

Prototipo de Lenguaje e interpretador para modelar redes AD-HOC usando agentes dispersos

Lenguaje ADD

Joseph Alejandro Gallego Mejía
Sergio Edmundo Pérez Fonseca
Jhon Fredy Ramírez Guzmán

20 de enero, 2013

1. Gramática del lenguaje propuesto

La gramática de nuestro lenguaje propuesto es una gramática *LL1*. Donde la primera (**L**) hace alusión al hecho de que el flujo de terminales se lee de izquierda a derecha, accediendo a la entrada por su izquierda. La segunda (**L**) se entiende que el método de análisis predictivo construye una derivación a izquierdas. El número (**1**) indica el número de terminales que debemos consultar para decidir que regla de producción se aplica.

Nuestra gramática en el formato Backus-Naur es la siguiente:

```
<declaraciones> ::= <detener_programa>
                  | <iniciar_programa>
                  | <crear_agente>
                  | <eliminar_agente>
                  | <recuperar_agente>
                  | <dispersar_agente>
                  | <crear_comunidad>
                  | <iniciar_servicio>
                  | <detener_servicio>
                  | <agregar_agente_comunidad>
                  | <agregar_servicio_comunidad>
                  | <obtener_lista_agentes_comunidad>
                  | <obtener_lista_agentes_host>
                  | <obtener_lista_movilidad_host>
                  | <obtener_lista_racionalidad_host>
<iniciar_programa> ::= 'iniciar_programa'
<detener_programa> ::= 'detener_programa'
<crear_agente> ::= 'crear_agente' ''' <identificador> '''
```

```

<eliminar_agente> ::= 'eliminar_agente' ''' <identificador> '''
<recuperar_agente> ::= 'recuperar_agente' ''' <identificador> '''
<dispersar_agente> ::= 'dispersar_agente' ''' <identificador> '''
<crear_comunidad> ::= 'crear_comunidad' ''' <identificador> '''
<iniciar_servicio> ::= 'iniciar_servicio' ''' <servicio> '''
<detener_servicio> ::= 'detener_servicio' ''' <servicio> '''
<agregar_agente_comunidad> ::= 'agregar_agente_comunidad'
                                ''' <identificador> ''' <identificador> '''
<agregar_servicio_comunidad> ::= 'agregar_servicio_comunidad'
                                ''' <identificador> ''' <identificador> '''
<obtener_lista_agentes_comunidad> ::= 'obtener_lista_agentes_comunidad'
                                ''' <identificador> '''
<obtener_lista_agentes_host> ::= 'obtener_lista_agentes_host'
                                ''' <identificador> '''
<obtener_lista_movilidad_host> ::= 'obtener_lista_movilidad_host'
                                ''' <identificador> '''
<obtener_lista_racionalidad_host> ::= 'obtener_lista_racionalidad_host'
                                ''' <identificador> '''
<servicio> ::= "Videoconferencia" | "Mensajes"
<identificador> ::= letra { letra | dígito }
<dígito> ::= [0-9]
<letra> ::= [A-Za-z_]

```

2. Identificación de Tokens

Los tokens válidos de nuestra gramática son:

- *identificador*: identificador que permite reconocer objetos dentro del lenguaje. Está constituido por mínimo un carácter del conjunto alfabético, seguido de la combinación de ninguno o muchos caracteres del conjunto alfabético y el conjunto de los dígitos.
- *servicio*: Identificador que permite reconocer un servicio determinado que ofrece una comunidad de agentes en la red Ad-Hoc. Solo puede tener los valores “*Mensajes*” o “*Videoconferencia*”
- *iniciar_programa*: Comando que permite iniciar la ejecución del programa.
- *detener_programa*: Comando que permite detener el programa en ejecución.
- *crear_agente*: Comando que permite crear un nuevo agente y agregarlo a la red Ad-Hoc.
- *eliminar_agente*: Comando que permite eliminar un agente de la red Ad-Hoc

