Conciencia Tecnológica
ISSN: 1405-5597
contec@mail.ita.mx
Instituto Tecnológico de Aguascalientes
México

Luna Rosas, Fco. Javier; Martínez Romo, Julio César; López Villalobos, José de Jesús
Shell para el procesamiento distribuido de imágenes (un enfoque heterogéneo)
Conciencia Tecnológica, núm. 18, diciembre, 2001, pp. 32-39
Instituto Tecnológico de Aguascalientes
Aguascalientes, México

Shell para el Procesamiento Distribuido de Imágenes (Un Enfoque Heterogéneo).

Fco. Javier Luna Rosas
fjluna@verona.fi-p.unam.mx

Julio Cesar Martínez Romo
jcmartin@verona.fi-p.unam.mx

José de Jesús López Villalobos
jjlopez@verona.fi-p.unam.mx

División de Estudios de Posgrado Facultad de Ingeniería (DEPFI-UNAM), Cd. Universitaria, Apdo. Postal 70-256, C.P. 04510, México, D.F. Tels. (525) 5622-30-12 y (49) 9723515, Fax(525) 5616-10-73.
Instituto Tecnológico de Aguascalientes, Av. A. López Mateos 1801 Ote. esq. Av. Tecnológico, Fracc. Ojocaliente, CP. 20256, Aguascalientes, Ags.

Resumen

El campo del Tratamiento Digital de Imágenes está en continua evolución, González y Woods dicen que durante los últimos 5 años ha aumentado significativamente el interés en la morfología de las imágenes, las redes neuronales, el procesamiento de imágenes en color, la compresión y el reconocimiento de Imágenes[7]. En la propuesta de este reporte se analiza el impacto potencial de la Tecnología de Objetos Distribuidos (DOT) en el Tratamiento Digital de Imágenes, construimos un Shell que permite el procesamiento distribuido de Imágenes. El Shell tiene como Back-End objetos distribuidos CORBA y como Front-End una interfaz gráfica de usuario construida en Java. La partición y almacenamiento de objetos se desarrolló apoyándose en El Proceso de Desarrollo de Software del Modelo Unificado [2] y el Lenguaje de Modelado Visual UML [1], además el almacenamiento de objetos se hace pensando en minimizar las siguientes métricas de costo: Costo Total de la Comunicación Inter-Objetos (ICO), la Ejecución Total y la ICO (E+ICO), Tiempo Global (CT), Balanceo de Carga (LB), Optimización del Canal y la Optimización del Tiempo de Respuesta de los Objetos.

1. – Introducción.

La tecnología de objetos distribuidos (DOT) es definida, considerando que incluyen 3 tecnologías, que han surgido con sinergia para tener una herramienta poderosa, más que la suma de las tres partes [2]. Estas tres tecnologías, son las siguientes:

- Tecnología de objetos.
- Tecnología distribuida.
- Tecnología Web.

La tecnología de objetos (OT). En el modelo orientado a objetos, los sistemas son vistos como objetos cooperativos que encapsulan estructura y comportamiento que corresponden a clases que están construidas jerárquicamente. En los últimos 20 años los beneficios de la tecnología de objetos han sido demostrados. La orientación a objetos ha cambiado la forma en que son construidos y mantenidos los sistemas actuales, teniendo experiencias substanciales en el Análisis Orientado a Objetos (*OOA*), Diseño Orientado a Objetos (*OOD*) y la Programación Orientada a Objetos (*OOP*). La transición de los enfoques estructurados hacia el enfoque OO no ha sido completa, aún no se ha explotado totalmente, especialmente en la legacía (legacy) de sistemas. Pero la tecnología de objetos está madurando rápidamente y es bien aceptada como tecnología que reduce la complejidad, mejora el mantenimiento, promete rehuso y reduce los costos de software durante el ciclo de vida. *La tecnología distribuida (DT),* envuelve computadoras autónomas que son conectadas por una red y no comparten memoria física. Tradicionalmente la DT emplea el modelo Cliente-Servidor, donde las computaciones sobre una máquina (Cliente) invocan computaciones en otra máquina (Servidor) de tal forma que a menudo es vista como LLamada de Procedimiento Remoto (RPC). Los desarrollos más recientes en tecnología distribuida han envuelto el uso de la tecnología Middleware. Middleware esta ampliamente definida por la OMG en [18], para las tecnologías que facilitan la comunicación de componentes (objetos) en sistemas distribuidos. *La tecnología Web (WT).* La tecnología Web ha globalizado y universalizado la computación como nunca antes. El uso de tecnología Web para construir grandes sistemas de aplicación o rediseñar los que ya existen, llega en 1995 con el lenguaje de programación orientado a objetos Java, que condujo a los "Applets" Java proporcionando un contenido ejecutable en la *Tecnología Web* estática.

