C. CÓDIGOS DEL SERVICIO DE EJEMPLO

C.1 *Servlet* de la aplicación: Servlet.java

```
1  package com.ericsson;
2
3  import java.io.IOException;
4  import java.util.Random;
5
6  import javax.servlet.ServletException;
7  import javax.servlet.sip.SipServlet;
8  import javax.servlet.sip.SipServletRequest;
9  import javax.servlet.sip.SipServletResponse;
10 import javax.servlet.sip.SipSession;
11
12 public class Servlet extends SipServlet
13 {
14     /**
15      * Generador de un número aleatorio
16      */
17     private Random random = new Random();
18
19     /**
20      * Indica si se recibió el ACK de la sesión
21      */
22     private final String ACK_RECEIVED = "ackReceived";
23     /**
24      * Atributo de la sesión donde se almacena el número a adivinar
25      */
26     private final String NUMBER_TO_GUESS = "number";
27     /**
28      * La sesión se acepta - genera número a adivinar
29      */
30     protected void doAck(SipServletRequest req) throws ServletException,
           IOException
31     {
32         SipSession session = req.getSession(true);
33         // Ensure we do not process the ACK twice for the dialog
34         Object ackAlreadyProcessed = session.getAttribute(ACK_RECEIVED);
35         if (ackAlreadyProcessed == null)
36         {
```

```
37              session.setAttribute(ACK_RECEIVED, true);
38              // proporciona un valor entre 0 (inclusive) y 11 (exclusive)
                    por lo que [0, 10]
39              session.setAttribute(NUMBER_TO_GUESS, random.nextInt(11));
40              SipServletRequest message = session.createRequest("MESSAGE");
41              message.setContent("Adivine un número entre 0 y 10".getBytes(),
                     "guess/start");
42              message.send();
43          }
44      }
45
46      /**
47       * Termina la sesión
48       */
49      protected void doBye(SipServletRequest req) throws ServletException,
            IOException
50      {
51          req.getSession().invalidate();
52      }
53
54      /**
55       * Invitación recibida - aceptarla
56       */
57      protected void doInvite(SipServletRequest req) throws ServletException
            , IOException
58      {
59          // acepta la invitación - envía 200 OK
60          SipServletResponse response = req.createResponse(200);
61          response.send();
62      }
63
64      /**
65       * Número recibido. Comprueba si el número coincide con el generado
             por el sistema
66       * y envía el mensaje apropiado
67       */
68      protected void doMessage(SipServletRequest req) throws
            ServletException, IOException
69      {
70          SipSession session = req.getSession(true);
71          Object numberObject = session.getAttribute(NUMBER_TO_GUESS);
72          // Se asegura que todo va bien
73          if(numberObject == null)
74          {
75              req.createResponse(400, "Sesión no establecida").send();
76          }
77          else
78          {
79              // Responde 200 OK
```

```
80              req.createResponse(200).send();
81
82              // Obtiene el número adivinado
83              String content = new String(req.getRawContent());
84              int guess = Integer.parseInt(content);
85              int number = (Integer)numberObject;
86
87              // Crea el mensaje que mandaremos al cliente
88              SipServletRequest message = session.createRequest("MESSAGE");
89
90              // Si se acierta
91              boolean gameOver = false;
92              if(guess == number)
93              {
94                  message.setContent("Ha acertado!".getBytes(), "guess/result
                        +good");
95                  gameOver = true;
96              }
97              else
98              {
99                  // De lo contrario el usuario tendrá que seguir adivinando
100                 String smaller = "Incorrecto, el número es... " + (guess<
                        number?"mayor":"menor");
101                 message.setContent(smaller.getBytes(), "guess/result+wrong"
                        );
102             }
103             // Envía el resultado
104             message.send();
105
106             // Cierra la sesión si el usuario no acierta
107             if(gameOver)
108             {
109                 session.createRequest("BYE").send();
110                 session.invalidate();
111             }
112         }
113     }
114 }
```

Listado C.1: Servlet de la aplicación.

