



UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

SISTEMAS MULTIAGENTE
GRADO EN INGENIERÍA INFORMÁTICA

Recuperación de información en Wikipedia

Antonio Manjavacas
Rubén Márquez

Octubre, 2019

Índice

1. INTRODUCCIÓN	2
2. OBJETIVOS GENERALES	2
3. COMUNICACIÓN ENTRE AGENTES	4
4. EJECUCIÓN DEL SISTEMA	6
4.1. <i>DashboardAgent</i>	6
4.2. <i>ProcesserAgent</i>	7
4.3. <i>WebAgent</i>	8
5. CLASES EMPLEADAS	9

1. INTRODUCCIÓN

Los *agentes de información* (también conocidos como *agentes de Internet*) son un tipo de agente cuyo objetivo reside en la recolección de información a través de la red; éstos se encargan de indexarla y ofrecérsela al usuario cuando realiza una determinada consulta.

Los agentes de Internet poseen un amplio rango de funcionalidades y pueden ser de lo más diversos, ya que las capacidades necesarias para la extracción de información entran dentro de las áreas de otros tipos de agentes. Surgen de la necesidad de procesar de manera automática la información y permiten facilitar el acceso a esta aprovechando la capacidad de los agentes.

La principal ventaja que estos agentes pueden ofrecernos, es la facilidad de acceso a información y datos no procesables manualmente.

2. OBJETIVOS GENERALES

El objetivo de esta práctica será la integración de servicios de tres agentes. La información obtenida por un agente web será utilizada por un agente que procese la información y una interfaz que la muestre al usuario. La estructura del sistema se detalla en la Figura 1.

Los agentes utilizados serán los siguientes:

- **Web agent.** Será el agente encargado de obtener la información sobre la población histórica de Boston, Chicago y Seattle.
- **ProcesserAgent.** Recibirá y procesará la información obtenida de las direcciones web. Dicho procesamiento consistirá en obtener el registro con mayor población para cada uno de los agentes y remitirlo al agente *Dashboard*.
- **DashboardAgent.** Se encargará de realizar peticiones de datos al *ProcesserAgent* y de mostrar la información recibida al usuario.

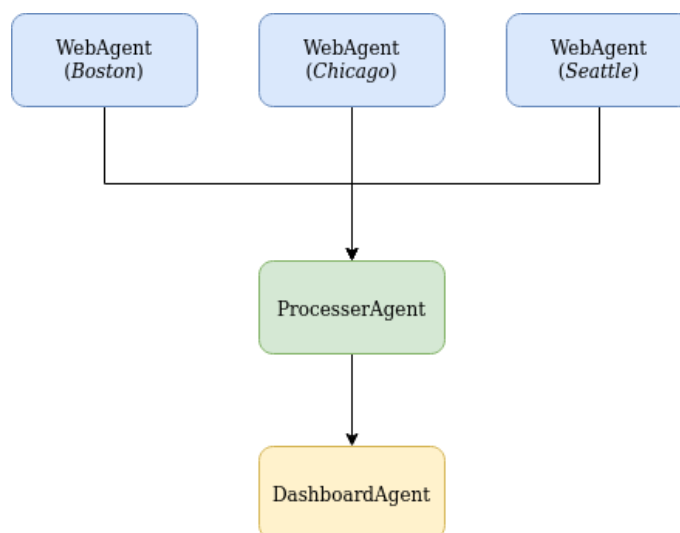


Figura 1: Sistema multiagente a desarrollar

La información recopilada por el *WebAgent* se obtendrá de las siguientes páginas de Wikipedia:

- <https://es.wikipedia.org/wiki/Boston>
- <https://es.wikipedia.org/wiki/Chicago>

- <https://es.wikipedia.org/wiki/Seattle>

De dichas fuentes se recuperará la información acerca de la población histórica de las ciudades anteriormente mencionadas, la cual se encuentra en formato de tabla en el apartado *Demografía* de cada una de las páginas web (un ejemplo de dichas tablas es el que se muestra en la Figura 2). De estas tablas se obtendrán los años junto con su correspondiente población. Finalmente, el agente encargado del procesamiento de los datos obtendrá el año con mayor población histórica de cada ciudad y enviará esta información al agente *Dashboard* para su presentación por pantalla.

Para recuperar la información, se ha hecho uso de las librerías *jsoup* (<https://jsoup.org/>), para obtener los ficheros HTML, y *java.util.regex* para extraer la información contenida en las tablas mediante el uso de expresiones regulares. La expresión regular utilizada ha sido: `"\\d{4} (\\s \\d{4}) | (\\s \\d{1,3}){2,3}) \\s [\\+D&&\\-W]"`.

Población histórica		
Año	Pob.	±%
1840	4470	—
1850	29 963	+570.3%
1860	112 172	+274.4%
1870	298 977	+166.5%
1880	503 185	+68.3%
1890	1 099 850	+118.6%
1900	1 698 575	+54.4%
1910	2 185 283	+28.7%
1920	2 701 705	+23.6%
1930	3 376 438	+25.0%
1940	3 396 808	+0.6%
1950	3 620 962	+6.6%
1960	3 550 404	-1.9%
1970	3 366 957	-5.2%
1980	3 005 072	-10.7%
1990	2 783 726	-7.4%
2000	2 896 016	+4.0%
2010	2 695 598	-6.9%
2015	2 720 546	+0.9%

Figura 2: Tabla de población histórica de Chicago

Por otro lado, de cara a introducir una componente aleatoria en cada uno de los agentes, se ha hecho uso de la API de [random.org](https://api.random.org/json-rpc/1/) (<https://api.random.org/json-rpc/1/>) para la generación de números aleatorios. La aleatoriedad que ofrece dicha página proviene del ruido atmosférico, lo que supone que para muchos propósitos sea mejor que los algoritmos de números pseudoaleatorios ofrecidos por lenguajes de programación como Java. El funcionamiento de esta componente aleatoria consistirá en la generación de un número que decida qué registros de las tablas recuperan los agentes.

Para cada página web se generará una lista de registros del tipo [fecha, población, URL] que serán enviados al agente *Processor* para que obtenga el de mayor población para cada ciudad.

3. COMUNICACIÓN ENTRE AGENTES

A continuación, en la Figura 3 se detalla el flujo de comunicación entre los agentes que conforman el sistema. Nótese el uso del protocolo de comunicación FIPA-Request a la hora de realizar la solicitud de datos procesados entre *Dashboard* y *Processor*, así como para la obtención de los datos que ofrecen los agentes de Internet por parte del *Processor*.