2.- Integrando CORBA y Java.

Java IDL es una tecnología de objetos distribuidos, esto es, objetos que interactúan sobre diferentes plataformas a través de una red. Java IDL es similar a RMI (Remote Method Invocation), que soportan

objetos distribuidos escritos enteramente en el lenguaje de programación Java, Java IDL activa objetos que interactuan entre ellos, independientemente si están escritos en Java o en algún otro lenguaje de programación tal como C, C++, Ada, COBOL y otros [5] ver Figura 1. Esto es posible ya que Java IDL es basado sobre Common Object Request Broker Architecture (CORBA), como comentamos previamente es la industria estándar para modelar objetos distribuidos. Una característica clave de CORBA es IDL, que es un lenguaje de definición de interfaces neutral. Cada lenguaje que soporta CORBA tiene su propio mapeo IDL a el lenguaje mismo. Java IDL proporciona un Object Request Broker, u ORB. El ORB es una librería de clase que activa la comunicación de bajo nivel entre las aplicaciones Java IDL y otras aplicaciones de CORBA (Ver Figura 1).

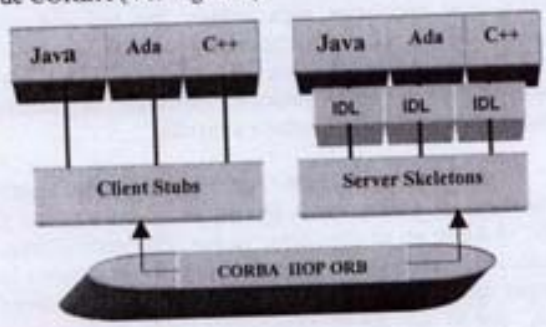

Figura 1. Interoperabilidad del IDL.

3.- El Desarrollo de la Arquitectura del Shell.

La arquitectura del Shell para el procesamiento distribuido de Imágenes, esta Diseñado de la siguiente manera:

3.1 Partición de Objetos. La partición de objetos se hace pensando en minimizar los siguientes tres objetivos:

- Minizar la comunicación Inter-objetos.
- Explotar la concurrencia.
- Limitar el tamaño de los objetos.

Para llevar a cabo la partición de objetos nos apoyamos en el Proceso de Desarrollo de Software del Modelo Unificado [1] y en el Lenguaje de Modelado Visual UML [2, 8] con el propósito de adecuarlo a un Método de Análisis y Diseño Orientado a Objetos (ADOO). La Fig 2 muestra el Filtro en Paso Bajas, Medio y Paso Altas, además del Filtrado Morfológico y Espacial, que son clases que componen la clase Filtrado como parte de la estructura estática (diagrama de clases) del Shell para el procesamiento de imágenes distribuidas.

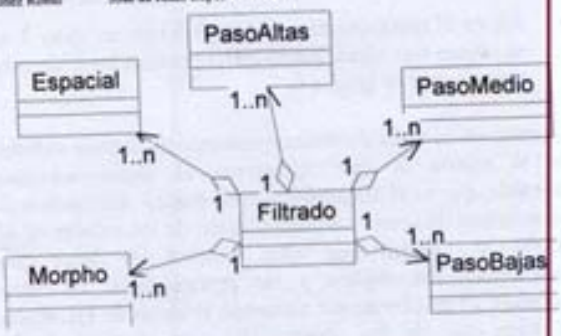

Figura 2. Filtrado del Shell.

