Sistemas Informáticos
Curso 2004 / 2005

Sistema de servicios Web/Grid para el análisis de diferencias entre imágenes

Realizado por:
Yair Kukielka
Alberto Sánchez
Lorena García
Dirigido por:
Prof. José Jaime Ruz Ortiz
Dpto. Arquitectura de Computadores y Automática

Facultad de Informática
Universidad Complutense de Madrid

INDICE

1. **Autorización a la UCM.**

Los autores de este proyecto: Yair Kukielka, Alberto Sánchez y Lorena García, autorizamos a la Universidad Complutense de Madrid a utilizar y difundir, con fines académicos (no comerciales), la presente memoria, la documentación y el prototipo desarrollado.

Fdo Yair Kukielka Fdo Lorena García Fdo Alberto Sánchez

2. **Resumen**

En este proyecto se ha diseñado e implementado un sistema distribuido que detecta las diferencias entre dos imágenes mediante el agoritmo de enfriamiento simulado. El programa aprovecha la capacidad de cálculo de procesadores remotos en la red de manera paralela para analizar las imágenes con mayor rapidez.

Está previsto que las imágenes a tratar provengan del mismo objetivo focal, y de tiempos diferentes. Por ejemplo, un satélite que está programado para realizar fotos de un área geográfica a la misma hora cada día. De esta manera se pueden localizar barcos enemigos, plagas o incendios en zonas de cultivo.

La aplicación está desarrollada en un entorno de ventanas. El usuario elige las dos imágenes a diferenciar y obtiene una tercera imagen en blanco y negro, donde la parte blanca muestra las diferencias y la negra muestra la zona en la que no hay cambios.

3. **Abstract**

In this project an image difference detection distributed system has been designed and implemented. The program uses the parallel computing capacity of remote processors within a network in order to speed up the image analyses.

It is foreseen that the images will come from the same focal objective, and at different times. For example, a satellite that is programmed to take pictures of a geographical area at the same time daily. In this way, enemy ships, plagues or fires could be detected.

The application has been developed in a window environment where the user chooses two images to be processed and obtains a third black and white image, where the white parts show the areas with differences and the black sections display the zones without changes.

4. **Lista de palabras clave**

- Programación distribuida
- Imágenes
- Servicios web
- Visual Studio 2003
- .NET
- C#
- Redes Grid

5. **Objetivos**

El desarrollo del proyecto tiene, como objetivo principal, **optimizar el tiempo de respuesta** del sistema al realizar la comparativa de dos imágenes de grandes dimensiones utilizando un algoritmo de comparación píxel a píxel complejo.

Partimos de la base de que las imágenes objeto del estudio serán de grandes dimensiones y de que el cálculo que realizará la aplicación para la detección de diferencias podría suponer gran cantidad de tiempo, podríamos estar hablando incluso de horas de procesamiento. Debido a esto se decide utilizar la idea de programación paralela distribuyendo el volumen de procesamiento entre varias máquinas conectadas a la red, con lo que se consigue un tiempo de respuesta menor al que se obtendría si la imagen fuera procesada desde una sola máquina.

En este proceso también se integra la idea de la plataforma .NET de utilizar una red de servicios. Esto es, hasta no hace mucho, se venían utilizado aplicaciones que se instalaban en un ordenador y que sólo utilizaban la potencia del procesador de dicho ordenador. Con la implantación de Internet a nivel mundial y el aumento de la velocidad de interconexión entre las computadoras se puede repartir la carga de procesamiento en aplicaciones con un enorme volumen de operaciones (servicios web). Este reparto se puede hacer en ordenadores que se estén desaprovechando en ese momento. Se puede utilizar la potencia de los ordenadores que permanecen ociosos la mayor parte del tiempo y así poder sacar provecho de su capacidad. La gran mayoría del tiempo los ordenadores pueden permanecer apagados, inactivos o infrautilizando la CPU, por ejemplo al editar textos con Word (Redes Grid).

Serían muchas las personas que se verían beneficiadas: aquellos ordenadores que pasen gran parte del tiempo sin ser usados podrían ser alquilados o prestados a otras personas o a empresas que necesiten, en momentos puntuales, una capacidad de cálculo mayor de la que disponen, sin necesidad de gastar innecesariamente grandes sumas de dinero en supercomputadoras. Bastaría con instalar esos servicios en las máquinas alquiladas y enviar solicitudes a estos servidores eventuales por medio de la red.

6. Tecnologías utilizadas

Las tecnologías que se han utilizado en el desarrollo del proyecto son:

- .NET
- C#
- Servicios Web
- Visual Studio 2003

6.1. .NET

Desde la introducción de Windows 3.1 en 1992 hasta Windows XP Microsoft ha usado como núcleo la misma API (Interfaz de Programación de Aplicaciones) de Windows. En la medida en que se iba progresando en nuevas versiones de su famoso sistema operativo Microsoft iba añadiendo grandes cantidades de información con nuevas funcionalidades a ese API. Esta estrategia de extensión de la API (en vez de sustitución) ha sido una de las razones de su éxito, cuidando la compatibilidad descendente (hacia versiones anteriores).

Este camino evolutivo tiene también una gran desventaja: al ir añadiendo características nuevas sobre las antiguas, el uso de éstas se va haciendo cada vez más complejo y menos comprensible.

En el año 2000 Microsoft hizo pública su estrategia, en la que se consideraba que es imprescindible “partir de un folio en blanco”, para adaptarse a las nuevas tecnologías y el software de última generación (Java, de Sun Microsystems, le estaba comiendo terreno) por medio de un entorno y de unos lenguajes y herramientas de desarrollo intuitivos y a la vez sofisticados.

La nueva plataforma se llamó .NET Framework, un entorno de ejecución de desarrollo y ejecución de código mucho más moderno y orientado a objetos, cuya versión 2.0 está a punto de salir. Está preparado para su uso tanto en sistemas operativos de Microsoft como en otros que no lo son (ya se ha portado a Linux, en el proyecto Mono). Proporciona a los programadores una nueva infraestructura que permite un nuevo modelo de programación basado en el lenguaje C# y en lenguajes estándares de Internet como son HTTP, XML, UDDI y SOAP, orientado hacia la creación de aplicaciones distribuidas y servicios web.

.NET es la nueva estrategia de la empresa de Bill Gates para mantener a Windows como sistema operativo dominante en el mercado, a medida que la informática comienza a alejarse del escritorio hacia dispositivos conectados a Internet como palms o PDAs.

Será la API de la siguiente versión de Windows, Longhorn, cuya salida al mercado se prevé para mediados del 2006.

Es importante destacar que la compatibilidad descendente no se ha perdido, ya que .NET encapsula la funcionalidad de componentes COM (comunicación entre componentes de las antiguas versiones de Windows) por lo que puede interactuar con ellos.

