

Documentación de código

Alegandro Gallego, Sergio Pérez, Jhon Ramírez

Aqui va la fecha

1. Documentación

A continuación se presenta la descripción del comportamiento de cada una de los archivos del proyecto, así como el una descripción del objetivo que se persigue en cada método, incluyendo la descripción de sus parámetros de entrada, y sus parámetros de salida que retorna una vez termina su ejecución.

1.1. Agente.py

Representa un agente en el sistema. Está conformado por un componente de **Racionalidad**, y un componente de **Movilidad**, tiene la característica de dispersarse en la red, esto es, mover cada una de sus componentes a otros nodos en la red, cada cierto intervalo de tiempo, determinado por una función de distribución de probabilidad.

```
1 import Pyro4
2 import threading
3 import time
4
5 class Agente(object):
6
7     tipoMovilidad = ["constante", "uniforme", "exponencial"]
8
9     def __init__(self, nombre, movilidadId, racionalidadId, hostUri)
10         :
11         self.hostUri = hostUri
12         self.nombre = nombre
13         self.movilidadId = movilidadId
14         self.racionalidadId = racionalidadId
15         thread = threading.Thread(target = self.wait2Seconds, args
16             = [])
17         thread.start()
18
19     def getMovilidadId(self):
20         return self.movilidadId
21
22     def getRacionalidadId(self):
23         return self.racionalidadId
```

```

23     def getNombre(self):
24         return self.nombre
25
26     def getType(self):
27         return 'head'
28
29     def getPyroId(self):
30         return str(self._pyroId)
31
32     def doIt(self):
33         ##place some call to legs and arms
34         racionalidadUri = Pyro4.Proxy(self.hostUri).resolve(self.
            racionalidadId)
35         movilidadUri = Pyro4.Proxy(self.hostUri).resolve(self.
            movilidadId)
36         if (racionalidadUri == False or movilidadUri == False):
37             return 'Algo esta perdido'
38         racionalidad = Pyro4.Proxy(racionalidadUri)
39         movilidad = Pyro4.Proxy(movilidadUri)
40         return [racionalidad.sayArms(), movilidad.sayLegs()]

```

Métodos

1.1.1. `init(nombre, movilidadId, racionalidadId, hostUri)`

Es el constructor de la clase, y a través de este método se permite instanciar objetos de esta clase.

```

1 def __init__(self,nombre, movilidadId, racionalidadId, hostUri):
2     self.hostUri = hostUri
3     self.nombre = nombre
4     self.movilidadId = movilidadId
5     self.racionalidadId = racionalidadId

```

Parámetros

- *nombre*: Nombre con el que se identifica el Agente en la red, debe ser único.
- *movilidadId*: Identificador del objeto que representa el componente correspondiente a la capacidad del agente de moverse en la red.
- *racionalidadId*: Identificador del objeto que representa el componente correspondiente a la capacidad del agente de tomar decisiones.
- *hostUri*: Identificador de recurso uniforme del *Host* donde está alojado el agente en un instante de tiempo específico.

1.1.2. `getMovilidad()`

Permite a los demás objetos en la red obtener el identificador del objeto de la movilidad del agente.

```
1 def getMovilidadId(self):  
2     return self.movilidadId
```

Retorno

Retorna el identificador del objeto que representa el componente correspondiente a la capacidad del agente de moverse en la red.

1.1.3. getRacionalidad()

Permite a los demás objetos en la red obtener el identificador del objeto de la racionalidad del agente.

```
1 def getRacionalidadId(self):  
2     return self.racionalidadId
```

Retorno

Retorna el identificador del objeto que representa el componente correspondiente a la capacidad del agente de tomar decisiones.

1.1.4. getNombre()

Permite a los demás objetos en la red obtener el nombre del agente.

```
1 def getNombre(self):  
2     return self.nombre
```

Retorno

Retorna el nombre con el que se puede identificar el agente en la red.

1.1.5. getType()

Permite a los demás objetos en la red reconocer si este objeto es la cabeza (componente fundamental) del agente.

```
1 def getType(self):  
2     return 'head'
```

Retorno

Retorna la cadena de texto "head" que indica que este objeto es la cabeza del agente.

1.1.6. getPyroId()

Permite a los demás objetos en la red obtener el identificador único del agente en el daemon.

```
1 def getPyroId(self):  
2     return str(self._pyroId)
```

Retorno

Retorna el identificador único del agente.