3.2 Almacenamiento de Objetos. Para un sistema de computación homogéneo el tiempo de ejecución de un objeto, puede ser visto independientemente del procesador en que se ejecute, pero en un sistema de computación heterogéneo, el tiempo de ejecución de los objetos, puede ser visto en función del host. Los objetos pueden ser asignados antes o durante la ejecución de un sistema, si el almacenamiento se hace antes de la ejecución, por ejemplo en tiempo de diseño, es llamado almacenamiento estático, si los objetos son almacenados durante la ejecución, el almacenamiento es llamado almacenamiento dinámico, nosotros utilizaremos el almacenamiento estático para almacenar nuestros objetos. El almacenamiento lo realizamos pensando en minimizar las siguientes métricas de costos.

- Costo total de la comunicación Inter-objetos (IOC)
- Ejecución total y la ICO (E + IOC)
- Tiempo Global (CT)
- Balanceo de carga(LB)

Una de las primeras métricas para la computación distribuida , fue el costo de comunicación. Rápidamente se conoció que la principal desventaja para el almacenamiento de objetos, en diferentes procesadores fué la sobrecarga introducida por la IOC. Por lo tanto fue totalmente natural querer usar esto como base para juzgar el desempeño de un almacenamiento. El *modelo algebraico* es una técnica de solución óptima que utilizamos para resolver el problema del *almacenamiento de objetos*, por lo tanto el costo de almacenamiento para la métrica E + IOC puede ser expresado como sigue[17].

$$\text{Costo} = \sum_{k}\sum_{i} (e_{ik}x_{ik} + \sum_{i<k}\sum_{j<i} c_{ij}x_{ik}x_{jl})$$

- La Cij es el costo de comunicación inter-objeto entre el objeto i y el objeto j.
- Cik debe ser el costo de ejecución del objeto i si es almacenado en el procesador K.

- Xij es el almacenamiento deseable con un valor 1 si el objeto i es almacenado en el procesador j, de otra manera este es igual a 0.

La meta de la tarea de almacenamiento es derivar valores de la matriz X que representa el almacenamiento deseable, que es el almacenamiento seguro que minimiza las métricas de costo. La computación de los valores de la matriz X tienen que estar basados en datos que caractericen los objetos y los procesadores. El costo global (CT) lo obtenemos sumando el costo de ejecución de cada uno de los objetos (E) más el costo de la comunicación Inter.-objetos (IOC). La Figura 3 muestra cuando obtenemos la métrica CT en objetos concurrentes, determinamos los costos E + IOC por separado, luego usamos el que tiene el valor mayor.

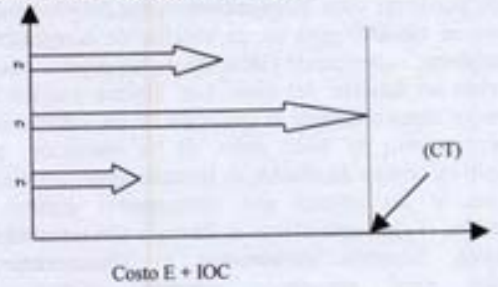

Figura 3. Costo Total en Objetos Concurrentes.

Para realizar el balanceo de carga utilizamos el método Efe [6]. Escogimos este método porque es fácil de entender. El método Efe adopta una estrategia general, primero forma categorías de módulos, basadas en minimizar los costos IOC y entonces ajustan las categorías basadas en las restricciones de balanceo de carga.

3.3 Optimizando la Respuesta de Objetos. En Java un píxel, es almacenado en un entero de 32-bits, el entero consiste en 4 bytes empacados, esos bytes representan los planos: Alpha, Red, Green y Blue (ARGB), como se muestra en la Figura 4. El Píxel empacado mostrado previamente, es una técnica de almacenamiento disponible para un píxel de 32-bits en Java, sin embargo hay problemas con este enfoque, ya que la codificación y decodificación, requieren operaciones Shift y enmascaramiento, incrementando el tiempo computacional.

