PARADIGMAS 2 (1) 13-30 (2010)
Revista del Departamento Académico de Ingeniería.
Universidad Ricardo Palma. ISSN: 2220-2056

Programación por Capas

Francisco Aguilar Vásquez[1]

Resumen

Definiciones básicas de programación, de capas y de programación por capas. Patrón arquitectónico "capas" de descomposición de sistemas complejos para su comprensión en diferentes perspectivas. Modelamiento de capas basadas en responsabilidades: capa de presentación, capa de aplicación y capa de datos.

Palabras claves

Programación, capa, programación por capas, modelo, patrón, proceso, clase, objeto, UML, lenguaje de programación.

Abstract

Basic definitions of programming, layers/tiers and layered/tiered programming. Architectural layers pattern of decomposition of complex systems for their understanding from different perspectives. Modeling layers based on responsibilities: presentation layer, application layer and data layer.

Key words

Programming, layer, tier, layered/tiered programming, model, pattern, process, class, object, UML, programming language

Introducción

El término programación se refiere al proceso de construcción de programas (software) para ejecutarse en el computador, en el que se destacan los siguientes pasos: edición de programas fuentes en un lenguaje de programación, compilación de los programas fuentes al lenguaje del computador y enlazado de los programas compilados con facilidades de ejecución para convertirlos en programas ejecutables. Este proceso se lleva acabo en ambientes de construcción de programas que disponen de editores, compiladores, enlazadores y otras facilidades, disponibles por medio de interfaces, particularmente visuales.

El término capa tienes varios significados entre los que se encuentran: aquello que cubre o protege algo, prenda de vestir, estrato superpuesto a otro u otros con los que forma un todo. La piel humana, un fragmento de la cual se presenta en la figura 1, es un ejemplo de aquello que cubre y protege algo - los órganos del cuerpo. Las capas que constituyen la tierra, esquemáticamente presentados en la figura 2, son ejemplos de estratos que forman un todo. En la figura 3 se muestra un estructura de madera compuesta por capas, cada una de las cuales, también, puede apreciarse como estrato. En el idioma Inglés, el termino capa tiene múltiples traducciones, siendo layer y tier las mas pertinentes

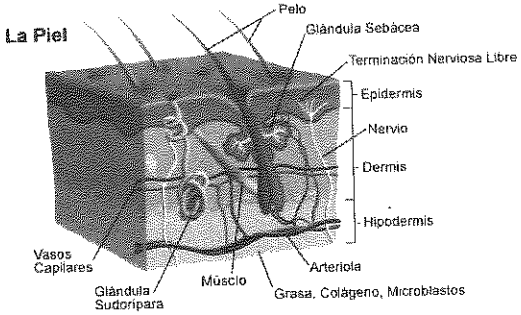

Figura 1. Capas de la piel.

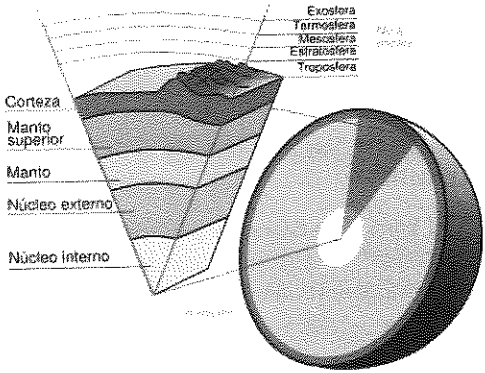

Figura 2. Capas de la tierra.

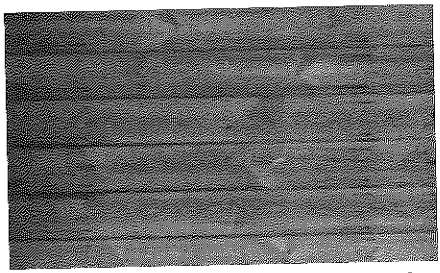

Figura 3. Capas de una estructura de madera.

Metafóricamente, el término capa ha ingresado en el ámbito de computación para referirse a una de las múltiples técnicas que permite tratar, a nivel general (arquitectónico), la complejidad de los sistemas hardware/software desde diferentes perspectivas, denominada patrón "capas". Un patrón representa una solución a un problema común que existe en un contexto particular. El esquema del patrón "capas" es el siguiente:

Contexto: Un sistema que requiere descomposición.
Problema: Un sistema complejo, difícil de mantener, con elementos inestables no delimitados, con elementos reusables difícil de identificar, a ser construido por diferentes equipos.
Solución: Estructurar/organizar el sistema en capas.