1.1.7. doIt()

Permite identificar en que host de la red se encuentra el componente de racionalidad del agente, y en que host de la red se encuentra el componente de movilidad del agente.

```
1 def doIt(self):  
2     ##place some call to legs and arms  
3     racionalidadUri = Pyro4.Proxy(self.hostUri).resolve(self.  
4         racionalidadId)  
5     movilidadUri = Pyro4.Proxy(self.hostUri).resolve(self.  
6         movilidadId)  
7     if (racionalidadUri == False or movilidadUri == False):  
8         return 'Algo esta perdido'  
9     racionalidad = Pyro4.Proxy(racionalidadUri)  
10    movilidad = Pyro4.Proxy(movilidadUri)  
11    return [racionalidad.sayArms(), movilidad.sayLegs()]
```

Retorno

Retorna una lista, con dos elementos: una cadena de texto con la ubicación en la red del componente de racionalidad, y una cadena de texto con la ubicación en la red del componente de movilidad. Si no encuentra alguno de los componentes que conforman el agente, se retorna "Algo está perdido".

1.2. ComunidadAgentes.py

Representa una comunidad de agentes para un servicio específico, es decir, una comunidad de agentes es un conjunto de agentes que tienen la responsabilidad de garantizar la prestación de un servicio determinado entre uno o más dispositivos en la red.

```
1 from Agente import Agente  
2 from Servicio import Servicio  
3  
4 import Pyro4  
5  
6 # we're using custom classes, so need to use pickle  
7 Pyro4.config.SERIALIZER='pickle'
```

```

8
9 # we're using custom classes, so need to use pickle
10 Pyro4.config.SERIALIZERS_ACCEPTED.add('pickle')
11
12 class ComunidadAgentes(object):
13     idAgente = 1
14     def __init__(self, nombre):
15         self.__nombre = nombre
16         ComunidadAgentes.idAgente+=1
17         self.agente = {}
18
19     def setServicio(self, servicio):
20         print "Inicializo servicio"
21         self.servicio = servicio
22
23     def getServicio(self):
24         return self.servicio
25
26     def getNombre(self):
27         return self.__nombre
28
29     def addAgente(self, agente, nombre):
30         self.agente[nombre] = agente
31
32     def getAgente(self, nombre):
33         return self.agente[nombre]

```

Métodos

1.2.1. `init(nombre)`

Es el constructor de la clase, y a través de este método se permite instanciar objetos de esta clase.

```

1 def __init__(self, nombre):
2     self.__nombre = nombre
3     ComunidadAgentes.idAgente+=1
4     self.agente = {}

```

Parámetros

- *nombre*: Nombre con el que se identifica la comunidad de agentes en la red, debe ser único.

1.2.2. `setServicio(servicio)`

Permite asignar el servicio que la comunidad de agentes debe proveer.

```

1 def setServicio(self, servicio):
2     print "Inicializo servicio"
3     self.servicio = servicio

```

Parámetros

- *servicio*: Nombre del servicio que la comunidad de agentes debe proveer

1.2.3. `getServicio()`

Permite a los demás objetos en la red obtener el nombre del servicio que la comunidad de agentes suministra.

```
1 def getServicio(self):  
2     return self.servicio
```

Retorno

Retorna el nombre del servicio que la comunidad de agentes debe proveer a los dispositivos en la red

1.2.4. `getNombre()`

Permite a los demás objetos en la red obtener el nombre de la comunidad de agentes.

```
1 def getNombre(self):  
2     return self._nombre
```

Retorno

Retorna el nombre con el que se puede identificar la comunidad de agentes en la red.

1.2.5. `addAgente(agente,nombre)`

Permite agregar un nuevo agente a la comunidad de agentes con nombre "*nombre*".

```
1 def addAgente(self, agente, nombre):  
2     self.agente[nombre] = agente
```

Parámetros

- *agente*: Instancia de la clase Agente. Representa el nuevo agente que será agregado a la comunidad de agentes.
- *nombre*: Nombre del nuevo agente que formará parte de la comunidad de agentes.

1.2.6. getAgente()

Permite a los demás objetos en la red obtener el agente cuyo nombre coincide con "*nombre*", y que a su vez hace parte de la comunidad de agentes.

```
1 def getAgente(self, nombre):  
2     return self.agente[nombre]
```

Parámetros

- *nombre*: Nombre del agente que se desea buscar en la comunidad de agentes.

Retorno

Retorna un objeto de la clase agente, el cuál es el resultado de la búsqueda en la comunidad de agentes de un agente que tenga como nombre "*nombre*".

1.3. CrearHost.py

Esta clase se encarga de publicar una instancia de la clase "*Host*" (Objeto python regular) para que pueda ser accesible remotamente, es decir convierte dicho objeto en un "**Objeto Pyro**".