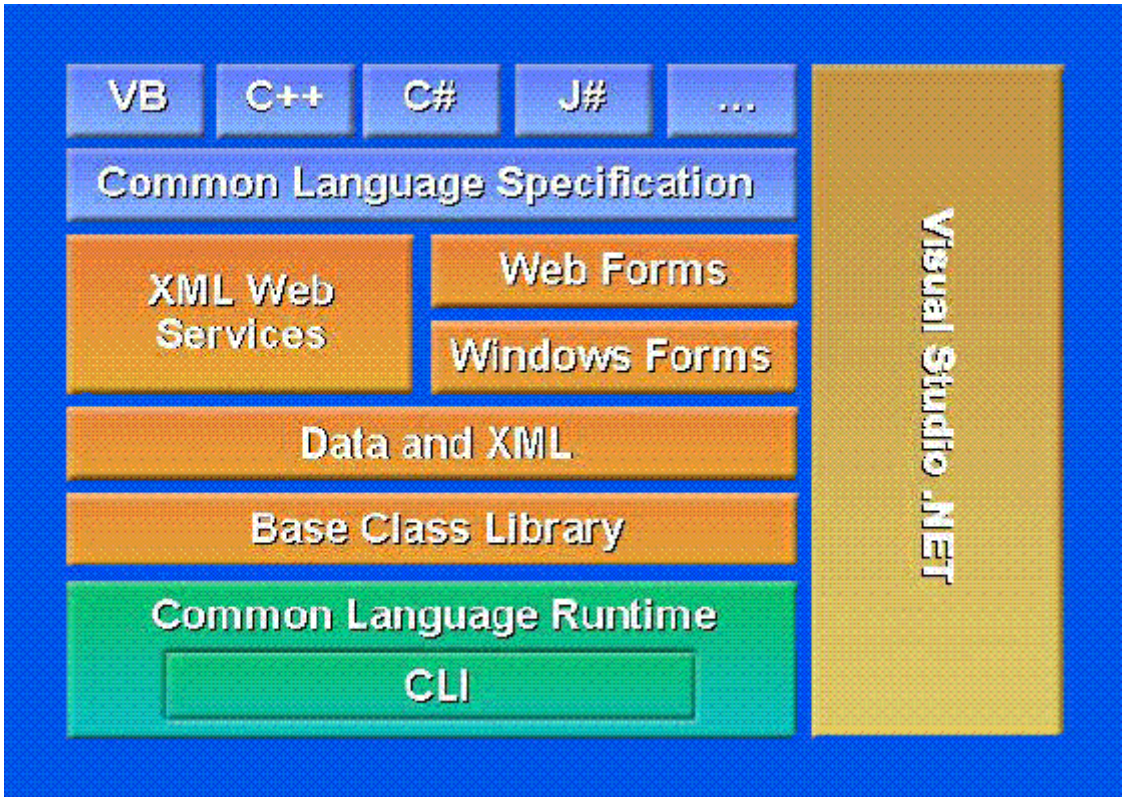
También hay que aclarar que .NET no constituye un sistema operativo en sí mismo (al menos por el momento) y que la API de Windows está aún detrás del telón.

El nuevo lenguaje creado desde cero especialmente para este entorno es C#, que aprovecha la experiencia de la programación de los últimos 20 años.

También, para no complicar la existencia a los programadores de Visual Basic se ha extendido este lenguaje al más poderoso VB.NET, aunque con las deficiencias derivadas de haber evolucionado a lo largo de los años.

En .NET también tienen cabida los lenguajes C++, JScript y J#.

La imagen siguiente muestra la estructura de .NET Framework:



Para más información sobre .NET: http://www.microsoft.com/Net/Basics.aspx

6.2. C#

En junio del año 2000 Microsoft anunció la plataforma .NET y el lenguaje creado específicamente para ella, C#. Éste es un lenguaje orientado a objetos y fuertemente tipado, diseñado para maximizar la simplicidad, expresividad y productividad de la programación. Tiene mucho en común con la manera de programar de C++ y Java, aunque también recoge características de otros lenguajes como Modula 2, C o Smalltalk.

La plataforma .NET está pensada alrededor del **CLR** (Common Language Runtime, similar a la máquina virtual de Java, **JVM**) y un amplio conjunto de librerías que pueden ser aprovechados por varios lenguajes que son capaces de trabajar juntos compilando todos a un lenguaje intermedio (**IL**, Intermediate Language).

Así, en un mismo proyecto se puede encontrar código de diferentes lenguajes como C#, VB.NET o C++ y todo se compilará a un mismo lenguaje que podrá ser interpretado por el CLR.

Los creadores de este lenguaje son Anders Hejlsberg y Scout Wiltamuth, quienes se dieron a conocer por crear también el lenguaje Delphi.

C# no puede ser considerado aisladamente, fuera del entorno de .NET. El compilador de C# tiene como destino específico el ambiente .NET, lo que significa que todo el código escrito en este lenguaje se ejecutará siempre en el contexto de .NET.

Algunas características del lenguaje C# son:

- Soporte completo para las clases y la programación orientado a objetos, incluyendo tanto herencia de interfaz como de implementación, métodos virtuales y sobrecarga de operadores.

- Un conjunto consistente y bien definido de tipos básicos y tipos construidos que cuelgan de una raíz común, la clase Object.

- Soporte intrínseco para la generación automática de documentación en XML.

- Recolector de basura (memoria reservada dinámicamente).

- Posibilidad de marcar clases o métodos con atributos definidos por el usuario. Esto puede ser útil de cara a la generación automática de documentación, y puede tener algunos efectos sobre la compilación (por ejemplo, si se marcan métodos para que sean compilados sólo durante la fase de depuración).

- Acceso total a la librería de clases base .NET, así como fácil acceso a la API de Windows, si esto fuera necesario.

- El manejo de punteros y el acceso directo a memoria están disponibles en caso de necesidad, como en este proyecto, al procesar eficientemente píxeles en imágenes.

- Reflexión: posibilidad de acceder a información sobre tipos en los programas o en unidades de ensamblado y también leer otros metadatos de los manifiestos.

- C# puede ser utilizado para crear páginas dinámicas ASP.NET.

Limitaciones de C#:

- No ha sido desarrollado para el diseño de aplicaciones de tiempo real o de alto rendimiento. No dispone de funciones en línea o destructores cuya ejecución se garantice en puntos específicos de código. Para este tipo de software se seguirá utilizando C++.

Lo que se necesita para escribir código en C#:

- Windows 98, 2000, XP ó superior.

- Instalar el SDK (Kit de desarrollo) de .NET Framework (al menos el runtime).

