AEMETAgent

Un sistema multiagente para obtener información del clima.

Sergio Fernández García.

Eduardo Sánchez López.

Índice

Índice de ilustraciones

1. Introducción y objetivos.

AEMETAgent es un sistema multiagente programado con Jade para la visualización de la información de las estaciones meteorológicas de la AEMET.

El objetivo de esta practica es usar las técnicas y conceptos aprendidos en la asignatura de *Sistemas Multiagentes* para desarrollar un conjunto de agentes que recopilen información y se comuniquen entre ellos. Aunque si bien este proyecto podría haberse realizado con un único agente esto no cumpliría con el requerimiento de que haya comunicación entre agentes.

2. Herramientas utilizadas.

1) ***Java Agent DEvelopment Framework (JADE):***

   Es una plataforma para el desarrollo y despliegue de agentes. Desarrollada en Java, JADE utiliza una implementación del lenguaje FIPA-ACL para la comunicación entre los agentes. Más adelante hablaremos con más profundidad de dicho lenguaje una vez veamos los diferentes pasos de mensaje entre agentes.

2) Jsoup:

   Es una librería hecha en Java la cuál usamos para manipular el contenido de los documentos HTML bajados desde la página web. Esta librería es esencial para poder obtener toda la información deseada y poder mostrarla más adelante en un formato que pueda ser entendido por las personas.

3) ***JxBrowser:***
   Es otra librería hecha en Java la cuál nos permitirá lograr toda la parte visual del proyecto. Esta librería es usada para poder añadir un navegador en nuestra aplicación y poder ver e interactuar con el mapa de España a través de la API de Google Maps. Sin esta librería colocar los marcadores sería un acto aún más costoso y complicado. Es importante recalcar que para poder usar esta librería hará falta una licencia. Se puede pedir una licencia gratis de 30 días en https://www.teamdev.com/jxbrowser#evaluate. Para reemplazar solo hay que cambiar el archivo license.jar por el nuevo.

3. Agentes.

Nuestro sistema se compone de 3 agentes que ejecutan tareas distintas. Un esquema del flujo de información entre ellos sería el siguiente:

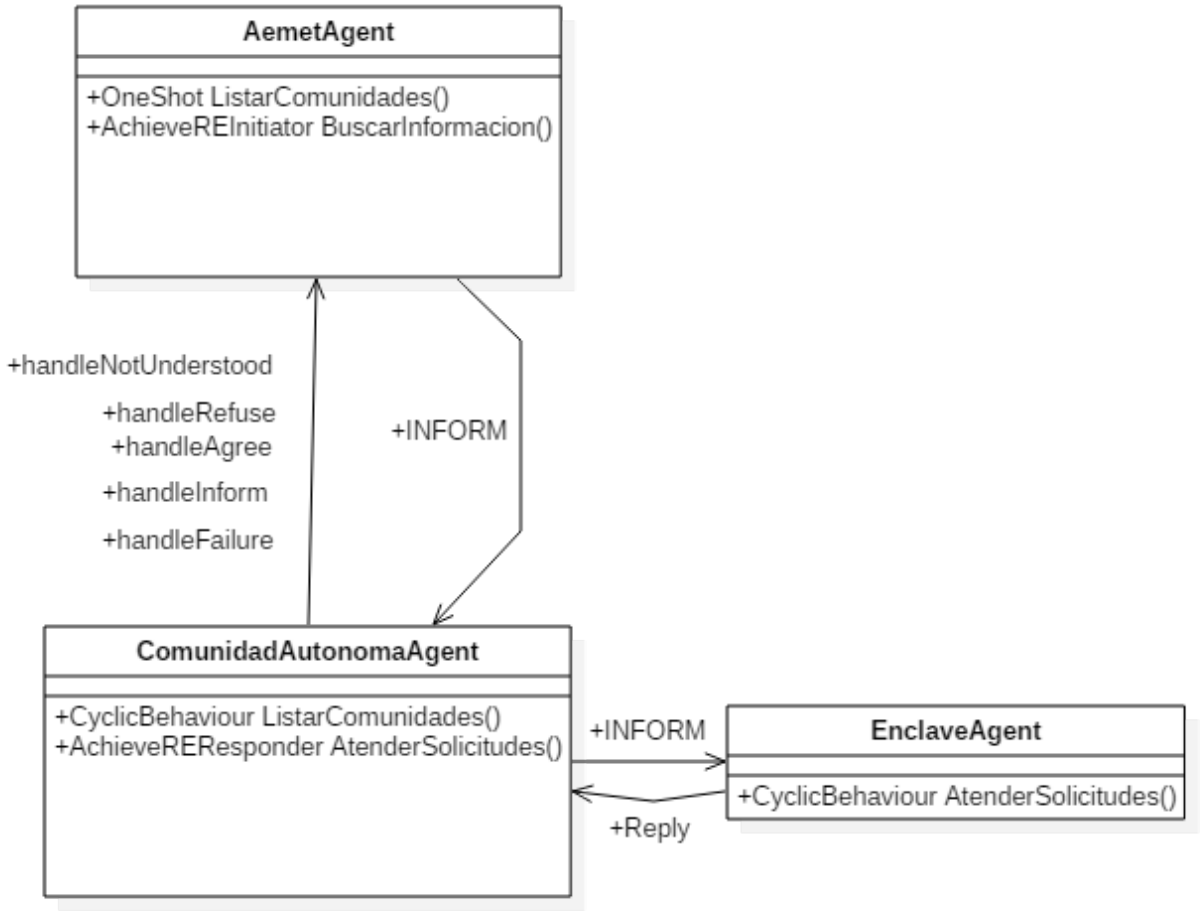

Ilustración 1: Arquitectura del sistema.

