":
48              break
49          msgClient = connexion.recv(1024)
50
51  # 6) Fermeture de la connexion :
52      connexion.send("Au revoir !")
53      print "Connexion interrompue."
54      connexion.close()
55
```

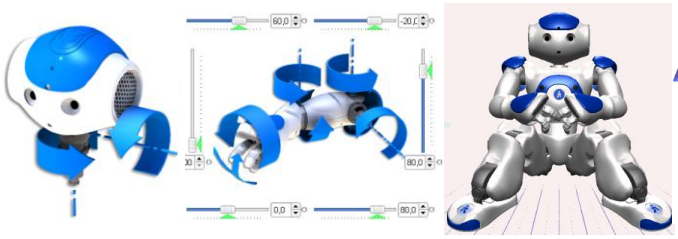
6 Le programme de démonstration NAO Choregraphe

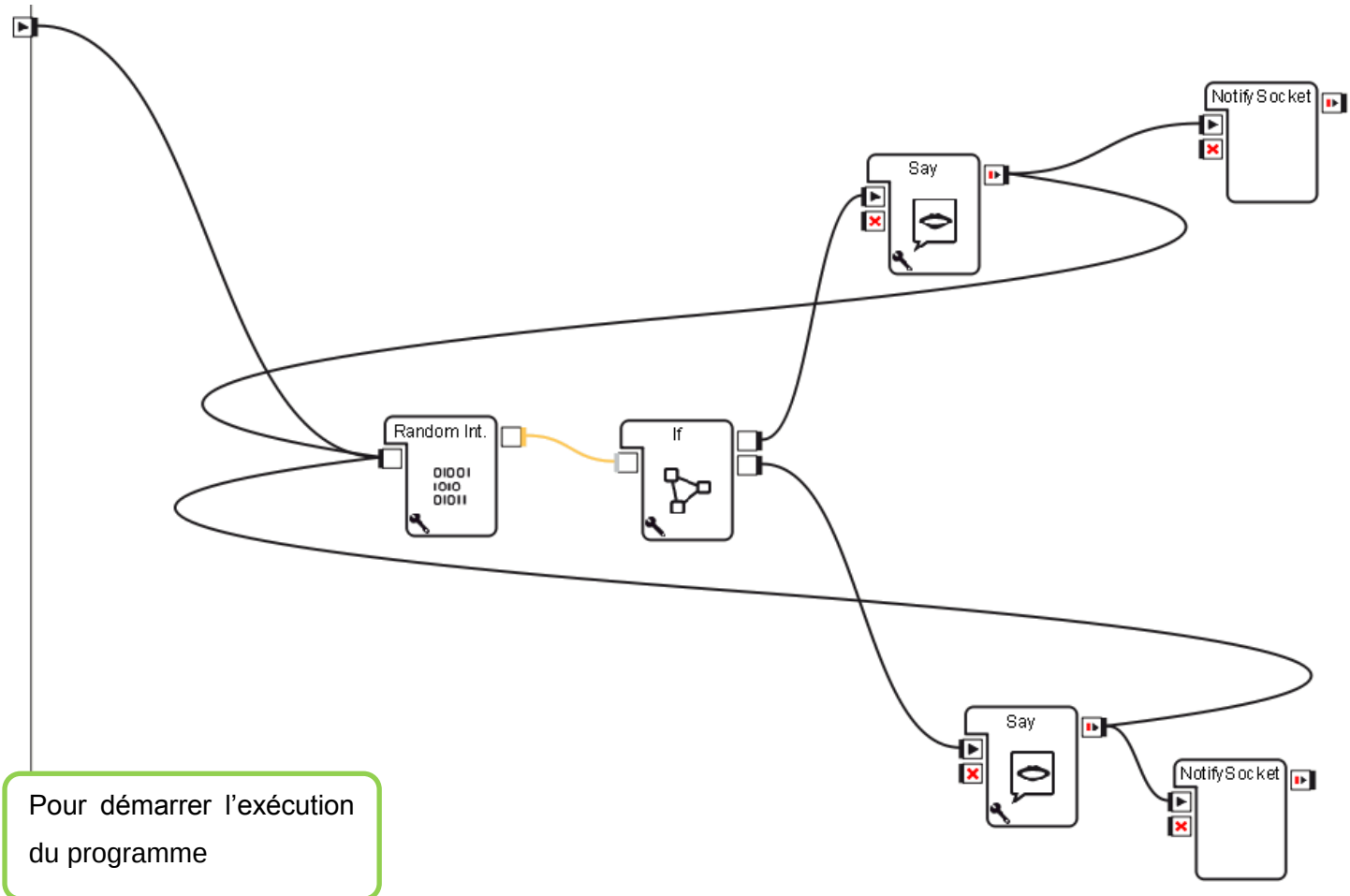
Pour la démonstration un nombre entier est tiré au sort puis selon sa valeur supérieure ou égale à 5 ou non un message vocal est émis et un message réseau est envoyé au serveur.



Math | Flow Control | Voice | Spécifique Aldebaran

STI2D SIN

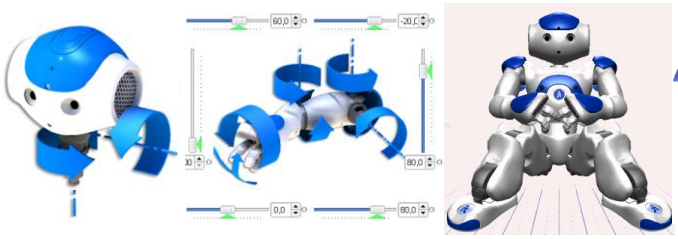




Pour démarrer l'exécution du programme

Pour stopper l'exécution du programme

STI2D SIN



7 Le programme client coté NAO sous Choregraphe

Voilà le code proposé par le support de la société Aldebaran :

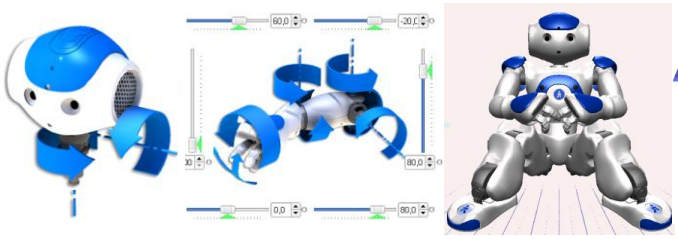
Script editor

If | NotifySocket

