Instituto Politécnico Nacional

Escuela Superior de Cómputo

Práctica 5 - Implementación de un diccionario distribuido

Unidad de aprendizaje: Aplicaciones para Comunicaciones de Red

Grupo: 3CM8

Alumnos(a):
Nicolás Sayago Abigail
Ramos Díaz Enrique

Profesor(a):
Moreno Cervantes Axel

28 de Mayo 2019

Índice

1. Introducción

Se implementará un servicio de diccionario con "n" servidores en línea, cada uno contendrá definiciones del tipo "palabra", "significado" y el contenido de cada servidor será distinto.

Cuando un cliente desee realizar una búsqueda en el diccionario, esta se realizará entre los servidores y se enviará el significado al cliente.

Si se realiza una visualización del diccionario, este se desplegará en orden alfabético.

Si se agregan términos al diccionario, éstos se agregaran solo al servidor con el que el cliente este conectado.

2. Desarrollo

2.1. Clase Controlador

Esta clase es la más importante, pues aquí se encuentran mapeados en un TreeMap<>() el servidor en el cual se encuentran ubicadas cada una de las palabras existentes en todo el diccionario distribuido.

Además, también están mapeados tanto el host como el puerto de cada uno de los servidores existentes en el sistema, siendo por el momento únicamente dos distintos.

Adicionalmente, la clase tiene sus respectivos getters y setters, para obtener el servidor de ubicación de una palabra, obtener todos los servidores en línea, obtener todas las palabras del diccionario distribuido, agregar nuevas palabras, obtener información especifica (host y puerto) de un servidor, etc.

Se podría decir que es una especie de "servidor central - controlador" que brinda la información del servidor al cuál el cliente deberá conectarse, con el pero de que ésta lógica esta implementada del lado del cliente.

También proporciona información para crear adecuadamente nuestros sockets del lado del cliente, hacia el servidor correcto.

```
1  //ServidorControlador
2  import java.net.*;
3  import java.io.*;
4  import java.util.*;
5
6  public class Controlador {
7   private TreeMap<String, String> ubicacion;
8   private HashMap<String, String[]> servidores;
9
10  public Controlador() {
11   ubicacion = new TreeMap<>();
```

```
12    ubicacion.put("Conejo", "servidor1");
13    ubicacion.put("Iguana", "servidor1");
14    ubicacion.put("Rana", "servidor1");
15    ubicacion.put("Dibujo", "servidor1");
16    ubicacion.put("Estrella", "servidor1");
17    ubicacion.put("Comida", "servidor1");
18    ubicacion.put("Familia", "servidor1");
19    ubicacion.put("Conversar", "servidor1");
20    ubicacion.put("Computadora", "servidor1");
21    ubicacion.put("Sombra", "servidor1");
22    ubicacion.put("Navegador", "servidor1");
23
24    ubicacion.put("Aguila", "servidor2");
25    ubicacion.put("Raiz", "servidor2");
26    ubicacion.put("Saltar", "servidor2");
27    ubicacion.put("Ropa", "servidor2");
28    ubicacion.put("Dia", "servidor2");
29    ubicacion.put("Amor", "servidor2");
30    ubicacion.put("Color", "servidor2");
31    ubicacion.put("Libro", "servidor2");
32    ubicacion.put("Lenguaje", "servidor2");
33    ubicacion.put("Lapiz", "servidor2");
34    ubicacion.put("Pais", "servidor2");
35    ubicacion.put("Oreja", "servidor2");
36
37    servidores = new HashMap<>();
38    String[] servidor1 = new String[]{"127.0.0.1", "4001"};
39    servidores.put("servidor1", servidor1);
40    String[] servidor2 = new String[]{"127.0.0.1", "4002"};
41    servidores.put("servidor2", servidor2);
42  }
43
44  public HashMap<String, String[]> getServidores() {
45    return servidores;
46  }
47
48  public String[] getPalabras() {
49    return ubicacion.keySet().toArray(new String[0]);
50  }
51
52  public boolean existePalabra(String palabra) {
53    return ubicacion.containsKey(palabra);
54  }
55
```

```
56  public void agregarPalabra(String palabra, String servidor) {
57   ubicacion.put(palabra, servidor);
58  }
59
60  public String obtenerServidor(String palabra) {
61   if(ubicacion.containsKey(palabra))
62    return ubicacion.get(palabra);
63   else
64    return "";
65  }
66
67  public String obtenerHostServidor(String servidor) {
68   return servidores.get(servidor)[0];
69  }
70
71  public int obtenerPtoServidor(String servidor) {
72   return Integer.parseInt(servidores.get(servidor)[1]);
73  }
74 }
```

2.2. Clase Cliente

En esta clase, que sirve más como una de utilería para las interfaces gráficas, se encuentran las siguientes funcionalidades:

2.2.1. Cargar y asignar servidores

Con ayuda de la clase Controlador, obtenemos todos los servidores existentes y los cargamos en un ComboBox.

Según la selección del éste por parte del usuario, será nuestro servidor "principal", en el cual se harán todas las búsquedas en primera instancia, así como el añadido de nuevas palabras, que serán exclusivas y sólo existirán en dicho servidor.

```
1  public class Cliente {
2   private static Controlador ctrl = new Controlador();
3   private static String servidorActual = "";
4   private static String hostActual = "";
5   private static int ptoActual = 0;
6   public static String[] nombresServidores;
7   public static String[][] infoServidor;
8
9   public static void cargarServidores() {
10   HashMap<String, String[]> servidores = ctrl.getServidores();
```

```
11    nombresServidores = servidores.keySet().toArray(new String[0]); //Nombres
       ↪ servidores
12    infoServidor = servidores.values().toArray(new String[0][0]); //Hosts [0]
       ↪ y puertos [1]
13    System.out.println("Informacion se servidores cargada\n");
14   }
15
16   public static void asignarServidor(int i) {
17    servidorActual = nombresServidores[i];
18    hostActual = infoServidor[i][0];
19    ptoActual = Integer.parseInt(infoServidor[i][1]);
20    System.out.println("Servidor actual: " + servidorActual);
21    System.out.println("Host: " + hostActual + ". Pto: " + ptoActual);
22   }
23  }
```

2.2.2. Buscar palabras - significados

Simplemente enviamos la palabra ingresada por el usuario al servidor al que se esta conectado actualmente, con un socket principal, a la espera de una respuesta.

Si la respuesta recibida es el significado, acabamos. Mostramos el significado en la interfaz del usuario.

En cambio, si la respuesta recibida es una cadena vacía, significa que la palabra no se encuentra en el servidor actual.

Luego de esto, le preguntamos a la clase Controlador si la palabra ingresada existe en algún otro servidor existente.

Si no existe, entonces le indicamos al usuario que la palabra ingresada no existe en ningún servidor.