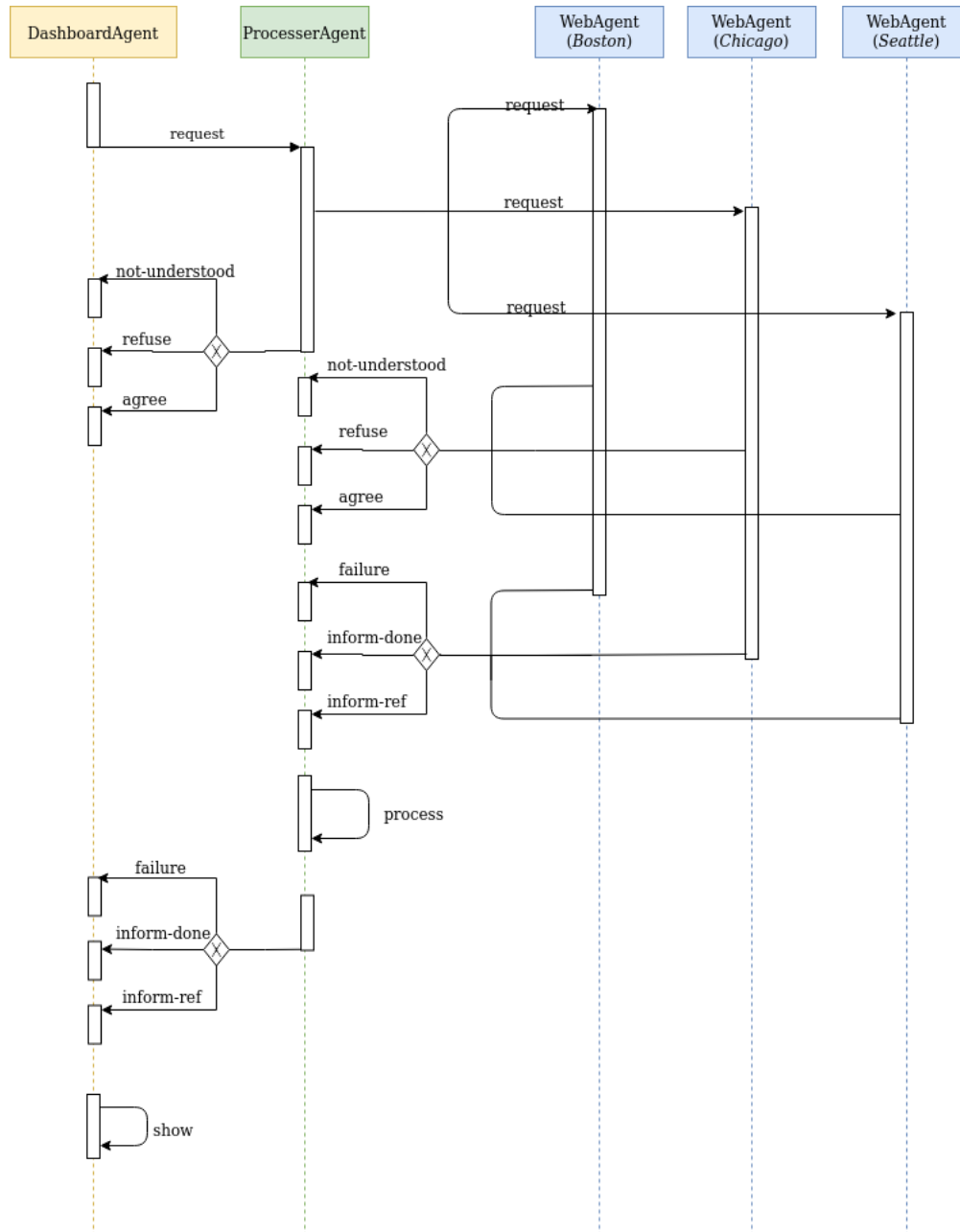


Figura 3: Flujo de comunicación entre agentes

El flujo de comunicación descrito incluye:

- La petición del *DashboardAgent* al *ProcesserAgent* de los datos procesados.
- La petición del *ProcesserAgent* al *WebAgent* de la información sobre población histórica para Boston, Chicago y Seattle.
- La respuesta con los datos demográficos al *ProcesserAgent*.

- La obtención del año con mayor población para cada una de las ciudades por parte del *ProcesserAgent* (*process*).
- La recepción de los datos procesados por el *DashboardAgent*.
- La representación de dichos datos (*show*).

Por otro lado, la Figura 4 incluye el diagrama de clases de diseño del sistema. Como puede observarse, cada agente incluye comportamientos y sus correspondientes manejadores para todos los tipos de mensajes definidos según el protocolo FIPA-Request:

- *handleNotUnderstood*
- *handleRefuse*
- *handleAgree*
- *handleFailure*
- *handleInform*
- *handleRequest*
- *prepareResultNotification*

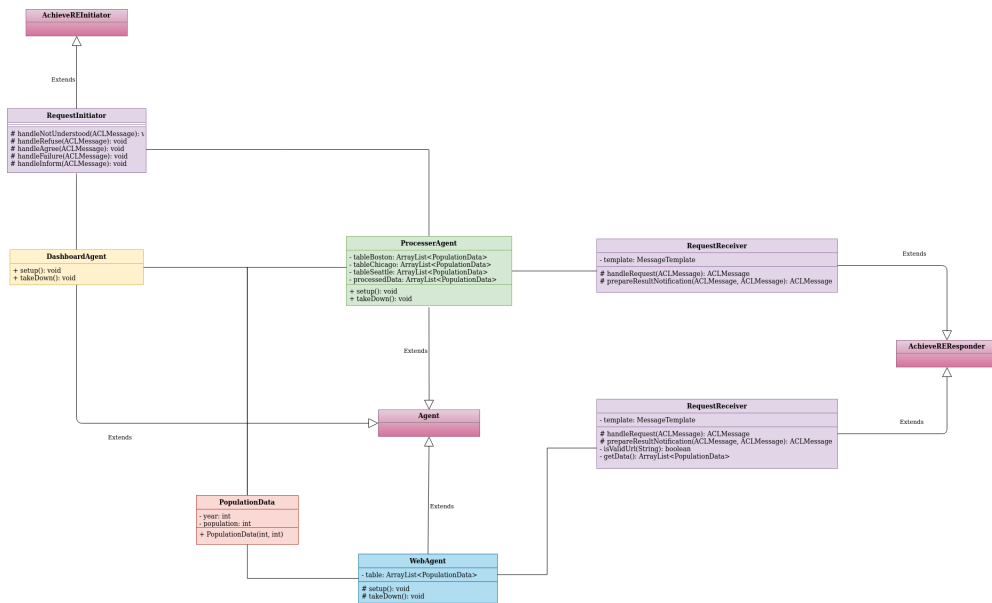


Figura 4: Diagrama de clases de diseño

4. EJECUCIÓN DEL SISTEMA

Una vez definido el comportamiento general de nuestro sistema, así como las herramientas a utilizar, veamos cómo sucede la ejecución e interacción entre agentes. El código de cada uno de ellos se encuentra incluido en la sección 5.