El orden a la hora de desplegarlos es siempre tanto *ComunidadAutonomaAgent* o *EnclaveAgent* primero que *AemetAgent.* Esto es debido a que *AemetAgent* es el que hace que se cree el MainFrame donde se encuentra toda la parte visual, y para ello necesita comunicarse con *ComunidadAutonomaAgent* primero para pedir el listado de comunidades autónomas. Si se creará primero *AemetAgent*, el mensaje no llegaría y el sistema no podría continuar.

A continuación hablaremos sobre cada uno de ellos.

3.1. AemetAgent

Es el agente principal con el cuál iniciamos toda la cadena de mensajes y con el que además pintaremos en el mapa los diferentes marcadores.

Lo primero que hace este agente es solicitar el listado de las comunidades autónomas al agente *ComunidadAutonomaAgent*, y lo coloca en el selection box del mainframe, donde se elige cuál es la comunidad que se quiere estudiar.

```
public class AemetAgent extends Agent {
	protected MainFrame mf;

	@Override
	protected void setup() {
		this.addBehaviour(new ListarComunidades(this));
		mf = new MainFrame(this);
	}
```

El listado se obtiene mediante el comportamiento *ListarComunidades*, por lo que lo llamaremos en el setup para que se llame una vez se cree el agente.

```
private class ListarComunidades extends OneShotBehaviour{

		public ListarComunidades(Agent agente) {
			super(agente);

		}
		@Override
		public void action() {

			AID id_comunidadAgente = new AID("ComunidadAutonomaAgent",
AID.ISLOCALNAME);

			//Solicita el listado de las comunidaddes Autónomas
aComunidadAutonomaAgent.
			ACLMessage msg_comunidades = new ACLMessage(ACLMessage.INFORM);
			msg_comunidades.setLanguage("COMUNIDADES");
			msg_comunidades.addReceiver(id_comunidadAgente);
			msg_comunidades.setSender(getAID());

			send(msg_comunidades);

			ACLMessage resp_comunidades = blockingReceive();
			if (resp_comunidades != null) {
				try {

	mf.setComunidades((Hashtable<String,String>)resp_comunidades.getContentObject());
				} catch (UnreadableException e) {

					e.printStackTrace();
				}
			}
			else{
				block();
			}
		}

	}
```

Cabe destacar que es un OneShotBehaviour porque solo necesitamos obtener el listado de las comunidades autónomas una sola vez para configurar el MainFrame.

BuscarInformacionEstaciones es un método que se ejecuta cuando se pulsa el botón “Mostrar” del MainFrame. Como parámetros a este método se pasan las comunidades autónomas seleccionadas en el comboBox.

Este método crea un ACLMessage para pedir al agente *ComunidadAutonomaAgent* que empiece a recopilar la información necesaria sobre las comunidades autónomas seleccionadas. Pero este mensaje se envía cuando añadimos el comportamiento BuscarInformación.

La necesidad de este método intermedio situado en el agente es que un objeto externo que no es un agente (Como se trata de en este caso MainFrame) no puede añadir comportamientos. Por eso se invoca a este método para que desde dentro del agente se añada el comportamiento.

```
public void buscarInformacionEstaciones(String objetivo) {
		AID id_comunidadAgente = new AID("ComunidadAutonomaAgent",
AID.ISLOCALNAME);

		//Pedir información al agente de la Comunidad Autónoma
(ComunidadAutonomaAgent)
		ACLMessage msg_peticion = new ACLMessage(ACLMessage.QUERY_IF);
		msg_peticion.setProtocol(FIPANames.InteractionProtocol.FIPA_QUERY);
		msg_peticion.setLanguage("INFO");
		msg_peticion.setContent(objetivo);
		msg_peticion.addReceiver(id_comunidadAgente);

		this.addBehaviour(new BuscarInformacion(this, msg_peticion));


	}
```

BuscarInformación hereda de AchieveREInitiator y se encarga de enviar el mensaje previamente creado hacia el agente *ComunidadAutonomaAgent* y de esperar su respuesta.

Si recibe un handleAgree entonces significa que *ComunidadAutonomaAgent* ha aceptado su petición y deberá de esperar a que se recojan todos los datos.

Un handleRefuse indica lo contrario, el agente se niega a recoger los datos por alguna circunstancia y el proceso no continuará.

Se recibe un handleNotUnderstood cuando algún proceso de la recogida de datos falló.