C.2 Cliente de la aplicación: Cliente.java

```
1  package com.ericsson;
2
3  import java.awt.BorderLayout;
4  import java.awt.Button;
5  import java.awt.FlowLayout;
```

```
6  import java.awt.Font;
7  import java.awt.Frame;
8  import java.awt.Panel;
9  import java.awt.TextArea;
10 import java.awt.event.ActionEvent;
11 import java.awt.event.ActionListener;
12 import java.awt.event.WindowAdapter;
13 import java.awt.event.WindowEvent;
14
15 import com.ericsson.icp.IBase;
16 import com.ericsson.icp.ICPFactory;
17 import com.ericsson.icp.IPlatform;
18 import com.ericsson.icp.IProfile;
19 import com.ericsson.icp.IService;
20 import com.ericsson.icp.ISession;
21 import com.ericsson.icp.util.ErrorReason;
22 import com.ericsson.icp.util.SdpFactory;
23
24 public class Cliente extends Frame
25 {
26
27     private IPlatform platform;
28     private IProfile profile;
29     /**
30      * Referencia al servicio ICP para manipular mensajes en modo búsqueda
31
32      */
33     private IService service;
34     /**
35      * La sesión del juego actual
36      */
37     private ISession session;
38
39     private Button startGameButton;
40
41     /**
42      * Main
43      */
44     public static void main(String[] args)
45     {
46         Cliente guess = new Cliente();
47         guess.setVisible(true);
48     }
49
50     public Cliente()
51     {
52         setFont(new Font("SansSerif", Font.PLAIN, 12));
53         // Crea la interfaz de usuario
54         createGui();
```

```
55          try
56          {
57              // Crea la plataforma y servicio ICP
58              platform = ICPFactory.createPlatform();
59              platform.registerClient("GuessClient");
60              platform.addListener(new PlatformAdapter());
61
62              // Conecta al perfil usado en la aplicación
63              profile = platform.createProfile("ImsSetting");
64              profile.addListener(new ProfileAdapter());
65
66              // Crea el servicio ICP. Se usará para crear una sesión en el
67          //  método startGame()
68              service = profile.createService("+g.cliente.ericsson.com", "
                    cliente.ericsson.com");
69              service.addListener(new ServiceAdapter());
70          }
71          catch (Exception e)
72          {
73              showError("No se ha podido arrancar la ICP", e);
74          }
75      }
76
77      /**
78       * Crea la GUI
79       */
80      private void createGui()
81      {
82          // Set the application title and size
83          // Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize
                ();
84          // setSize(new Dimension(screenSize.width / 2, screenSize.height /
                2));
85          setTitle("Cliente ¡Adivine un número!");
86          setLayout(new BorderLayout());
87
88          Panel buttonPanel = new Panel();
89          buttonPanel.setLayout(new FlowLayout());
90          // Botones
91          startGameButton = new Button("Nuevo Juego");
92          startGameButton.addActionListener(new ActionListener()
93          {
94
95              public void actionPerformed(ActionEvent event)
96              {
97                  startGame();
98              }
99          });
100         buttonPanel.add(startGameButton);
```

```
101         Button quit = new Button("Salir");
102         quit.addActionListener(new ActionListener()
103         {
104             public void actionPerformed(ActionEvent event)
105             {
106                 quit();
107             }
108         });
109 
110         // Cierra la aplicación cuando se cierra la ventana
111         this.addWindowListener(new WindowAdapter()
112         {
113             public void windowClosing(WindowEvent event)
114             {
115                 quit();
116             }
117         });
118         buttonPanel.add(quit);
119 
120         // crea la etiqueta de explicación
121         TextArea explanationText = new TextArea();
122         explanationText.setEditable(false);
123         explanationText.setRows(8);
124         explanationText
125                 .setText("El objetivo de este juego es adivinar \n un
                    número aleatorio generado \npor el servidor.\nPara
                    comenzar el juego,\npresione el botón.\nSi desea salir,
                    pulse 'Salir'");
126         add(explanationText, BorderLayout.SOUTH);
127         add(buttonPanel, BorderLayout.NORTH);
128 
129         pack();
130     }
131 
132     /**
133      * Libera el objeto ICP cuando se cierra la aplicación y para la
            máquina virtual
134 
135      */
136     private void quit()
137     {
138         release(session);
139         release(service);
140         release(platform);
141         setVisible(false);
142         dispose();
143         System.exit(0);
144     }
145 
```

```
146     /**
147      * Libera un objeto ICP
148      *
149      * @param object El objeto a liberar
150      */
151     private void release(IBase object)
152     {