Primero compilaremos todas las clases en el paquete *agents* incluyendo las librerías en el *classpath*:

- *lib/jade.jar*
- *lib/gson-2.8.6.jar*
- *lib/jsoup-1.12.1.jar*
- *lib/random-org.jar*

A continuación, iniciaremos la plataforma *JADE* y crearemos cada uno de los agentes:

- Definiremos tres agentes del tipo *WebAgent* con los nombres: **WebAgent1**, **WebAgent2** y **WebAgent3**.
- Nuestro *ProcesserAgent* tendrá por nombre: **Processer**.
- Finalmente, el *DashboardAgent*, con nombre: **Dashboard**.

Puede utilizarse el siguiente comando:

```
java jade.Boot -agents WebAgent1:agents.WebAgent;  
WebAgent2:agents.WebAgent;WebAgent3:agents.WebAgent;  
Processer:agents.ProcesserAgent;Dashboard:agents.DashboardAgent
```

4.1. *DashboardAgent*

El comportamiento de este agente es el siguiente: al ser iniciado, despliega una interfaz gráfica con tres cuadros de texto donde el usuario puede introducir las URLs de las que se extraerá la información (ver Figura 5). En nuestro caso, las URLs por defecto ya incluidas en la interfaz serán las correspondientes a las páginas de Wikipedia de las ciudades de Boston, Seattle y Chicago. Una vez introducidas las URLs, pulsar el botón ubicado en la esquina inferior derecha del formulario dará lugar al envío de una petición *FIPA-REQUEST* al agente *Processer* con las URLs introducidas.

Tras la obtención y procesamiento de los datos por el resto de agentes, se mostrará un segundo formulario con los datos de población histórica recibidos, reflejando los años de mayor población obtenidos para cada una de las ciudades (ver Figura 6).

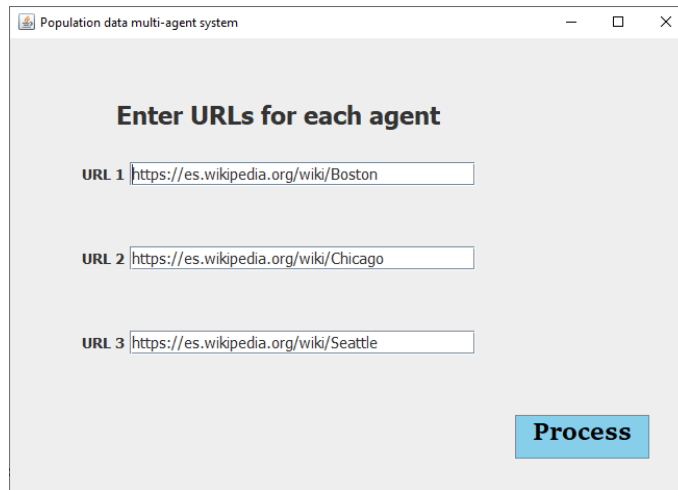


Figura 5: Interfaz de usuario inicial

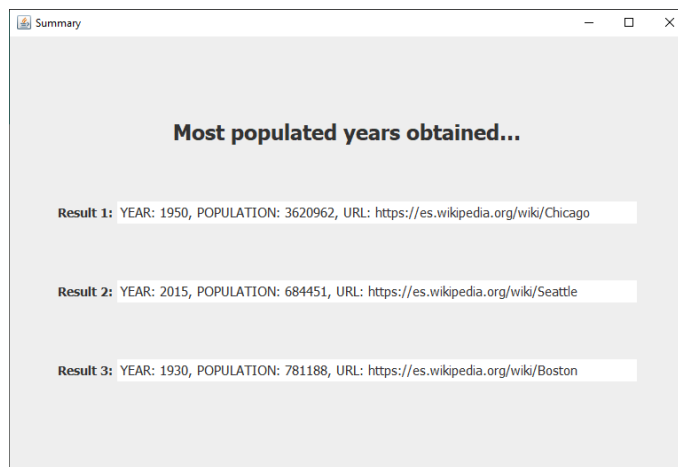


Figura 6: Interfaz de usuario con los resultados

4.2. *ProcesserAgent*

Este agente actuará como intermediario entre la interfaz y el agente de Internet. Cuando el *Processer* recibe una petición *FIPA-REQUEST* del Dashboard, obtiene las tres URLs contenidas en el mensaje y genera las peticiones de tipo *FIPA-REQUEST* que envía a cada uno de los agentes de Internet por medio de un comportamiento compuesto. Dicho comportamiento, ya sea paralelo o secuencial, estará compuesto por tres subcomportamientos del tipo *AchieveREInitiator*, que enviarán una URL a cada *WebAgent*.

La comparación en el orden de envío y recepción de mensajes según el tipo de comportamiento compuesto empleado puede observarse en las figuras Figura 7 y Figura 8.

Una problemática ante la que nos encontramos fue la necesidad de esperar a la respuesta de todos los agentes web antes de responder al Dashboard. Como puede observarse en el código, fue necesario redefinir el método *prepareResultNotification(ACLMessage request, ACLMessage response)* del *AchieveREResponder* mediante *registerPrepareResultNotification(Behaviour b)* y acceder directamente al *DataStore* (cola de mensajes) del comportamiento *AchieveREResponder* para introducir la respuesta al Dashboard solamente cuando los agentes web hubiesen terminado su tarea.

Una vez recibidas todas las respuestas de los agentes web, el *Processer* obtiene los registros con mayor población para cada URL y responde al Dashboard con los datos obtenidos.