- Es recomendable tener un entorno de programación, por ejemplo Visual Studio (http://msdn.microsoft.com/vstudio), o X-Develop (http://www.omnicore.com). Este último tiene distribuciones para Windows, Linux y Mac OS X. También existen entornos gratuitos como Eclipse (www.eclipse.org) con un pluggin para C# (http://www.improve-technologies.com/alpha/esharp).

- Para ejecutar una aplicación escrita en C# en una plataforma que no tenga instalado el paquete mínimo de .NET se debe incrustar el runtime o motor de ejecución .NET en el ejecutable.

En este momento se está esperando la versión 2.0 del lenguaje C#.

6.3. Servicios web

El entorno dentro del cual se encuentra .NET es un Internet que está cambiando de ser centrada en las personas y basada en los contenidos, a estar centrada en las aplicaciones y basada en los servicios. Estas aplicaciones y servicios forman parte de lo que se está llamando servicios web.

Un servicio web es un servicio, con un interfaz definido y conocido, al que se puede acceder a través de Internet. Igual que una página web está definida por una **URL** (Uniform Resource Locator), un servicio web está definido por un **URI** (Uniform Resource Identification) y por su interfaz, a través del cual se puede acceder a él. Igual que una página web puede ofrecer cotizaciones de la bolsa, un servicio web que haga lo mismo presentará un interfaz para que se pueda acceder fácilmente a esos datos desde la aplicación. De esta forma, las aplicaciones se convierten en clientes que integran servicios web procedentes de diferentes proveedores, y además, cabe la posibilidad de que se cobre por uso del servicio, no por cada copia de la aplicación vendida. Este es uno de los aspectos que más gusta a Microsoft: la posibilidad de acabar de una vez por todas con la piratería, a base de alojar partes importantes de las aplicaciones en sus propios servidores, no en el ordenador del cliente.

Los servicios web son un mecanismo novedoso para la realización de llamadas remotas a métodos. Se dividen en ***servicios de transporte*** (los protocolos del nivel más bajo, que codifican la información independientemente de su formato, y que pueden ser comunes a otros servicios), de ***mensajería***, de ***descripción*** y de ***descubrimiento*****.** En la parte más baja se encuentran los servicios de transporte, que establecen la conexión y el puerto usado. Generalmente se usa HTTP, el mismo protocolo que la WWW, pero en se puede usar también SMTP (Simple Mail Transfer Protocol -el mismo protocolo que el correo electrónico), FTP (File Transfer Protocol), o BEEP (Blocks Extensible Exchange Protocol) un protocolo específico para servicios web, que, a diferencia de los anteriores, no es cliente-servidor, sino "entre pares"; los dos ordenadores entre los que se establece la comunicación actúan como clientes y servidores a la vez. Es además extensible, y está especificado en XML; por eso se está haciendo mucho más popular para aplicaciones web.

De ahí hacia arriba, están los servicios de mensajería, que especifican cómo se tiene que codificar el mensaje, que contiene los datos que se intercambian, entre el cliente y el servidor. El protocolo más usado en esta capa es el **SOAP** (Simple Object Access Protocol). Este protocolo puede usar cualquiera de los transportes anteriores, se pueden escribir clientes y servidores en cualquier lenguaje, y usa XML como lenguaje para especificar los mensajes.

Los servicios deben especificarse para que una aplicación sepa de forma automática qué formato usar para comunicarse con un servicio. Para ello se usa principalmente **WSDL** (W*eb Services Description Language*), que

permite especificar la dirección de un servicio y el interfaz que se usa para acceder a él, sea SOAP o HTML.

Por último, en la capa más alta, está **UDDI** (*Universal Description, Discovery, and Integration*), un protocolo que lleva WSDL un poco más allá, permitiendo no sólo describir servicios web, sino productos, la empresa en sí, y cómo está dispuesta a llevar a cabo transacciones. El Registro UDDI permite buscar negocios, servicios por categorías, y te devuelve informaciones sobre cómo acceder a ellos.

En el pasado, la invocación remota ha sido problemática. Con DCOM (Distributed COM) instanciar un objeto en el servidor, llamar a uno de sus métodos y obtener los resultados no era nada sencillo y la configuración necesaria estaba llena de trucos.

SOAP simplifica las cosas enormemente. Es un estándar basado en XML que detalla cómo pueden realizarse llamadas a métodos sobre el protocolo HTTP. Un servidor remoto es capaz de comprender estas llamadas y de realizar por nosotros todo el trabajo sucio, como por ejemplo la instanciación del objeto necesario, la realización de la llamada y la devolución de la respuesta formateada al cliente.

.NET Framework nos ofrece todo esto de manera muy sencilla. Con en ASP.NET, tenemos la posibilidad de utilizar el conjunto completo de técnicas de C# y .NET, pero lo más importante es que el consumo de servicios web puede lograrse desde cualquier plataforma que ofrezca acceso al servidor a través de HTTP. En otras palabras, los servicios web desarrollados para el entorno .NET pueden ser invocados, por ejemplo, desde equipos Linux, móviles con acceso HTTP, o incluso desde neveras conectadas a Internet.

Otra ventaja que permite la transmisión de la información a través de HTTP es que los firewalls o cortafuegos, que a veces filtran la información sospechosa que no se reciba por el puerto 80 permitirán este tipo de comunicación, ya que se produce por este mismo puerto.

Los antivirus de última generación suelen integrar firewalls que pueden bloquear cualquier tipo de comunicación que no se produzca por los puertos utilizados habitualmente.

Así, desde equipos situados en redes corporativas o departamentales dentro de empresas, que suelen estar protegidos por sistemas restrictivos de seguridad se podría acceder sin problemas a estos servicios, siempre que se tenga acceso a Internet.

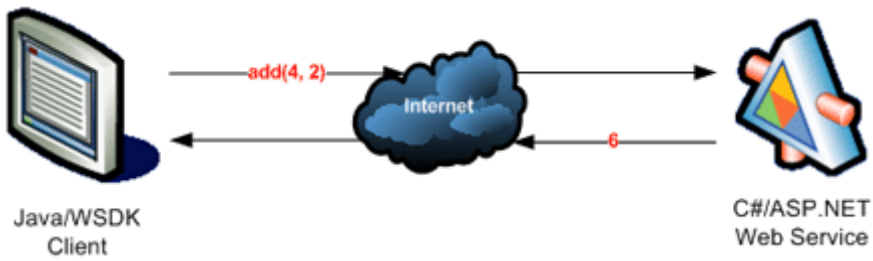
Esto es así porque en el fondo, la información que se transmite por este método es un simple **texto** en un formato determinado (SOAP).

Los servicios web se describen utilizando el lenguaje **WSDL** (Web Service Description Language), lo que hace posible el descubrimiento dinámico de servicios web en tiempo de ejecución. Esto añade un gran potencial a esta

tecnología ya que tendremos un método estándar para preguntar a un servidor qué servicios ofrece e invocarlos si se necesitan.