```
import socket

class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)

    def onLoad(self):
        #put initialization code here
        pass

    def onUnload(self):
        #put clean-up code here
        pass

    def onInput_onStart(self):
        #self.onStopped() #activate the output of the box
        HOST = '192.168.1.7'
        PORT = 2000

        mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        mySocket.connect((HOST, PORT))

        message = "2"
        mySocket.send(message)
        mySocket.close()

    def onInput_onStop(self):
        self.onUnload() #it is recommended to reuse the clean-up as the box is stopped
        self.onStopped() #activate the output of the box
```

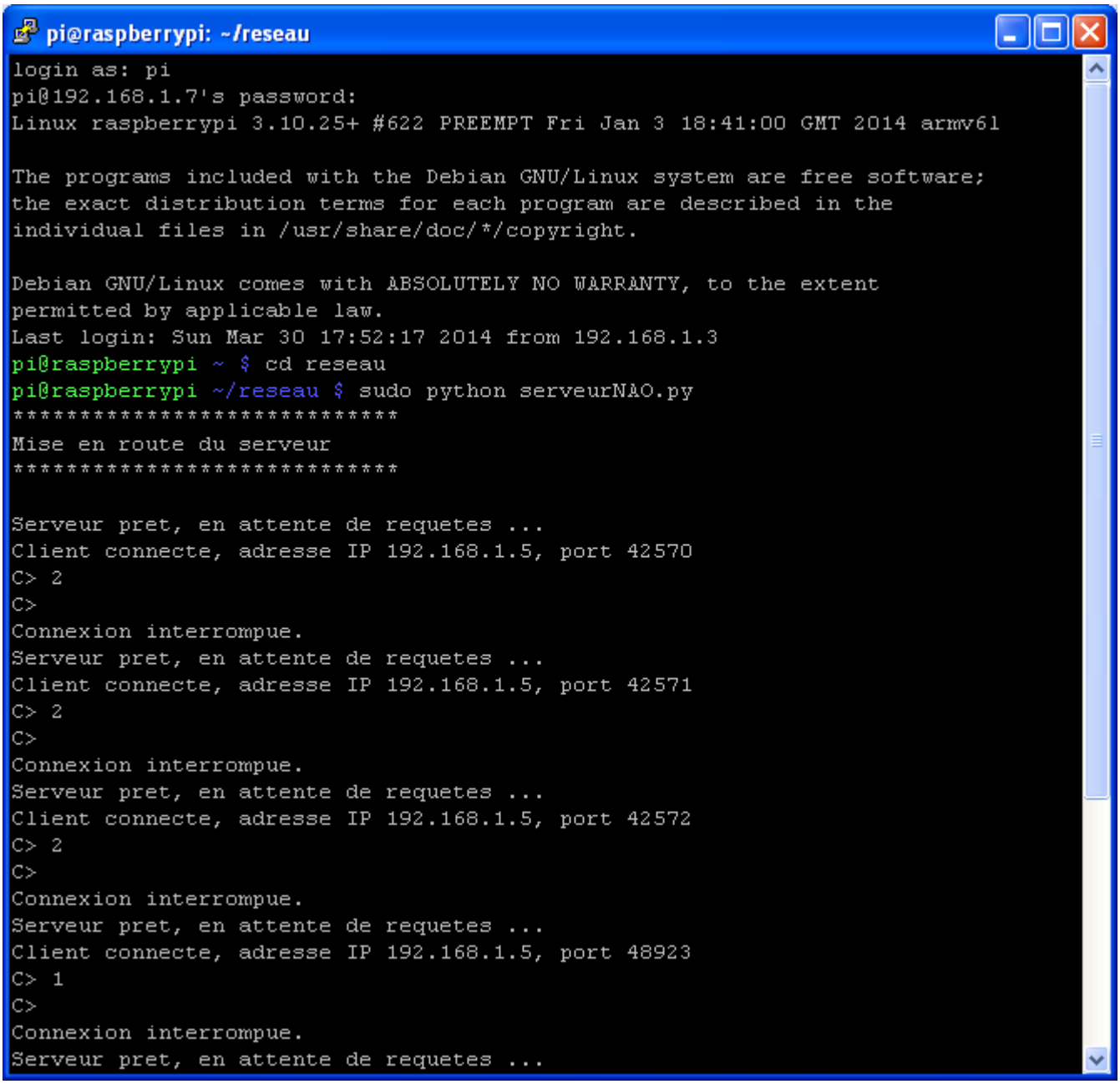
Find:

STI2D SIN

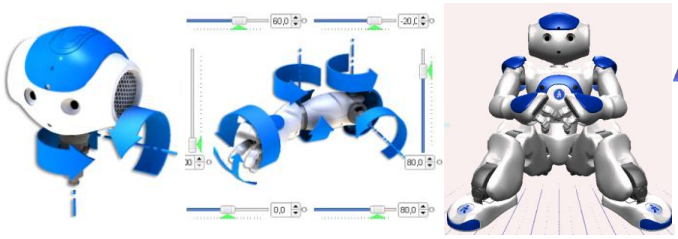


8 Résultat du fonctionnement

Pour chacun des messages émis vers le serveur celui-ci affiche le contenu reçut qui correspond au message vocal émis par NAO.



STI2D SIN

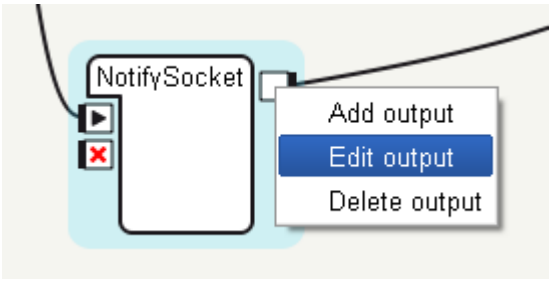


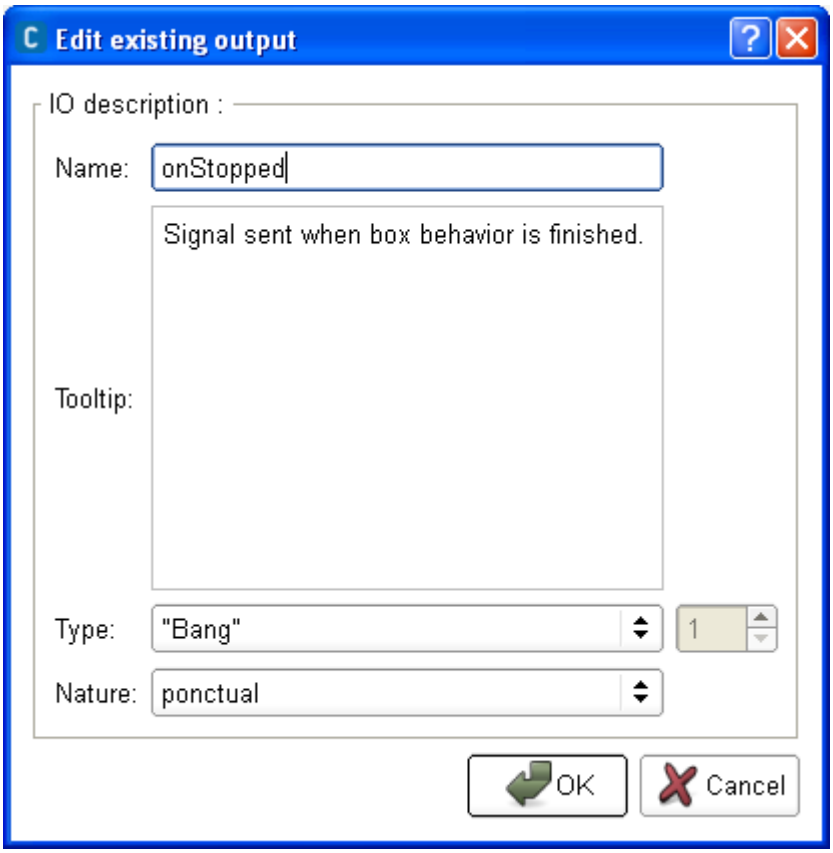
9 Amélioration du code Choregraphe