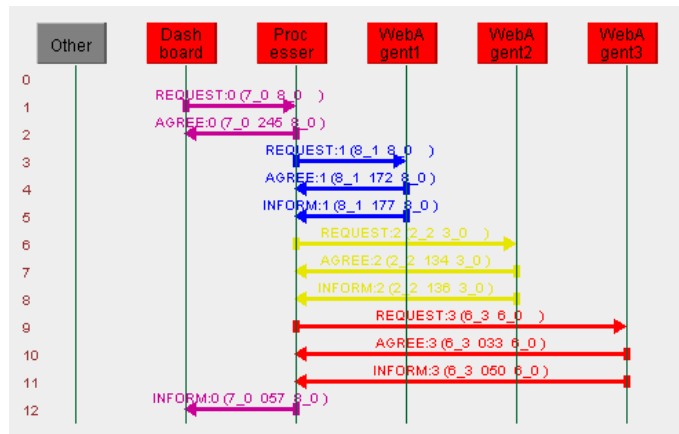


Figura 7: Intercambio de mensajes con comportamiento secuencial

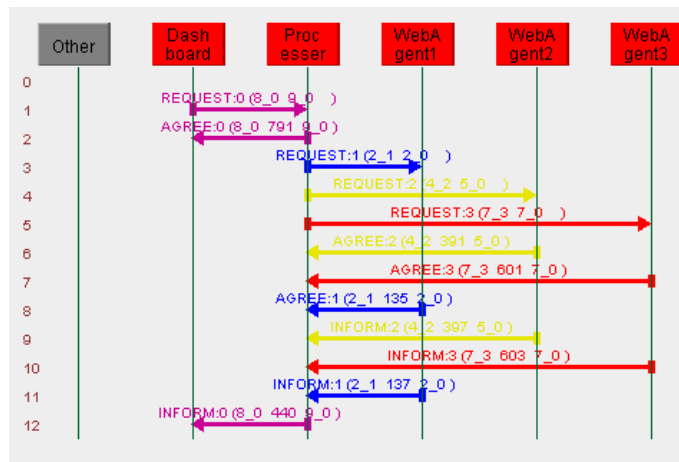


Figura 8: Intercambio de mensajes con comportamiento paralelo

4.3. WebAgent

Las peticiones del Processor son recibidas por los agentes web: WebAgent1, WebAgent2 y WebAgent3. Dichos agentes extraen los datos asociados a la URL recibida mediante el método *getData()*, que almacena cada uno de los registros de las tablas de población histórica en objetos del tipo *PopulationData*. Finalmente, el conjunto de registros se envía en un mensaje de respuesta al Processor.

5. CLASES EMPLEADAS

Clase *DashboardAgent*: implementa la interfaz con el usuario y la petición de los datos

```
1 package agents;
2
3 import java.io.IOException;
4 import java.util.ArrayList;
5
6 // Agent imports
7 import jade.core.Agent;
8
9 import jade.core.AID;
10 import jade.lang.acl.ACLMessage;
11 import jade.lang.acl.UnreadableException;
12 import jade.proto.AchieveREInitiator;
13 import jade.domain.FIPANames;
14
15 // Interface imports
16 import java.awt.BorderLayout;
17 import javax.swing.JFrame;
18 import javax.swing.JPanel;
19 import javax.swing.border.EmptyBorder;
20 import java.awt.GridBagLayout;
21 import javax.swing.JTextField;
22 import javax.swing.JTextPane;
23
24 import java.awt.GridBagConstraints;
25 import java.awt.Insets;
26 import javax.swing.JLabel;
27 import java.awt.Font;
28 import javax.swing.JButton;
29 import java.awt.Color;
30 import java.awt.Cursor;
31 import java.awt.event.ActionListener;
32 import java.awt.event.ActionEvent;
33
34 public class DashboardAgent extends Agent {
35
36     final static String DASHBOARD = "Dashboard";
37     final static String PROCESSER = "Processer";
38
39     final static String WEB_AGENT_1 = "WebAgent1";
40     final static String WEB_AGENT_2 = "WebAgent2";
41     final static String WEB_AGENT_3 = "WebAgent3";
42
43     protected void setup() {
44
45         Form frame = null;
46
47         try {
48             frame = new Form();
49             frame.setVisible(true);
50         } catch (Exception e) {
51             System.out.println("[DASHBOARD-AGENT] Form view failed: " + e.
52                 getMessage());
53         }
54
55         while (!frame.ready) { // wait until button event
56             try {
```

```

56     Thread.sleep(2000);
57 } catch (InterruptedException e) {
58     e.printStackTrace();
59 }
60 }
61
62 System.out.println("[DASHBOARD-AGENT] Received URLs! Sending to
63     processor...");
64
65 ArrayList<String> msgContent = new ArrayList<String>();
66 msgContent.add(WEB_AGENT_1 + ">" + frame.url_1);
67 msgContent.add(WEB_AGENT_2 + ">" + frame.url_2);
68 msgContent.add(WEB_AGENT_3 + ">" + frame.url_3);
69
70 ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
71 msg.addReceiver(new AID(PROCESSER, AID.ISLOCALNAME));
72 msg.setProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST);
73 msg.setSender(new AID(DASHBOARD, AID.ISLOCALNAME));
74
75 try {
76     msg.setContentObject(msgContent);
77 } catch (IOException e) {
78     System.out.println("[DASHBOARD-AGENT] Error in the urls encapsulation
79     .");
80 }
81
82 addBehaviour(new RequestInitiator(this, msg));
83
84 }
85
86 protected void takeDown() {
87     System.out.println("[DASHBOARD-AGENT] Taking down...");
88 }
89
90 public class RequestInitiator extends AchieveREInitiator {
91
92     public RequestInitiator(Agent a, ACLMessage msg) {
93         super(a, msg);
94     }
95
96     protected void handleAgree(ACLMessage agree) {
97         System.out.println("[DASHBOARD-AGENT] " + agree.getSender().getName()
98         + " has accepted the request.");
99     }
100
101     protected void handleRefuse(ACLMessage refuse) {
102         System.out.println("[DASHBOARD-AGENT] " + refuse.getSender().getName()
103         + " has rejected the request.");
104         System.out.println("[DASHBOARD-AGENT] Reason - " + refuse.getContent()
105         ());
106     }
107
108     protected void handleNotUnderstood(ACLMessage notUnderstood) {
109         System.out.println(
110         "[DASHBOARD-AGENT] " + notUnderstood.getSender().getName() + "
111         didn't understood the request.");
112         System.out.println("[DASHBOARD-AGENT] Reason - " + notUnderstood.
113         getContent());
114     }
115
116     protected void handleInform(ACLMessage inform) {