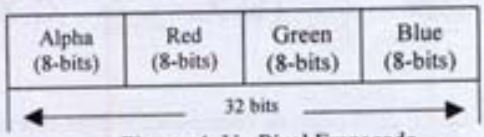

Figura. 4 Un Píxel Empacado.

Por ejemplo, suponemos que tenemos un píxel a color empacado, como el mostrado en la Figura 4, y queremos convertir este a un valor monocromático promediando los tres colores. Un método que acopla esta tarea es como sigue.

```
Public int filterRGB (int rgb) {
        int red      = (rgb & 0xff0000) >>16;
        int green    = (rgb &  0xff00) >>8;
        int blue     = (rgb & 0xff);
        int gray = (red + green + blue) / 3;
        return ( 0xff000000     |
                 (gray << 16)   |
                 (gray <<8 )    | gray);
}
```

El método filterRGB des-empaca el píxel, ejecutando la operación de filtrado, y entonces re-empaca el píxel. Tres operaciones AND, tres operaciones OR y 4 operaciones SHIFT, son necesarias para accesar los tres colores. Considerando que las Imágenes tienen cientos de bytes o algunas tienen Mega-bytes. El enfoque de des-empacamiento y empacamiento, provoca demasiado tiempo de ejecución en el acceso a pixeles.

Para proporcionar un mejor rendimiento separamos los pixeles de la imagen en el Red, Green y Blue, el Alpha no se considero, ya que representa el tinte, saturación y brillo de la composición RGB, separamos la imagen aprovechando el tipo de datos escalar que tiene Java, por lo que des-empaquetamos todos los enteros y los empacamos dentro de Bytes.

```
r = new short[width][height];
g = new short[width][height];
b = new short[width][height];
```

Tomando en cuenta que Java permite mandar una imagen que tiene en memoria a un arreglo de la forma siguiente:

```
int pels[] = new int[width * height];
cm = ColorModel.getRGBdefault();
PixelGrabber grabber =
        new PixelGrabber(
        image, 0, 0,
        width, height, pels, 0, width);
        try {grabber.grabPixels();}
        catch (InterruptedException e){};
        int i = 0;
        for (int x = 0; x < width; x++)
                for (int y = 0; y < height; y++) {
        i = x + y * width;
        b[x][y]=(short)cm.getBlue(pels[i]);
        g[x][y]= (short)cm.getGreen(pels[i]);
        r[x][y]= (short)cm.getRed(pels[i]);
        }
}
```

Separando los planos RGB, podemos nuevamente implementar el color monocromático en un método llamado grises():

```
public void grises() {
for (int x=0; x < width; x++)
  for (int y=0; y < height; y++) {
        r[x][y] = (short)
        ((r[x][y] + g[x][y]  + b[x][y]) / 3);
        g[x][y] = r[x][y];
        b[x][y] = r[x][y];
        }
}
```

Como observamos en el código anterior basta con sumar el contenido de los tres planos RGB y después hacer una asignación, para convertir nuevamente la imagen, Java proporciona un método que permite crear una imagen de un arreglo, entonces debemos re-empaquetar nuevamente los bytes en enteros de la forma siguiente:

```
int pels[] = new int[width*height];
 for (int x = 0; x < width; x++)
        for (int y = 0; y < height; y++) {
         pels[ x + y * width] =
         0xff000000
         | (r[x][y] << 16)
         | (g[x][y] << 8)
         |  b[x][y];
        }
  Image i =tk.createImage(new MemoryImageSource
        (width, height, cm, pels, 0, width));
```

En el código anterior se utilizan tres OR, para volver a re-empaquetar el pixel en un entero, de esta manera podemos observar, que desaparecieron 4 operaciones SHIFT y tres operaciones AND, en total serian 7 operaciones, si consideramos que tenemos, imágenes de varios cientos de bytes, o a veces, Mega-bytes, el ahorro computacional seria muy rentable.