La organización estructurada del computador desde la perspectiva de programación y el modelo de red OSI-7 capas de ISO, presentados en las figuras 4 y 5, respectivamente, son ejemplos muy conocidos de la aplicación del modelo arquitectónico "capas". En el primer caso cada nivel representa una máquina (el nivel 0 – máquina real y los otros niveles – máquinas virtuales) con su propio lenguaje de programación que se traduce en el lenguaje de la máquina bajo ella. El segundo caso define un conjunto de protocolos de red, donde cada capa se enfoca en un aspecto específico de

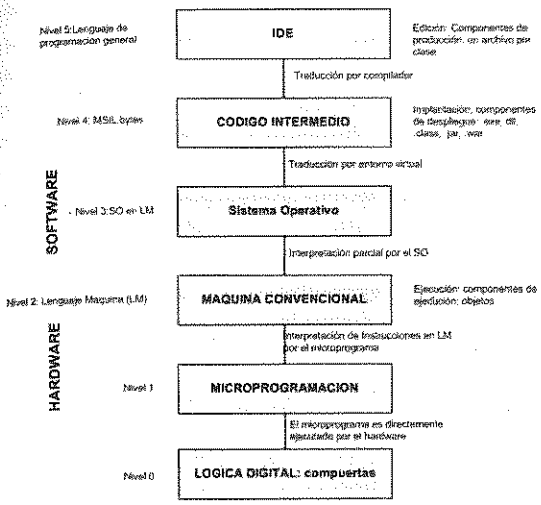

Figura 4. Capas del computador desde la perspectiva de programación.

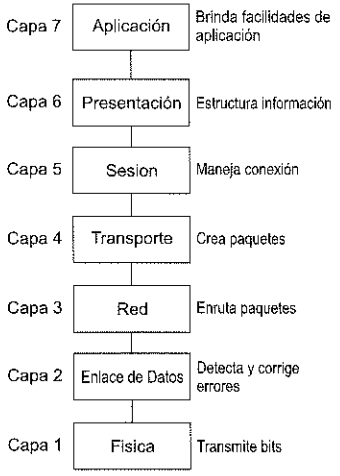

Figura 5. Capas del modelo OSI de ISO.

Las estrategias para organizar los sistemas en capas consideran diferentes características tales como responsabilidad, nivel reuso, seguridad, pertenencia, conjunto de habilidades. Cada estrategia se expresa mejor por medio de modelos específicos del sistema. Un modelo representa un sistema completo en una perspectiva. La figura 6 muestra cinco modelos de un sistema en igual número de perspectivas:

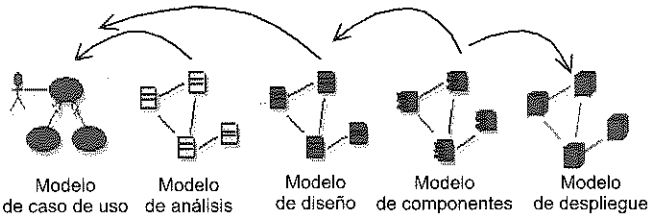

Figura 6. Modelos de un sistema en diferentes perspectivas.

El modelo de casos de uso captura los requerimientos del sistema. El modelo de análisis expresa la realización conceptual de los requerimientos. El modelo de diseño muestra el diseño del sistema. El modelo de componentes presenta la implementación del sistema. El modelo de despliegue muestra su distribución en la infraestructura computacional - computadores. Entre otros modelos, el modelo de datos captura los aspectos de persistencia del sistema.

La programación por capas involucra técnicas efectivas de programación en la implementación del modelo de componentes, coherentes con el patrón arquitectónico en capas del modelo de diseño, así como con el diseño detallado de las capas. Las capas tienen un significado esencialmente lógico (**layer**) y su distribución puede resultar en sistemas de un proceso o de múltiples procesos (aplicaciones distribuidas). Los procesos múltiples pueden ejecutarse en un computador o en

múltiples computadores. No existe acuerdo en el uso del término inglés **tier**, unos suelen usarlo para identificar capas lógicas basadas en responsabilidades, independientemente de su distribución en procesos; otros lo usan para identificar distribuciones físicas de los sistemas distribuidos en procesos. Estos últimos suelen usar los nombres two-tier, three-tier y n-tier para identificar los sistemas cliente – servidor, los sistemas distribuidos en tres capas físicas (presentación, negocio y datos) y los sistemas distribuidos en n-capas físicas (presentación, negocio distribuido y datos), respectivamente. En este artículo optaremos por la primera alternativa, explicitando la distribución en procesos y computadores.

Los siguientes párrafos desarrollan la estrategia basada en responsabilidades para estructurar en capas (layers o tiers) una sencilla aplicación "RegPrimos" que realiza la siguiente tarea "recibe del usuario un número entero y si es primo lo registra en un archivo". Se empieza mostrando un modelamiento parcial del diseño en capas y luego se desarrolla la programación en capas. La programación se desarrolla en dos versiones: programación en tres capas lógicas ubicadas en una aplicación monolítica (un proceso) y programación en tres capa lógicas distribuidas en tres procesos, los cuales pueden ejecutarse hasta en tres computadores diferentes. Para fines de modelamiento se emplea el UML y la programación se realiza en C# del ambiente ms vs.net 2005 y posteriores.