Similar a handleNotUnderstood, handleFailure llegará cuándo ocurra algún fallo en la petición, mayormente causado por errores en la red.

```
//Comportamiento que solicita información al resto de agentes
	private class BuscarInformacion extends AchieveREInitiator{


		public BuscarInformacion(Agent agente, ACLMessage aclmsg) {
			super(agente, aclmsg);
		}
```

```
	protected void handleAgree(ACLMessage agree) {
		System.out.println("Se ha aceptado");
		mf.mostrarMensaje("Buscando datos.", true);

	}

	protected void handleRefuse(ACLMessage refuse) {
		System.out.println("Se ha rechazado");
		mf.mostrarMensaje("Se ha rechazado la petición.", false);
	}

	protected void handleNotUnderstood(ACLMessage not_understood) {
		System.out.println("No se ha comprendido el mensaje");
		mf.mostrarMensaje("Ha ocurrido algún problema con la recogida de
				datos.", false);
	}

	protected void handleInform(ACLMessage inform) {
		System.out.println("Información del servicio");
		try {
			Hashtable<String,LinkedList> comunidades_table =
					new Hashtable<String, LinkedList>();

			comunidades_table =
			(Hashtable<String, LinkedList>) inform.getContentObject();

			mf.mostrarMensaje("Se han obtenido los datos.", true);
			mf.setInfo(comunidades_table);

			removeBehaviour(this);
		} catch (UnreadableException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	protected void handleFailure(ACLMessage fail) {
		System.out.println("Fallo en la petición");
		mf.mostrarMensaje("Fallo en la petición", false);
	}


}
```

El handle más importante es handleInform, el cuál se envía a *AemetAgent* cuando la información solicitada esta ya lista. Dicha información viene contenida en el propio inform recibido. Una vez desempaquetado dicho mensaje se llama el método setInfo del MainFrame para que aparezcan por pantalla.

3.2. ComunidadAutonomaAgent

Este Agente tiene dos funciones. La primera es otorgar al *AemetAgent* con el listado de las comunidades autónomas cuando este lo solicite, para ello utilizamos este comportamiento.

```
//Comportamiento para listar Comunidades Autónomas
	private class ListarComunidades extends CyclicBehaviour{

		MessageTemplate templateListar = null;

		public ListarComunidades(Agent agent){
			super(agent);
			MessageTemplate filtroSender = MessageTemplate.MatchSender(new
						AID("AemetAgent", AID.ISLOCALNAME));

			MessageTemplate filtroPerfomative =
				MessageTemplate.MatchPerformative(ACLMessage.INFORM);
			MessageTemplate filtroLanguage =
				MessageTemplate.MatchLanguage("COMUNIDADES");

			templateListar = MessageTemplate.and(filtroSender, filtroLanguage);
			templateListar = MessageTemplate.and(templateListar,
							filtroPerfomative);
		}
		@Override
		public void action() {
			ACLMessage msg = receive(templateListar);
			if(msg != null) {
				try {
					Document doc = Jsoup.connect(AEMET +
							ID_CC_AA_DIR).get();
					Elements form = doc.select("form[name=\"frm1\"]");

					Elements comunidades =
						form.select("option:not(option[selected])");

					for(Element comunidad: comunidades ) {
						comunidades_table.put(comunidad.attr("value"),
								comunidad.text());
					}
					ACLMessage res = msg.createReply();
					res.setContentObject(comunidades_table);
					send(res);
				} catch (IOException e) {
					e.printStackTrace();
				}
				finally {
					ACLMessage res = msg.createReply();
					send(res);
				}
			}
			else{
				block();
			}

		}
```

```
	}
```

En el constructor crearemos la plantilla la cuál debe de encajar con las características del mensaje que le tiene que llegar para poder en marcha todo el proceso de listado. En el método action esperaremos hasta que llegue un mensaje que encaje con dicha plantilla para avisar de que se requiere del listado de comunidades autónomas.

Después de dicho aviso, con la ayuda de Jsoup se parsea la página web para poder obtener así todos los nombres. Una vez hecho este proceso se envía un mensaje de vuelta a *AemetAgent* con todos los nombres.

La segunda función de este agente es atender a las solicitudes que llegan desde arriba (Desde el *AemetAgent*) sobre la obtención de información sobre una comunidad autonómica en particular. Para ello creamos este comportamiento como la extensión de un AchieveREResponder.

```
//Atendemos las peticiones de AEMETAgent.
	private class AtenderSolicitudes extends AchieveREResponder{
		Hashtable<String,LinkedList> comunidades_stations_table;

		public AtenderSolicitudes(Agent agent, MessageTemplate plantilla) {
			super(agent, plantilla);
		}

		protected ACLMessage handleRequest(ACLMessage request) {
			String solicitud = request.getContent();
			ACLMessage agree = request.createReply();

			if(comunidades_table.isEmpty() || solicitud.equals("all") ||
						comunidades_table.containsKey(solicitud)) {
				agree.setPerformative(ACLMessage.AGREE);
			} else {
				agree.setPerformative(ACLMessage.REFUSE);
			}
			return agree;
		}

		protected ACLMessage prepareResultNotification(ACLMessage request,
								ACLMessage response) {
			String filtro = request.getContent();
			ACLMessage inform = request.createReply();
			try {
				comunidades_stations_table = aemetAgent(filtro);
				if(comunidades_stations_table != null){
					inform.setContentObject(comunidades_stations_table);
					inform.setPerformative(ACLMessage.INFORM);
				}else {
					inform.setPerformative(ACLMessage.FAILURE);
				}
			} catch (IOException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
			return inform;
		}
```