- *recuperar_agente*: Comando que permite reconstruir el agente en el host a partir de sus componentes dispersos en la red Ad-Hoc.
- *dispersar_agente*: Comando que permite dispersar los componentes del agente en la red Ad-Hoc.
- *crear_comunidad*: Comando que permite crear una nueva comunidad de agentes y agregarla a la red Ad-Hoc.
- *iniciar_servicio*: Comando que permite iniciar un servicio determinado entre dos nodos en la red Ad-Hoc.
- *detener_servicio*: Comando que permite detener un servicio determinado que se encuentra en ejecución entre dos nodos en la red Ad-Hoc.
- *agregar_agente_comunidad*: Comando que permite agregar un agente determinado a una comunidad de agentes determinada.
- *agregar_servicio_comunidad*: Comando que permite agregar un servicio determinado a una comunidad de agentes determinada.
- *obtener_lista_agentes_comunidad*: Comando que permite obtener la lista de agentes que pertenecen a una comunidad de agentes determinada.
- *obtener_lista_agentes_host*: Comando que permite obtener la lista de agentes que se encuentran en el host en un determinado instante de tiempo.
- *obtener_lista_movilidad_host*: Comando que permite obtener la lista de componentes de movilidad de los agentes, que se encuentran en el host en un instante de tiempo determinado.
- *obtener_lista_racionalidad_host*: Comando que permite obtener la lista de componentes de racionalidad de los agentes, que se encuentran en el host en un instante de tiempo determinado.

3. Árboles sintácticos

4. ¿Qué es Pyro?

Pyro significa **P**ython **R**emote **O**bject, es una librería que permite construir aplicaciones, en las cuales los objetos pueden comunicarse entre ellos a través de una red, con un mínimo esfuerzo de programación. Con ayuda de esta librería Se puede hacer el llamado a los métodos de Python normalmente, con casi todos los posibles tipo de parámetros y valores de retorno, y Pyro se encarga de localizar el objeto correcto en el computador correcto para ejecutar el método.

Pyro está diseñado para ser muy fácil de usar, y generalmente no interponerse en el camino. También provee un conjunto de características poderosas que

permiten construir aplicaciones distribuidas rápidamente y sin mucho esfuerzo. Pyro está desarrollada 100 % en Python puro y por lo tanto puede ser ejecutado en muchas plataformas y versiones de Python, incluyendo **Python 2.x**, **Python 3.x**, **IronPython**, **Jython 2.7+** and **Pypy**.

- Pyro es propiedad de Irmén de Jong irmen@razorvine.net - <http://www.razorvine.net>
- El repositorio con el código fuente se encuentra en GitHub: <http://github.com/irmen/Pyro4>
- La documentación de la librería se puede encontrar en: <http://pythonhosted.org/Pyro4/>
- Pyro puede ser encontrado en **Pypi** como Pyro4

5. Un poco de historia

Pyro fue iniciado en 1998, hace más de diez años, cuando la tecnología de invocación de métodos remotos, tales como *RMI* de Java y *CORBA*, eran muy populares. El autor, quería algo así en Python, y como no había nada disponible, entonces decidió escribir el suyo propio. En el transcurso de los años lentamente se le fueron añadiendo nuevas características hasta la versión 3.10. En ese punto, era claro que el código base se había vuelto un poco viejo y no permitía la adición de nuevas características de una manera fácil, entonces, para inicios de 2010 se dió origen a Pyro4, escrito totalmente desde cero. Después de un par de versiones Pyro4 llegó a ser lo suficientemente estable para convertirse en la nueva versión "principal".

Pyro es el nombre del paquete de la versión antigua (*3.x*) de Pyro. Pyro4 es el nombre del nuevo paquete, es decir, la versión actual. Su API y comportamiento es similar a Pyro 3.x, pero no es compatible hacia atrás. Por ello, para evitar conflictos, la nueva versión de Pyro tiene un nombre diferente.

6. Documentación de clases

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.