De manera muy sencilla el procedimiento es el siguiente: se crean uno o más objetos, los cuales se desean publicar como objetos Pyro, se crea un "*daemon*", se registran los objetos con este, y se inicia el ciclo de las peticiones de dicho daemon.

```
1 import Pyro4  
2 from Agente import Agente  
3 from Servicio import Servicio  
4 from ComunidadAgentes import ComunidadAgentes  
5 from Host import Host  
6  
7 HostLogik = Host("Logik")  
8 #Metodo pyro  
9 #Se crea de la manera simple con serveSimple, sin tener en cuenta  
   el host.  
10 Pyro4.Daemon.serveSimple({  
11     HostLogik : "host." + str(HostLogik.getNombre())  
12 }, host="0.0.0.0")
```

Métodos

1.3.1. serveSimple(*objetos*,*host*)

Permite exponer un objeto regular de Python como un objeto Pyro, de tal manera que sea accesible remotamente a los demás objetos en la red.

```

1 Pyro4.Daemon.serveSimple({
2     HostLogik : "host." + str(HostLogik.getNombre())
3 }, host="0.0.0.0")

```

Parámetros

- *objects*: Es un diccionario que contiene los objetos que serán expuestos, los cuales serán registrados como llave, y los nombres de dichos objetos, que serán registrados como valores.
- *host*: Es el host donde se iniciará el daemon.

1.4. Host.py

Representa un host en el sistema, es decir, un nodo de la red (instancias de esta clase). Tiene la propiedad de almacenar en su interior una lista de los diferentes componentes que conforman un agente (cabeza, movilidad y racionalidad), así como una lista de los name server que define Pyro.

```

1 import Pyro4
2 import Agente
3 import Racionalidad
4 import Movilidad
5 import random
6
7 class Host(object):
8
9     def getNombre(self):
10         return self.nombre
11
12     def __init__(self, nombre):
13         self.nombre = nombre
14         self.listNS = {}
15         self.listAgentes = {}
16         self.listMovilidad = {}
17         self.listRacionalidad = {}
18
19     def resolve(self, name):#had to add this methos on the host, so
        it can act sorta like a NameServer
20         ret = self.find(name)
21         if(ret == False):
22             print('Precaucion!: objeto no esta en el host. Esto
                puede afectar el rendimiento.')
23             for nameServer, nameServer_uri in self.listNS.items():
24                 try:
25                     findHost = Pyro4.Proxy(Pyro4.Proxy(
                        nameServer_uri).list()['host.' + self.
                        nombre])
26                     ret = findHost.find(name)
27                 except:
28                     print "Error: no se localizo el host"
29                     ret = False
30             if(ret!=False):

```



```

31         return ret
32     return ret
33
34
35
36 def find(self, name):
37     """ returns uri of object or false """
38     try:
39         return self.listAgentes[name]
40     except:
41         try:
42             return self.listMovilidad[name]
43         except:
44             try:
45                 return self.listRacionalidad[name]
46             except:
47                 return False
48
49
50 def getListNS(self):
51     return self.listNS;
52
53 def addNS(self, ip, ns):
54     """ """
55     try:
56         if self.listNS[ip]:
57             print "El NS existe en la lista"
58     except KeyError:
59         self.listNS[ip] = "PYRO:Pyro.NameServer@" + str(ip) +
60             ":" + str(ns.port)
61
62 def setListNS(self, lNS):
63     self.listNS = lNS
64
65 def deleteNS(self, nombre):
66     """ """
67     try:
68         del self.listNS[nombre]
69         print "Se elimino el NS de la lista"
70     except KeyError:
71         print "No existe el NS en la lista"
72
73 def getListAgentes(self):
74     """ """
75     return self.listAgentes;
76
77 def addAgente(self, agente, create = True):
78     if(self.resolve(agente) == False):
79         movilidadId = 'legs_' + agente
80         racionalidadId = 'arms_' + agente
81         hostUri = 'PYRO:' + self._pyroId + '@' + self.
            _pyroDaemon.locationStr
82         agent = Agente.Agente(agente, movilidadId,
83                                racionalidadId, hostUri)
84         if(create):
85             self.addRacionalidad(racionalidadId)
86             self.addMovilidad(movilidadId)