Si la comunidad autónoma no existe se responderá con un Refuse, ya que obviamente no es posible devolver información sobre algo que no existe. En caso contrario, o en caso en el que se solicite la información sobre todas las comunidades autónomas, se enviará un Agree y se irá preparando el trámite para obtener la información final, como podemos ver en prepareResultNotificaction.

```
private Hashtable<String, LinkedList> aemetAgent(String filtro) {

                    Hashtable<String, String>comunidades_table = new Hashtable<String,
String>();
                    Hashtable<String, LinkedList>comunidades_stations_table =
                                        new Hashtable<String, LinkedList>();

                    System.out.println("\\nAGENTE AEMET\n----------------------");
                    try{
                          Document doc = Jsoup.connect(AEMET + ID_CC_AA_DIR).get();


                          Elements form = doc.select("form[name=\"frm1\"]");

                          Elements comunidades =
                                form.select("option:not(option[selected])");

                          for(Element comunidad: comunidades ) {
                                if(filtro.equals("all") ||
                                            filtro.equals(comunidad.attr("value")))

                                      comunidades_table.put(comunidad.attr("value"),
                                                                  comunidad.text());
                          }

                          LinkedList weatherStation_list = new LinkedList();

                          for (String comunidad_key : comunidades_table.keySet()) {
                                weatherStation_list = comunidadAutonoma(comunidad_key,
                                            comunidades_table.get(comunidad_key));

                                comunidades_stations_table.put(comunidad_key,
                                                        weatherStation_list);
                          }

                          return comunidades_stations_table;


                    }catch(Exception e){
                          System.out.println("Exception "+e);
                          return null;
                    }

              }
```

En este método llamaremos a otro llamado aemetAgent, el cuál tratará de manera individual cada comunidad para crear el mensaje que se va a devolver a *AemetAgent*. A su vez este método llama a otro método llamado comunidadAutonoma, donde el sistema divide las comunidades en enclaves (estaciones meteorológicas como tal) para así dar el último paso necesario.

```
private LinkedList comunidadAutonoma(String comunidad_key, String comunidad_name) {
                        String url_enclave = "";

                        Hashtable<String,String> localidades_table = new Hashtable<String,String>();
                        Hashtable<String, Hashtable<String,String>> provincias_table = new Hashtable<String,
Hashtable<String,String>>();
                        LinkedList weatherStation_list = new LinkedList();

                        String url_comunidad = AEMET + ULTIMOS_DATOS + "?k="+ comunidad_key +"&w=0";

                        System.out.println("\nAGENTE C.C.A.A.\n----------------------");
                        try{
                                Document doc = Jsoup.connect(url_comunidad).get();

                                provincias_table.clear();
                                Elements provincias = doc.select("form[name=\"frm1\"] optgroup");

                                for(Element elem_provincia: provincias) {
                                        String provincia = elem_provincia.attr("label");

                                        localidades_table.clear();
                                        Elements localidades = doc.select("form[name=\"frm1\"]
                                                        optgroup[label=\"" + provincia + "\"] option");

                                        for(Element elem_localidad: localidades) {
                                                String id_localidad = elem_localidad.attr("value");
                                                String localidad = elem_localidad.text();

                                                localidades_table.put(id_localidad, localidad);
                                        }

                                        provincias_table.put(provincia, localidades_table);

                                for(String localidad_key : provincias_table.get(provincia).keySet()) {
                                                url_enclave = AEMET + ULTIMOS_DATOS + "?k="+ comunidad_key +
                                                                                        "&l=" +
                                                localidad_key + "&w=0&datos=det&x=h24&f=temperatura";
                                                String localidad =
                                                        provincias_table.get(provincia).get(localidad_key);
                                                System.out.println(comunidad_key + ", " + provincia + ", " +
                                                                                                localidad);
                                                System.out.println(url_enclave);
                                                Location loc =  new Location();
                                                WeatherInformation w_info = new WeatherInformation();

                                                //Solicitud de la informacion referente a la estacion.
                                                ACLMessage aclmsg = new ACLMessage(ACLMessage.INFORM);
                                                aclmsg.addReceiver(new AID("EnclaveAgent",AID.ISLOCALNAME));
                                                aclmsg.setContent(url_enclave);
                                                send(aclmsg);


                                                ACLMessage msgloc = blockingReceive();
                                                if(msgloc!= null) {
                                                        LinkedList empaquetado = new LinkedList();

                                                        empaquetado = (LinkedList)msgloc.getContentObject();

                                                        loc = (Location)empaquetado.get(0);
                                                        w_info = (WeatherInformation) empaquetado.get(1);

                                                }
                                                else {
                                                        block();
                                                }
                                                //-------------------------------------------------
                                                System.out.println("\t Lat: " + loc.getLatitude() + " /
                                                                                Long: "+ loc.getLongitude());
                                                weatherStation_list.add(new WeatherStation(comunidad_key,
                                        comunidad_name, provincia, localidad_key, localidad, loc, w_info));
                                        }

                                }
```

```
                        }catch(Exception e){
                                System.out.println("Exception "+e);
                        }

                        return weatherStation_list;
                }
        }

}
```

Mediante jsoup se obtienen las localidades de cada comunidad para después enviar una petición **por cada localidad** al tercer agente, *EnclaveAgent*. Esa petición consiste en un mensaje ACLM (Como todos los que hemos enviado previamente) el cuál contiene la url que lleva hasta la información de dicha localidad en la página de la AEMET. Una vez todas las localidades han sido recorridas, se devuelve la lista de estaciones metereológicas.