6.1. 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
24
25     def getNombre(self):
26         return self.nombre
27
28     def getType(self):
29         return 'head'
30
31     def getPyroId(self):
32         return str(self._pyroId)
33
34     def doIt(self):
35         ##place some call to legs and arms
36         racionalidadUri = Pyro4.Proxy(self.hostUri).resolve(self.
37             racionalidadId)
38         movilidadUri = Pyro4.Proxy(self.hostUri).resolve(self.
39             movilidadId)
40         if (racionalidadUri == False or movilidadUri == False):
41             return 'Algo esta perdido'
42         racionalidad = Pyro4.Proxy(racionalidadUri)
43         movilidad = Pyro4.Proxy(movilidadUri)
44         return [racionalidad.sayArms(), movilidad.sayLegs()]
```

Métodos

6.1.1. `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.

6.1.2. `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.

6.1.3. `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.

6.1.4. `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.

6.1.5. `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.

6.1.6. `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.

6.1.7. `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.
        racionalidadId)
4     movilidadUri = Pyro4.Proxy(self.hostUri).resolve(self.
        movilidadId)
5     if (racionalidadUri == False or movilidadUri == False):
6         return 'Algo esta perdido'
7     racionalidad = Pyro4.Proxy(racionalidadUri)
8     movilidad = Pyro4.Proxy(movilidadUri)
9     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".

6.2. 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

6.2.1. `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.

6.2.2. `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

6.2.3. `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

6.2.4. 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.

6.2.5. 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.

6.2.6. 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*".

6.3. 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

6.3.1. 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.

6.4. 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.
82                 _pyroDaemon.locationStr
83             agent = Agente.Agente(agente, movilidadId,
84                 racionalidadId, hostUri)
85             if(create):
86                 self.addRacionalidad(racionalidadId)
87                 self.addMovilidad(movilidadId)
88
89             print('Adding head ' + agent.getNombre() + ' to Daemon
90                 in ' + str(self._pyroDaemon.locationStr))
91             uri = self._pyroDaemon.register(agent)
92             self.listAgentes[agent.getNombre()] = uri.asString()
93             return uri.asString()
94         else:
95             #Corregir
96             return self.resolve(agente)
97
98     def setListAgente(self, lAgente):
99         self.listAgentes = lAgente
100
101     def deleteAgente(self, nombre):
102         uri = self.listAgentes[nombre]
103         print('Removing ' + nombre + ' from Daemon at ' + self.
104             _pyroDaemon.locationStr)
105         self._pyroDaemon.unregister(uri[5:uri.find('@')])
106         self.listAgentes.pop(nombre)
107
108     def getListMovilidad(self):
109         """
110         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()[
159             'host.' + self.nombre])
160         newHost.addMovilidad(nombre)
161     except:
162         print('No existe la movilidad en este Host')
163
164     def moveRacionalidad(self, nombre, hostTo):
165         try:
166             uri = self.listRacionalidad[nombre]
167             print('Moviendo ' + nombre + ' to ' + hostTo + '...')
168             self.deleteRacionalidad(nombre)
169             newHost = Pyro4.Proxy(Pyro4.locateNS(hostTo).list()[
170                 'host.' + self.nombre])
171             newHost.addRacionalidad(nombre)
172         except:
173             print('No existe la racionalidad en este Host')
174
175     def disperseAgente(self, agentId):
176         agent = Pyro4.Proxy(self.listAgentes[agentId])
177         movilidadId = agent.getMovilidadId()
178         racionalidadId = agent.getRacionalidadId()
179         self.moveMovilidad(movilidadId, random.sample(self.listNS.
180             keys(), 1)[0])
181         self.moveRacionalidad(racionalidadId, random.sample(self.
182             listNS.keys(), 1)[0])
183         return self.moveAgente(agentId, random.sample(self.listNS.
184             keys(), 1)[0])
185
186     #Servicio
187     def searchHead(self, name):#had to add this methos on the host,
188         so it can act sorta like a NameServer
189         ret = self.findHead(name)
190         if(ret == False):
191             print('Precaucion!: Head del objeto no esta en el host.
192                 Esto puede afectar el rendimiento.')
193             for nameServer, nameServer_uri in self.listNS.items():
194                 try:
195                     findHost = Pyro4.Proxy(Pyro4.Proxy(
196                         nameServer_uri).list()['host.' + self.
197                             nombre])
198                     ret = findHost.findHead(name)
199                 except:
200                     print "Error: no se localizo el host"
201                     ret = False
202             if(ret != False):
203                 return [ret, Pyro4.Proxy(nameServer_uri).list()[
204                     'host.' + self.nombre]]
205
206         return ret
207
208     def findHead(self, name):
209         """ returns uri of object or false """
210         try:
211             return self.listAgentes[name]
212         except:

```

```

204         return False
205
206     def searchMovilidad(self, name):#had to add this methos on the
207         host, so it can act sorta like a NameServer
208         ret = self.findMovilidad(name)
209         if(ret == False):
210             print('Precaucion!: Movilidad del objeto no esta en el
211                 host. Esto puede afectar el rendimiento.')
212             for nameServer, nameServer_uri in self.listNS.items():
213                 try:
214                     findHost = Pyro4.Proxy(Pyro4.Proxy(
215                         nameServer_uri).list()['host.' + self.
216                         nombre])
217                     ret = findHost.findMovilidad(name)
218                 except:
219                     print "Error: no se localizo el host"
220                     ret = False
221             if(ret != False):
222                 return [ret, Pyro4.Proxy(nameServer_uri).list()['
223                     host.' + self.nombre]]
224
225     return ret
226
227     def findMovilidad(self, name):
228         """ returns uri of object or false """
229         try:
230             return self.listMovilidad[name]
231         except:
232             return False
233
234     def searchRacionalidad(self, name):#had to add this methos on
235         the host, so it can act sorta like a NameServer
236         ret = self.findRacionalidad(name)
237         if(ret == False):
238             print('Precaucion!: Movilidad del objeto no esta en el
239                 host. Esto puede afectar el rendimiento.')
240             for nameServer, nameServer_uri in self.listNS.items():
241                 try:
242                     findHost = Pyro4.Proxy(Pyro4.Proxy(
243                         nameServer_uri).list()['host.' + self.
244                         nombre])
245                     ret = findHost.findRacionalidad(name)
246                 except:
247                     print "Error: no se localizo el host"
248                     ret = False
249             if(ret != False):
250                 return ret
251
252     return ret
253
254     def findRacionalidad(self, name):
255         """ returns uri of object or false """
256         try:
257             return self.listRacionalidad[name]
258         except:
259             return False