Si sí existe, creamos otro socket "secundario" que se va a conectar al servidor en donde se encuentre la palabra. Realizamos el mismo procedimiento que al principio, y es un hecho que obtendremos como respuesta la definición. La mostramos al usuario.

```
1  public static String buscarPalabra(String palabra) {
2   String definicion = "";
3   try {
4    Socket cl = new Socket(hostActual, ptoActual);
5    System.out.println("\nConexion establecida al " + servidorActual);
6    DataOutputStream dos = new DataOutputStream(cl.getOutputStream());
      ↪ //OutputStream
7    DataInputStream dis = new DataInputStream(cl.getInputStream()); //
      ↪ InputStream
8
```

```
9   //Bandera = 0 - Buscar definicion
10  dos.writeInt(0);
11  System.out.println("Enviando palabra: " + palabra);
12  //Enviar palabra y la intento buscar en este servidor
13  dos.writeUTF(palabra);
14  dos.flush();
15  //Recibo respuesta del servidor actual, puede o no ser la definicion
16  String respuesta = dis.readUTF();
17  dos.close();
18  dis.close();
19  cl.close();
20  definicion = respuesta;
21  System.out.println("Definicion recibida: " + definicion);
22
23  //La definicion es vacia, por lo tanto hay que buscar en otro servidor
24  if(respuesta.equals("")) {
25   //Intentamos hallar el servidor
26   String servidorRemoto = ctrl.obtenerServidor(palabra);
27   /*Si la respuesta es vacia, la palabra no esta registada en ningun
28   servidor y terminamos*/
29   if(servidorRemoto.equals("")) {
30    System.out.println("Palabra no encontrada en ningun servidor. Regresando
      ↪ cadena vacia...");
31    definicion = "No encontrada";
32   }
33   else {
34    /*Sino, nos conectamos a ese servidor, y es seguro que la palabra
35     esta ahi*/
36    String hostRemoto = ctrl.obtenerHostServidor(servidorRemoto);
37    int ptoRemoto = ctrl.obtenerPtoServidor(servidorRemoto);
38
39    //Nuevo socket
40    Socket cl2 = new Socket(hostRemoto, ptoRemoto);
41    System.out.println("\nDefinicion no encontrada. Conectando al " +
      ↪ servidorRemoto);
42    System.out.println("Host: " + hostRemoto + ". Puerto: " + ptoRemoto);
43    DataOutputStream dos2 = new DataOutputStream(cl2.getOutputStream());
      ↪ //OutputStream
44    DataInputStream dis2 = new DataInputStream(cl2.getInputStream()); //
      ↪ InputStream
45
46    //Bandera = 0 - Buscar definicion
47    dos2.writeInt(0);
48    System.out.println("Enviando palabra: " + palabra);
```

```
49      //Enviar palabra y la intento buscar en otro servidor
50      dos2.writeUTF(palabra);
51      dos2.flush();
52      //Recibo definicion del servidor
53      definicion = dis2.readUTF();
54      System.out.println("Definicion recibida: " + definicion);
55      dos2.close();
56      dis2.close();
57      cl2.close();
58     }
59    }
60   } catch(Exception e){
61    e.printStackTrace();
62   }//catch
63   return definicion;
64  }
```

2.2.3. Agregar palabras - significados

Primero, le preguntamos al controlador si la palabra ya existe en algún servidor.

Si ya existe, terminamos. Se lo indicamos al usuario.

Si no existe, entonces enviamos la palabra junto a su significado al servidor actual, y esta se agrega en él. Además, agregamos la palabra y significado al TreeMap<>() de la clase Controlador.

```
1   public static boolean agregarPalabra(String palabra, String definicion) {
2    boolean success = false;
3    try {
4     if(ctrl.existePalabra(palabra) == false) {
5      Socket cl = new Socket(hostActual, ptoActual);
6      System.out.println("\nConexion establecida al " + servidorActual);
7
8      DataOutputStream dos = new DataOutputStream(cl.getOutputStream());
        ↪ //OutputStream
9      DataInputStream dis = new DataInputStream(cl.getInputStream()); //
        ↪ InputStream
10     // Bandera = 1 - Agregar palabra
11     dos.writeInt(1);
12
13     System.out.println("Enviando palabra: " + palabra);
14     // Enviar palabra y la agrega
15     dos.writeUTF(palabra);
16     dos.flush();
```

```
17      System.out.println("Enviando definicion: " + definicion);
18      // Enviar definicion
19      dos.writeUTF(definicion);
20      dos.flush();
21
22      dos.close();
23      dis.close();
24      cl.close();
25      //Registrando palabra en el Controlador
26      ctrl.agregarPalabra(palabra, servidorActual);
27      success = true;
28     }
29    } catch(Exception e) {
30     e.printStackTrace();
31    }//catch
32    return success;
33  }
```

2.2.4. Visualizar todas las palabras del diccionario

Simplemente obtenemos todo el TreeMap<>() con las palabras de la clase Controlador, y las mostramos en un JList con ayuda de un objeto de tipo DefaultListModel¡String¿asociado a la lista.

2.3. Servidores

Todos los servidores tienen implementada la misma funcionalidad, lo único que cambia son las palabras - significados registradas en un HashMap<>() en cada uno de ellos, así como el puerto y la dirección host.

Dentro el main, creamos un ServerSocket con el host y el puerto, y posteriormente declaramos un ciclo infinito, a la espera de peticiones de los clientes.

Con ayuda de una bandera, vamos a indicarle al servidor que realice dos distintas acciones: agregar una palabra junto a su definición, y devolver la definición de una palabra.

Cuando la bandera es igual a cero, recibimos la palabra y la buscamos dentro del HashMap<>() por medio del método **containsKey(palabra)**. Esto regresa el significado si es que la palabra (llave) existe en el mapa, si no, devuelve nulo, por medio del método **get(palabra)**.

Por último, le enviamos el significado al cliente.

Cuando la bandera es igual a 1, significa que el cliente quiere añadir una nueva palabra junto a su significado utilizando el método **put(palabra)**.

Todos estos métodos pertenecen a la clase HashMap<>().