3.3. EnclaveAgent

Es un agente muy similar a *ComunidadAutonomaAgent* pero atendiendo solicitudes sobre estaciones meteorológicas específicas.

El funcionamiento básico es el mismo, nada más desplegar el agente se le añade un comportamiento llamado *AtenderSolicitudes* que estará a la espera de que le llegue un mensaje.

```
protected void setup() {
              System.out.println("[Enclave] - Servicio Activo");
              this.addBehaviour(new AtenderSolicitudes());

       }
       //            ---------------- [ COMPORTAMIENTO ] ----------------
       // Comportamiento para atender las solicitudes de información
       private class AtenderSolicitudes extends CyclicBehaviour{

              @Override
              public void action() {
                     ACLMessage msg_recive = receive();

                     if (msg_recive!= null) {
                            Location loc = new Location();
                            WeatherInformation w_info = new WeatherInformation();

                            String url_enclave = msg_recive.getContent();
                            try {
                                   loc = enclaveInfo(url_enclave);
                                   w_info = enclaveWeatherInfo(url_enclave);

                                   LinkedList empaquetado = new LinkedList();
                                   empaquetado.add(loc);
                                   empaquetado.add(w_info);

                                   ACLMessage msg_out = msg_recive.createReply();
                                   msg_out.setContentObject(empaquetado);
```

```
                        send(msg_out);
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }else {
                    block();
                }
            }
```

Una vez que le llegue un mensaje realizará todo el proceso de parseo con Jsoup pero esta vez con la información final, a diferencia de los parseos anteriores que simplemente eran para obtener nombres y urls para que otro agente accediera.

```
private Location enclaveInfo(String url_enclave) {
                    Double latitude = null;
                    Double longitude = null;

//                  System.out.println("\nAGENTE ENCLAVE\n---------------------");
                    try{
                        Document doc = Jsoup.connect(url_enclave).get();

                        Elements latitudes = doc.select(".latitude");
                        for(Element elem_latitud: latitudes) {
                            latitude =
                        Double.parseDouble(elem_latitud.attr("title"));
                        }


                        Elements longitudes = doc.select(".longitude");
                        for(Element elem_longitudes: longitudes) {
                            longitude =
                        Double.parseDouble(elem_longitudes.attr("title"));
                        }

                    }catch(Exception e){
                        System.out.println("Exception "+e);
                    }

                    return new Location(latitude,longitude);
            }
```

La información que necesitamos son principalmente la localización para colocar la estación en su correcto lugar en el mapa y los datos relacionados con el clima de dicha estación. La localización la obtenemos con el método enclaveInfo que nos devuelve un objeto del tipo Location y los demás datos con el método enclaveWeatherInfo el cuál nos devuelve un objeto del tipo WeatherInformation.

```
private WeatherInformation enclaveWeatherInfo(String url_enclave) {
                    String fecha = null;
                    float temperatura_c = 0;
                    float velocidad_viento = 0;
                    String dir_viento = null;
                    String dir_viento_img = null;
                    float racha = 0;
                    String dir_racha = null;
                    String dir_racha_img = null;
                    float precipitacion = 0;
                    float presion = 0;
                    float tendencia = 0;
                    float humedad = 0;


                    try{
                           Document doc = Jsoup.connect(url_enclave).get();

                           int row = -1;
                           Elements datos = new Elements();
                           do {
                                  row++;
                                  datos = doc.select("tbody tr:eq("+ row + ") td");
                           }while(datos.get(1).text().isEmpty() && row < 23);

                           if (row == 23) {
                                  return null;
                           }
                           fecha = datos.get(0).text();
                           temperatura_c = (datos.get(1).text().isEmpty()) ? 0:
                           Float.valueOf(datos.get(1).text());

                           velocidad_viento = (datos.get(2).text().isEmpty()) ? 0:
                           Float.valueOf(datos.get(2).text());
                           // Dirección e imagen de la dirección del viento se toma
                           después.

                           racha = (datos.get(4).text().isEmpty()) ? 0:
                           Float.valueOf(datos.get(4).text());

                           //Dirección de la racha e imagen de la dirección de la racha
                           se toma después.

                           precipitacion = (datos.get(6).text().isEmpty()) ? 0:
                           Float.valueOf(datos.get(6).text());

                           presion = (datos.get(7).text().isEmpty()) ? 0:
                           Float.valueOf(datos.get(7).text());

                           tendencia = (datos.get(8).text().isEmpty()) ? 0:
                           Float.valueOf(datos.get(8).text());

                           humedad = (datos.get(9).text().isEmpty()) ? 0:
                           Float.valueOf(datos.get(9).text());

                           Elements imagenes = doc.select("tbody tr:eq("+ row + ") td
                           img");
                           int inicio  = 0;
                           if (!imagenes.isEmpty()){
```

```
                                    if(!datos.get(3).text().isEmpty()) {
                                          dir_viento =
(imagenes.get(inicio).text().isEmpty())? null : imagenes.get(inicio).attr("title");

                                          dir_viento_img =
(imagenes.get(inicio).text().isEmpty())? null :  "http://www.aemet.es" +
imagenes.get(inicio).attr("src");
                                          inicio++;
                                    }

                                    if(!datos.get(5).text().isEmpty()) {
                                          dir_racha =
(imagenes.get(inicio).text().isEmpty())? null : imagenes.get(inicio).attr("title");

                                          dir_racha_img =
(imagenes.get(inicio).text().isEmpty())? null :  "http://www.aemet.es" +
imagenes.get(inicio).attr("src");
                                    }
                              }

                        }catch(Exception e){
                              System.out.println("EnclaveInfo");
                              System.out.println("Exception "+e);
                        }

                        return new WeatherInformation(fecha, temperatura_c,
                                    velocidad_viento, dir_viento, dir_viento_img,
                                    racha, dir_racha, dir_racha_img, precipitacion,
                                    presion, tendencia, humedad);
                  }
            }
```