Modelamiento de RegPrimos en capas basadas en responsabilidades.

Tal como se muestra en el modelo de la figura 7, la estructuración en capas basada en responsabilidades de un sistema considera las siguientes capas:

- **Lógica de presentación** que contiene los elementos para brindar alguna forma de presentación: interfaz de RegPrimos con el usuario.
- **Lógica de negocio** que contiene elementos para ejecutar algún procesamiento del negocio y aplicar sus reglas: verificación si el número es primo.
- **Lógica de datos** que contiene los elementos que brindan acceso a las fuentes de datos: registro del numero primo, sin duplicados, en archivo.

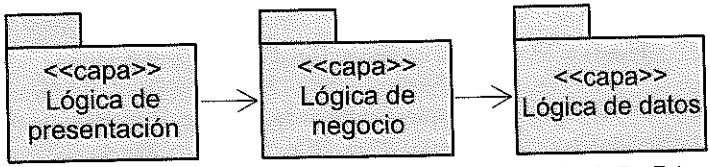

Figura 7. Estructuración en capas basada en responsabilidades de RegPrimos.

Dado que la capa se puede modelar de varias formas, aquí lo representamos por un paquete UML con el estereotipo «capa».

La dirección de las dependencias en la figura 7 significa que un elemento de una capa particular solamente puede acceder a los elementos de la misma capa, y/o de la capa de su derecha. Los elementos de la capa lógica del negocio no pueden acceder a los elementos de la capa lógica de

Esta estrategia de estructuración basada en responsabilidades mejora el desarrollo y mantenimiento de un sistema ya que sus diferentes responsabilidades están delimitadas unas de otras, brindando a cada capa alta cohesión y reduciendo el acoplamiento entre las capas. La aplicación de esta estrategia puede influenciar el modelo de diseño, el modelo de componentes y el modelo de despliegue.

El modelo de diseño puede ser organizado usando varios enfoques, uno de ello muestra los elementos contenidos en cada capa tal como se muestra en las figuras 8 y 9 para RegPrimos. Las figuras muestran el contenido de cada una de las tres capa lógicas en términos de clases diseño: NumeroPrimoIGU, NumeroPrimo y NumeroPrimoReg.

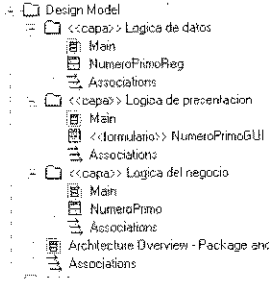

Figura 8. RegPrimos: modelo de diseño.

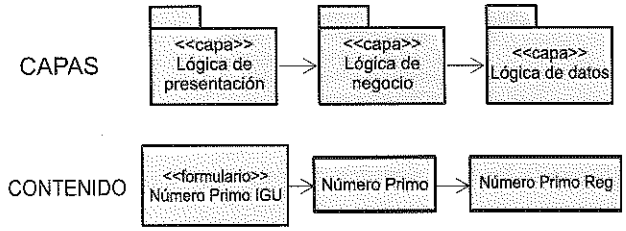

Figura 9. Capas lógicas y su contenido de RegPrimos.

NumeroPrimoIGU es la clase que administra un formulario mostrado por la figura 10 que permite, ingresar un número, iniciar el proceso que se realiza con la ayuda de NumeroPrimo y ver un mensaje coherente de respuesta.

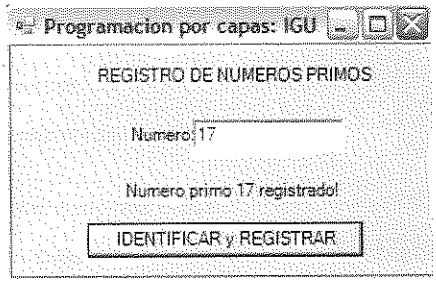

Figura 10. Interfaz visual de RegPrimos.

NumeroPrimoIGU es la clase que administra un formulario mostrado por la figura 10 que permite, ingresar un número, iniciar el proceso que se realiza con la ayuda de NumeroPrimo y ver un mensaje coherente de respuesta.

NúmeroPrimo es la clase que verifica si el número facilitado por NumeroPrimoIGU es primo. Si el número es primo es transferido a NumeroPrimoReg.

NúmeroPrimoReg crea un archivo para almacenar los números primos y registra el número si es que no está registrado.

Programación de RegPrimos en tres capas lógicas ubicadas en un proceso.

La estructura del modelo de implementación de la figura 11 muestra los componentes contenidos en los espacios de nombres correspondientes a las capas de diseño. Además en esta figura se muestra el componente ProcesoAplicación que inicia el proceso de la aplicación. En la figura 12 se muestran las relaciones de traducción (compilación) entre los componentes de producción en C# y el componente desplegable, que es un componente ejecutable en un proceso (en el único hilo del proceso).