2.3.1. Servidor 1

```
1  import java.net.*;
2  import java.io.*;
3  import java.util.*;
4
5  public class Servidor1 {
6   private static HashMap<String, String> diccionario = new HashMap<>();
7
8   public static void agregarDefinicion(String palabra, String definicion) {
9    diccionario.put(palabra, definicion);
10  }
11
12  public static void buscarDefinicion(String palabra, DataOutputStream dos) {
13   String respuesta = "";
14   System.out.println("Buscando definicion para: " + palabra);
15   if(diccionario.containsKey(palabra))
16    respuesta = diccionario.get(palabra);
17
18   try {
19    System.out.println("Definicion encontrada: " + respuesta);
20    dos.writeUTF(respuesta);
21    dos.flush();
22   } catch(Exception e) {
23    e.printStackTrace();
24   }
25  }
26
27  public static void main(String[] args) {
28   // Definiciones
29   diccionario.put("Conejo", "Mamifero de cuerpo alargado y arqueado de unos
      ↪ 40 cm de longitud, pelo suave y espeso, orejas largas, cola corta y
      ↪ patas traseras mas desarrolladas que las delanteras.");
30   diccionario.put("Iguana", "Reptil escamoso americano que puede alcanzar
      ↪ hasta 1,80 m de longitud, con la lengua no protractil y los dientes
      ↪ sobre la superficie interna de las mandibulas.");
31   diccionario.put("Rana", "Anfibio sin cola, de piel lisa y brillante,
      ↪ tronco rechoncho, cabeza grande y ojos saltones, con las extremidades
      ↪ posteriores muy desarrolladas para saltar.");
32   diccionario.put("Dibujo", "Forma que resulta de combinarse las lineas,
      ↪ figuras y otros elementos que adornan una cosa o que son constitutivos
      ↪ de ella.");
33   diccionario.put("Estrella", "Astro o cuerpo celeste que brilla con luz
      ↪ propia en el firmamento.");
```

```
34  diccionario.put("Comida", "Sustancia sólida que se come y sirve de
     ↪  alimento.");
35  diccionario.put("Familia", "Grupo de personas formado por una pareja
     ↪  (normalmente unida por lazos legales o religiosos), que convive y
     ↪  tiene un proyecto de vida en comun, y sus hijos, cuando los tienen.");
36  diccionario.put("Conversar", "Hablar [una persona] con otra sobre algo
     ↪  alternando los turnos de palabra.");
37  diccionario.put("Computadora", "Maquina electronica capaz de almacenar
     ↪  informacion y tratarla automaticamente mediante operaciones
     ↪  matematicas y logicas controladas por programas informaticos.");
38  diccionario.put("Sombra", "Imagen oscura que proyecta un cuerpo opaco
     ↪  sobre una superficie al interceptar los rayos de luz.");
39  diccionario.put("Navegador", "Programa que permite navegar por internet u
     ↪  otra red informatica de comunicaciones.");
40
41  try {
42   ServerSocket s = new ServerSocket(4001);
43   s.setReuseAddress(true);
44   System.out.println("Servidor1 iniciado, esperando cliente...");
45
46   for( ; ; ) {
47    Socket cl = s.accept();
48    System.out.println("\nCliente conectado desde " + cl.getInetAddress() +
      ↪  " " + cl.getPort());
49    DataOutputStream dos = new DataOutputStream(cl.getOutputStream());
      ↪  //OutputStream
50    DataInputStream dis = new DataInputStream(cl.getInputStream()); //
      ↪  InputStream
51    int bandera = dis.readInt();
52
53    if(bandera == 0) {
54     // Buscar definicion
55     String palabra = dis.readUTF();
56     buscarDefinicion(palabra, dos);
57    }
58    else if(bandera == 1) {
59     // Agregar definiciones
60     String palabra = dis.readUTF();
61     String definicion = dis.readUTF();
62     agregarDefinicion(palabra, definicion);
63    }
64    dis.close();
65    dos.close();
66    cl.close();
```

```
67     }//for
68    } catch(Exception e) {
69     e.printStackTrace();
70    }
71   }
72  }
```

2.3.2. Servidor 2

```
1  import java.net.*;
2  import java.io.*;
3  import java.util.*;
4
5  public class Servidor2 {
6   protected static HashMap<String, String> diccionario = new HashMap<>();
7
8   public static void agregarDefinicion(String palabra, String definicion) {
9    diccionario.put(palabra, definicion);
10  }
11
12  public static void buscarDefinicion(String palabra, DataOutputStream dos) {
13   String respuesta = "";
14   System.out.println("Buscando definicion para: " + palabra);
15   if(diccionario.containsKey(palabra))
16    respuesta = diccionario.get(palabra);
17
18   try {
19    System.out.println("Definicion encontrada: " + respuesta);
20    dos.writeUTF(respuesta);
21    dos.flush();
22   } catch(Exception e) {
23    e.printStackTrace();
24   }
25  }
26
27  public static void main(String[] args) {
28   // Definiciones
29   diccionario.put("Aguila", "Ave rapaz falconiforme, de aproximadamente 2 m
      ↪ de envergadura, con vista muy aguda.");
30   diccionario.put("Raiz", "Organo de las plantas que crece hacia el interior
      ↪ de la tierra, por el que se fijan al suelo y absorben las sustancias
      ↪ necesarias para su crecimiento.");
31   diccionario.put("Saltar", "Elevarse del suelo u otra superficie con
      ↪ impulso para caer en el mismo lugar o en otro.");
```

```
32    diccionario.put("Ropa", "Nombre generico de cualquier pieza de tela
       ↪  confeccionada que viste a una persona, un objeto o un lugar.");
33    diccionario.put("Dia", "Tiempo que emplea la Tierra en dar una vuelta
       ↪  sobre si misma, equivalente a 24 horas, y que se utiliza como unidad
       ↪  de tiempo; se cuenta normalmente desde las doce de la noche hasta
       ↪  veinticuatro horas despues.");
34    diccionario.put("Amor", "Sentimiento de vivo afecto e inclinacion hacia
       ↪  una persona o cosa a la que se le desea todo lo bueno.");
35    diccionario.put("Color", "Impresion que producen en la retina los rayos de
       ↪  luz reflejados y absorbidos por un cuerpo, segun la longitud de onda
       ↪  de estos rayos.");
36    diccionario.put("Libro", "Conjunto de hojas de papel, pergamino, vitela,
       ↪  etc., manuscritas o impresas, unidas por uno de sus lados y
       ↪  normalmente encuadernadas, formando un solo volumen.");
37    diccionario.put("Lenguaje", "Capacidad propia del ser humano para expresar
       ↪  pensamientos y sentimientos por medio de la palabra.");
38    diccionario.put("Lapiz", "Utensilio para escribir, dibujar o pintar que
       ↪  consiste en una barra delgada y larga generalmente de madera, con una
       ↪  mina cilindrica fina de grafito u otra sustancia mineral en el
       ↪  interior que sobresale por uno de los extremos de esta barra cuando
       ↪  esta afilado.");
39    diccionario.put("Pais", "Comunidad social con una organizacion politica
       ↪  comun y un territorio.");
40    diccionario.put("Oreja", "Parte externa del oido del hombre y otros
       ↪  mamiferos, formada por un repliegue cutaneo sostenido por una lamina
       ↪  cartilaginosa.");
41
42    try {
43     ServerSocket s = new ServerSocket(4002);
44     s.setReuseAddress(true);
45     System.out.println("Servidor2 iniciado, esperando cliente...");
46
47     for( ; ; ) {
48      Socket cl = s.accept();
49      System.out.println("\nCliente conectado desde " + cl.getInetAddress() +
         ↪  " " + cl.getPort());
50      DataOutputStream dos = new DataOutputStream(cl.getOutputStream());
         ↪  //OutputStream
51      DataInputStream dis = new DataInputStream(cl.getInputStream()); //
         ↪  InputStream
52      int bandera = dis.readInt();
53
54      if(bandera == 0) {
55       // Buscar definicion
```

```
56       String palabra = dis.readUTF();
57       buscarDefinicion(palabra, dos);
58      }
59      else if(bandera == 1) {
60       // Agregar definiciones
61       String palabra = dis.readUTF();
62       String definicion = dis.readUTF();
63       agregarDefinicion(palabra, definicion);
64      }
65      dis.close();
66      dos.close();
67      cl.close();
68     }//for
69    } catch(Exception e) {
70     e.printStackTrace();
71    }
72   }
73  }
```