153         if(object != null)
154         {
155             try
156             {
157                 object.release();
158             }
159             catch (Exception e)
160             {
161 
162             }
163         }
164     }
165 
166     /**
167      * Finaliza el juego actual.
168      */
169     private void endGame()
170     {
171         try
172         {
173             session.end();
174             startGameButton.setEnabled(true);
175         }
176         catch (Exception e)
177         {
178             showError("Imposible terminar la sesión", e);
179         }
180     }
181 
182     /**
183      * Empieza un juego nuevo. esto creará una sesión ICP que se usará
         para intercambiar mensajes entre el cliente ICP y el servlet SIP
184      */
185     private void startGame()
186     {
187         try
188         {
189             // Crea la sesión y añade su listener sobre ella
190         session = service.createSession();
191             session.addListener(new SessionAdapter()
192             {
```

```
193                 public void processSessionStartFailed(ErrorReason aError,
                        long retryAfter)
194                 {
195                     GuessClient.this.showError("Could not start session",
                            new Exception(aError.getReasonString()));
196                 }
197
198                 public void processError(ErrorReason aError)
199                 {
200                     GuessClient.this.showError("Session Error", new
                            Exception(aError.getReasonString()));
201                 }
202
203                 public void processSessionMessage(String aContentType, byte
                        [] aMessage, int aLength)
204                 {
205                     // se recibió un MESSAGE, que se procesa...
206                     super.processSessionMessage(aContentType, aMessage,
                            aLength);
207                     String message = new String(aMessage);
208                     // envía el nuevo intento si no acertó
209                     if(aContentType.indexOf("result+good") == -1)
210                     {
211                         // Abre "Dialogo" para el número
212                         GuessNumberDialog dialog = new GuessNumberDialog(
                                GuessClient.this);
213                         dialog.setMessageLabel(message);
214                         dialog.setVisible(true);
215
216                         // Si no se escribe nada termina
217                         String input = dialog.getContent();
218                         if((input == null || input.trim().length() == 0))
219                         {
220                             endGame();
221                             return;
222                         }
223                         // Envía lo que escribió el usuario como parte de un
                                mensaje MESSAGE
224                         byte[] sendingContent = input.getBytes();
225                         try
226                         {
227                             session.sendMessage("guess/try", sendingContent,
                                    sendingContent.length);
228                         }
229                         catch (Exception e)
230                         {
231                             showError("No se pudo enviar el mensaje", e);
232
233                         }
```

```
234                     }
235                     else
236                     {
237                         // Número acertado, mostrar el "MensajeDialog" de
                            felicitiaciones
238                         final MessageDialog winDialog = new MessageDialog(
                            GuessClient.this, "Has ganado!",  "Acertaste el
                            número correcto!");
239                         winDialog.setVisible(true);
240                         startGameButton.setEnabled(true);
241                     }
242                 }
243             });
244 
245             // Empieza el juego
246             session.start("sip:adivina@juegosims.com", null, profile.
                getIdentity(), SdpFactory.createMIMEContainer());
247             startGameButton.setEnabled(false);
248         }
249         catch (Exception e)
250         {
251             showError("No se pudo crear la sesión", e);
252         }
253     }
254 
255     /**
256      * Muestra diálogo de error.
257      *
258      * @param message
259      * @param e
260      */
261     private void showError(String message, Exception e)
262     {
263         String exceptionMessage = e.getMessage();
264         if(exceptionMessage.length() == 0)
265         {
266             exceptionMessage = e.toString();
267         }
268         MessageDialog errorDialog = new MessageDialog(this, "ICP Error",
                message + "\r\n\r\n: " + exceptionMessage);
269         errorDialog.setVisible(true);
270     }
271 }
```

