Corba

Table des matières

1 Introduction

L'architecture est déjà mise en place via un client/serveur générique.

2 Modèle client/serveur générique

Il faut comprendre que le controleur, tout comme son interface graphique sont à la fois client et serveur.

- Serveur spécifique
  - serveur de Socket par exemple
- Contrôleur
  - client Socket par exemple
  - serveur Corba
- GUI du contrôleur
  - client Corba
  - serveur Corba
- GUI caméra
  - client Corba

3 Interfaces Dash : :Controller

```
$ echo $HESSUSER
/home/roche/hess

$ cd ~/hess
$ cvs co dash
$ cd dash && make
```

L'interface spécifie l'objet vu par le serveur.
Cf fichier */hess/dash/idl/Dash.idl*
Les méthodes disponible pour la classe **Controller** sont implémentées ici :

- */hess/dash/python/Server.py* Cette classe implémente les mécanismes de base propre à chaque serveur CORBA.
  - Le constructeur contacte les servers POA (Portable Object Adapter).

    ```
    print "Initializing ORB"
    orb = CORBA.ORB_init(options, CORBA.ORB_ID)
    poa = orb.resolve_initial_references("RootPOA")
    rootContext=orb.resolve_initial_references("NameService")._narrow(CosNaming.NamingContex
    poaManager = poa._get_the_POAManager()
    poaManager.activate()
    thread.start_new_thread(startorb,())
    ```
  - `find_server(self,name)` : Find a server with a given name, and try to contact it

  Voici les méthodes publiques :
  - `Delete            // Terminate the server process`
  - `Ping              // Check if the connection is up`
  - `Initialized       // Check if the process is initialized ; To be overloaded`
- */hess/dash/python/Message.py*

  Cette classe ajoute une couche permettent le débugage équivalente au démon SYSLOG.

  Voici les méthodes publiques :
  - AddMessageReceiver
  - RemoveMessageReceiver
  - TakeMessage
  - Take
- */hess/dash/python/StateController.py*

  Cette classe fait de chaque controleur une machine à état.

  Par ailleurs, cet objet fait intervenir la notion de thread. Les commandes sont alors suspendues/annulées... par ce biais.

  Voici les méthodes publiques :
  - GetState
  - GetProcessStatus
  - SetProcessStatus
  - GetTransition
  - IsInTransition
  - Action
  - TakeRunParameters
  - GetRunParameters

```
Server
└─Message
  └─StateController
    └─Controller
```

Figure 1 : héritages

- GotoSafe
- Configure
- Start
- Stop
- Resume
- Interrupt
- IniGotoSafe
- IniConfigure
- IniStart
- IniStop
- IniPause
- IniResume

- **Controller**

  Cette classe virtuelle pure, en plus des classes décrites ci-dessus, hérite d'autres classes mais qui ne sont pas implémentées.

  Nos objets implémenterons cette classe.

remarque : En python les méthodes virtuelles pures peuvent ne pas être implémentées.

4 Utilisation de Dash

Nous intégrons l'architecure générique via la classe **Controller** :

- Dans notre interface IDL

```
#include "Dash.idl"

module Onlinecalibrator
{
  interface Controller: Dash::Controller
```

- Dans notre serveur, la place de la première classe héritée étant imposée (sinon le client ne se connecte pas), on en déduit que c'est elle qui joue le rôle déterminant.

```
from dash.python import StateController

class OnlinecalibratorController_i(Onlinecalibrator__POA.Controller,
                                   StateController.StateController_i):

    def __init__(self, name, ledServer, ledPort):
        StateController.StateController_i.__init__(self,name,options=["-ORBclientCallTimeOut
```

- Dans notre client on retrouve également l'implémentation de la classe `StateController`. La méthode `find_server` de l'instance locale (`self.controller`) permet d'initialiser l'instance distante (`self.server`). Dès lors l'instance locale ne sert plus.

```
from dash.python import StateController,Server,Message
import Dash, Dash__POA
```

```
class OnlinecalibratorUIController(StateController.StateController_i):

    def __init__(self,
                 name = "Onlinecalibrator/GUI",
                 options = ["-ORBclientCallTimeOutPeriod","600000"]):

        StateController.StateController_i.__init__(self,name = name,options = options)


class OnlinecalibratorUI:

    def __init__(self,name = "Onlinecalibrator/GUI"):

        self.controller = OnlinecalibratorUIController(name = name)
        self.find_controller()

    def find_controller(self):
        """Try to find the controller """
        name = "Onlinecalibrator/Controller"

        self.server = self.controller.find_server(name)
        if self.server is not None:
            self.server = self.server._narrow(Onlinecalibrator.Controller)
```

Un exemple d'utilisation est donné dans la section contrôleur