3.4 Optimizando el Canal de Comunicación. Según Couloris, el rango de transferencia de datos, es la velocidad en la cual la información puede ser transferida entre dos computadoras en red [4]. Para optimizar el ancho de banda, nuestro SHELL, no envía toda la imagen por el canal de comunicación, lo máximo que envía son tres cuartos de la imagen del emisor al receptor, es decir de nuestro Objeto Stub, a nuestro objeto de Implementación (Objeto Skeleton). Cuando nuestro Objeto de Implementación regresa la respuesta a nuestro Objeto Stub, de la misma forma regresa un máximo de tres cuartos de la imagen. Por ejemplo consideremos nuestra forma de hacer negativa una imagen (Figura5).

Figura 5. Negativo de una Imagen.

F(Vij) = 255 – Vij.
Donde:
Vij = Valor del pixel en la posición i,j. Tal que Vij E [0..255].

Basta con enviar 3 cuartos de la imagen para extraer el negativo del RGB, más aún, podemos separar el RGB, y enviar un cuarto de imagen, es decir enviaríamos, un cuarto de imagen que equivale al color Rojo "Red", otro cuarto para el color Verde "Green" y el último cuarto para el color Azul "Blue", indudablemente al enviar un cuarto de imagen del Objeto Stub al Objeto Implementación equivaldría a enviar uno de los tres colores, en el regreso, se obtendría un cuarto de la imagen en un Objeto Procesado, es decir el negativo del objeto. Como se puede observar, podemos aprovechar al máximo el canal de comunicación, reduciendo la imagen a su mínima expresión y evitando de esta manera la saturación del canal de comunicación.

3.5 Criterios de Desempeño para Migrar Objetos a Plataformas Rápidas. Douglas A. Lyon define la convolución como la operación que se realiza entre dos imágenes, y una imagen es típicamente más pequeña que la otra. La imagen más pequeña es llamada el kernel de la convolución[11]. La convolución consiste en la suma de los pesos de una área, alrededor de un píxel de entrada, dicho de otra manera la convolución es la correlación entre la imagen y el kernel invertido. En el dominio discreto podemos calcular la convolución como:

$$h(x,y) = f * g = \sum_{U=0}^{Umax-1} \sum_{V=0}^{Vmax-1} f(u,v)g([x-u],[y-v])$$

Si aplicamos la convolución a la segunda derivada parcial de una señal (también conocido como operador Laplaciano [7]), para la detección de bordes, con una conectividad de 4 pixeles (N4(p)), tendríamos:

```
public void laplacian() {
    float k[][] = {
            { 0, -1,  0},
            {-1,  4, -1},
            { 0, -1,  0}
            };
}
```

$f(x,y) = 4f[x][y] - f[x-1][y] - f[x+1][y] - f[x][y-1] - f[x][y+1]$.

Si utilizamos el Laplacian y el Prewitt con una conectividad de 8 pixeles (N8(p)), tendríamos:

```
public void laplacianPrewitt() {
float k[][] = {
            { -1, -1, -1},
            { -1,  8, -1},
            { -1, -1, -1}
            };
```

$f(x,y) = 8f[x][y] - f[x-1][y] - f[x+1][y] - f[x][y-1] - f[x][y+1] - f[x-1][y-1] - f[x-1][y+1] - f[x+1][y-1] - f[x+1][y+1]$.

Claramente el kernel (N4(p)), ocupa menos tiempo de ejecución que el kernel (N8(p)), por lo que podemos determinar que el objeto que contiene el kernel con conectividad de 8 pixeles es más pesado, basado en este criterio podemos elegir un procesador con menos carga de trabajo, o bien un procesador mas rápido, que nos permita acelerar el desempeño del objeto.

4.- Análisis, Diseño y Construcción del Shell para el Procesamiento Distribuido de Imágenes.

En esta sección se hace el análisis, diseño y construcción del Shell. El desarrollo se apoya en el Proceso de Desarrollo de Software del Modelo Unificado [BRJ99A], en el Lenguaje de Modelado Visual UML [2, 8], en el Lenguaje de Programación Orientado a Objetos JAVA [12, 9] y en la Tecnología Middleware CORBA que permite la comunicación de objetos en sistemas distribuidos [14, 19].