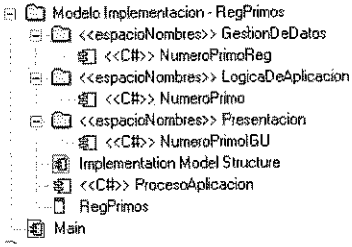

Figura 11. RegPrimos-modelo de componentes.

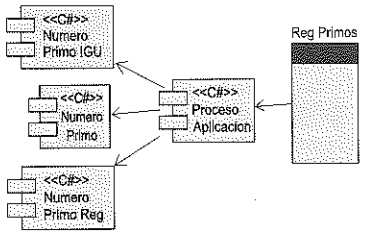

Figura 12. RegPrimos-diagrama de componentes

A continuación se presenta el código fuente en C# de todos los componentes de producción con los respectivos comentarios:

Clase ProcesoAplicación, que contiene el método Main (linea 10) que crea el proceso en el cual se ubican las capas lógicas y constituye el punto de entrada al hilo de este proceso. En este método se crea un objeto de la clase NumeroPrimoGUI (linea 12) que implementa la capa de presentación – formulario interfaz entre el usuario y la aplicación.

```
1       using System;
2       using System.Collections.Generic;
3       using System.Text;
4       using System.Windows.Forms;
5       using Presentacion;

6       namespace RegPrimos
7       {
8         class ProcesoAplicacion
9         {
10          static void Main()
11          {
12            Application.Run(new NumeroPrimoIGU());
13          }
14        }
15      }
```

Clase NumeroPrimoIGU, que implementa el formulario – capa de presentación - interfaz entre el usuario y la aplicación. El método InitializeComponent implementado entre las líneas 57 y 130, es invocado en la línea 14 para dibujar el formulario y el método procesador de eventos, entre las líneas 16 y 30, depura el número ingresado por medio de un esquema de excepciones y usa la capa de Lógica de la aplicación, en las líneas 23 y 24, para continuar con el proceso.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using LogicaDeAplicacion;

namespace Presentacion
{
  public partial class NumeroPrimoIGU : Form
  {
    public NumeroPrimoIGU().
    {
      InitializeComponent();
    }

    private void btnRegistrar_Click(object sender, EventArgs e)
    {
      int numero;
      try
      {
        numero = int.Parse(txtNumero.Text);

        if (numero < 1) throw new InvalidEnumArgumentException();

        NumeroPrimo objPrimo = new NumeroPrimo(numero);
        lblMensaje.Text = objPrimo.AdministrarPrimo();
      }
      catch
      {
        lblMensaje.Text = "Ingrese un numero entero positivo!";
      }
    }
  }
)
namespace Presentacion
{
  partial class NumeroPrimoIGU
  {
    /// <summary>
    /// Required designer variable.
    /// </summary>
    private System.ComponentModel.IContainer components = null;

    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    protected override void Dispose(bool disposing)
    {
      if (disposing && (components != null))
      {
        components.Dispose();
      }
      base.Dispose(disposing);
    }

    #region Windows Form Designer generated code
```

```
53          /// <summary>
54          /// Required method for Designer support - do not modify
55          /// the contents of this method with the code editor.
56          /// </summary>
57          private void InitializeComponent()
58          {
59            this.label1 = new System.Windows.Forms.Label();
60            this.label2 = new System.Windows.Forms.Label();
61            this.lblMensaje = new System.Windows.Forms.Label();
62            this.txtNumero = new System.Windows.Forms.TextBox();
63            this.btnRegistrar = new System.Windows.Forms.Button();
64            this.SuspendLayout();
65            //
66            // label1
67            //
68            this.label1.Location = new System.Drawing.Point(47, 9);
69            this.label1.Name = "label1";
70            this.label1.Size = new System.Drawing.Size(201, 21);
71            this.label1.TabIndex = 0;
72            this.label1.Text = "REGISTRO DE NUMEROS PRIMOS";
73            this.label1.TextAlign = Drawing.ContentAlignment.MiddleCenter;
74            //
75            // label2
76            //
77            this.label2.Location = new System.Drawing.Point(67, 48);
78            this.label2.Name = "label2";
79            this.label2.Size = new System.Drawing.Size(56, 21);
80            this.label2.TabIndex = 1;
81            this.label2.Text = "Numero: ";
82            this.label2.TextAlign = Drawing.ContentAlignment.MiddleRight;
83            //
84            // lblMensaje
85            //
86            this.lblMensaje.Location = new System.Drawing.Point(3, 85);
87            this.lblMensaje.Name = "lblMensaje";
88            this.lblMensaje.Size = new System.Drawing.Size(285, 21);
89            this.lblMensaje.TabIndex = 2;
90            this.lblMensaje.TextAlign = ContentAlignment.MiddleCenter;
91            //
92            // txtNumero
93            //
94            this.txtNumero.Location = new System.Drawing.Point(120, 49);
95            this.txtNumero.Name = "txtNumero";
96            this.txtNumero.Size = new System.Drawing.Size(100, 20);
97            this.txtNumero.TabIndex = 3;
98            //
99            // btnRegistrar
100         //
101          this.btnRegistrar.Location = new System.Drawing.Point(50, 117);
102           this.btnRegistrar.Name = "btnRegistrar";
103           this.btnRegistrar.Size = new System.Drawing.Size(182, 23);
104           this.btnRegistrar.TabIndex = 4;
105           this.btnRegistrar.Text = "IDENTIFICAR y REGISTRAR";
106           this.btnRegistrar.UseVisualStyleBackColor = true;
107   btnRegistrar.Click += new EventHandler(this.btnRegistrar_Click);
108           //
109           // IGU
110           //
```