3. Pruebas

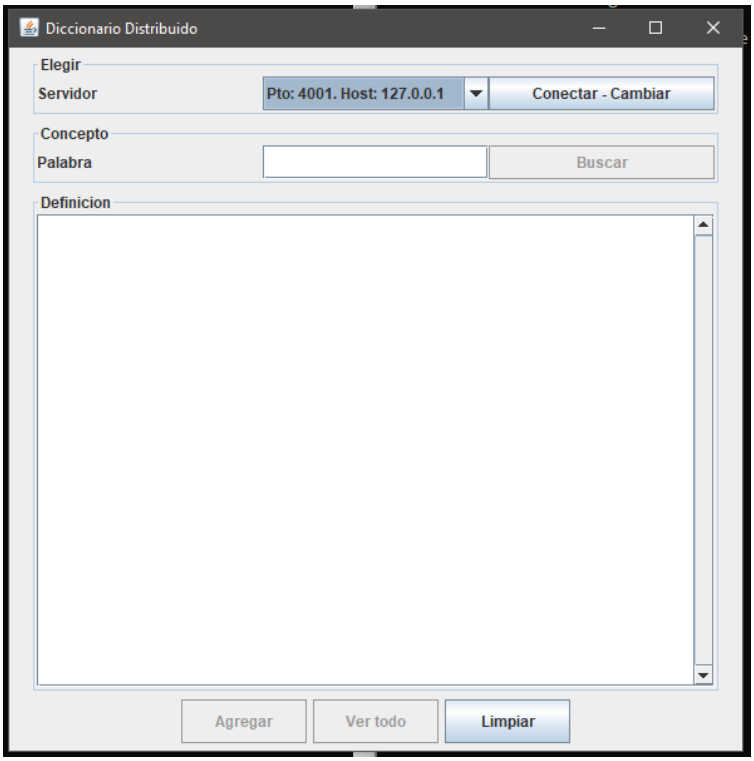

Figura 1: Interfaz diccionario distribuido

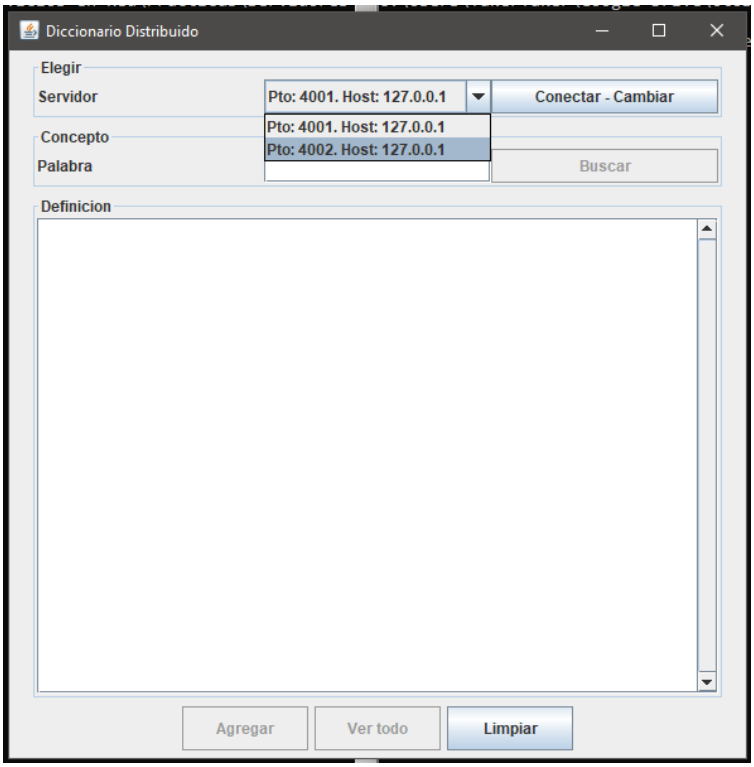

Figura 2: Servidores en línea disponibles

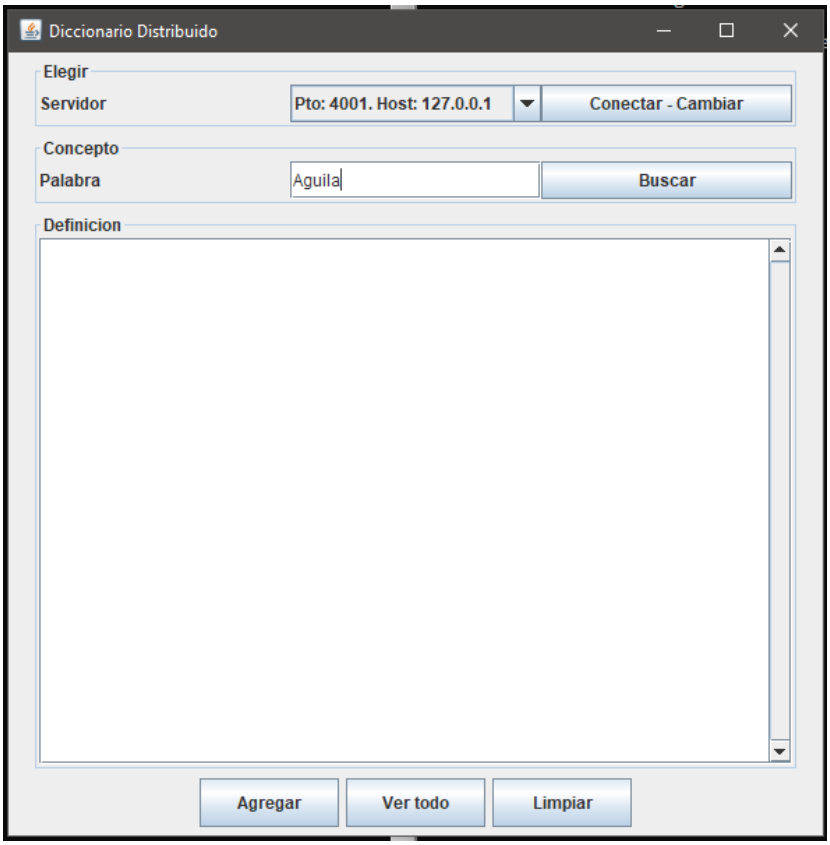