```

```

85
86         print('Adding head ' + agent.getNombre() + ' to Daemon
            in ' + str(self._pyroDaemon.locationStr))
87         uri = self._pyroDaemon.register(agent)
88         self.listAgentes[agent.getNombre()] = uri.asString()
89         return uri.asString()
90     else:
91         #Corregir
92         return self.resolve(agente)
93
94     def setListAgente(self, lAgente):
95         self.listAgentes = lAgente
96
97     def deleteAgente(self, nombre):
98         uri = self.listAgentes[nombre]
99         print('Removing ' + nombre + ' from Daemon at ' + self.
              _pyroDaemon.locationStr)
100        self._pyroDaemon.unregister(uri[5:uri.find('@')])
101        self.listAgentes.pop(nombre)
102
103    def getListMovilidad(self):
104        """ """
105        return self.listMovilidad;
106
107    def addMovilidad(self, movilidadId):
108        movilidad = Movilidad.Movilidad(movilidadId)
109        print('Adding Movility ' + movilidad.getId() + ' to Daemon
            in ' + str(self._pyroDaemon.locationStr))
110        uri = self._pyroDaemon.register(movilidad)
111        self.listMovilidad[movilidad.getId()] = uri.asString()
112
113    def setListMovilidad(self, lMovilidad):
114        self.listMovilidad = lMovilidad
115
116    def deleteMovilidad(self, nombre):
117        uri = self.listMovilidad[nombre]
118        print('Removing ' + nombre + ' from Daemon at ' + self.
              _pyroDaemon.locationStr)
119        self._pyroDaemon.unregister(uri[5:uri.find('@')])
120        self.listMovilidad.pop(nombre)
121
122
123
124    def getListRacionalidad(self):
125        return self.listRacionalidad
126
127
128    def addRacionalidad(self, racionalidadId):
129        racionalidad = Racionalidad.Racionalidad(racionalidadId)
130        print('Adding rationality ' + racionalidad.getId() + ' to
            Daemon in ' + str(self._pyroDaemon.locationStr))
131        uri = self._pyroDaemon.register(racionalidad)
132        self.listRacionalidad[racionalidad.getId()] = uri.asString()
133
134    def setListRacionalidad(self, lRacionalidad):
135        self.listRacionalidad = lRacionalidad

```

```

136
137 def deleteRacionalidad(self, nombre):
138     uri = self.listRacionalidad[nombre]
139     print('Removing ' + nombre + ' from Daemon at ' + self.
        _pyroDaemon.locationStr)
140     self._pyroDaemon.unregister(uri[5:uri.find('@')])
141     self.listRacionalidad.pop(nombre)
142
143 def moveAgente(self, nombre, hostTo):
144     try:
145         uri = self.listAgentes[nombre]
146         print('Moviendo ' + nombre + ' to ' + hostTo + '...')
147         self.deleteAgente(nombre)
148         newHost = Pyro4.Proxy(Pyro4.locateNS(hostTo).list()[
            host.' + self.nombre])
149         return newHost.addAgente(nombre, False)
150     except:
151         print('No existe el agente en este Host')
152
153 def moveMovilidad(self, nombre, hostTo):
154     try:
155         uri = self.listMovilidad[nombre]
156         print('Moviendo ' + nombre + ' to ' + hostTo + '...')
157         self.deleteMovilidad(nombre)
158         newHost = Pyro4.Proxy(Pyro4.locateNS(hostTo).list()[
            host.' + self.nombre])
159         newHost.addMovilidad(nombre)
160     except:
161         print('No existe la movilidad en este Host')
162
163 def moveRacionalidad(self, nombre, hostTo):
164     try:
165         uri = self.listRacionalidad[nombre]
166         print('Moviendo ' + nombre + ' to ' + hostTo + '...')
167         self.deleteRacionalidad(nombre)
168         newHost = Pyro4.Proxy(Pyro4.locateNS(hostTo).list()[
            host.' + self.nombre])
169         newHost.addRacionalidad(nombre)
170     except:
171         print('No existe la racionalidad en este Host')
172
173 def disperseAgente(self, agentId):
174     agent = Pyro4.Proxy(self.listAgentes[agentId])
175     movilidadId = agent.getMovilidadId()
176     racionalidadId = agent.getRacionalidadId()
177     self.moveMovilidad(movilidadId, random.sample(self.listNS.
        keys(), 1)[0])
178     self.moveRacionalidad(racionalidadId, random.sample(self.
        listNS.keys(), 1)[0])
179     return self.moveAgente(agentId, random.sample(self.listNS.
        keys(), 1)[0])
180
181
182 #Servicio
183 def searchHead(self, name):#had to add this methos on the host,
    so it can act sorta like a NameServer
184     ret = self.findHead(name)