Le code précédent est fonctionnel, mais la boite NotifySocket ne retransmet pas le flux à la fin de son activation. Pour le modifier voilà la solution mise en œuvre :

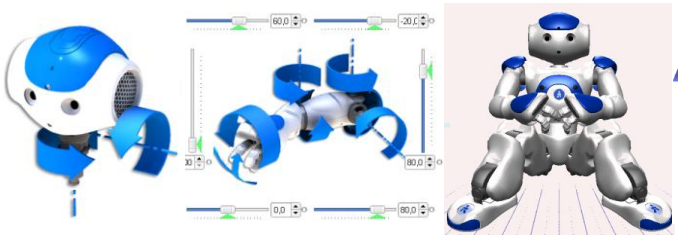
9.1 Modification des propriétés de la sortie

Un clic-droit sur la sortie permet de modifier sa nature et de le positionner sur normal :





STI2D SIN



Il faut ensuite ajouter la transmission du flux :

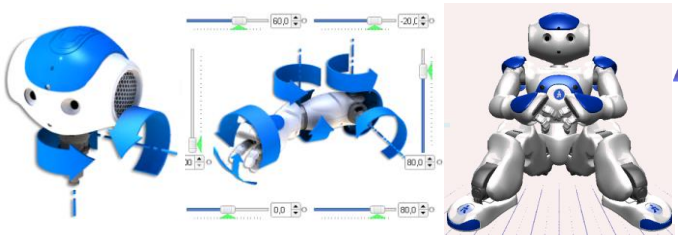
Script editor

NotifySocket

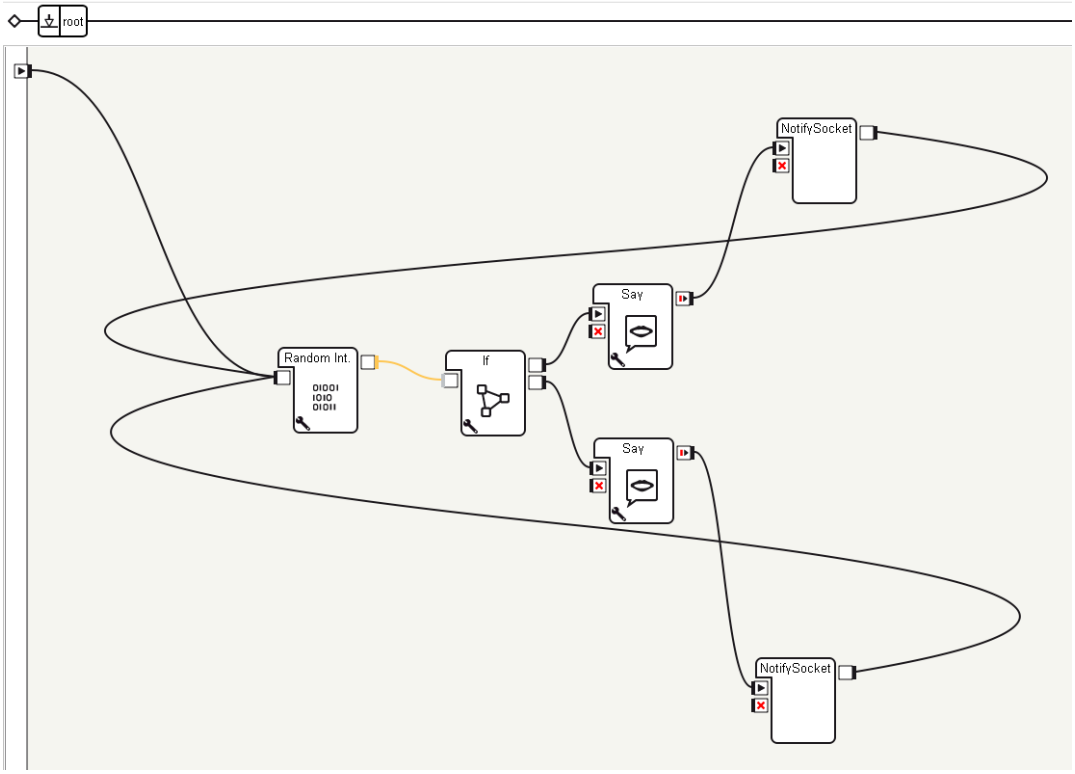
```
import socket