```

```

109
110     System.out.println("[DASHBOARD-AGENT] Received results from " +
inform.getSender().getName());
111
112     ArrayList<PopulationData> info = null;
113
114     try {
115         info = (ArrayList<PopulationData>) inform.getContentObject();
116     } catch (UnreadableException e) {
117         System.out.println("[DASHBOARD-AGENT] Error unpacking message");
118     }
119
120     Summary sum = null;
121
122     try {
123         System.out.println("[DASHBOARD-AGENT] Showing results");
124
125         sum = new Summary();
126         sum.textPane.setText(info.get(0).toString());
127         sum.textPane_1.setText(info.get(1).toString());
128         sum.textPane_2.setText(info.get(2).toString());
129         sum.setVisible(true);
130     } catch (Exception e) {
131         System.out.println("[DASHBOARD-AGENT] Summary view failed: " + e.
getMessage());
132     }
133
134 }
135
136 protected void handleFailure(ACLMessage failure) {
137     System.out.println("[DASHBOARD-AGENT] " + failure.getSender().getName
() + " failed during execution!");
138     System.out.println("[DASHBOARD-AGENT] Reason - " + failure.getContent
());
139 }
140
141 }
142
143 /** Input information form */
144 class Form extends JFrame {
145
146     private static final long serialVersionUID = 1L;
147     private JPanel contentPane;
148     private JTextField textField1;
149     private JTextField textField2;
150     private JTextField textField3;
151
152     public String url_1 = "";
153     public String url_2 = "";
154     public String url_3 = "";
155
156     private String default_url_1 = "https://es.wikipedia.org/wiki/Boston";
157     private String default_url_2 = "https://es.wikipedia.org/wiki/Chicago";
158     private String default_url_3 = "https://es.wikipedia.org/wiki/Seattle";
159
160     public boolean ready = false;
161
162     /**
163      * Create the frame
164      */

```

```

165 public Form() {
166     setForeground(new Color(173, 216, 230));
167     setTitle("Population data multi-agent system");
168     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
169     setBounds(100, 100, 695, 476);
170     contentPane = new JPanel();
171     contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
172     contentPane.setLayout(new BorderLayout(0, 0));
173     setContentPane(contentPane);
174
175     JPanel panel = new JPanel();
176     contentPane.add(panel, BorderLayout.CENTER);
177     GridBagLayout gbl_panel = new GridBagLayout();
178     gbl_panel.columnWidths = new int[] { 64, 0, 0, 0, 0 };
179     gbl_panel.rowHeights = new int[] { 0, 0, 0, 0, 0, 0, 0 };
180     gbl_panel.columnWeights = new double[] { 0.0, 0.0, 1.0, 1.0, Double.
MIN_VALUE };
181     gbl_panel.rowWeights = new double[] { 1.0, 0.0, 1.0, 1.0, 1.0, 1.0,
Double.MIN_VALUE };
182     panel.setLayout(gbl_panel);
183
184     JLabel lblTitle = new JLabel("Enter URLs for each agent");
185     lblTitle.setFont(new Font("Tahoma", Font.BOLD, 24));
186     GridBagConstraints gbc_lblTitle = new GridBagConstraints();
187     gbc_lblTitle.gridwidth = 2;
188     gbc_lblTitle.insets = new Insets(0, 0, 5, 5);
189     gbc_lblTitle.gridx = 1;
190     gbc_lblTitle.gridy = 1;
191     panel.add(lblTitle, gbc_lblTitle);
192
193     JLabel lblUrl1 = new JLabel("URL 1");
194     lblUrl1.setFont(new Font("Tahoma", Font.BOLD, 14));
195     GridBagConstraints gbc_lblUrl1 = new GridBagConstraints();
196     gbc_lblUrl1.insets = new Insets(0, 0, 5, 5);
197     gbc_lblUrl1.gridx = 1;
198     gbc_lblUrl1.gridy = 2;
199     panel.add(lblUrl1, gbc_lblUrl1);
200
201     textField1 = new JTextField(default_url_1);
202     textField1.setFont(new Font("Tahoma", Font.PLAIN, 15));
203     GridBagConstraints gbc_textField1 = new GridBagConstraints();
204     gbc_textField1.fill = GridBagConstraints.HORIZONTAL;
205     gbc_textField1.insets = new Insets(0, 0, 5, 5);
206     gbc_textField1.gridx = 2;
207     gbc_textField1.gridy = 2;
208     panel.add(textField1, gbc_textField1);
209     textField1.setColumns(10);
210
211     JLabel lblUrl2 = new JLabel("URL 2");
212     lblUrl2.setFont(new Font("Tahoma", Font.BOLD, 14));
213     GridBagConstraints gbc_lblUrl2 = new GridBagConstraints();
214     gbc_lblUrl2.insets = new Insets(0, 0, 5, 5);
215     gbc_lblUrl2.gridx = 1;
216     gbc_lblUrl2.gridy = 3;
217     panel.add(lblUrl2, gbc_lblUrl2);
218
219     textField2 = new JTextField(default_url_2);
220     textField2.setFont(new Font("Tahoma", Font.PLAIN, 15));
221     GridBagConstraints gbc_textField2 = new GridBagConstraints();
222     gbc_textField2.fill = GridBagConstraints.HORIZONTAL;