Figura 3: Búsqueda de una palabra en el servidor actual

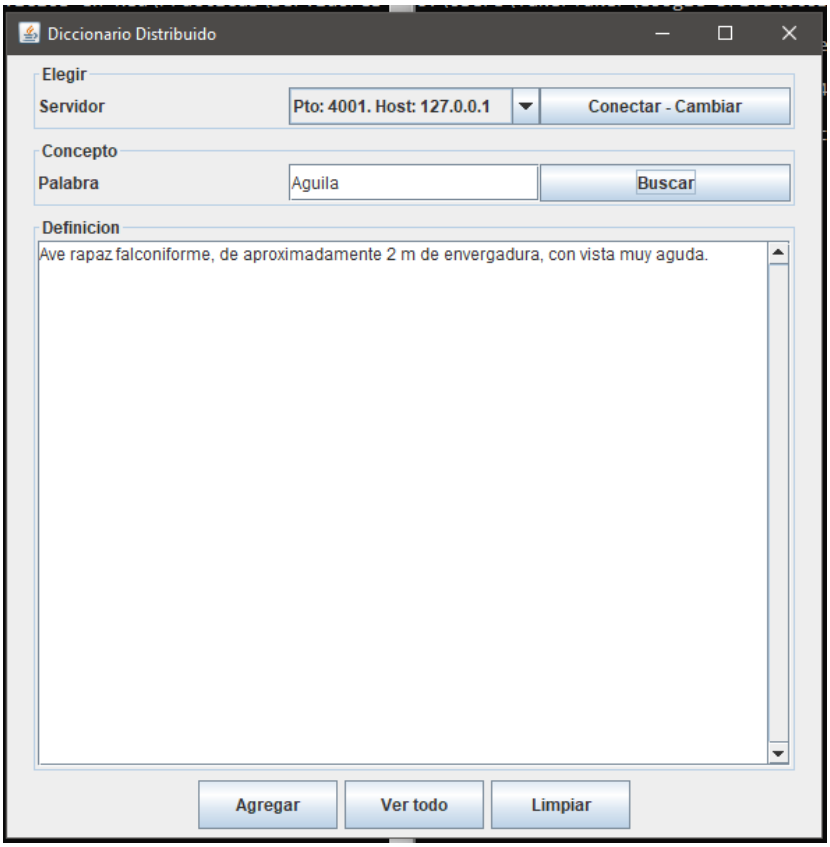

Figura 4: Definición devuelta el servidor actual

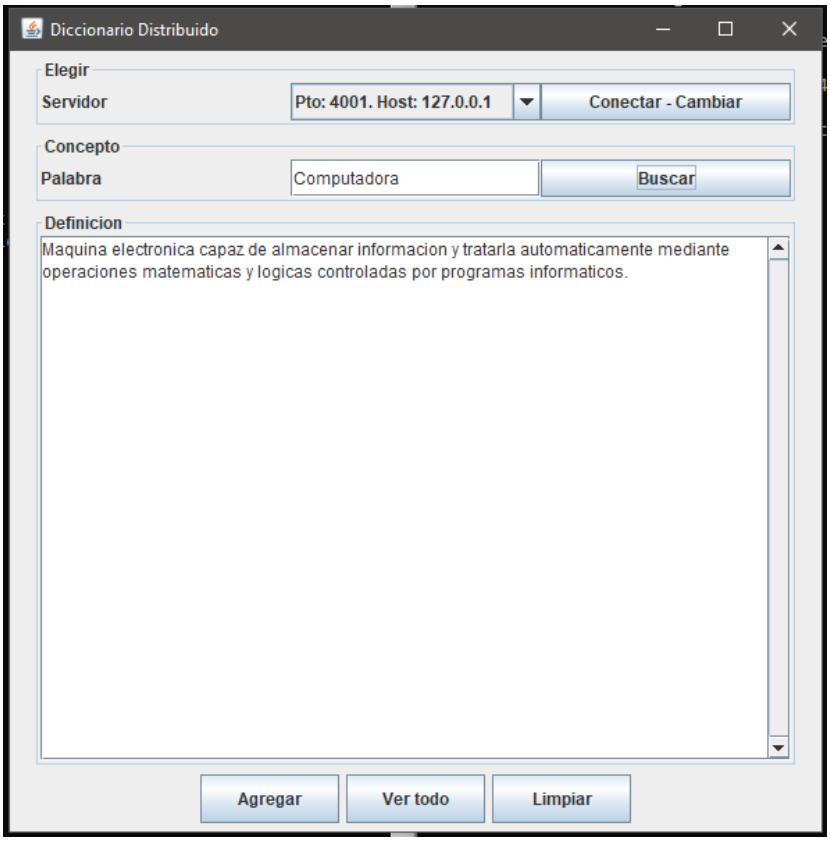

Figura 5: Búsqueda de una palabra en otro servidor y definición devuelta por este