WSDL permite describir todos los métodos (también los tipos necesarios para llamar a esos métodos) usando XML con XML Schemas. Hay una amplia gama de tipos disponibles para los servicios web, que van desde los tipos primitivos hasta objetos complejos como DataSet, de forma que bases de datos en memoria enteras pueden ser serializadas hacia un cliente, lo q puede resultar en una reducción dramática de la carga sobre el servidor de bases de datos.

La imagen siguiente muestra la interoperabilidad que ofrecen los servicios web. A la izquierda vemos un cliente creado con Java Web Service Development Pack que hace una petición de suma de 4 y 2. A la derecha un servidor con tecnología C# y ASP.NET devolviendo la solución:



Por último, se debe tener en cuenta que los consumidores de un servicio web no tienen porqué ser únicamente aplicaciones web. No hay ninguna razón por la que no se puedan utilizar servicios web desde aplicaciones Windows o Linux – lo cual constituye una opción muy atractiva para la implementación de una Intranet corporativa.

6.4. Visual Studio 2003

Visual Studio .NET es un **IDE** (Integrated Development Environment, entorno de programación) creado por Microsoft con el objetivo de hacer más fácil e intuitiva la programación. Provee a los programadores de un amplio conjunto de herramientas para construir aplicaciones distribuidas, para la web (servicios web con XML) y para dispositivos móviles.

Hoy en día es quizás el entorno más completo junto con Eclipse (http://www.eclipse.org), herramienta gratuita orientada a la programación Java. La versión 2005 de Visual Studio está en su versión Beta 2, a punto de salir al mercado con la versión 2.0 de .NET Framework.

Se ofrecen varias versiones del programa: Visual Studio .NET 2003 Professional, Visual Studio .NET 2003 Enterprise Developer, Visual Studio.NET

Academic y Visual Studio .NET 2003 Enterprise Architect y. Nosotros hemos trabajado con la última, ya que la universidad tiene licencia y además es la más completa.

Estas son algunas de las características ofrecidas por Visual Studio. NET 2003:

1- Capacidad de modelado visual de aplicaciones, requisitos de negocios, diseño de bases de datos y **UML** (Unified Modeling Language) para especificar la arquitectura de las aplicaciones y su funcionalidad, reduciendo el tiempo de desarrollo y generando directamente clases, funciones y métodos. También provee unas 60 plantillas de diferentes tipos de proyectos.

2- Soporte para programación de dispositivos móviles:

- Incluye la construcción de aplicaciones para dispositivos inteligentes (Smart Devices) orientado a la plataforma Microsoft.NET Compact Framework (versión de .NET para dispositivos móviles), con el que se pueden desarrollar y depurar aplicaciones destinadas a Pocket PC, o teléfonos móviles. Integra un emulador de este tipo de dispositivos para probar y depurar los programas sin disponer físicamente de estos terminales.

- Soporte para programación de aplicaciones clientes ligeras orientadas a la web, como teléfonos **WAP** (Wireles Application Protocol) o asistentes personales inalámbricos (**PDA**s).

3- Soporte para el desarrollo orientado a las empresas:

- Orientado a la versión .NET Framework 1.1, que incluye mejoras con respecto a su versión 1.0, como la escalabilidad, seguridad y rendimiento.

- Desarrollo sobre Windows Server 2003.

- Migración sencilla para aplicaciones de .NET Framework 1.0.

- Nuevos controladores para crear fácilmente conexiones a fuentes de datos OLEDB y ODBC incluyendo SQL Server, Access, Jet, DB2 y Oracle.

- Soporte mejorado para trabajar con servicios web y XML. Ofrece los últimos estándares como WS-Routing, WS-Security, WS-Attachments y Direct Internet Message Encapsulation (DIME).

4- Cualquier programador puede crear un pluggin para el IDE.

5- Soporte para documentación automatizada:

- Visual Studio.NET 2003 (VS2003) ofrece la posibilidad de generar automáticamente documentación de código en formato XML. ¿Qué ventajas tiene esto? El lenguaje XML ofrece la posibilidad de separar la presentación de los documentos y su contenido, con lo que podremos presentar nuestra documentación de innumerables maneras. Por ejemplo VS2003 permite generar la documentación en formato HTML a partir de estos comentarios en XML. Con un par de clicks tendremos generado un conjunto completo de páginas HTML que documentan nuestro proyecto y que se pueden visualizar cómodamente en un navegador web.

En nuestro caso hemos experimentado algunos problemas con este tipo de documentación. La generación automática de las páginas web no creaba bien los hipervínculos, razón por la cual no las hemos entregado.

7. **El proyecto**

7.1. **Desarrollo**

Como se ha explicado brevemente en el capítulo 5, el proyecto implementa un sistema distribuido que compara dos imágenes utilizando un algoritmo de detección de cambios píxel a píxel complejo y devuelve una tercera imagen que pone de manifiesto las diferencias entre las dos primeras.

Para minimizar el tiempo de cálculo de las diferencias, las dos imágenes son procesadas por varios ordenadores autónomos, conectados mediante una red de comunicaciones y equipados con Servicios Web como vehículo de comunicación entre ellos.

Estos ordenadores no serán supercomputadores utilizados exclusivamente para esto, sino que serán máquinas que pasan la mayor parte del tiempo ociosas o desaprovechando sus recursos. Por ejemplo, una empresa que quiera aprovechar la capacidad computacional por las noches, que es cuando sus máquinas permanecen apagadas. De esta manera, la empresa puede alquilarlas e incrementar sus beneficios. Y la empresa que necesite mayore recursos los puede obtener de forma más económica que si se hiciera con una supercomputadora.

Los fundamentos de este proyecto se basan en la arquitectura cliente/servidor, es decir, un sistema distribuido donde un cliente se encarga de realizar peticiones a un sistema denominado *servidor* que se encarga de darles respuesta (cliente y servidor pueden coincidir físicamente en la misma máquina).

A continuación se muestra un esquema que muestra la estructura de esta arquitectura:

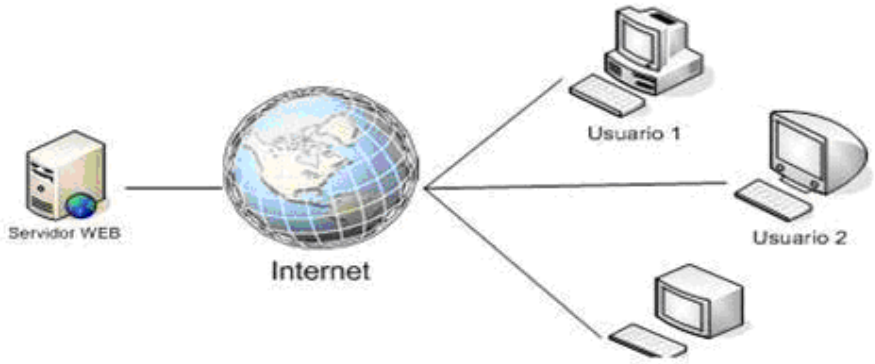


Figura 7.1.1

Por tanto, se puede hacer una distinción entre la aplicación del lado del cliente y la aplicación del lado del servidor. La aplicación del lado del cliente se encarga, entre otras cosas, de la parte más ligera del proceso, mientras que los servidores realizan la parte más pesada. A continuación se detallan, por separado, cada una de sus tareas.