class MyClass(GeneratedClass):
    def __init__(self):
        GeneratedClass.__init__(self)

    def onLoad(self):
        #put initialization code here
        pass

    def onUnload(self):
        #put clean-up code here
        pass

    def onInput_onStart(self):
        #self.onStopped() #activate the output of the box
        HOST = '192.168.1.3'
        PORT = 2000

        mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        mySocket.connect((HOST, PORT))

        message = "2"
        mySocket.send(message)
        mySocket.close()
        self.onStopped()

    def onInput_onStop(self):
        self.onUnload() #it is recommended to reuse the clean-up as the box is stopped
        self.onStopped() #activate the output of the box
```

Ln 26 Find:

STI2D SIN



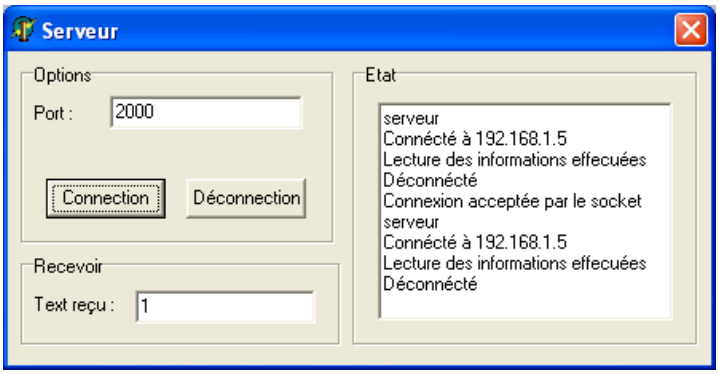
9.2 Le nouveau code Choregraphe :



L'enchaînement des box est plus simple les boites se succèdent les unes après les autres.

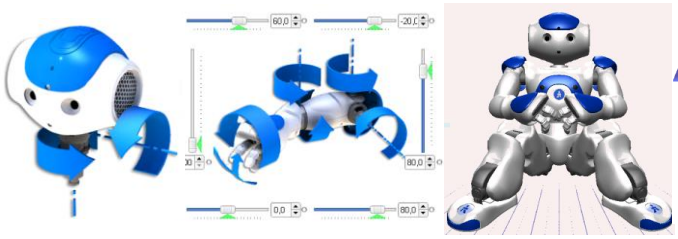
10 Mise en place du serveur[2] sur PC

Le serveur est démarré et se met en écoute sur le port 2000. Le message reçu est ensuite affiché dans le champ texte reçu.



[2] Voir le lien http://delphipage.free.fr/sockets.html

STI2D SIN



Indique un document ressource

Retour au sommaire

Retour à la page courante