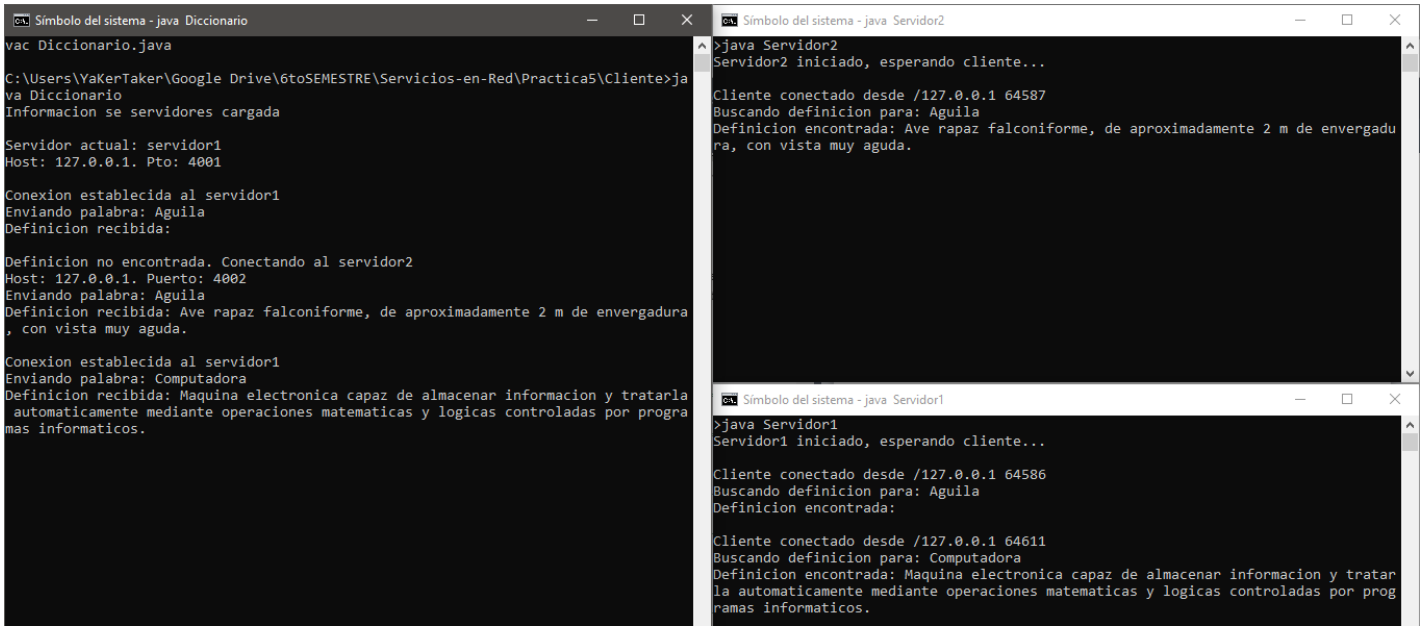

Figura 6: Peticiones del cliente y respuestas de los servidores

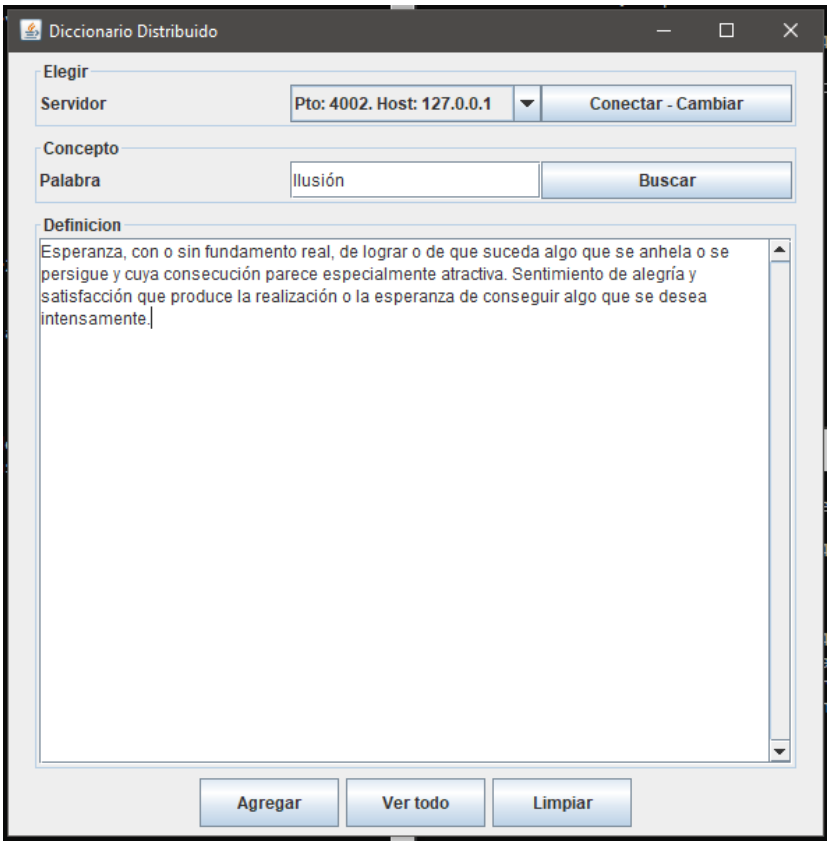

Figura 7: Ingreso de una nueva palabra - definición

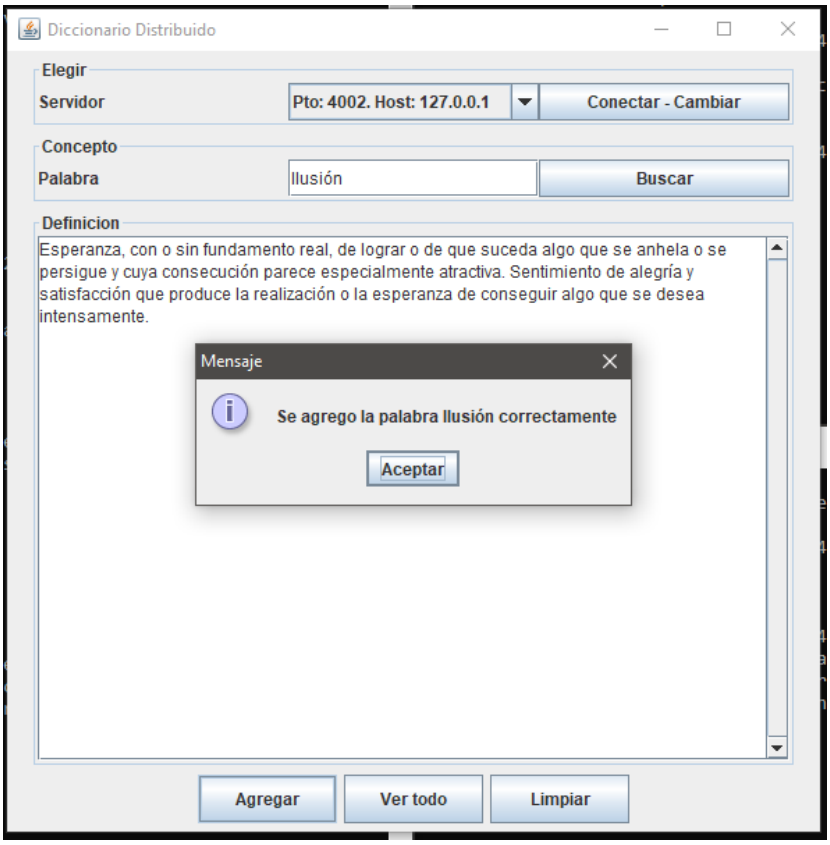

Figura 8: Palabra y definición agregada al servidor actual

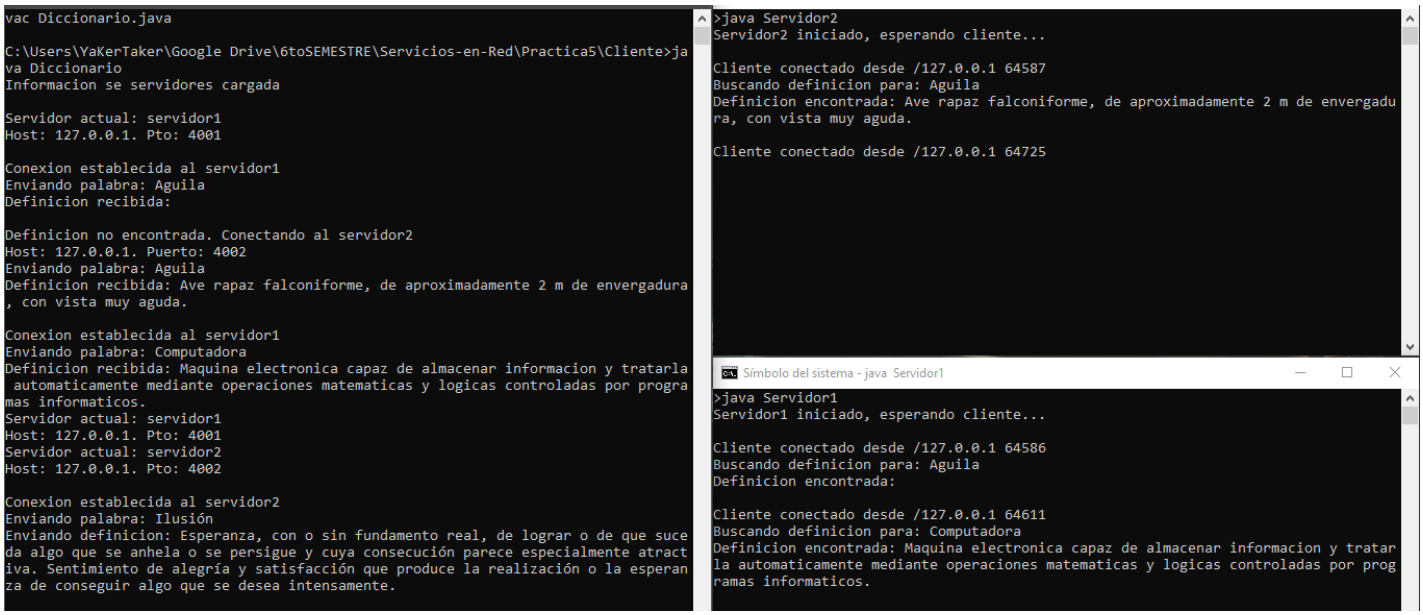

Figura 9: Cambio entre servidor actual desde el cliente