- **Aplicación del lado del cliente:**

Esta se encarga de recibir como parámetros de entrada, dos imágenes. Las imágenes que se pueden comparar han de tener las mismas dimensiones y cualquiera de los formatos de imagen con extensión: bmp, jpeg, tiff, png y gif.

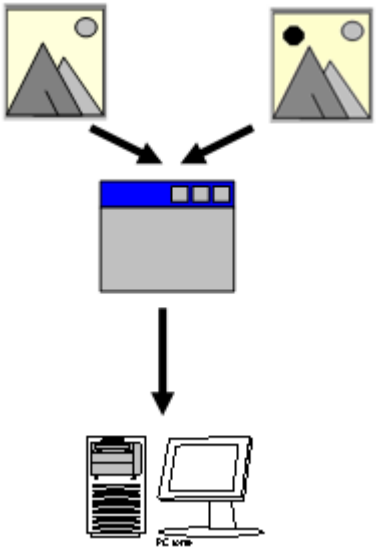


Figura 7.1.2

Los pasos que sigue el sistema son los siguientes:

1. Las imágenes recibidas como parámetros de entrada se transforman en imágenes equivalentes en escala de grises. Esta

transformación se realiza ya que el algoritmo de detección de cambios (algoritmo de enfriamiento simulado) sólo tiene capacidad para reconocer cambios en imágenes con una sola componente de color (escala de grises). De esta manera, en el programa se introducen imágenes a color, que son transformadas en sus equivalentes en blanco y negro. Estas imágenes equivalentes en blanco y negro son las que se introducen en el algoritmo de detección de cambios. Por tanto, aunque el programa reciba imágenes a color, la solución realmente es la obtenida de comparar sus equivalentes a blanco y negro. Se ha desarrollado de esta manera, para no limitar que las entradas al programa solo fueran fotos en blanco y negro. (Revisar punto: Agrisar imagen)

Es más aconsejable comparar imágenes en escala de grises, ya que las imágenes, al ser transformadas a blanco y negro, dejan de ser fieles a las originales. Para poder comparar imágenes a color, se hubieran tenido que tratar, en el algoritmo de detección de cambios, las tres componentes de color, y en este caso solo se ha tenido en cuenta una.

2. El siguiente paso que ejecuta el programa, es el de realizar la primera llamada a un servidor para recibir la información sobre los posibles servidores que son susceptibles de ser utilizados (Figura 7.1.2). Con esto se consigue tener actualizada siempre la lista de servidores que se pueden utilizar sin tener que retocar el código de la aplicación cliente. (Revisar servicio Web).

Se ha pensado de esta manera poruqe hemos considerado que somos una gran empresa que ha distribuido gran cantidad de copias del programa. Si esta lista estuviera ubicada en el software del cliente; cuando se aumente el número de servidores disponibles, habría que actualizar todas las copias del software cliente para que tuvieran cononcimiento de los nuevos servidores. Esta actualización debería hacerla el propio usuario a través de la descarga de la aplicación vía web. Y esto supondría un engorro por parte del usuario (como es el caso de las molestas actualizaciones críticas de los sitemas operativos de Microsoft).

De la manera que lo hemos implementado, nosotros como desarrolladores, solo tendríamos que modificar la lista de los servidores del servicio web; y el usuario no tendría que ser el responsable de las actualizaciones.