```
112             this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
113             this.ClientSize = new System.Drawing.Size(292, 152);
114             this.Controls.Add(this.btnRegistrar);
115             this.Controls.Add(this.txtNumero);
116             this.Controls.Add(this.lblMensaje);
117             this.Controls.Add(this.label2);
118             this.Controls.Add(this.label1);
119             this.Name = "IGU";
120             this.Text = "Programacion por capas: IGU";
121             this.ResumeLayout(false);
122             this.PerformLayout();

123         }

124         #endregion

125         private System.Windows.Forms.Label label1;
126         private System.Windows.Forms.Label label2;
127         private System.Windows.Forms.Label lblMensaje;
128         private System.Windows.Forms.TextBox txtNumero;
129         private System.Windows.Forms.Button btnRegistrar;
130     }
131 }
```

Clase NumeroPrimo, implementa la lógica de la aplicación. Esta clase tiene un constructor, líneas 10 a 13, que inicializa la variable número; el método esPrimo, líneas 14 a 27, y el método AdministrarPrimo, líneas 28 a 38. Este último método, al ser invocado por la capa de presentación, verifica si el número es primo con la ayuda del método esPrimo(). Si el número es primo se invoca a la capa de datos, en caso contrario devuelve el respectivo mensaje.


```
1   using System;
2   using System.Collections.Generic;
3   using System.Text;
4   using GestionDeDatos;

5   namespace LogicaDeAplicacion //capa
6   {
7     class NumeroPrimo
8     {
9       int numero;

10      public NumeroPrimo(int num)
11      {
12        numero = num;
13      }

14      private bool esPrimo()
15      {
16        bool esPrimo = true;

17        if (numero > 3)
18          for (int n = 2; n <= numero / 2; n++)
19          {
20            if (numero % n == 0)
21            {
22              esPrimo = false;
```

```
23                  break;
24                }
25              }
26            return esPrimo;
27          }

28          public string AdministrarPrimo()
29          {
30            if (esPrimo())
31            {
32              NumeroPrimoReg objPrimoReg = new NumeroPrimoReg(numero);
33              return objPrimoReg.RegistrarPrimo();
34            }
35            else
36              return "Numero " + numero + " NO es primo!";
37          }
38        }
39      }
```

Clase numeroPrimoReg, implementa la capa de datos. Dispone de tres métodos: un constructor en las líneas 11 a 16, buscarPrimo en las líneas 17 a 32 y RegistrarPrimo en las líneas 33 a 42. El constructor inicializa número y crea el archivo "primos.txt" si es que no existe. El método buscar primo verifica que el número primo no exista en el archivo "primos.txt". Si el número primo no existe en el archivo, el método RegistrarPrimo adiciona el número al archivo "primos.txt"


```
1       using System;
2       using System.Collections.Generic;
3       using System.Text;
4       using System.IO;

5       namespace GestionDeDatos
6       {
7         class NumeroPrimoReg
8         {
9           int numero;
10          string archivo = "Primos.txt";

11          public NumeroPrimoReg(int num)
12          {
13            numero = num;
14            FileStream fs = new FileStream(archivo, FileMode.OpenOrCreate);
15            fs.Close();
16          }

17          private bool buscarPrimo()
18          {
19            StreamReader sr = new StreamReader(archivo);

20            bool existeUnPrimo = false;
21            string s;
22            while ((s = sr.ReadLine()) != null)
23            {
24              if (numero == int.Parse(s))
25              {
26                existeUnPrimo = true;
27                break;
```

```
29            }
30            sr.Close();
31            return existeUnPrimo;
32          }

33          public string RegistrarPrimo()
34          {
35            if (buscarPrimo())
36              return "Numero primo " + numero + " duplicado!";

37            StreamWriter sw = new StreamWriter(archivo, true);
38            sw.WriteLine(numero.ToString());
39            sw.Close();