```

```

185         if(ret == False):
186             print('Precaucion!: Head del objeto no esta en el host.
              Esto puede afectar el rendimiento.')
187         for nameServer,nameServer_uri in self.listNS.items():
188             try:
189                 findHost = Pyro4.Proxy(Pyro4.Proxy(
                    nameServer_uri).list()['host.' + self.
                    nombre])
190                 ret = findHost.findHead(name)
191             except:
192                 print "Error: no se localizo el host"
193                 ret = False
194             if(ret != False):
195                 return [ret,Pyro4.Proxy(nameServer_uri).list()['
                    host.' + self.nombre]]
196         return ret
197
198     def findHead(self, name):
199         """ returns uri of object or false """
200         try:
201             return self.listAgentes[name]
202         except:
203             return False
204
205     def searchMovilidad(self, name):#had to add this methos on the
206         host, so it can act sorta like a NameServer
207         ret = self.findMovilidad(name)
208         if(ret == False):
209             print('Precaucion!: Movilidad del objeto no esta en el
              host. Esto puede afectar el rendimiento.')
210         for nameServer,nameServer_uri in self.listNS.items():
211             try:
212                 findHost = Pyro4.Proxy(Pyro4.Proxy(
                    nameServer_uri).list()['host.' + self.
                    nombre])
213                 ret = findHost.findMovilidad(name)
214             except:
215                 print "Error: no se localizo el host"
216                 ret = False
217             if(ret != False):
218                 return [ret,Pyro4.Proxy(nameServer_uri).list()['
                    host.' + self.nombre]]
219         return ret
220
221     def findMovilidad(self, name):
222         """ returns uri of object or false """
223         try:
224             return self.listMovilidad[name]
225         except:
226             return False
227
228     def searchRacionalidad(self, name):#had to add this methos on
229         the host, so it can act sorta like a NameServer
230         ret = self.findRacionalidad(name)
231         if(ret == False):

```

```

232         print('Precaucion!: Movilidad del objeto no esta en el
233               host. Esto puede afectar el rendimiento.')
234     for nameServer, nameServer_uri in self.listNS.items():
235         try:
236             findHost = Pyro4.Proxy(Pyro4.Proxy(
237                 nameServer_uri).list()['host.' + self.
238                 nombre])
239             ret = findHost.findRacionalidad(name)
240         except:
241             print "Error: no se localizo el host"
242             ret = False
243         if(ret != False):
244             return ret
245     return ret
246
247 def findRacionalidad(self, name):
248     """ returns uri of object or false """
249     try:
250         return self.listRacionalidad[name]
251     except:
252         return False
253
254 #Funciones para la comunidad de agentes.
255 def retrieveAgente(self, agentId):
256     #Primero se recupera la cabeza o encabezado del agente.
257     agent = self.searchHead(agentId)
258     if(agent == False):
259         print 'No se encontro el agente'
260     else:
261         print 'Moviendo el encabezado al host ' + self.nombre
262         agent = self.moveAgente(agentId, self.listNS[self.nombre
263         ])
264         movilidadId = Pyro4.Proxy(agent).getMovilidadId()
265         [movilidad, hostMovilidad] = self.searchMovilidad(
266             movilidadId)
267         if(movilidad == False):
268             print 'No se encontro la movilidad '
269             print 'Dado que no se encontro la movilidad no se
270             procedera a encontrar la racionalidad'
271         else:
272             self.moveMovilidad(movilidadId, self.listNS[self.
273             nombre])
274             racionalidadId = Pyro4.Proxy(agent).
275             getRacionalidadId()
276             try:
277                 Pyro4.Proxy(hostMovilidad).getListMovilidad()[
278                 racionalidadId]
279                 self.moveRacionalidad(racionalidadId, self.
280                 listNS[self.nombre])
281             except:
282                 'La racionalidad no se encuentra con la
283                 movilidad por lo tanto no se podra mover'
284
285     return agent

```

Métodos

1.4.1. getNombre()

Permite a los demás objetos en la red obtener el nombre del host.

```
1 def getNombre( self ) :  
2     return self.nombre
```

Retorno

Retorna el nombre con el que se puede identificar el host en la red.

1.4.2. init(*nombre*)

Es el constructor de la clase, y a través de este método se permite instanciar objetos de esta clase.

```
1 def __init__(self , nombre):  
2     self.nombre = nombre  
3     self.listNS = {}  
4     self.listAgentes = {}  
5     self.listMovilidad = {}  
6     self.listRacionalidad = {}
```

Parámetros

- *nombre*: Nombre con el que se identifica el host en la red, debe ser único.