La Figura 6 muestra la interfaz grafica (Front-End) de los clientes que actúan en el sistema distribuido del Shell. Cada Cliente es una versión *Multihilo*, las capacidades *Multihilo* permiten al Cliente recibir *Callbacks* en un hilo separado. Un hilo invoca un método remoto sobre uno o mas Servidores de Objetos remotos, mientras otro espera registrar un nuevo Cliente en el *Coordinador(Servidor)*.

La interfaz del Shell puede realizar el pre-procesamiento que se requiera para cualquier imagen. Este pre-procesamiento puede realizarlo el mismo Cliente, o bien lo realiza otro *Applet*, que sirve como control para realizar pre-procesamiento en todas las imágenes que se encuentran distribuidas en la red. Por ejemplo si quisiéramos sacar el Histograma de cada una de las imágenes, el Applet que sirve como control de las imágenes envía un mensaje al servidor *Coordinador* de imágenes indicándole que se quiere sacar el histograma de cada una de las imágenes que tenemos registradas, el *Coordinador*, le indica a cada uno de los Clientes registrados mediante un objeto Callback que se calcule el Histograma, entonces los Clientes invocan a uno o varios Objetos encargados de calcular el Histograma, los objetos se encuentran distribuidos en la red o bien se encuentran almacenados en uno o varios Servidores de Objetos (Back-End).

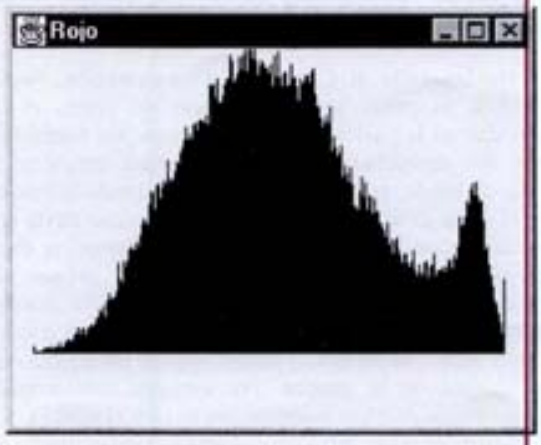

Figura 6. Histograma del Mandril (Plano Rojo).

La Figura 6 muestra el Histograma del mandril, este es calculado solamente para el plano rojo(red), también calculamos el histograma para los demás planos , pero por razones de espacio no pusimos los otros dos. González y Woods [7] dicen que el histograma de una imagen es una medida estática y es también conocido como "Probability Mass Function (PMF) de la imagen. El PMF es calculado sumando el número total de pixeles de un valor particular y entonces dividirlo por el número total de pixeles de la imagen.

$$PMF=p(V=i)=Pv(i)=\frac{1}{WH}\sum_{x=0}^{W=1}\sum_{y=0}^{H=1}\begin{cases}1 \text{ if } Vxy=i\\ 0 \text{ if } V_{xy}<>i\end{cases}$$

Donde:

Vxy = Valor del píxel en la localización x, y.

Con:

W = Ancho de la imagen en pixeles.
H = Alto de la imagen en pixeles.
p(V=i)= probabilidad en que ocurra un valor particular.

4.1 Análisis del Shell. Según Mellor el propósito del análisis es proporcionar una descripción del problema [3]. La descripción debe ser concreta, consistente, legible y revisable, por las diversas partes interesadas y ser comparada con la realidad. En nuestros términos es proporcionar un modelo de la forma en que se comporta el sistema. Al enfocarnos sobre tal comportamiento identificamos los puntos clave que representan alguna actividad primaria en el sistema o con frecuencia denotan el mapeo de entradas y salidas que representan las transformaciones que el sistema hace en su ambiente.