```



```

252 #Funciones para la comunidad de agentes.
253 def retrieveAgente(self, agentId):
254     #Primero se recupera la cabeza o encabezado del agente.
255     agent = self.searchHead(agentId)
256     if(agent == False):
257         print 'No se encontro el agente'
258     else:
259         print 'Moviendo el encabezado al host ' + self.nombre
260         agent = self.moveAgente(agentId, self.listNS[self.nombre
261                                     ])
262         movilidadId = Pyro4.Proxy(agent).getMovilidadId()
263         [movilidad, hostMovilidad] = self.searchMovilidad(
264             movilidadId)
265         if(movilidad == False):
266             print 'No se encontro la movilidad '
267             print 'Dado que no se encontro la movilidad no se
268                 procedera a encontrar la racionalidad'
269         else:
270             self.moveMovilidad(movilidadId, self.listNS[self.
271                 nombre])
272             racionalidadId = Pyro4.Proxy(agent).
273                 getRacionalidadId()
274             try:
275                 Pyro4.Proxy(hostMovilidad).getListMovilidad()[
276                     racionalidadId]
277                 self.moveRacionalidad(racionalidadId, self.
278                     listNS[self.nombre])
279             except:
280                 'La racionalidad no se encuentra con la
281                     movilidad por lo tanto no se podra mover'
282
283     return agent

```

Métodos

6.4.1. 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.

6.4.2. 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.

6.4.3. 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.

6.4.4. 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.

6.4.5. 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 direccion 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.

6.4.6. 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.

6.4.7. 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.

6.4.8. 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.

6.4.9. 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.

6.4.10. 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 (uri) "*agente*" que ha sido agregado a la lista de agentes en el host.

6.4.11. 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 seran asignadas al host.

6.4.12. 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.

6.4.13. 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.

6.4.14. addMovilidad(*movilidadId*)

Adiciona a la lista de componentes de movilidad de los agentes el componente de movilidad que coincida con 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.

6.4.15. 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.

6.4.16. 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.
4           _pyroDaemon.locationStr)
5     self._pyroDaemon.unregister(uri[5:uri.find('@')])
6     self.listMovilidad.pop(nombre)
```

Parámetros

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

6.4.17. 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.

6.4.18. addRacionalidad(*racionalidadId*)

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

```
1 def addRacionalidad(self, racionalidadId):
2     racionalidad = Racionalidad.Racionalidad(racionalidadId)
3     print('Adding rationality ' + racionalidad.getId() + ' to
4           Daemon in ' + str(self._pyroDaemon.locationStr))
5     uri = self._pyroDaemon.register(racionalidad)
6     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.

6.4.19. `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.

6.4.20. `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.
4           _pyroDaemon.locationStr)
5     self._pyroDaemon.unregister(uri[5:uri.find('@')])
6     self.listRacionalidad.pop(nombre)
```

Parámetros

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

6.4.21. `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.' +
7           self.nombre])
8         return newHost.addAgente(nombre, False)
9     except:
10        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 el nuevo identificador de recurso uniforme (uri) del agente.

6.4.22. 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.' +
7             self.nombre])
8         newHost.addMovilidad(nombre)
9     except:
10        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.

6.4.23. 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.

6.4.24. **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
6         (), 1)[0])
7     self.moveRacionalidad(racionalidadId , random.sample(self.listNS.
8         keys() , 1)[0])
9     return self.moveAgente(agentId , random.sample(self.listNS.keys
10         (), 1)[0])
```

Parámetros

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

Retorno

Retorna el nuevo identificador de recurso uniforme (uri) del agente que ha sido dispersado.