Listado C.2: El cliente de la aplicación.

C.3 Cuadros de diálogo

C.3.1 Diálogo principal: Dialogo.java

```
1  package com.ericsson.sds.samples.guess;
2
3  import java.awt.Button;
4  import java.awt.Dialog;
5  import java.awt.Dimension;
6  import java.awt.Frame;
7  import java.awt.GridLayout;
8  import java.awt.Label;
9  import java.awt.TextField;
10 import java.awt.event.ActionEvent;
11 import java.awt.event.ActionListener;
12 import java.awt.event.WindowAdapter;
13 import java.awt.event.WindowEvent;
14
15 /**
16  * Dialog class to ask the user for a number
17  */
18 class GuessNumberDialog extends Dialog
19 {
20     /**
21      *
22      */
23     private static final long serialVersionUID = -7522292705252076615L;
24     private Label messageLabel;
25     private String content;
26
27     public GuessNumberDialog(Frame owner)
28     {
29         super(owner, "¡Adivine un número!", true);
30         init();
31     }
32     /**
33      * Initialize the dialog GUI. This GUI is pretty simple,
34      * it has a label and a text field to enter the number.
35      *
36      */
37     private void init()
38     {
39         setLayout(new GridLayout(3, 1));
40         messageLabel = new Label();
41         final TextField textField = new TextField();
42         ActionListener closeActionListrener = new ActionListener()
43         {
44             public void actionPerformed(ActionEvent e)
45             {
```

```
46                content = textField.getText();
47                setVisible(false);
48            }
49        };
50        textField.addActionListener(closeActionListrener);
51        Button okButton = new Button("OK");
52        okButton.addActionListener(closeActionListrener);
53        add(messageLabel);
54        add(textField);
55        add(okButton);
56        this.setSize(new Dimension(250, 100));
57        this.addWindowListener(new WindowAdapter()
58        {
59            public void windowClosing(WindowEvent arg0)
60            {
61                GuessNumberDialog.this.setVisible(false);
62            }
63        });
64    }
65
66    /**
67     * Get the content typed by the user
68     */
69    public String getContent()
70    {
71        return content;
72    }
73
74    /**
75     * Set the message to display
76     */
77    public void setMessageLabel(String message)
78    {
79        messageLabel.setText(message);
80    }
81}
```

Listado C.3: El cuadro de diálogo principal.

C.3.2 DialogoMensajes.java

```
1 package com.ericsson.sds.samples.guess;
2
3 import java.awt.Button;
4 import java.awt.Dialog;
5 import java.awt.Frame;
6 import java.awt.GridBagConstraints;
7 import java.awt.GridBagLayout;
```

```
8  import java.awt.Insets;
9  import java.awt.Label;
10 import java.awt.event.ActionEvent;
11 import java.awt.event.ActionListener;
12 import java.awt.event.WindowAdapter;
13 import java.awt.event.WindowEvent;
14
15 class MessageDialog extends Dialog
16 {
17     /**
18      *
19      */
20     private static final long serialVersionUID = 3331590581131014830L;
21
22     /**
23      * Dialog displayed when the use has guessed correctly
24      */
25     public MessageDialog(Frame parent, String title, String message)
26     {
27         super(parent, title, true);
28         setLayout(new GridBagLayout());
29         Button ok = new Button("OK");
30         ok.addActionListener(new ActionListener()
31         {
32             public void actionPerformed(ActionEvent e)
33             {
34                 MessageDialog.this.setVisible(false);
35             }
36         });
37         Label messageLabel = new Label();
38         messageLabel.setText(message);
39         GridBagConstraints constrains = new GridBagConstraints();
40         constrains.gridx = 0;
41         constrains.gridy = 0;
42         constrains.weightx = 1;
43         constrains.insets = new Insets(5, 5, 5, 5);
44         constrains.fill = GridBagConstraints.HORIZONTAL;
45         add(messageLabel, constrains);
46         constrains = new GridBagConstraints();
47         constrains.gridx = 0;
48         constrains.gridy = 1;
49         constrains.anchor = GridBagConstraints.CENTER;
50         add(ok, constrains);
51         addWindowListener(new WindowAdapter()
52         {
53             public void windowClosing(WindowEvent e)
54             {
55                 MessageDialog.this.setVisible(false);
56             }
```

```
57          });
58          pack();
59      }
60
61  }
```

Listado C.4: El cuadro de diálogo de mensaje de éxito.