```

```

223     gbc_textField2.insets = new Insets(0, 0, 5, 5);
224     gbc_textField2.gridx = 2;
225     gbc_textField2.gridy = 3;
226     panel.add(textField2 , gbc_textField2);
227     textField2.setColumns(10);
228
229     JLabel lblUrl3 = new JLabel("URL 3");
230     lblUrl3.setFont(new Font("Tahoma", Font.BOLD, 14));
231     GridBagConstraints gbc_lblUrl3 = new GridBagConstraints();
232     gbc_lblUrl3.insets = new Insets(0, 0, 5, 5);
233     gbc_lblUrl3.gridx = 1;
234     gbc_lblUrl3.gridy = 4;
235     panel.add(lblUrl3 , gbc_lblUrl3);
236
237     textField3 = new JTextField(default_url_3);
238     textField3.setFont(new Font("Tahoma", Font.PLAIN, 15));
239     GridBagConstraints gbc_textField3 = new GridBagConstraints();
240     gbc_textField3.insets = new Insets(0, 0, 5, 5);
241     gbc_textField3.fill = GridBagConstraints.HORIZONTAL;
242     gbc_textField3.gridx = 2;
243     gbc_textField3.gridy = 4;
244     panel.add(textField3 , gbc_textField3);
245     textField3.setColumns(10);
246
247     JButton btnProcess = new JButton("Process");
248     btnProcess.setFont(new Font("Sitka Text", Font.BOLD, 25));
249     btnProcess.addActionListener(new ActionListener() {
250         public void actionPerformed(ActionEvent arg0) {
251             url_1 = textField1.getText();
252             url_2 = textField2.getText();
253             url_3 = textField3.getText();
254             ready = true;
255             dispose();
256         }
257     });
258
259     btnProcess.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
260     btnProcess.setBackground(new Color(135, 206, 235));
261     btnProcess.setForeground(new Color(0, 0, 0));
262     GridBagConstraints gbc_btnProcess = new GridBagConstraints();
263     gbc_btnProcess.gridx = 3;
264     gbc_btnProcess.gridy = 5;
265     panel.add(btnProcess , gbc_btnProcess);
266 }
267
268 }
269
270 /* Output information form */
271 class Summary extends JFrame {
272
273     private static final long serialVersionUID = 1L;
274     private JPanel contentPane;
275
276     public JTextPane textPane;
277     public JTextPane textPane_1;
278     public JTextPane textPane_2;
279
280     /**
281      * Create the frame.
282      */

```

```

283 public Summary() {
284     setTitle("Summary");
285     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
286     setBounds(100, 100, 783, 528);
287     contentPane = new JPanel();
288     contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
289     setContentPane(contentPane);
290     GridBagLayout gbl_contentPane = new GridBagLayout();
291     gbl_contentPane.columnWidths = new int[] { 0, 0, 0, 0, 0 };
292     gbl_contentPane.rowHeights = new int[] { 0, 0, 0, 0, 0, 0, 0 };
293     gbl_contentPane.columnWeights = new double[] { 1.0, 0.0, 1.0, 1.0,
Double.MIN_VALUE };
294     gbl_contentPane.rowWeights = new double[] { 1.0, 1.0, 1.0, 1.0, 1.0,
1.0, Double.MIN_VALUE };
295     contentPane.setLayout(gbl_contentPane);
296
297     JLabel lblTitle = new JLabel("Most populated years obtained...");
298     lblTitle.setFont(new Font("Tahoma", Font.BOLD, 24));
299     GridBagConstraints gbc_lblTitle = new GridBagConstraints();
300     gbc_lblTitle.gridwidth = 2;
301     gbc_lblTitle.insets = new Insets(0, 0, 5, 5);
302     gbc_lblTitle.gridx = 1;
303     gbc_lblTitle.gridy = 1;
304     contentPane.add(lblTitle, gbc_lblTitle);
305
306     JLabel lblResult1 = new JLabel("Result 1:");
307     lblResult1.setFont(new Font("Tahoma", Font.BOLD, 14));
308     GridBagConstraints gbc_lblResult1 = new GridBagConstraints();
309     gbc_lblResult1.insets = new Insets(0, 0, 5, 5);
310     gbc_lblResult1.gridx = 1;
311     gbc_lblResult1.gridy = 2;
312     contentPane.add(lblResult1, gbc_lblResult1);
313
314     textPane = new JTextPane();
315     textPane.setEditable(false);
316     textPane.setFont(new Font("Tahoma", Font.PLAIN, 15));
317     GridBagConstraints gbc_textPane = new GridBagConstraints();
318     gbc_textPane.insets = new Insets(0, 0, 5, 5);
319     gbc_textPane.fill = GridBagConstraints.HORIZONTAL;
320     gbc_textPane.gridx = 2;
321     gbc_textPane.gridy = 2;
322     contentPane.add(textPane, gbc_textPane);
323
324     JLabel lblResult2 = new JLabel("Result 2:");
325     lblResult2.setFont(new Font("Tahoma", Font.BOLD, 14));
326     GridBagConstraints gbc_lblResult2 = new GridBagConstraints();
327     gbc_lblResult2.insets = new Insets(0, 0, 5, 5);
328     gbc_lblResult2.gridx = 1;
329     gbc_lblResult2.gridy = 3;
330     contentPane.add(lblResult2, gbc_lblResult2);
331
332     textPane_1 = new JTextPane();
333     textPane_1.setEditable(false);
334     textPane_1.setFont(new Font("Tahoma", Font.PLAIN, 15));
335     GridBagConstraints gbc_textPane_1 = new GridBagConstraints();
336     gbc_textPane_1.insets = new Insets(0, 0, 5, 5);
337     gbc_textPane_1.fill = GridBagConstraints.HORIZONTAL;
338     gbc_textPane_1.gridx = 2;
339     gbc_textPane_1.gridy = 3;
340     contentPane.add(textPane_1, gbc_textPane_1);

```

```

341
342 JLabel lblResult3 = new JLabel("Result 3:");
343 lblResult3.setFont(new Font("Tahoma", Font.BOLD, 14));
344 GridBagConstraints gbc_lblResult3 = new GridBagConstraints();
345 gbc_lblResult3.insets = new Insets(0, 0, 5, 5);
346 gbc_lblResult3.gridx = 1;
347 gbc_lblResult3.gridy = 4;
348 contentPane.add(lblResult3, gbc_lblResult3);
349
350 textPane_2 = new JTextPane();
351 textPane_2.setEditable(false);
352 textPane_2.setFont(new Font("Tahoma", Font.PLAIN, 15));
353 GridBagConstraints gbc_textPane_2 = new GridBagConstraints();
354 gbc_textPane_2.insets = new Insets(0, 0, 5, 5);
355 gbc_textPane_2.fill = GridBagConstraints.HORIZONTAL;
356 gbc_textPane_2.gridx = 2;
357 gbc_textPane_2.gridy = 4;
358 contentPane.add(textPane_2, gbc_textPane_2);
359 }
360
361 }
362 }

