CtrlCorba

Table des matières

1 Introduction

```
$ ./bin/Controller -n Security/Controller
                                          $ python gui/GUI.py
```

1.1 Envoie d'ordres

Les ordres transitent sous forme d'un pseudo codage XML semblable tout ou parti au contenu de fichier de configuration. Envoyer les ordres consiste à écrire ce texte dans une SOCKET.

1.2 Remontée des alarmes

Le client doit pouvoir s'inscrire au près du serveur afin de recevoir les alertes. A priori il s'agit du processus decrit ci-dessus mais inversé.

Remarque : notre client devient par conséquent également un serveur.

1.3 Récupération des valeurs

Afin de proposer un contrôle visuel, le client doit aussi être en mesure de récupérer les valeurs en vigueure sur le serveur. Celà consiste je suppose à envoyer un ordre puis à déchiffrer la chaîne reçue en retour.

2 Architecture du controleur Corba

2.1 Fichiers sources

```
ctrlSecurity
├── Makefile
├── idl
│   └── Security.idl
├── include .2 Controller.hh .1 src
│   └── Controller.C
├── scripts
│   ├── connect_controller.py
│   └── test_controller.py
└── gui
    └── GUI.py
```

FIGURE 1 : *fichiers source*

2.2 Interface Corba

Le Makefile génère deux interface en C++ et en PYTHON :

```
$ omniidl -bcxx -Wba -I/usr/local/hess/dash/idl -Cstub idl/Security.idl
$ omniidl -bpython -I/usr/local/hess/dash/idl idl/Security.idl
```

```
security
├── stub
│   ├── Security.hh
│   ├── SecuritySK.cc
│   └── SecurityDynSK.cc
├── Security_idl.py
├── Security
│   └── __init__.py
└── Security__POA
    └── __init__.py
```

FIGURE 2 : *fichiers de l'interface* CORBA

2.3 Serveur C++

Le Makefile génère deux objets et un executable *(cf cvs/dash/Makefile.dash)* :

```
g++ ... -c stub/SecuritySK.cc -o out/SecuritySK.o
g++ ... -c src/ctrlSecurity.C -o out/ctrlSecurity.o
g++ ... out/ctrlSecurity.o out/SecuritySK.o -o bin/ctrlSecurity
```

```
security
├── out
│   ├── SecuritySK.o
│   └── Controller.o
└── bin
    └── Controller
```

FIGURE 3 : *fichiers du serveur* C++

Notez bien le paramètre !

```
$ ~cvs/security/bin/Controller -n Security/Controller
```

```
  roche
└─ Security
   └─ Controleur
```

FIGURE 4 : *noms* CORBA *du serveur* C++

```
$ nameclt list
roche/

$ nameclt list roche
Security/

$ nameclt list roche/Security
Controller
```

2.4 Clients Python

Il y a redite puisque le python est un langage interprété.

```
  security
├─ scripts
│  ├─ connect_controller.py
│  └─ test_controller.py
├─ gui
└─ GUI.py
```

FIGURE 5 : *fichiers des clients* PYTHON

2.4.1 Simple connection

Ce script à la mérite d'être indépendant de la classe *Dash* dans le sens où il ne fait pas appel à ses fonctions pour initier la connexion au controleur CORBA.

fichier *cvs/security/scripts/connect_controller.py* :

```
...
from omniORB import CORBA
import CosNaming
import Security,Security__POA
import Dash

global orb
orb = CORBA.ORB_init(sys.argv, CORBA.ORB_ID)
poa = orb.resolve_initial_references("RootPOA")
rootContext=orb.resolve_initial_references("NameService")._narrow(CosNaming.NamingContext)
poaManager = poa._get_the_POAManager()
poaManager.activate()

name = []
name.insert(0,CosNaming.NameComponent("Controller", ""))
name.insert(0,CosNaming.NameComponent("Security", ""))
name.insert(0,CosNaming.NameComponent("roche", ""))
```

```
obj=rootContext.resolve(name)
controller = obj._narrow(Security.Controller)
if controller.GetState() == Dash.StateController.Safe:
    controller.Configure()

print "Position 1:",controller.GetPosition(1)
```

2.4.2 Interface non graphique

Ce script utilise les fonctions de la classe *Dash* pour établir la connection.

```
#!/usr/bin/env python

import os, sys

sys.path.append(os.environ["HESSUSER"])
sys.path.append(os.environ["HESSUSER"]+"/dash")
sys.path.append(os.environ["HOME"]+"/src/security")

from dash.python import StateController,Server,Message,LoopHandler
import Dash, Dash__POA
import Security, Security__POA

""" This is the Corba client"""
class SecurityUIClient(StateController.StateController_i):

    def __init__(self):

        StateController.StateController_i.__init__(
            self,
            name = "Security/UI",
            options = ["-ORBclientCallTimeOutPeriod","600000"]
            )

"""The main object gather Corba client and connected Corba server"""
class SecurityUI:

    def find_controller(self):
        """Client will try to find the controller """
        name = "Security/Controller"

        self.server = self.client.find_server(name)
        if self.server is not None:
            """Polymorphisme """
            self.server = self.server._narrow(Security.Controller)

    def __init__(self):

        self.client = SecurityUIClient()
        self.find_controller()

if __name__ == '__main__':
    os.environ["LC_ALL"] = "C"
```

```
    """ connect to the controller"""
    ui = SecurityUI()

    if ui.server == None:
        print "cannot contact Corba server"

    else:
        print "state :"
        ui.server.GetState()

    print ui.server

    print "Terminating (please ignore following insults)\n\t\t---\n"
```

```
  roche
└─Security
   └─UI
```

FIGURE 6 : *noms* CORBA *du client* PYTHON

```
$ nameclt list
roche/

$ nameclt list roche
Security/

$ nameclt list roche/Security
Controller
UI
```

2.4.3 Interface graphique

```
  roche
└─Security
   └─GUI
```

FIGURE 7 : *noms* CORBA *du* GUI PYTHON

```
$ nameclt list
roche/

$ nameclt list roche
Security/

$ nameclt list roche/Security
Controller
GUI
```

3 Implémentation du controleur Corba

3.1 Fonction ConfigureHardware