C.4 Adaptadores de la ICP

C.4.1 PlatformAdapter.java

```
1  package com.ericsson.sds.samples.guess;
2
3  import com.ericsson.icp.IPlatformListener;
4  import com.ericsson.icp.util.ErrorReason;
5
6  public class PlatformAdapter implements IPlatformListener
7  {
8      public void processPlatformTerminated(ErrorReason aReasonCode)
9      {
10     }
11
12     public void processError(ErrorReason aError)
13     {
14     }
15
16     public void processApplicationData(String aApplication, byte[] aData,
           int aLength)
17     {
18     }
19
20     public void processIncomingProfile(String aProfileName)
21     {
22     }
23 }
```

Listado C.5: Adaptador de la plataforma.

C.4.2 ProfileAdapter.java

```
1  package com.ericsson.sds.samples.guess;
2
3  import com.ericsson.icp.IProfileListener;
4  import com.ericsson.icp.IProfile.State;
5  import com.ericsson.icp.util.ErrorReason;
```

```
6
7  public class ProfileAdapter implements IProfileListener
8  {
9
10     public void processApplicationData(String aApplication, byte[] aData,
           int aLength)
11     {
12     }
13
14     public void processEvent(String aEvent, String aSource, ErrorReason
           aReasonCode)
15     {
16     }
17
18     public void processStateChanged(State aState)
19     {
20     }
21
22     public void processError(ErrorReason aError)
23     {
24     }
25 }
```

Listado C.6: Adaptador del perfil de la plataforma.

C.4.3 **ServiceAdapter.java**

```
1  package com.ericsson.sds.samples.guess;
2
3  import com.ericsson.icp.IServiceListener;
4  import com.ericsson.icp.ISession;
5  import com.ericsson.icp.util.ErrorReason;
6
7  public class ServiceAdapter implements IServiceListener
8  {
9
10     public void processIncomingSession(ISession aSession)
11     {
12     }
13
14     public void processMessage(String arg0, String arg1, byte[] arg2, int
           arg3, String arg4)
15     {
16     }
17
18     public void processMessage(String aRemote, String aMsgType, byte[]
           aMessage, int aLength)
19     {
```

```
20      }
21
22      public void processOptions(String aPreferedContact, String aRemote,
            String aType, byte[] aContent, int aLength)
23      {
24      }
25
26      public void processPublishResult(boolean aStatus, String aRemote,
            String aEvent, long retryAfter)
27      {
28      }
29
30      public void processRefer(String aReferID, String aRemote, String
            aThirdParty, String aContentType, byte[] aContent, int aLength)
31      {
32      }
33
34      public void processReferEnded(String aReferID)
35      {
36      }
37
38      public void processReferNotification(String aReferId, int aState)
39      {
40      }
41
42      public void processReferNotifyResult(boolean status, String aReferID,
            long retryAfter)
43      {
44      }
45
46      public void processReferResult(boolean aStatus, String aReferId, long
            retryAfter)
47      {
48      }
49
50      public void processSendMessageResult(boolean aStatus, long retryAfter)
51      {
52      }
53
54      public void processSendOptionsResult(boolean aStatus, String
            aPreferedContact, String aRemote, String aType, byte[] aContent,
            int aLength, long retryAfter)
55      {
56      }
57
58      public void processSubscribeEnded(String aPreferedContact, String
            aRemote, String aEvent)
59      {
60      }
```

```
61
62      public void processSubscribeNotification(String aRemote, String
            aContact, String aEvent, String aType, byte[] aContent, int aLength
            )
63      {
64      }
65
66      public void processSubscribeResult(boolean aStatus, String aRemote,
            String aEvent, long retryAfter)
67      {
68      }
69
70      public void processUnsubscribeResult(boolean aStatus, String aRemote,
            String aEvent, long retryAfter)
71      {
72      }
73
74      public void processError(ErrorReason aError)
75      {
76      }
77  }
```

Listado C.7: Adaptador de servicio.