```


Clase *ProcesserAgent*: intermediario entre agentes encargado de procesar los datos de la web

```
1 package agents;
2
3 import jade.core.Agent;
4 import jade.core.behaviours.ParallelBehaviour;
5 import java.io.IOException;
6 import java.util.ArrayList;
7
8 import jade.core.AID;
9 import jade.lang.acl.ACLMessage;
10 import jade.lang.acl.MessageTemplate;
11 import jade.lang.acl.UnreadableException;
12 import jade.proto.AchieveREResponder;
13 import jade.proto.AchieveREInitiator;
14
15 import jade.domain.FIPANames;
16 import jade.domain.FIPAAgentManagement.NotUnderstoodException;
17 import jade.domain.FIPAAgentManagement.RefuseException;
18
19 public class ProcesserAgent extends Agent {
20
21     final static String DASHBOARD = "Dashboard";
22     final static String PROCESSER = "Processer";
23
24     final static int N_WEB_AGENTS = 3;
25
26     private static RequestReceiver requestReceiver;
27     private static ArrayList<PopulationData> maxPopulationsList = new
        ArrayList<PopulationData>();
28
29     protected void setup() {
30
31         MessageTemplate protocol = MessageTemplate.MatchProtocol(FIPANames.
            InteractionProtocol.FIPA_REQUEST);
32         MessageTemplate performative = MessageTemplate.MatchPerformative(
            ACLMessage.REQUEST);
33         MessageTemplate sender = MessageTemplate.MatchSender(new AID(DASHBOARD,
            AID.ISLOCALNAME));
34         MessageTemplate temp = MessageTemplate.and(protocol, performative);
35         temp = MessageTemplate.and(temp, sender);
36
37         requestReceiver = new RequestReceiver(this, temp);
38         addBehaviour(requestReceiver);
39     }
40
41     protected void takeDown() {
42         System.out.println("[PROCESSER-AGENT] Taking down...");
43     }
44
45     public class RequestReceiver extends AchieveREResponder {
46
47         public RequestReceiver(Agent a, MessageTemplate temp) {
48             super(a, temp);
49             System.out.println("[PROCESSER-AGENT] Waiting request...");
50         }
51
52         protected ACLMessage handleRequest(ACLMessage request) throws
            NotUnderstoodException, RefuseException {
53
54             System.out.println("[PROCESSER-AGENT] " + request.getSender().getName
```

```

55     () + " has sent a request.");
56     ArrayList<String> msgContent = null;
57     try {
58         msgContent = (ArrayList<String>) request.getContentObject();
59     } catch (UnreadableException e) {
60         throw new NotUnderstoodException("[PROCESSER-AGENT]
NotUnderstoodException: unreadable content of the message.");
61     }
62
63     ParallelBehaviour pb = new ParallelBehaviour();
64
65     if (msgContent.size() == N_WEB_AGENTS) {
66
67         String webAgentName, webAgentUrl;
68
69         for (String content : msgContent) {
70
71             String[] contentParts = content.split(">");
72             webAgentName = contentParts[0];
73             webAgentUrl = contentParts[1];
74
75             System.out.println(
76                 "[PROCESSER-AGENT] URL: " + webAgentUrl + " will be processed
by: " + webAgentName);
77
78             ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
79             msg.addReceiver(new AID((String) webAgentName, AID.ISLOCALNAME));
80             msg.setProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST);
81             msg.setSender(new AID(PROCESSER, AID.ISLOCALNAME));
82             msg.setContent(webAgentUrl);
83
84             pb.addSubBehaviour(new RequestInitiator(myAgent, msg));
85         }
86
87     } else {
88         throw new RefuseException("[PROCESSER-AGENT] RefuseException: the
message content contains information for "
89             + msgContent.size() + " web agents but " + N_WEB_AGENTS + " web
agents are needed!");
90     }
91
92     this.registerPrepareResultNotification(pb);
93
94     ACLMessage agree = request.createReply();
95     agree.setPerformative(ACLMessage.AGREE);
96     return agree;
97 }
98
99 }
100
101 public static class RequestInitiator extends AchieveREInitiator {
102
103     private static int count = 0;
104
105     public RequestInitiator(Agent a, ACLMessage msg) {
106         super(a, msg);
107     }
108
109     protected void handleAgree(ACLMessage agree) {

```

```

110     System.out.println("[PROCESSER-AGENT] " + agree.getSender().getName()
111     + " has accepted the request.");
112 }
113
114 protected void handleRefuse(ACLMessage refuse) {
115     System.out.println("[PROCESSER-AGENT] " + refuse.getSender().getName()
116     + " has rejected the request.");
117     System.out.println("[PROCESSER-AGENT] Reason - " + refuse.getContent());
118 }
119
120 protected void handleNotUnderstood(ACLMessage notUnderstood) {
121     System.out.println(
122     "[PROCESSER-AGENT] " + notUnderstood.getSender().getName() + "
123     didn't understood the request.");
124     System.out.println("[PROCESSER-AGENT] Reason - " + notUnderstood.
125     getContent());
126 }
127
128 protected void handleInform(ACLMessage inform) {
129     count++;
130
131     System.out.println("[PROCESSER-AGENT] Received results from " +
132     inform.getSender().getName());
133
134     ArrayList<PopulationData> table = null;
135
136     try {
137         table = (ArrayList<PopulationData>) inform.getContentObject();
138     } catch (UnreadableException e) {
139         System.out.println(e.getMessage());
140     }
141
142     // Data processing: get greatest population
143     PopulationData greatest = null;
144     for (PopulationData pd : table) {
145         if (greatest == null) {
146             greatest = pd;
147         }
148         if (pd.getPopulation() > greatest.getPopulation()) {
149             greatest = pd;
150         }
151     }
152
153     maxPopulationsList.add(greatest);
154
155     if (count >= N_WEB_AGENTS) {
156         System.out.println("[PROCESSER-AGENT] Preparing result...");
157
158         String incomingRequestkey = (String) requestReceiver.REQUEST_KEY;
159         ACLMessage incomingRequest = (ACLMessage) requestReceiver.
160         getDataStore().get(incomingRequestkey);
161         // Prepare the notification to the request originator and store it
162         in the
163         // DataStore
164         ACLMessage notification = incomingRequest.createReply();
165         notification.setPerformative(ACLMessage.INFORM);
166         try {
167             notification.setContentObject(maxPopulationsList);

```

```

162     } catch (IOException e) {
163         System.out.println(e.getMessage());
164     }
165     String notificationkey = (String) requestReceiver.
RESULT_NOTIFICATION_KEY;
166     requestReceiver.getDataStore().put(notificationkey, notification);
167 }
168
169 }
170
171 protected void handleFailure(ACLMessage failure) {
172     System.out.println("[PROCESSER-AGENT] " + failure.getSender().getName
() + " has failed!");
173     System.out.println("[PROCESSER-AGENT] Reason - " + failure.getContent
());
174 }
175
176 }
177
178 }