40            return "Numero primo " + numero + " registrado!";
41          }
42        }
43      }
```

Programación de RegPrimos en tres capas distribuidas en tres procesos.

Esta versión de RegPrimos consta de tres capas lógicas, cada una hospedada en un proceso independiente. Esta vez los tres procesos se ejecutan en el computador local; pero fácilmente pueden distribuirse hasta en tres computadores diferentes. Los procesos anfitriones de las capas lógicas se comunican empleando el protocolo TCP.

En la figura 13 se muestra las interfaces de los tres procesos al registrar el número primo 5. La parte superior corresponde al formulario de la capa de presentación ejecutada en una aplicación visual (Windows). La parte intermedia es la consola del proceso que hospeda la capa lógica de aplicación. La parte interior corresponde a la consola del proceso que hospeda a la capa de datos. El proceso Windows se comunica con el proceso consola que realiza la lógica de la aplicación por medio del puerto 14000. Este último proceso se comunica con el proceso que realiza la capa de datos por medio del puerto 15000. Los tres procesos implementan tres aplicaciones integradas en una aplicación distribuida RegPrimos, cada una de las cuales hospeda una capa lógica.

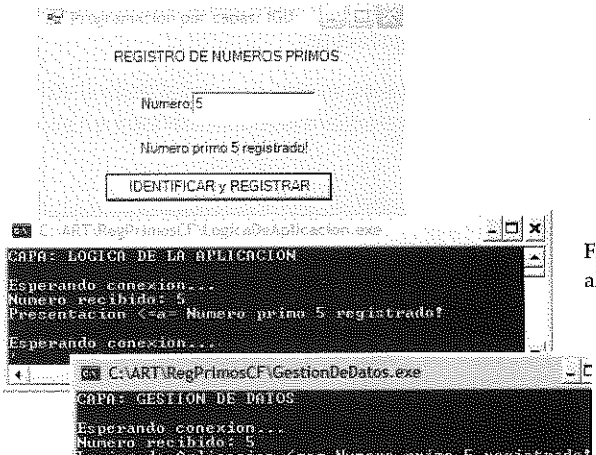

Figura 13. Interfaces de los tres procesos al registrar el número primo 5.

La figura 14 muestra el modelo de implementación de la aplicación visual que realiza la capa de presentación y la figura 15 presenta la relación de construcción entre los componentes de producción y desplegable de esta aplicación.

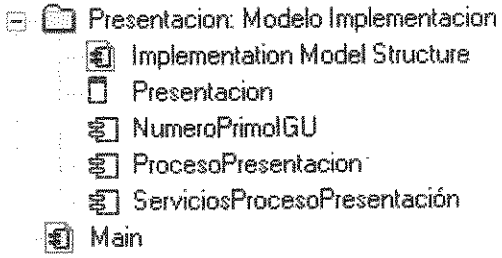

Figura 14. Presentación-modelo de componentes.

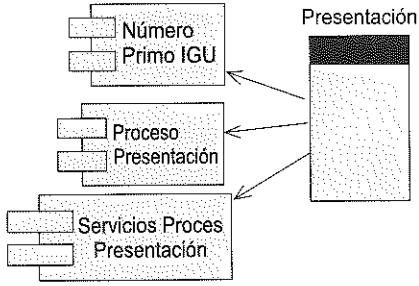

Figura 15. Presentación-diagrama de componentes

La clase ProcesoPresentación coincide con la clase ProcesoAplicacion de la versión monoproceso de RegPrimos.

La clase NumeroPrimoIGU se diferencia de la clase del mismo nombre de RegPrimos monolítica en las líneas 5 y 6 del siguiente segmento de código:


```
1       try
2       {
3         numero = int.Parse(txtNumero.Text);

4         if (numero < 1) throw new InvalidEnumArgumentException();

5       ServiciosProcesoPresentacion objServicio = new
ServiciosProcesoPresentacion();
6         lblMensaje.Text = objServicio.InvocarLogAp(numero);
7       }
```

La clase ServiciosProcesoPresentacion, implementa la comunicación con el proceso que hospeda la capa lógica del negocio y se define por el siguiente código fuente:

```
1       class ServiciosProcesoPresentacion
2       {
3         public string InvocarLogAp(int numero)
4         {
5           byte[] bDatos;
6           //cliente de Logica de Aplicacion
7           TcpClient cliente = new TcpClient();
8           UnicodeEncoding codificador = new UnicodeEncoding();

9           try
10          {
11            //conectarse al puerto 14000 de localhost
12            cliente.Connect("localhost", 14000);

13            //objeto flujo de red para comunicarse con Logica de Aplicacion
```

```
15            bDatos = codificador.GetBytes(numero.ToString());