1.4.3. resolve(*name*)

Realiza una búsqueda en todos los nodos que conforman la red permitiendo a los demás objetos en la red obtener el identificador de recurso uniforme (uri) del objeto cuyo nombre coincida con "*name*".

```
1 def resolve(self , name):  
2     ret = self.find(name)  
3     if(ret == False):  
4         print('Precaucion!: objeto no esta en el host. Esto puede  
5             afectar el rendimiento.')6         for nameServer, nameServer_uri in self.listNS.items():  
7             try:  
8                 findHost = Pyro4.Proxy(Pyro4.Proxy(nameServer_uri).list()[ '  
9                     host.' + self.nombre])  
10                ret = findHost.find(name)  
11            except:  
12                print "Error: no se localizo el host"  
13                ret = False  
14                if (ret!=False):  
15                    return ret  
16        return ret
```

Parámetros

- *name*: Nombre del objeto que se desea encontrar en la red.

Retorno

Retorna el identificador de recurso uniforme (uri) del objeto cuyo nombre coincida con el parámetro "*name*". Si no se encuentra en la red dicho objeto, es decir, en ningún host, entonces se retorna Falso.

1.4.4. find(*name*)

Realiza una búsqueda en este host permitiendo a los demás objetos en la red obtener el identificador de recurso uniforme (uri) del objeto cuyo nombre coincida con "*name*".

```
1 def find(self, name):  
2     try:  
3         return self.listAgentes[name]  
4     except:  
5         try:  
6             return self.listMovilidad[name]  
7         except:  
8             try:  
9                 return self.listRacionalidad[name]  
10            except:  
11                return False
```

Parámetros

- *name*: Nombre del objeto que se desea encontrar en el host.

Retorno

Retorna el identificador de recurso uniforme (uri) del objeto cuyo nombre coincida con el parámetro "*name*" que se encuentre ubicado en el host. Si no se encuentra en el host dicho objeto se retorna Falso.

1.4.5. getListNS()

Permite a los demás objetos en la red obtener la lista de los name servers definida por Pyro. En cada name server se encuentra la dirección de cada uno de los hosts que se encuentran distribuidos en la red.

```
1 def getListNS(self):  
2     return self.listNS;
```

Retorno

Retorna la lista de los name server existentes hasta el momento.

1.4.6. addNS(*host*, *ns*)

Añade a la lista de name servers del host un nuevo name server.

```
1 def addNS(self, ip, ns):
2     try:
3         if self.listNS[ip]:
4             print "El NS existe en la lista"
5     except KeyError:
6         self.listNS[ip] = "PYRO:Pyro.NameServer@" + str(ip) + ":" + str(
            ns.port)
```

Parámetros

- *host*: Dirección IP del host donde esta corriendo el name server "*ns*"
- *ns*: Name server que será agregado a la lista de name servers del host.

1.4.7. setListNS(*INS*)

Permite asignar la lista de name servers al host.

```
1 def setListNS(self, INS):
2     self.listNS = INS
```

Parámetros

- *INS*: Lista de name servers que se quiere asignar a este host.

1.4.8. deleteNS(*nombre*)

Permite eliminar el name server con el nombre "*nombre*" de la lista de name servers del host. Si no existe el name server en la lista, se produce una excepción.

```
1 def deleteNS(self, nombre):
2     try:
3         del self.listNS[nombre]
4         print "Se elimino el NS de la lista"
5     except KeyError:
6         print "No existe el NS en la lista"
```

Parámetros

- *nombre*: Nombre del name server que se quiere remover de la lista de name servers del host.

1.4.9. getListAgentes()

Permite a los demás objetos en la red obtener la lista de las cabezas de los agentes que se encuentran en el host.

```
1 def getListAgentes(self):  
2     return self.listAgentes;
```

Retorno

Retorna la lista de las cabezas de los agentes que se encuentran alojados en el host.

1.4.10. addAgente(*agente*, *create* = **True**)

Adiciona a la lista de agentes el objeto "*agente*". Si el agente ya existe en la red se retorna su identificador de recurso uniforme, si no existe, el agente se expone como objeto Pyro y se retorna la *uri* resultante.

```
1 def addAgente(self, agente, create = True):  
2     if(self.resolve(agente) == False):  
3         movilidadId = 'legs_' + agente  
4         racionalidadId = 'arms_' + agente  
5         hostUri = 'PYRO:' + self._pyroId + '@' + self._pyroDaemon.  
            locationStr  
6         agent = Agente.Agente(agente, movilidadId, racionalidadId,  
            hostUri)  
7         if(create):  
8             self.addRacionalidad(racionalidadId)  
9             self.addMovilidad(movilidadId)  
10  
11         print('Adding head ' + agent.getNombre() + ' to Daemon in ' +  
            str(self._pyroDaemon.locationStr))  
12         uri = self._pyroDaemon.register(agent)  
13         self.listAgentes[agent.getNombre()] = uri.asString()  
14         return uri.asString()  
15     else:  
16         #Corregir  
17         return self.resolve(agente)
```

Parámetros

- *agente*: Instancia de la clase objeto que será adicionada a la lista de agentes del host.
- *create*: Parámetro que indica si se debe crear los componentes de movilidad y racionalidad del agente, y añadirlos a la lista de movilidades y racionalidades del host respectivamente. Por defecto es **True**.