Los comportamientos AtenderSolicitudes son cíclicos porque deben de atender más de una petición. Si los hicieramos OneShot entonces solo atenderían al primer mensaje que les llegara por lo que faltaría muchísimos datos.

4. Visualización.

Para ello contamos con una clase llamada MainFrame, mencionada anteriormente en este documento, la cuál se encarga de toda la parte gráfica del sistema.

Se crea una instancia de esta clase cuando desplegamos *AemetAgent*. Este es el único agente con el que nos podemos comunicar a través de la interfaz gráfica mediante el botón Mostrar, el cuál se usa para mandar una petición para poder retribuir los datos de la comunidad autónoma seleccionada. Una vez procesados esos datos aparecerán automáticamente los marcadores de los enclaves en pantalla.

5. Almacenamiento de datos.

AEMETAgent tiene 3 clases para este proposito.

1.- Location guarda una localización que se usará después para poder situar una estación meteorológica en su lugar correspondiendote en el mapa.

2.- WeatherInformation sirve para almacenar todos los datos correspondientes con el clima de un lugar en específico. Dichos datos son:

- ✔ Hora en la que se obtuvo estos datos.
- ✔ La temperatura en grados Celsius.
- ✔ La velocidad y dirección del viento.
- ✔ La racha y dirección de la racha.
- ✔ Las precipitaciones.
- ✔ La presión.
- ✔ La tendencia.
- ✔ La humedad.

3.- WeatherStation contiene información sobre la estación meteorológica. A parte de tener un objeto Location y WeatherInformation, también contiene:

- ✔ Id y nombre de la comunidad autónoma a la que pertenece.
- ✔ Provincia a la que pertenece.
- ✔ Id y nombre del enclave al que pertenece.

Los marcadores del MainFrame son una representación gráfica de esta clase.

6. Manual de Usuario.

Requisitos:

- Tener JDK y JRE instalados en el equipo.

- Opcional: Tener un IDE para el desarrollo de programas en Java. Para el desarrollo y demostración del proyecto se usó Eclipse.

6. 1. Abriendo el proyecto

1º. Importar el archivo *.zip* en el IDE de elección.