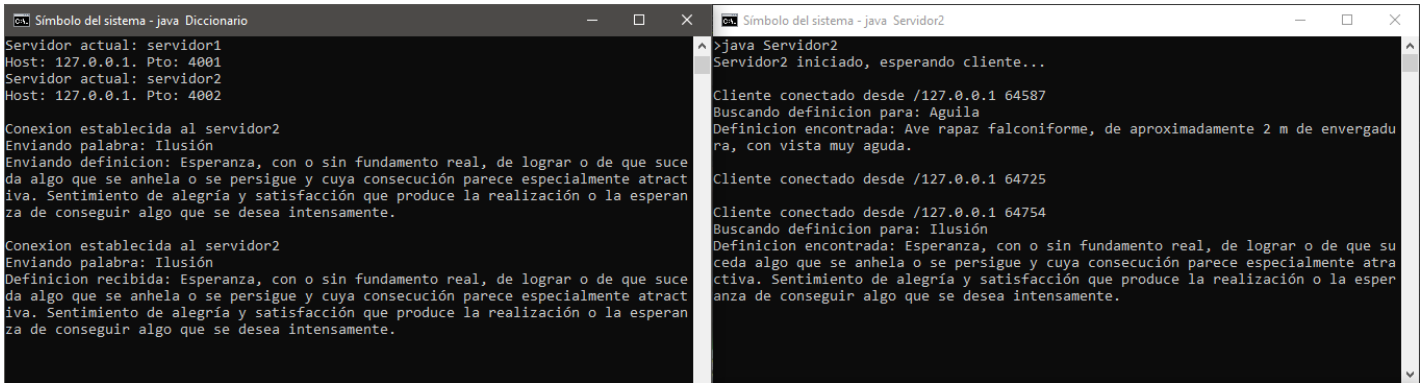

Figura 10: Palabra y definición enviada y agregada al servidor actual

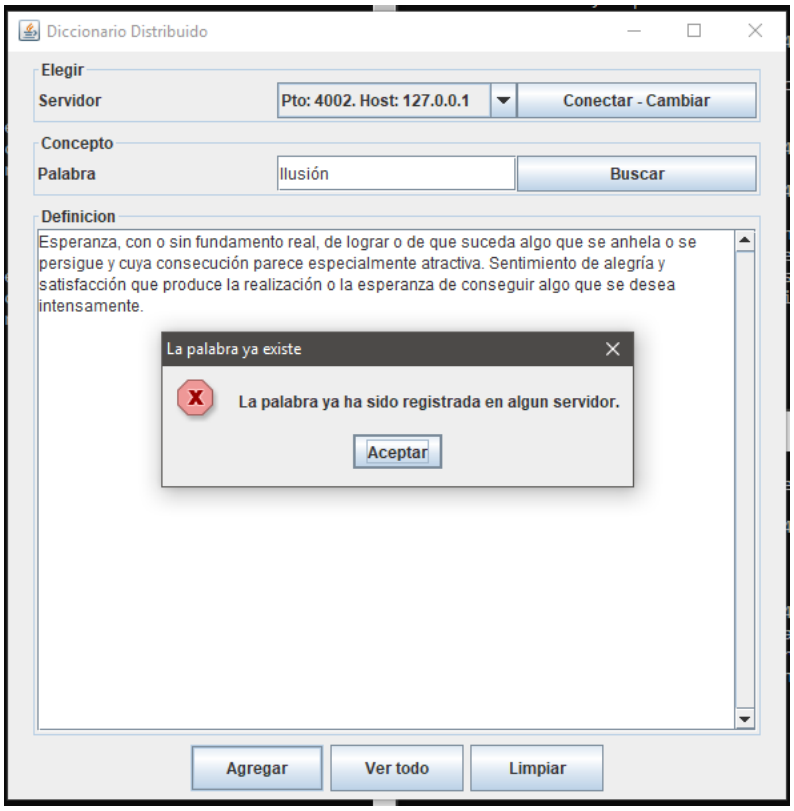

Figura 11: Mensaje de error al intentar añadir una palabra que ya existe en los servidores distribuidos

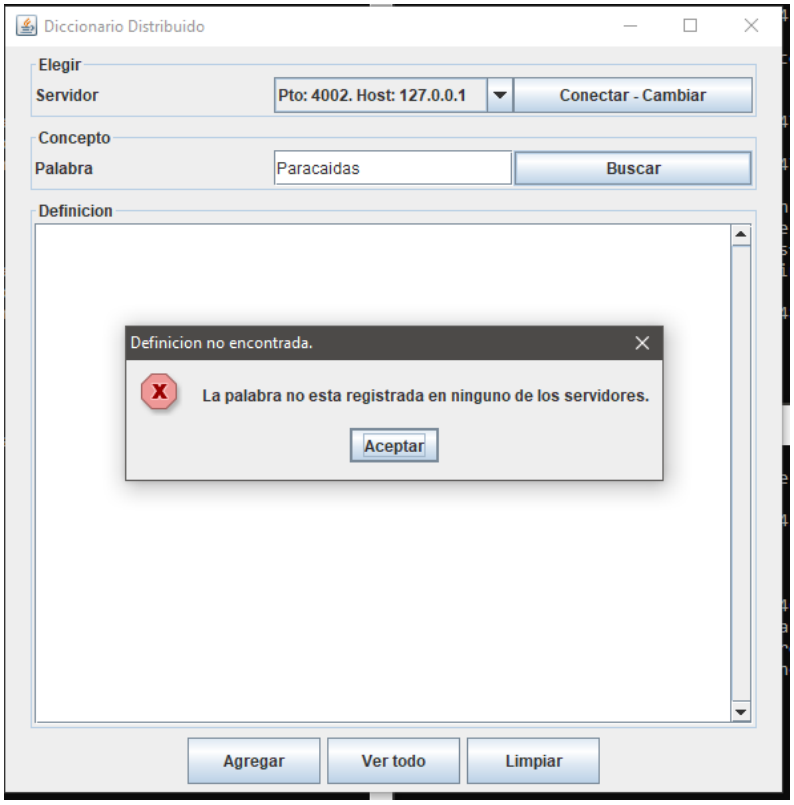

Figura 12: Mensaje de error al intentar buscar una palabra que no existe en los servidores distribuidos