C.4.4 SessionAdapter.java

```
1   package com.ericsson.sds.samples.guess;
2
3   import com.ericsson.icp.ISessionListener;
4   import com.ericsson.icp.util.ErrorReason;
5   import com.ericsson.icp.util.ISessionDescription;
6   import com.ericsson.icp.util.MIMEContainer;
7
8   public class SessionAdapter implements ISessionListener
9   {
10
11      public void processSessionAlerting()
12      {
13      }
14
15      public void processSessionCancelled()
16      {
17      }
18
19      public void processSessionEnded()
20      {
21      }
22
```

```
23      public void processSessionInformation(String arg0, byte[] arg1, int
            arg2)
24      {
25      }
26
27      public void processSessionInformationFailed(ErrorReason arg0, long
            arg1)
28      {
29      }
30
31      public void processSessionInformationSuccessful(String arg0, byte[]
            arg1, int arg2)
32      {
33      }
34
35      public void processSessionInvitation(String arg0, boolean arg1,
            ISessionDescription arg2, MIMEContainer arg3)
36      {
37      }
38
39      public void processSessionMediaNegotiation(ISessionDescription arg0)
40      {
41      }
42
43      public void processSessionMessage(String arg0, byte[] arg1, int arg2)
44      {
45      }
46
47      public void processSessionMessage(String arg0, byte[] arg1, int arg2,
            String arg3)
48      {
49      }
50
51      public void processSessionMessageFailed(ErrorReason arg0, long arg1)
52      {
53      }
54
55      public void processSessionMessageSuccessful(String arg0, byte[] arg1,
            int arg2)
56      {
57      }
58
59      public void processSessionOptions(String arg0, ISessionDescription
            arg1)
60      {
61      }
62
63      public void processSessionOptionsFailed(ErrorReason arg0, long arg1)
64      {
```

```
65      }
66
67      public void processSessionOptionsSuccessful(String arg0,
            ISessionDescription arg1)
68      {
69      }
70
71      public void processSessionPublishFailed(String arg0, byte[] arg1, int
            arg2, ErrorReason arg3, long arg4)
72      {
73      }
74
75      public void processSessionPublishSuccessful(String arg0, int arg1,
            byte[] arg2, int arg3)
76      {
77      }
78
79      public void processSessionReceivedPRACK(ISessionDescription arg0)
80      {
81      }
82
83      public void processSessionReceivedPRACKResponse(ISessionDescription
            arg0)
84      {
85      }
86
87      public void processSessionRefer(String arg0, String arg1, byte[] arg2,
             int arg3)
88      {
89      }
90
91      public void processSessionReferEnded(String arg0)
92      {
93      }
94
95      public void processSessionReferFailed(String arg0, ErrorReason arg1,
            long arg2)
96      {
97      }
98
99      public void processSessionReferNotify(String arg0, int arg1, String
            arg2)
100     {
101     }
102
103     public void processSessionReferNotifyFailed(String arg0, ErrorReason
            arg1, long arg2)
104     {
105     }
```

```
106
107     public void processSessionReferNotifySuccessful(String arg0)
108     {
109     }
110
111     public void processSessionReferSuccessful(String arg0)
112     {
113     }
114
115     public void processSessionStartFailed(ErrorReason arg0, long arg1)
116     {
117     }
118
119     public void processSessionStarted(ISessionDescription arg0)
120     {
121     }
122
123     public void processSessionSubscribeDeactived(String arg0, String arg1,
            byte[] arg2, int arg3)
124     {
125     }
126
127     public void processSessionSubscribeFailed(String arg0, String arg1,
            byte[] arg2, int arg3, ErrorReason arg4, long arg5)
128     {
129     }
130
131     public void processSessionSubscribeNotification(String arg0, String
            arg1, byte[] arg2, int arg3)
132     {
133     }
134
135     public void processSessionSubscribeSuccessful(String arg0, String arg1
            , byte[] arg2, int arg3)
136     {
137     }
138
139     public void processSessionUpdate(ISessionDescription arg0)
140     {
141     }
142
143     public void processSessionUpdateFailed(ErrorReason arg0, long arg1)
144     {
145     }
146
147     public void processSessionUpdateSuccessful(ISessionDescription arg0)
148     {
149     }
150
```

```
151     public void processError(ErrorReason arg0)
152     {
153     }
154 
155 }
```

Listado C.8: Adaptador de sesión.