2º. Este proyecto posee

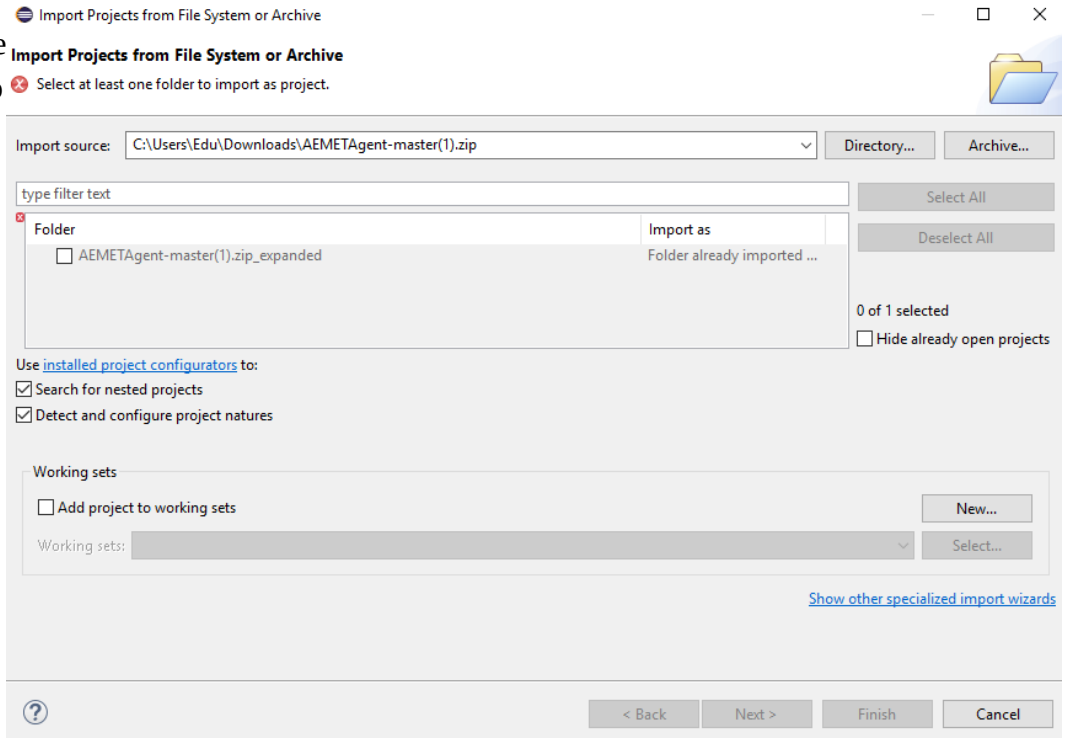

Ilustración 2: IMportación del archivo.

capacidad de despliegue automático, no hace falta crear los agentes manualmente en Jade. Para ello tenemos que añadir la siguiente cadena en los argumentos del programa.

-gui -host 127.0.0.1 -port 12344

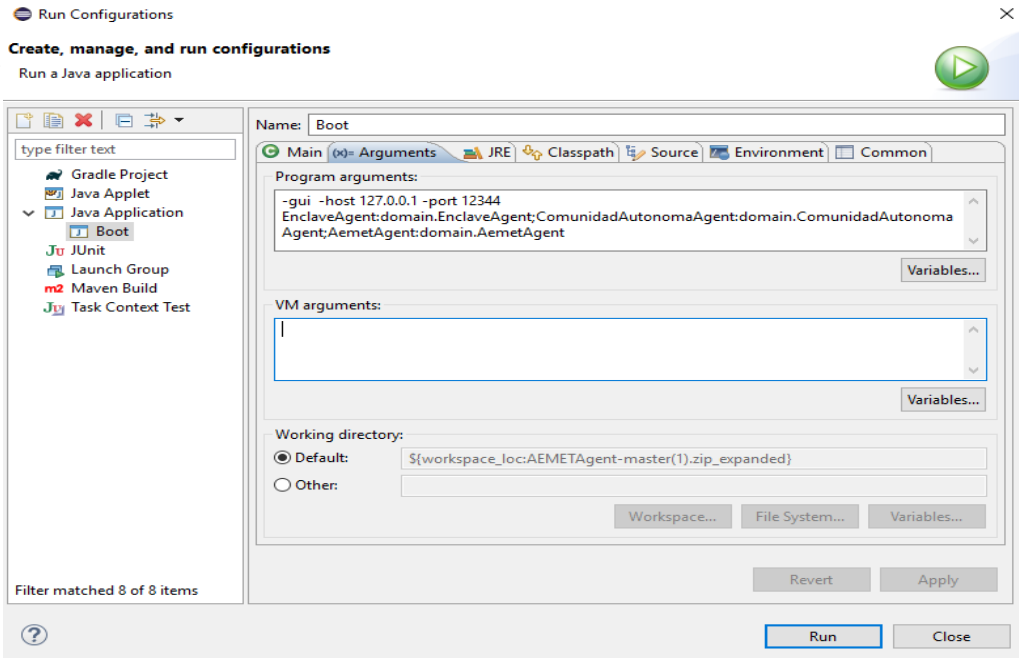

Ilustración 3: Configuración del entorno para la ejecución.

EnclaveAgent:domain.EnclaveAgent;ComunidadAutonomaAgent:domain.ComunidadAutonomaAgent;AemetAgent:domain.AemetAgent

3º. Una vez hecho todo esto se le puede dar al comando “Run” para empezar a usar el programa.