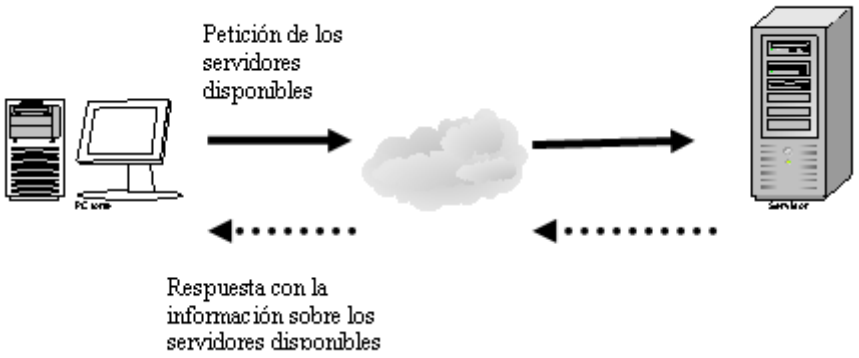


Figura 7.1.3

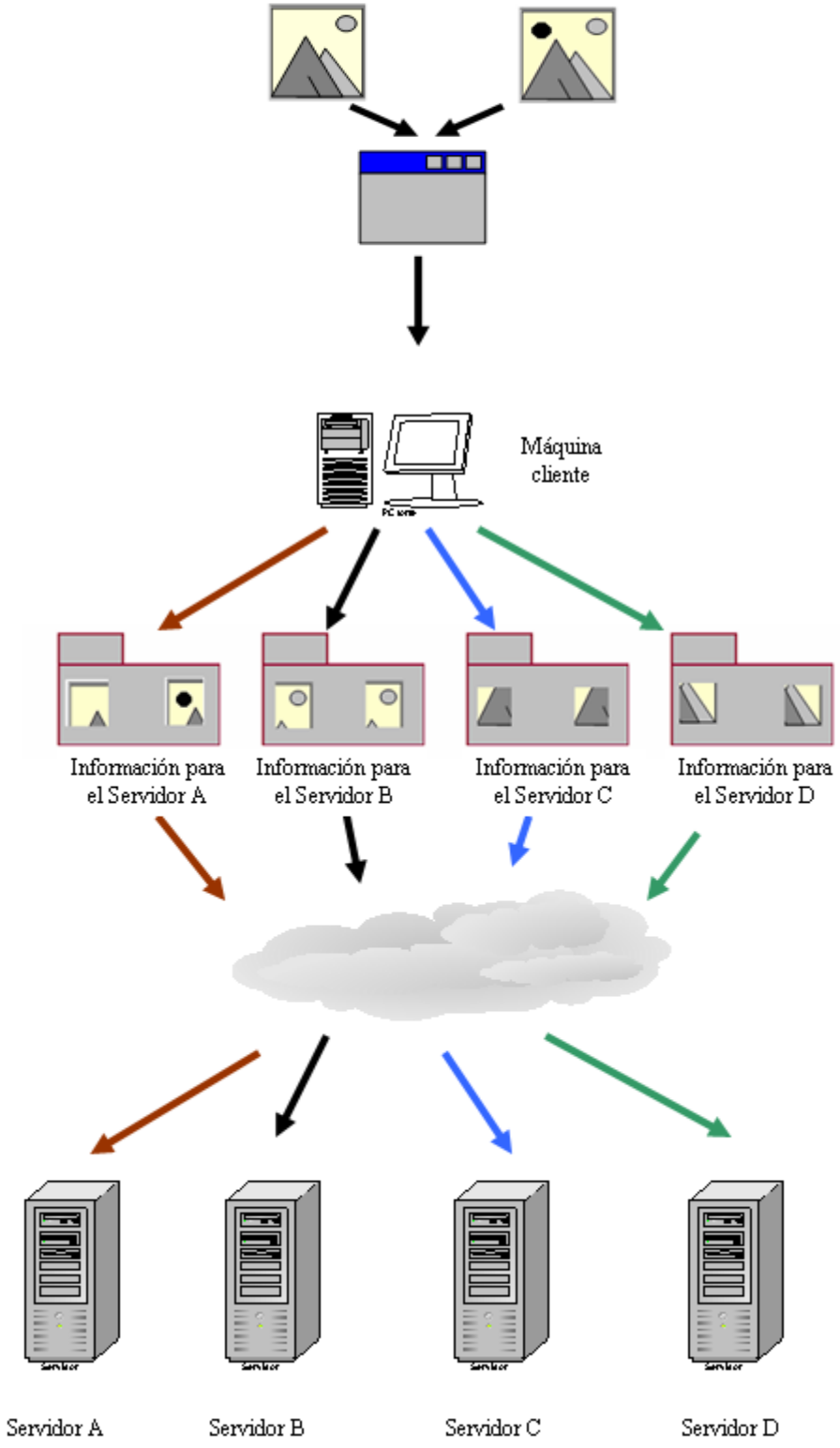
3. Una vez que tiene conocimiento de las direcciones de todos los servidores que contienen el servicio Web, se calcula la capacidad de procesamiento (o potencia) de cada uno de ellos, para más tarde repartir la imagen en relación a la rapidez en que pueden procesar los datos. Previamente la máquina cliente ha comprobado qué direcciones de las recibidas en el punto dos están disponibles y cuales no, esto es, que procesadores están operativos en ese momento y cuales no.

4. A continuación, se hace un reparto del número de filas total de la imagen entre el número de PC´s que tenemos para procesar. Este reparto se hace en función del tiempo que hayan tardado en realizar un cálculo, idéntico para todos (indicado en el punto anterior). Así conseguimos un reparto de carga equilibrado de la imagen, de tal manera que al más rápido se le asignan más filas y al más lento menos. Y, sabiendo el número de filas que debe procesar cada ordenador, se asigna un intervalo de filas específico a cada uno de ellos.

Este es el reparto que se hace por defecto, pero el usuario puede posteriormente modificarlo y asignar las filas que considere oportunas para cada procesador, pero teniendo en cuenta que se esta modificando el balanceo de carga.

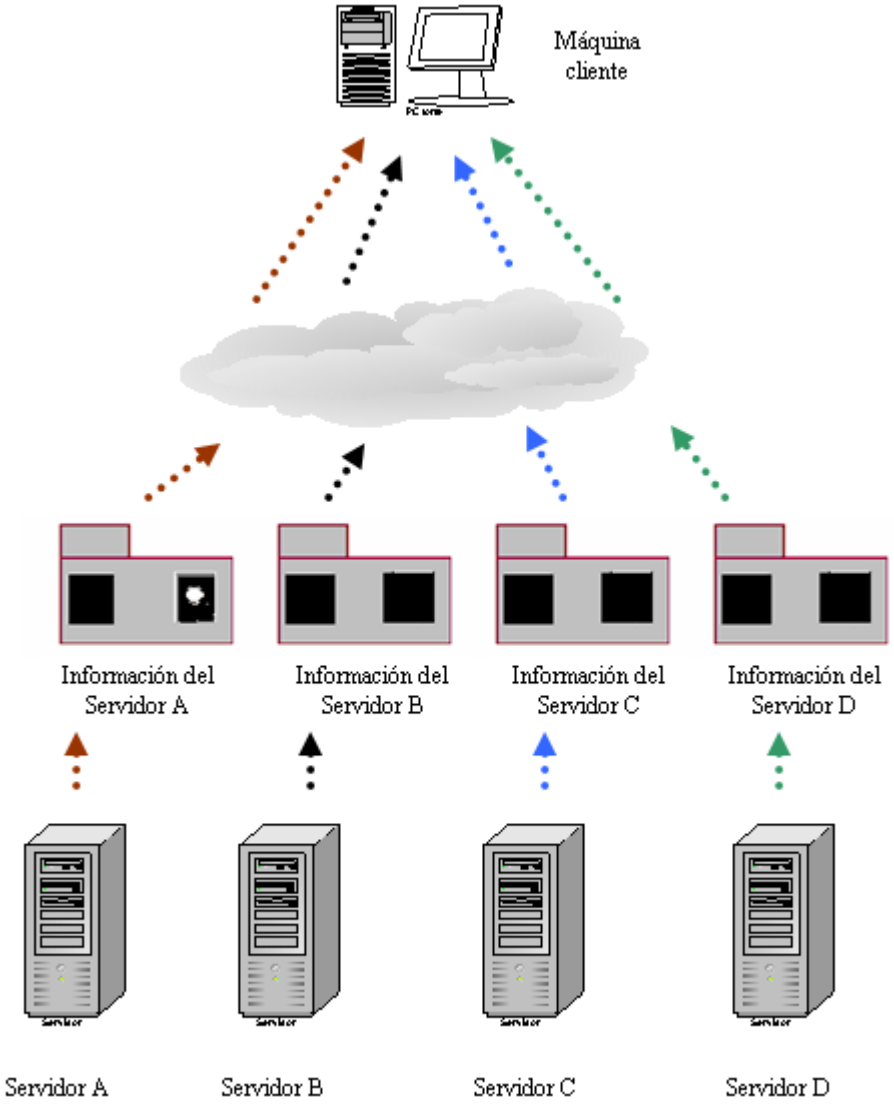
5. Ahora que ya se tiene la información necesaria para saber con qué procesadores se cuenta para el cálculo, y el intervalo de filas exactas que debe calcular cada uno, se serializa la información para mandársela a los diferentes servidores. Esto quiere decir que se transforman las imágenes a una secuencia de caracteres para que puedan ser enviarlos al servicio web a traves de la red, puesto que el servicio web no puede enviar ni recibir el tipo de datos que se manejan en el proceso, se tranforman a tipos más simples.