Retorno

Retorna el identificador de recurso uniforme del objeto "*agente*" que ha sido.

1.4.11. **setListAgente(*lAgente*)**

Permite asignar la lista de agentes al host.

```
1 def setListAgente(self, lAgente):  
2     self.listAgentes = lAgente
```

Parámetros

- *lAgente*: Lista de agentes que serán asignadas al host.

1.4.12. **deleteAgente(*nombre*)**

Permite eliminar el agente cuyo nombre coincida con "*nombre*" de la lista de agentes del host.

```
1 def deleteAgente(self, nombre):  
2     uri = self.listAgentes[nombre]  
3     print('Removing ' + nombre + ' from Daemon at ' + self.  
         _pyroDaemon.locationStr)  
4     self._pyroDaemon.unregister(uri[5:uri.find('@')])  
5     self.listAgentes.pop(nombre)
```

Parámetros

- *nombre*: Nombre del agente que será removido de la lista de agentes del host.

1.4.13. **getListMovilidad()**

Permite a los demás objetos en la red obtener la lista de los componentes de movilidad de los agentes existentes en la red, que se encuentran en el host.

```
1 def getListMovilidad(self):  
2     return self.listMovilidad;
```

Retorno

Retorna la lista de los componentes de movilidad de los agentes que se encuentran alojados en el host.

1.4.14. **addMovilidad(*movilidadId*)**

Adiciona a la lista de componentes de movilidad de los agentes el componente de movilidad que coincida con el id "*movilidadId*".

```
1 def addMovilidad(self, movilidadId):  
2     movilidad = Movilidad.Movilidad(movilidadId)  
3     print('Adding Movility ' + movilidad.getId() + ' to Daemon in ' +  
         str(self._pyroDaemon.locationStr))
```

```

4 | uri = self._pyroDaemon.register(movilidad)
5 | self.listMovilidad[movilidad.getId()] = uri.asString()

```

Parámetros

- *movilidadId*: Id del componente de movilidad que se desea agregar a la lista de componentes de movilidad en el host.

1.4.15. setListMovilidad(*lMovilidad*)

Permite asignar la lista de componentes de movilidad de los agentes al host.

```

1 | def setListMovilidad(self, lMovilidad):
2 |     self.listMovilidad = lMovilidad

```

Parámetros

- *lMovilidad*: Lista de los componentes de movilidad que serán asignados al host.

1.4.16. deleteMovilidad(*nombre*)

Permite eliminar el componente de movilidad cuyo nombre coincida con "*nombre*" de la lista de componentes de movilidad del host.

```

1 | def deleteMovilidad(self, nombre):
2 |     uri = self.listMovilidad[nombre]
3 |     print('Removing ' + nombre + ' from Daemon at ' + self._pyroDaemon.locationStr)
4 |     self._pyroDaemon.unregister(uri[5:uri.find('@')])
5 |     self.listMovilidad.pop(nombre)

```

Parámetros

- *nombre*: Nombre del componente de movilidad que será removido de la lista de componentes de movilidad del host.

1.4.17. getListRacionalidad()

Permite a los demás objetos en la red obtener la lista de los componentes de racionalidad de los agentes existentes en la red, que se encuentran en el host.

```

1 | def getListRacionalidad(self):
2 |     return self.listRacionalidad

```

Retorno

Retorna la lista de los componentes de racionalidad de los agentes que se encuentran alojados en el host.

1.4.18. `addRacionalidad(racionalidadId)`

Adiciona a la lista de componentes de racionalidad de los agentes el componente de racionalidad que coincida con con el id "*racionalidadId*".

```
1 def addRacionalidad(self, racionalidadId):
2     racionalidad = Racionalidad.Racionalidad(racionalidadId)
3     print('Adding rationality ' + racionalidad.getId() + ' to
        Daemon in ' + str(self._pyroDaemon.locationStr))
4     uri = self._pyroDaemon.register(racionalidad)
5     self.listRacionalidad[racionalidad.getId()] = uri.asString()
```

Parámetros

- *racionalidadId*: Id del componente de racionalidad que se desea agregar a la lista de componentes de racionalidad en el host.

1.4.19. `setListRacionalidad(lRacionalidad)`

Permite asignar la lista de componentes de racionalidad de los agentes al host.

```
1 def setListRacionalidad(self, lRacionalidad):
2     self.listRacionalidad = lRacionalidad
```

Parámetros

- *lRacionalidad*: Lista de los componentes de racionalidad que serán asignados al host.