```

Clase *WebAgent*: agente de Internet

```
1 package agents;
2
3 import jade.core.Agent;
4 import jade.core.AID;
5 import jade.lang.acl.ACLMessage;
6 import jade.lang.acl.MessageTemplate;
7 import jade.proto.AchieveREResponder;
8
9 import jade.domain.FIPANames;
10 import jade.domain.FIPAAgentManagement.NotUnderstoodException;
11 import jade.domain.FIPAAgentManagement.RefuseException;
12 import jade.domain.FIPAAgentManagement.FailureException;
13
14 import java.io.IOException;
15 import java.util.ArrayList;
16 import java.util.regex.Matcher;
17 import java.util.regex.Pattern;
18
19 import org.jsoup.Jsoup;
20 import org.jsoup.nodes.Document;
21 import org.random.api.RandomOrgClient;
22
23 public class WebAgent extends Agent {
24
25     final static String REG_EXP = "\\d{4}((\\s\\d{4})|((\\s\\d{1,3}){2,3}))\\s[\\D&\\W]";
26     final static String REG_EXP_URL = "^((https?|ftp|file)://[-a-zA-Z0-9+&@#/%?=_!:,;]*[-a-zA-Z0-9+&@#/%?=_!])";
27     final static String API_KEY = "31c9d05e-9079-4778-9db0-7395dbdb9580";
28
29     final static String DASHBOARD = "Dashboard";
30     final static String PROCESSER = "Processor";
31
32     final static int MAX = 10;
33     final static int MIN = 0;
34     final static int NUMS = 10;
35
36     protected void setup() {
37
38         MessageTemplate sender = MessageTemplate.MatchSender(new AID(
39             PROCESSER, AID.ISLOCALNAME));
40         MessageTemplate protocol = MessageTemplate.MatchProtocol(FIPANames.
41             InteractionProtocol.FIPA_REQUEST);
42         MessageTemplate performative = MessageTemplate.MatchPerformative(
43             ACLMessage.REQUEST);
44         MessageTemplate temp = MessageTemplate.and(sender, protocol);
45         temp = MessageTemplate.and(temp, performative);
46
47         addBehaviour(new RequestReceiver(this, temp));
48     }
49
50     protected void takeDown() {
51         System.out.println("[ " + getLocalName() + "-AGENT] Taking down...");
52     }
53
54     public class RequestReceiver extends AchieveREResponder {
55
56         ArrayList<PopulationData> data;
```

```

54
55     public RequestReceiver(Agent a, MessageTemplate temp) {
56         super(a, temp);
57     }
58
59     protected ACLMessage handleRequest(ACLMessage request) throws
60     NotUnderstoodException, RefuseException {
61
62         String url = request.getContent();
63
64         if (isValidUrl(url)) {
65             try {
66                 data = getData(url);
67             } catch (IOException exception) {
68                 throw new RefuseException "[" + getLocalName() + "-
69                 AGENT] Refuse exception: error getting information from URL.");
70             }
71         } else {
72             throw new NotUnderstoodException "[" + getLocalName() + "-
73             AGENT] NotUnderstoodException: invalid URL received.");
74
75         ACLMessage agree = request.createReply();
76         agree.setPerformative(ACLMessage.AGREE);
77
78         return agree;
79     }
80
81     protected ACLMessage prepareResultNotification(ACLMessage request,
82     ACLMessage response)
83     throws FailureException {
84
85         System.out.println "[" + getLocalName() + "-AGENT]" + "
86         Preparing result ... ";
87         ACLMessage inform = request.createReply();
88         inform.setPerformative(ACLMessage.INFORM);
89
90         try {
91             inform.setContentObject(data);
92         } catch (IOException e) {
93             throw new FailureException "[" + getLocalName() + "-AGENT]
94             FailureException: serialization error.");
95         }
96
97         return inform;
98     }
99
100     /** Checks if and URL is valid */
101     private boolean isValidUrl(String url) {
102         Pattern p = Pattern.compile(REG_EXP_URL);
103         Matcher m = p.matcher(url);
104         return m.matches();
105     }
106
107     /** Get population and date registers */
108     private ArrayList<PopulationData> getData(String url) throws
109     IOException {
110
111         Document doc = Jsoup.connect(url).get();
112
113         // Body and pattern to be found

```

```

107         String body = doc.body().text();
108         String myPattern = REG_EXP;
109
110         // Pattern object and matcher creation
111         Pattern p = Pattern.compile(myPattern);
112         Matcher m = p.matcher(body);
113
114         // Find pattern and store data
115         ArrayList<PopulationData> table = new ArrayList<PopulationData
>();
116         PopulationData reg = null;
117
118         // Random number generation from www.random.org
119         RandomOrgClient client = RandomOrgClient.getRandomOrgClient(
API_KEY);
120         int[] rands = client.generateIntegers(NUMS, MIN, MAX);
121         int n = 0;
122
123         while (m.find()) {
124             if (rands[n++ % NUMS] % 2 == 0) { // random condition
125                 String cadena = m.group(0);
126                 String populationCad = cadena.substring(5, cadena.
length() - 2);
127                 reg = new PopulationData(Integer.parseInt(cadena.
substring(0, 4)),
128                                         Integer.parseInt(populationCad.replace(" ", ""))
), url);
129                 table.add(reg);
130             }
131         }
132
133         return table;
134     }
135
136 }
137
138 }

```

Clase *PopulationData*: representa los registros de población anual obtenidos

```
1 package agents;
2
3 import java.io.Serializable;
4
5 public class PopulationData implements Serializable {
6
7     private int year, population;
8     private String url;
9
10    public PopulationData(int year, int population, String url) {
11        this.year = year;
12        this.population = population;
13        this.url = url;
14    }
15
16    public int getYear() {
17        return year;
18    }
19
20    public void setYear(int year) {
21        this.year = year;
22    }
23
24    public int getPopulation() {
25        return population;
26    }
27
28    public void setPopulation(int population) {
29        this.population = population;
30    }
31
32    public String getUrl() {
33        return url;
34    }
35
36    public void setUrl(String url) {
37        this.url = url;
38    }
39
40    @Override
41    public String toString() {
42        return "YEAR: " + year + ", POPULATION: " + population + ", URL: " +
43            url;
44    }
45 }
```