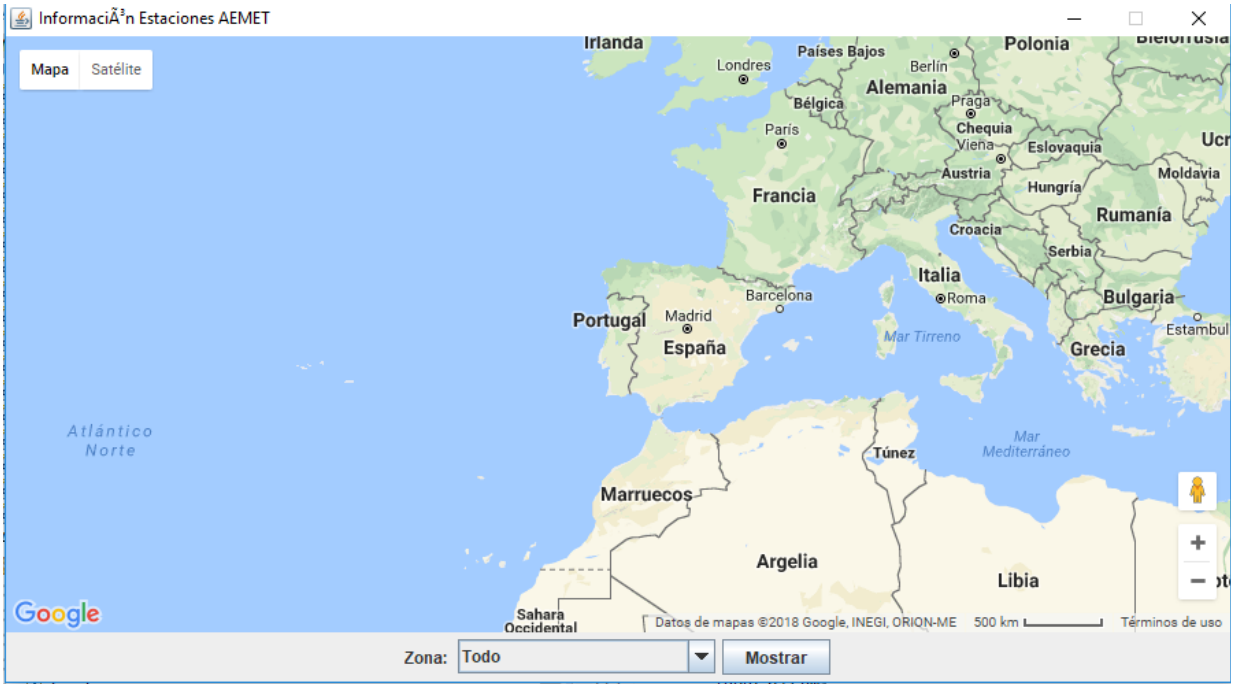

Ilustración 4: Pantalla inicial.

Alternativamente se puede bajar la versión de consumo la cuál solo requiere ejecutar el archivo “run.bat” para poder tener acceso al programa.

6. 2. Usando AEMETAgent

1º. Seleccionar la comunidad autónoma de la cuál se quiere obtener la información usando la barra de despliegue situada abajo en el centro y darle al botón “Mostrar”. Se puede seleccionar todo el conjunto de España, pero esto tardará bastante.

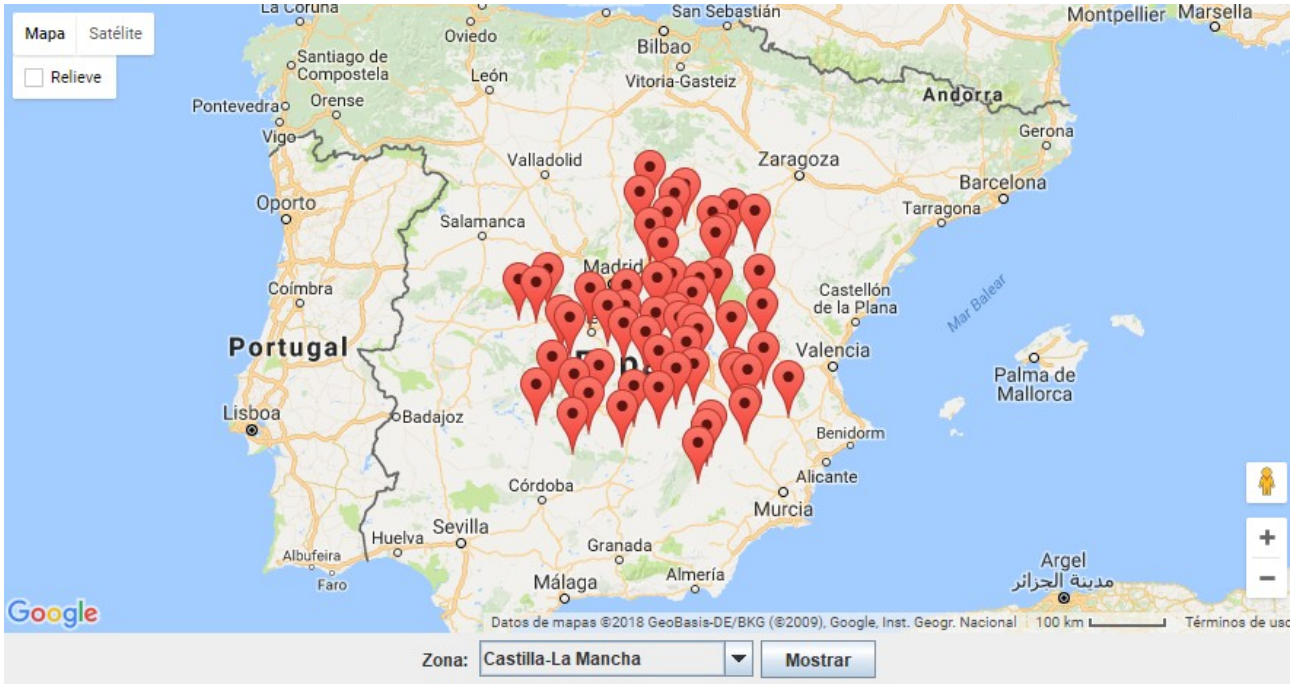

Ilustración 5: Presentación de las diferentes estaciones en el mapa.

2º. Una vez todos los marcadores han sido cargados, la información pertinente a la zona deseada se puede observar seleccionando el marcador correspondiente.

Ilustración 6: Representación de los datos obtenidos en la estación seleccionada.