16            flujo.Write(bDatos, 0, bDatos.Length);//enviar numero

17            //obtener respuesta de Logica de Aplicacion
18            byte[] bbDatos = new byte[1024];
19            int nBytes = flujo.Read(bbDatos, 0, bbDatos.Length);
20            if (nBytes > 0)
21              return codificador.GetString(bbDatos, 0, nBytes);
22            else
23              return "Respuesta de Logica de Aplicacion fallo";
24          }
25          catch (SocketException ex)
26          {
27            return ex.Message;
28          }
29          finally
30          {
31            cliente.Close();
32          }
33        }
34      }
```

De forma análoga la aplicación de lógica de la aplicación se implementa de acuerdo al modelo de la figura 16 y el diagrama de la figura 17.

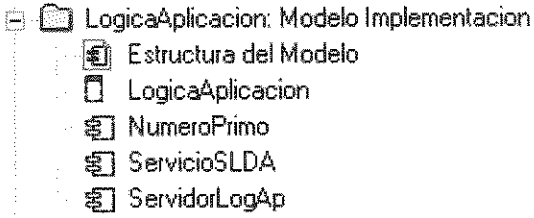

Figura 16. Lógica Aplicación-modelo de componentes.

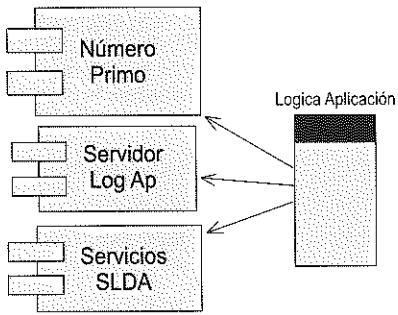

Figura 17. Lógica Aplicación-diagrama de componentes

El código de la clase NúmeroPrimo es parecido a la versión anterior. El código fuente de las clases ServidorLogAp y ServicioSLDA es el siguiente:

```
1       using System;
2       using System.Collections.Generic;
3       using System.Text;
4       using System.Net;
5       using System.Net.Sockets;

6       namespace LogicaDeAplicacion
7       {
8         class ServidorLogAp
9         {
```

```
10          static void Main(string[] args)
11          {
12            Console.WriteLine("CAPA: LOGICA DE LA APLICACION");
13            TcpListener servidorLogAp = null;
14            try
15            {
16              // Escuchar en el puerto 14000 de localhost.
17      servidorLogAp = new TcpListener(IPAddress.Parse("127.0.0.1"), 14000);
18              servidorLogAp.Start();

19              // Conversor de datos
20              UnicodeEncoding codificador = new UnicodeEncoding();

21              while (true)//bucle de esucha
22              {
23                Console.WriteLine("\nEsperando conexion...");

24                // Llamada sincrona a aceptar pedido de conexion
25                TcpClient conexion = servidorLogAp.AcceptTcpClient();
26                byte[] bDatos = new byte[128];
27                string sDatos = null;

28                //objete flujo de red para comunicacion con Presentacion
29                NetworkStream flujo = conexion.GetStream();

30                //recibir numero de Presentacion (en bytes).
31                int nBytes = flujo.Read(bDatos, 0, bDatos.Length);
32                if (nBytes > 0)
33                {
34                  // Convertir bytes a string.
35                  sDatos = codificador.GetString(bDatos, 0, nBytes);
36                  Console.WriteLine("Numero recibido: {0}", sDatos);

37                  int numero = int.Parse(sDatos);//numero primo como entero

38                  //usar capa de Logica de Aplicacion
39                  NumeroPrimo objPrimo = new NumeroPrimo(numero);
40                  sDatos = objPrimo.AdministrarPrimo();

41                  //devolver mensaje a Presentacion
42                  bDatos = codificador.GetBytes(sDatos);
43                  flujo.Write(bDatos, 0, bDatos.Length);
44                  Console.WriteLine("Presentacion <=a= {0}", sDatos);
45                }
46                conexion.Close(); //cerrar conexion
47              }
48            }
49            catch (SocketException ex)
50            {
51              Console.WriteLine("SocketException: {0}", ex);
52            }
53            finally
54            {
55              servidorLogAp.Stop(); //parar ServidorLogAp
56            }
57          }
58        }