Para efectuar el análisis, Booch [3], establece que el análisis de requerimientos, y el análisis del dominio del sistema, son dos pasos que deben ser liberados durante esta etapa. Por una parte el análisis de requerimientos del sistema debe proporcionar la escritura básica de las funciones que el sistema deberá realizar y por su parte el análisis del dominio del sistema proporciona las estructuras lógicas que son claves para el sistema. En nuestro ejemplo especificamos el análisis de requerimientos utilizando un modelo de casos de uso, los casos de uso nos representan la funcionalidad que debe tener el sistema [1]. La Figura 7 muestra un caso de uso para registrar los clientes que intervienen en el Shell para el procesamiento distribuido de imágenes.

Por otra parte el análisis del dominio lo realizamos utilizando diagramas de estructura estática. Un diagrama de estructura estática muestra el conjunto de clases y objetos que son parte de un sistema, junto con sus relaciones existentes entre estas clases y objetos [2].

Figura 7. Registro de Clientes.

4.2 Diseño del Shell. El propósito del diseño es crear una arquitectura para implementar el comportamiento que requiere el modelo del análisis [3]. Para tal propósito creamos nuestros escenarios, un escenario es usado para describir como los casos de uso se realizan mediante interacciones entre sociedades de objetos , para capturar los escenarios utilizamos dos tipos de diagramas de interacción: diagramas de secuencia que describen las interacciones de los objetos a través del tiempo y los diagramas colaborativos que muestran la interacción de objetos organizada a través de objetos y sus relaciones. La Figura 8 muestra el diagrama de secuencia para controlar y ejecutar los clientes en el Shell:

1.- Crea el hilo ClientControlThread. El constructor de MultiClient crea un nuevo objeto ClientControlThread y corre el hilo en alta prioridad.
2.- Crea un hilo ClienteEjecuciónThread. El constructor MultiClient también crea un nuevo objeto ClienteEjecuciónThread y lo corre en prioridad normal.
3.- Registrar el Callback con el Coordinador. El objeto ClienteControlThread invoca el método register() en el Coordinador y pasa la referencia a la interfaz.
4.- El Coordinador le dice a todos que comience la ejecución. El Coordinador relaciona los Clientes que tiene registrados invocando el método ClientControl-Thread.grises() via Callback.
5.- Pasar el comando al hilo principal. El ClientControlThread invoca el método iniciaGrises() sobre el objeto MultiClient.
6.- Activar el ClienteEjecuciónThread. El hilo principal invoca el método grises() sobre el objeto ClienteEjecuciónThread para activar el hilo. Si el hilo a sido previamente activado, entonces se invoca el método suspend(). La clase ClientCountThread hereda los métodos start(), suspend() y resume() de la clase padre Thread.
7.- Invocar el método remoto grises(). La actividad del objeto ClientEjecuciónThread es invocar el método grises() sobre un Servidor de objetos remoto

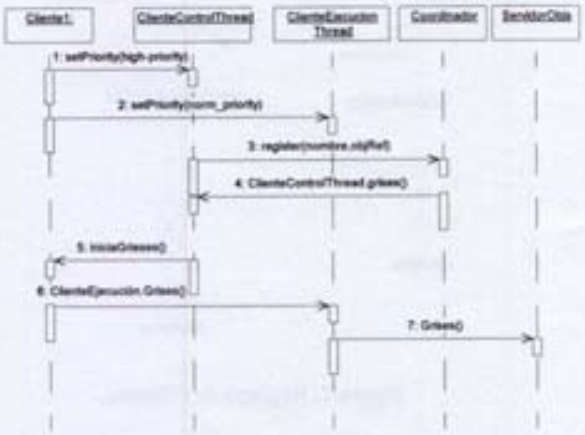

Figura 8. Control de Clientes.