6. Más tarde, se lanzan a ejecución a la vez cada una de las partes de la imagen a los diferentes procesadores (Figura 7.1.3). Debido al reparto equitativo que se hizo de las partes de la imagen, deberían tardar aproximadamente el mismo tiempo todos los procesadores en devolver la respuesta debido a las estimaciones realizadas previamente.



Envío de información a los procesadores. Figura 7.1.4

7. Cuando le llegan las soluciones resultado de los procesadores, se realiza el proceso inverso de serializacion, para deserializar los datos recibidos del servicio web.



Respuesta obtenida de los procesadores. Figura 7.1.5

8. A medida que van llegando las respuestas de los servidores, se muestra ese fragmento de la imagen por pantalla. Y así, con cada una de las partes, hasta que les llega la última. Una vez que se tienen todas las partes, se construye una imagen en blanco y negro que es el resultado de ensamblar cada una de ellas (Figura 7.1.5).

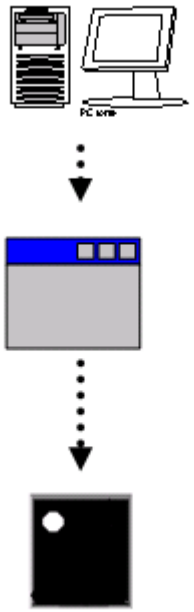


Figura 7.1.6

- **Aplicación del lado del servidor:**

La aplicación del lado del servidor contiene los métodos precisos de los que la aplicación cliente hace uso remotamente (métodos del servicio web).

Estos métodos son:

1. El método `DameProcesadoresAUtilizar`, que meramente proporciona una lista con las direcciones de los procesadores que se pueden utilizar.

2. Un método muy sencillo que simplemente se utiliza con la finalidad de comprobar que la aplicación cliente conecta con el servidor, denominado `ComprobarExistencia`. Devuelve la cadena de caracteres “Hola”.

3. Un método que realiza un cálculo, con el fin de que la aplicación cliente tome medidas del tiempo que tarda el procesador en realizarlo, y en base a todas las medidas tomadas de cada procesador, pueda comparar la rapidez de cada servidor para más tarde saber cuanta carga de proceso enviar a cada procesador. El método se llama `ComprobarPotencia`.

4. El método que implementa el algoritmo del Simulated Annealing, que recibe el nombre de `CalculaEnfSimulado`. Recibe como entradas un array de enteros con las filas de cada imagen que debe procesar y devuelve en el mismo formato la imagen diferencia de esas filas.

7.1.1. Inicialización

En este apartado se explica como se recogen los datos de cada uno de los servidores que guardaremos en variables.

7.1.1.1. Inicialización

Antes de poder hacer la llamada al método del servicio web del cálculo de diferencias entre imágenes, es necesario conocer los procesadores disponibles de los que vamos a poder hacer uso. Para conocer esta información se dispone de una lista con los servidores que se pueden emplear. Esta lista no puede ser modificada por el usuario, por lo que no va a poder añadir ningún servidor complementario, como se ha explicado en el capítulo anterior esto se realiza con la idea de evitar que la lista este en la aplicación software que tendrán instalado los usuarios en su máquina.

En un servidor, en principio *golls2*, va a estar disponible la lista con los procesadores que a priori pueden ser utilizados. Para acceder a esta lista hacemos una llamada a un método del servicio web de golls2 (Figura 7.1.1.1.1). Al mantener la lista en un servidor conseguimos que siempre esté actualizada en todos los clientes, ya que cada uno que acceda a la lista a través del servicio web encontrará la última versión de esta lista de servidores disponibles. Esto evita guardar la misma lista en varios ordenadores y la dificultad de tener que cambiarlas todas al mismo tiempo cuando se requiera modificar la información de la lista, ya que si no se modificasen a la vez perderíamos consistencia en la información. A continuación se muestra como se realiza la llamada al método `DameProcesadoresAUtilizar` del servicio web, que nos devuelve los procesadores de la lista:

```
servicio = new referencia.Service1("http://golls2.esi.ucm.es/Grupo2/
                ServicioWebEnfSimulado/Service1.asmx");

procesadores =servicio.DameProcesadoresAUtilizar();
```

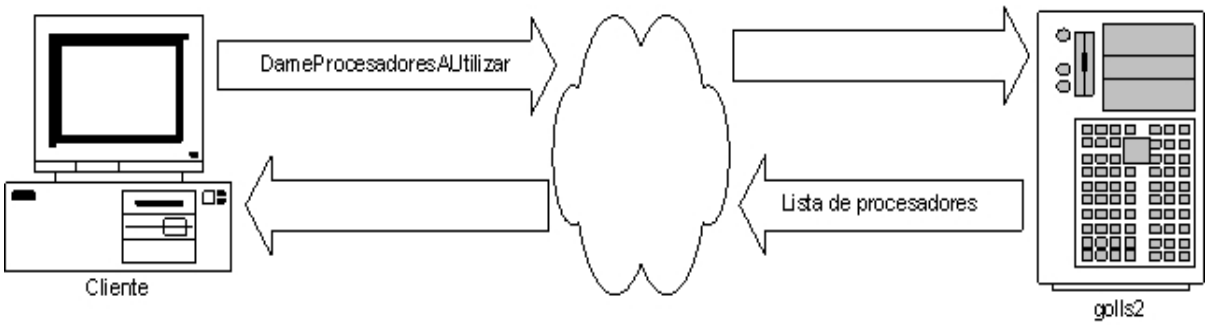


Figura 7.1.1.1.1

Si no está disponible *golls2* accedemos a un segundo equipo (en nuestro caso con IP 147.96.188.72) en el que también tendremos actualizada la lista de servidores.

Una vez que hemos obtenido la lista de servidores, tenemos que asegurarnos de que realmente están disponibles. No basta con obtener la lista de procesadores ya que puede que haya alguno que no funcione debido a un problema, por ejemplo, si ha perdido la conexión de red, está demasiado ocupado, está apagado o ha sufrido un error de hardware. Para saber si un servidor esta operativo se hace una llamada a través de un servicio web, a un método que nos devuelve la cadena de caracteres “Hola” si el procesador esta operativo, de manera que si no está operativo lo descartaremos enseguida. Si pasado un tiempo prudencial (5 segundos) no hemos obtenido respuesta de algún servidor lo consideramos no operativo. Para cada servidor que pueda ser usado lanzamos un hilo que ejecute la llamada a ese servicio web, de esta forma el proceso se realiza concurrentemente y obtenemos las respuestas de una forma más rápida (Figura 7.1.1.1.2):

```
/*creamos un hilo para comprobar que el servidor existe y no está
caído*/

hiloExistencia[i] = new Thread(new
                                ThreadStart(ComprobarExistenciaHilo));

hiloExistencia[i].Name = i.ToString();

hiloExistencia[i].Start();

responde[i] = hiloExistencia[i].Join(tiempoPrudencial);
```