59        class ServicioSLDA
```

```
60      {
61        public string InvocarGestionDatos(int numero)
62        {
63          byte[] bDatos;
64          //cliente de Gestion de Datos
65          TcpClient cliente = new TcpClient();
66          UnicodeEncoding codificador = new UnicodeEncoding();
67          try
68          {
69            //conectarse al puerto 15000 de localhost
70            cliente.Connect("localhost", 15000);
71            //objeto flujo de red para comunicarse con Gestion de Datos
72            NetworkStream flujo = cliente.GetStream();
73            bDatos = codificador.GetBytes(numero.ToString());
74            flujo.Write(bDatos, 0, bDatos.Length);//enviar numero
75            //obtener respuesta de Gestion de Datos
76            byte[] bbDatos = new byte[1024];
77            int nBytes = flujo.Read(bbDatos, 0, bbDatos.Length);
78            if (nBytes <= 0)
79              return "Error devuelto por Gestion de Datos!";
80            else
81              return codificador.GetString(bbDatos, 0, nBytes);
82          }
83          catch (SocketException ex)
84          {
85            Console.WriteLine("SocketException: {0}", ex);
86            return ex.Message;
87          }
88          finally
89          {
90            cliente.Close();
91          }
92        }
93      }
94    }
```


La organización de la aplicación de Gestión de Datos se presenta en el modelo de la figura 18 y en el diagrama de la figura 19.

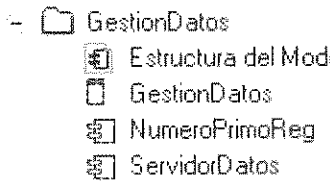

Figura 18. Gestión Datos-modelo de componentes.

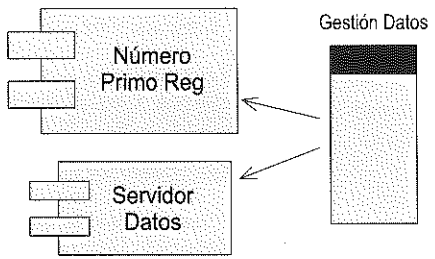

Figura 19. Gestión Datos-diagrama de componentes

El código de la clase NúmeroPrimoReg es el mismo de la versión anterior. El código fuente de la clase ServidorDatos es el siguiente:

```
1       using System;
2       using System.Collections.Generic;
3       using System.Text;
4       using System.Net;
5       using System.Net.Sockets;

6       namespace GestionDeDatos
7       {
8         public class ServidorDatos
9         {
10          static void Main(string[] args)
11          {
12            Console.WriteLine("CAPA: GESTION DE DATOS");
13            TcpListener servidorDatos = null;
14            try
15            {
16              // Escuchar en el puerto 15000 de localhost.
17      servidorDatos = new TcpListener(IPAddress.Parse("127.0.0.1"), 15000);
18              servidorDatos.Start();

19              // conversor de datos
20              UnicodeEncoding codificador = new UnicodeEncoding();

21              while (true) // Bucle de escucha
22              {
23                Console.WriteLine("\nEsperando conexion...");

24                //aceptar pedido de conexion
25                TcpClient conexion = servidorDatos.AcceptTcpClient();

26                // Bufferes para datos
27                byte[] bDatos = new byte[128];
28                string sDatos = null;

29               // objeto flujo de red para comunicacion Logica de Aplicacion
30                NetworkStream flujo = conexion.GetStream();

31                //recibir numero primo de Logica de Aplicacion (en bytes)
32                int nBytes = flujo.Read(bDatos, 0, bDatos.Length);
33                if (nBytes > 0)
34                {
35                  // Convertir bytes a string.
36                  sDatos = codificador.GetString(bDatos, 0, nBytes);
37                  Console.WriteLine("Numero recibido: {0}", sDatos);

38                  int numero = int.Parse(sDatos);//numero primo como entero

39                  //usar capa de Gestion de Datos
40                  NumeroPrimoReg objPrimoReg = new NumeroPrimoReg(numero);
41                  sDatos = objPrimoReg.RegistrarPrimo();

42                  //devolver mensaje a Logica de Aplicacion
```

```
43                  bDatos = codificador.GetBytes(sDatos);
44                  flujo.Write(bDatos, 0, bDatos.Length);
45                 Console.WriteLine("Logica de Aplicacion <=a= {0}", sDatos);
46                }
47                conexion.Close();//cerrar conexion
48              }
49            }
50            catch (SocketException e)
51            {
52              Console.WriteLine("SocketException: {0}", e);
53            }
54            finally
55            {
56              servidorDatos.Stop(); //Parar ServidorDatos
57            }
58          }
59        }
60      }
```

BIBLIOGRAFIA.

DEITEL H. M. Y DEITEL P.J. C# how to program, Prentice Hall, USA, 2002.

DEITEL H. M. Y DEITEL P.J. Como programar en JAVA, Prentice Hall, Mexico, 2004.

DUTHIE G. A, MICROSOFT ASP.NET programming with ms Visual C#.NET step by step, Microsoft Press, USA, 2003.

Eeles P. Racional Software White Paper, 2000

LARMAN C., APPLYING UML AND PATTERNS, Third Edition, Prentice Hall, USA, 2005
Microsoft, MSDN, Librería del IDE ms vs.2008, USA, 2008.

ROBINSON SIMON ET ALL, "Professional C#", Wrox Press, USA, 2001.

RUMBAUGH J., JACOBSON I., BOOCH G., El lenguaje unificado de modelado. Manual de referencia, 2000.

DIRECCIONES ELECTRÓNICAS:

Knowledge sharing communities: http://it.toolbox.com/blogs/high-thinking/layers-or-tiers-11718

David Hayden: http://davidhayden.com/blog/dave/archive/2005/07/22/2401.aspx