4.3 Interfaces IDL. La interface IDL define el contrato entre el cliente y el servidor (ORBs) en el Shell, especificando que operaciones y atributos estarán disponibles. El IDL OMG es un lenguaje de declaración independiente, este deberá ser mapeado al lenguaje antes de que se implemente el código, enseguida mencionamos solo algunas interfaces que utiliza el Shell para coordinar y comunicar los clientes.

```
        // Coordinador.idl

module Client
  { interface ClientControl
        {
                boolean grises();
                boolean negativo();
                boolean darken();
                boolean histogram();
                boolean thresh();
                boolean binariza();
                boolean average();
                boolean skeleton();
        };
       };

module MultiCoordinador
  { interface Coordinador
        {
         boolean register(in string clientNumber,
         in Client::ClientControl clientObjRef);
        boolean grises();
        boolean negativo();
        boolean darken();
        boolean histogram();
        boolean thresh();
        boolean binariza();
        boolean average();
        boolean skeleton();
  };
};
```

5.- Conclusión.

El uso combinado de *Tecnologías Orientadas a Objetos, Tecnologías Distribuidas y la Tecnología Web*(OT, DT y WT) cambia la forma en que los sistemas son diseñados. Los objetos han sido agregados a las redes e integrados con Middleware, a menudo llamados *Object Request Broker (ORB's).* Los objetos han sido usados en sistemas distribuidos para representar unidades de distribución, movimiento y comunicación. La adición de WT proporciona portabilidad en la aplicación. Existe en efecto un nuevo paradigma de computación, en el cual inter-operan los objetos este nuevo paradigma es la *Tecnología de Objetos Distribuida (DOT).* La DOT nos da la oportunidad de movernos rápido y hacia el futuro en la distribución y globalización de las aplicaciones. En nuestro Shell la DOT nos da la oportunidad de mantener *portabilidad, interoperabilidad y transparencia* entre plataformas, además de permitir el procesamiento distribuido en el tratamiento digital de imágenes.

Bibliografía.

[1] Jacobson, Booch y Rumbaugh. El Lenguaje de Modelado Unificado. Addsin Wesley 1999.

[2] Jacobson, Booch y Rumbaugh. The Unified Software Development Process. Addsin Wesley 1999.

[3]Booch, Grady. Object oriented analysis and design with applications. Benjamin Cummings, (1994).

[4] Coulouris, Dollimore, Kindberg. Distributed Systems Concepts and Design, Second Edition. Addison-wesley (1996).

[5] Curtis, David. Java, RMI and CORBA A White Paper by Platform Technology Object Management Group, Available WWW. http: //www.omg.org/library/wpjava.html.

[6] Efe. K. Heuristic Models of Task Assignment Scheduling in Distributed Systems. IEEE Computer, June, pp. 50-56.

[7] Tratamiento digital de imágenes. Addison-Wesley 1996.

[8] Quatrani, Terry. Visual Modeling with Rational Rose and UML. Addison-Wesley 1998.

[9] Lea, Doug. Concurrent Programming in Java, Design Principles and Patterns. Addison-Wesley 1997.

[10] Lindhold, Yellin. 1996. The Java Virtual Machine. The java series, Addison-Wesley.

[11] Lyon A. Dougias. Image Processing in JAVA. Prentice-Hall 1999.
[12] Jaworski Jamie. Java 1.2 Al Descubierto. Primera Edición Prentice Hall, SAMS.

[13] Maravall Darío. Reconocimiento de Formas y Visión Artificial. Addisson Wesley 1994.

[14] Orfali Robert, Harkey Dan. Client/Server Programming with JAVA and CORBA. Second Edition, Wiley Computer Publishing.

[15] Welcome to OMG´s Home Page [online]. Available WWW. http : // www.omg.org(1997).

[16] Rumbaugh, James. *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991.

[17] Sol M. Shatz. Development of Distributed Software Concepts and Tools. University of Illinois at Chicago. Macmillan Publishing company, NY. 1993.

[18] The Common Object Request Broker: Architecture and Specification. Revision 2.0, SunSoft,Inc, July 1995.

[19] Vogel andreas, Duddy Keith. JAVA Programming with CORBA. Second Edition, Wiley Computer Publishing.

[20] Wallnau, Kurt. Distributed Object Technology With CORBA and Java: Key Concepts and Implications [online]. Available WWW, http://www.sei.cmu.edu/publications/documents/97tr004title.