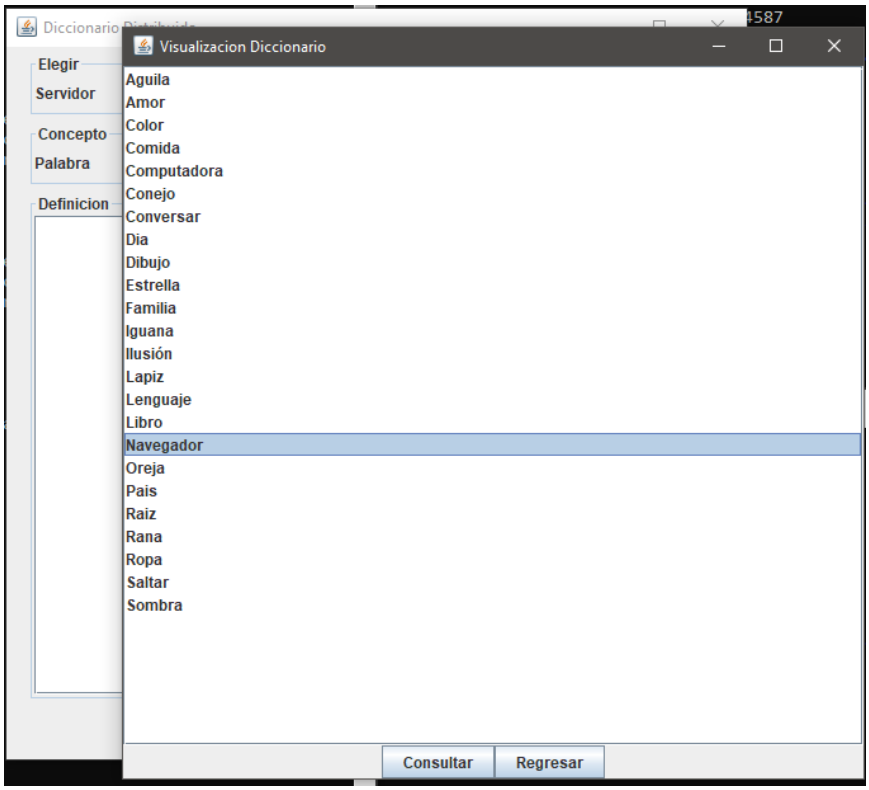

Figura 13: Visualización completa del diccionario distribuido, junto a todas las palabras en él

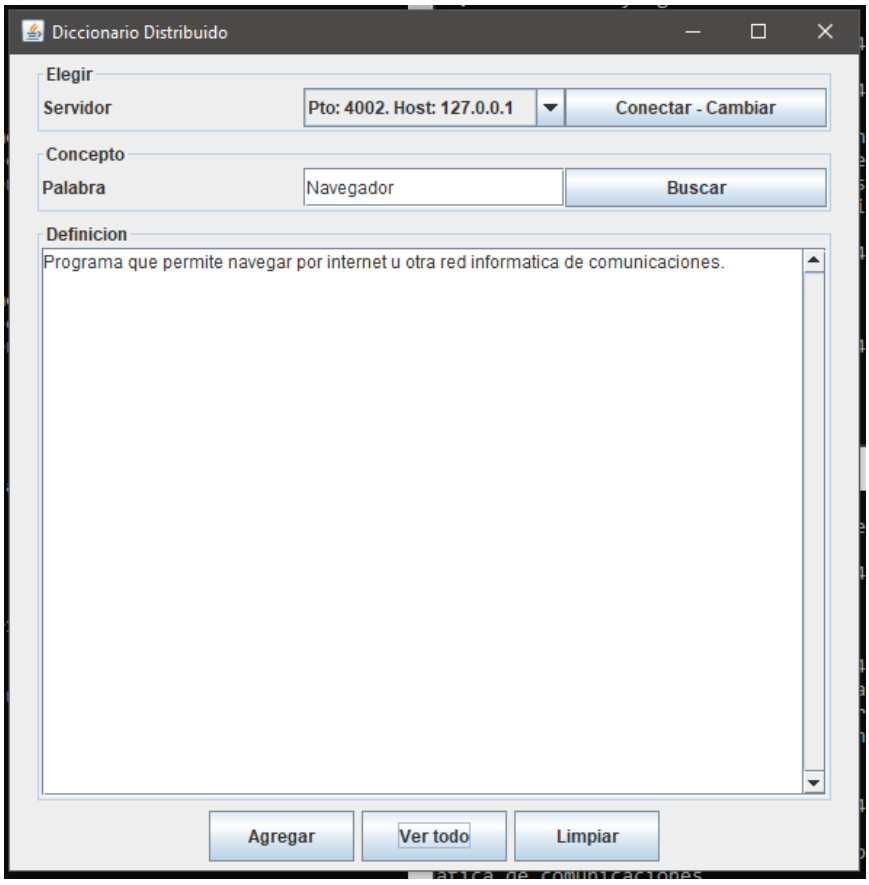

Figura 14: Búsqueda de definición desde la visualización completa del diccionario distribuido

4. Posibles mejoras

1. Agregar más de una definición a una misma palabra
2. Editar la definición de una palabra
3. Eliminar la definición de una palabra
4. Eliminar una palabra del diccionario distribuido
5. Permitir al usuario ordenar como desee las palabras en la visualización del diccionario completo
6. Añadir un buscador al diccionario para buscar palabras manualmente, y encontrar sus significados
7. Añadir diccionario de sinónimos y antónimos
8. Trasladar la lógica de la clase Controlador al lado del servidor en vez del cliente, teniendo así un servidor central que se encargue de distribuir las peticiones del cliente a los distintos servidores

5. Conclusiones

5.1. Nicolás Sayago Abigail

Al finalizar esta práctica pude comprender de mejor manera los sockets de flujo puesto que la práctica pasada ya los habíamos usado. Los métodos que se usaron fueron intuitivos y fáciles de utilizar.

Considero que una de las grandes ventajas que como equipo tuvimos, fue que primero discutimos el diseño de la aplicación en general, como las clases, la forma de cargar los datos, los diseños de las interfaces y la asignación de tareas para cada integrante.

5.2. Ramos Diaz Enrique

El trabajo de comunicación y transmisión de archivos no fue problema, pues la clases Socket y ServerSockets (sockets de flujo bloqueante) en Java hacen todo éste proceso con métodos muy simples de utilizar y comprender.

En general, manejar los sockets con sus respectivos métodos fue fácil gracias a las clases implementadas de tal forma que al comunicar el servidor con el cliente, se envía y recibían objetos que fueron implementados anteriormente. Uno de los retos a los que nos enfrentamos fue a un diseño agradable e intuitivo para el usuario.