1.4.20. `deleteRacionalidad(nombre)`

Permite eliminar el componente de racionalidad cuyo nombre coincida con "*nombre*" de la lista de componentes de racionalidad del host.

```
1 def deleteRacionalidad(self, nombre):
2     uri = self.listRacionalidad[nombre]
3     print('Removing ' + nombre + ' from Daemon at ' + self.
        _pyroDaemon.locationStr)
4     self._pyroDaemon.unregister(uri[5:uri.find('@')])
5     self.listRacionalidad.pop(nombre)
```

Parámetros

- *nombre*: Nombre del componente de racionalidad que será removido de la lista de componentes de movilidad del host.

1.4.21. `moveAgente(nombre, hostTo)`

Permite mover el agente, de la lista de agentes del host, cuyo nombre coincida con “*nombre*” al host de destino “*hostTo*”. Si el agente no existe en el host se lanza una excepción.

```
1 def moveAgente(self, nombre, hostTo):
2     try:
3         uri = self.listAgentes[nombre]
4         print('Moviendo ' + nombre + ' to ' + hostTo + '...')
5         self.deleteAgente(nombre)
6         newHost = Pyro4.Proxy(Pyro4.locateNS(hostTo).list()['host.' +
            self.nombre])
7         return newHost.addAgente(nombre, False)
8     except:
9         print('No existe el agente en este Host')
```

Parámetros

- *nombre*: Nombre del agente que se desea mover.
- *hostTo*: Host de destino al cual se desea mover el agente.

Retorno

Retorna

1.4.22. `moveMovilidad(nombre, hostTo)`

Permite mover el componente de movilidad, de la lista de componentes de movilidad del host, cuyo nombre coincida con “*nombre*” al host de destino “*hostTo*”. Si el componente de movilidad no existe en el host se lanza una excepción.

```
1 def moveMovilidad(self, nombre, hostTo):
2     try:
3         uri = self.listMovilidad[nombre]
4         print('Moviendo ' + nombre + ' to ' + hostTo + '...')
5         self.deleteMovilidad(nombre)
6         newHost = Pyro4.Proxy(Pyro4.locateNS(hostTo).list()['host.' +
            self.nombre])
7         newHost.addMovilidad(nombre)
8     except:
9         print('No existe la movilidad en este Host')
```

Parámetros

- *nombre*: Nombre del componente de movilidad que se desea mover.
- *hostTo*: Host de destino al cual se desea mover el componente de movilidad.

1.4.23. moveRacionalidad(*nombre*, *hostTo*)

Permite mover el componente de racionalidad, de la lista de componentes de racionalidad del host, cuyo nombre coincida con “*nombre*” al host de destino “*hostTo*”. Si el componente de racionalidad no existe en el host se lanza una excepción.

```
1 def moveRacionalidad(self, nombre, hostTo):
2     try:
3         uri = self.listRacionalidad[nombre]
4         print('Moviendo ' + nombre + ' to ' + hostTo + '...')
5         self.deleteRacionalidad(nombre)
6         newHost = Pyro4.Proxy(Pyro4.locateNS(hostTo).list()[ 'host.' +
7             self.nombre])
8             newHost.addRacionalidad(nombre)
9     except:
10         print('No existe la racionalidad en este Host')
```

Parámetros

- *nombre*: Nombre del componente de racionalidad que se desea mover.
- *hostTo*: Host de destino al cual se desea mover el componente de racionalidad.

1.4.24. disperseAgente(*agentId*)

Permite dispersar el agente identificado con id “*agentId*”^{en} la red, es decir, los componentes del agente (cabeza, movilidad y racionalidad) son enviados aleatoriamente a otros host en la red.

```
1 def disperseAgente(self, agentId):
2     agent = Pyro4.Proxy(self.listAgentes[agentId])
3     movilidadId = agent.getMovilidadId()
4     racionalidadId = agent.getRacionalidadId()
5     self.moveMovilidad(movilidadId, random.sample(self.listNS.keys(), 1)[0])
6     self.moveRacionalidad(racionalidadId, random.sample(self.listNS.keys(), 1)[0])
7     return self.moveAgente(agentId, random.sample(self.listNS.keys(), 1)[0])
```

Parámetros

- *agentId*: Identificador del agente que será dispersado en la red.

Retorno

Retorna

1.5. nombreclase.py

```
1 Codigo
```

Métodos

1.5.1. nombre()

```
1 codigo
```

Parámetros

- *nombre*:

Retorno