Código que muestra la llamada al método del servicio web que nos devuelve el *string* de respuesta:

```
comprobador[i] = new referencia.Service1(procs[i].direccion);

respuesta[i]=comprobador[i].ComprobarExistenciaWS();//El método
                                                    devuelve un string
```

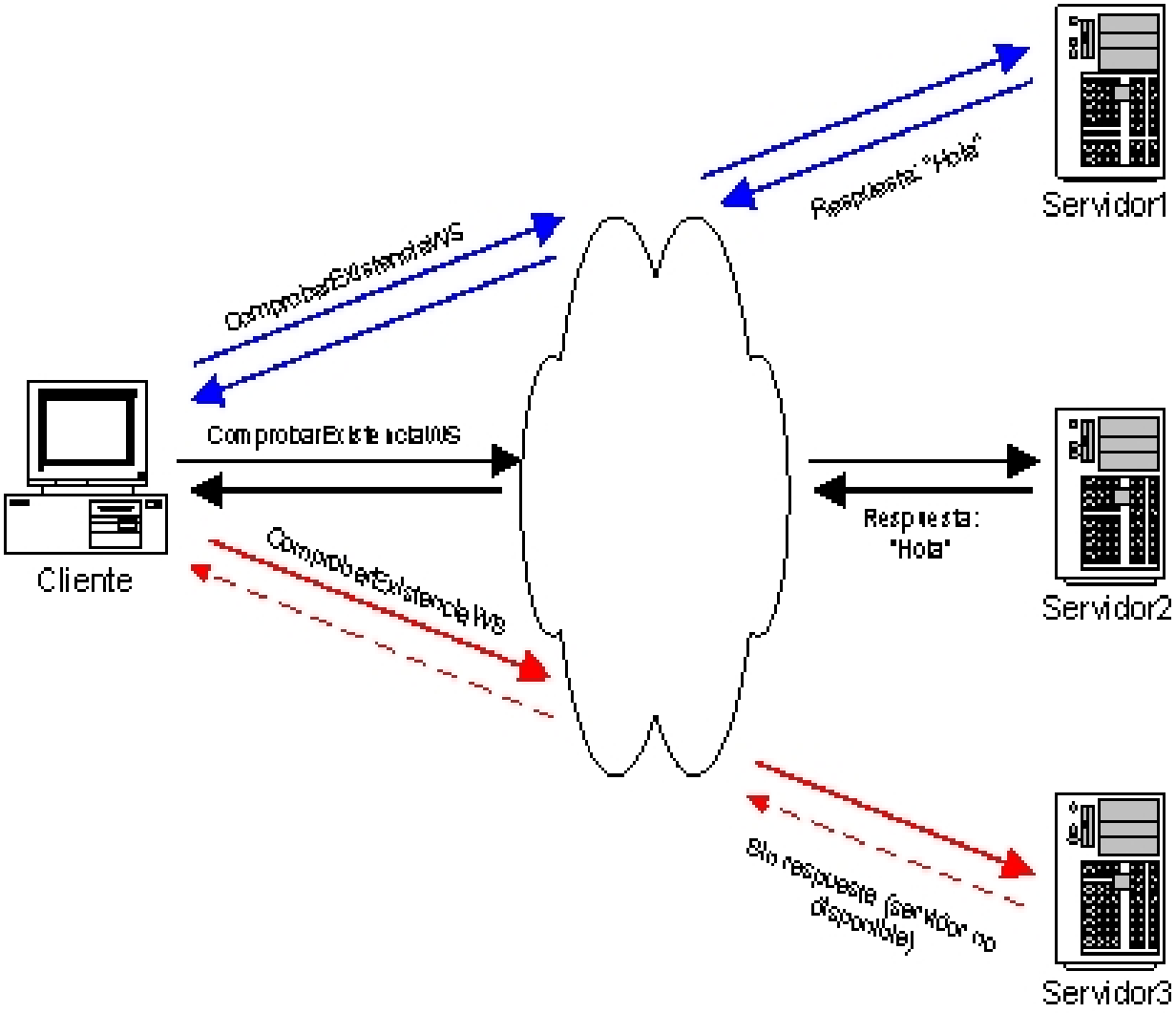


Figura 7.1.1.1.2

Después de esto ya tendremos los nombres de los servidores, así como la disponibilidad de cada uno de ellos. En esta lista también se incluye el procesador del cliente que ejecuta la aplicación, con lo cual la máquina cliente también va a realizar cálculos para la resolución del algoritmo sin quedarse en espera hasta que acaben los servidores. La información de cada procesador la guardamos en un *struct*, y los *struct* de todos los servidores en un *array de procesadores*. A continuación se muestran los datos que guarda un *struct* de cada procesador:

```
public struct Processor
{
      public string direccion;//Contiene el nombre del servidor
      public bool disponible;//Indicara si el servidor esta activo o
                                    //no
      public int tiempoEmpleado;//Tiempo que emplea en realizar el
                                     //cálculo de prueba de carga.
      public int filaIni;//primera fila de la imagen asignada a este
                                //procesador
      public int filaFin;//última fila de la imagen asignada a este
                                //procesador
      public int numFilas; //filaFin-filaIni
}
```

Cuando se acaban de inicializar los procesadores sólo tendrán valores los campos *direccion* y *disponible*, el resto se rellenarán en los pasos sucesivos.

7.1.1.2. Estimación de tiempos de procesadores

Cuando conocemos qué procesadores están operativos para el proceso de cálculo debemos estimar qué parte de las imágenes calcula cada uno. De tal manera que haya un balance de carga en función de las potencias de los procesadores en ese momento. Antes de continuar explicaremos qué son las matrices de las imágenes.

Como se explica en el punto de tratamiento de las imágenes (7.1.2), cada imagen está representada por una matriz de dos dimensiones (ancho y alto) de píxeles. Esta matriz se obtiene a partir de cada imagen en la fase de agrisamiento. Esta matriz esta compuesta por números enteros comprendidos entre 0 y 255, que representan píxeles en escala de grises.

Cada servidor va a procesar una parte de estas matrices. Hay muchas maneras de de hacer las divisiones en la matriz. Nosotros hemos elegido la que creemos más fácil, por filas. Más adelante se explicará el método elegido para hacer el reparto de filas (7.1.4). Por ahora sólo es necesario saber que para realizar el reparto tenemos que estimar la carga que puede ejecutar cada uno de los procesadores, para conseguir un tiempo de respuesta similar de cada uno de ellos. Para ello, se hace una llamada, a cada servidor disponible, al método del servicio web encargado de proporcionar unos tiempos que indican la potencia de cálculo que tienen en ese momento:

```
referencia.Service1 comprobadorPot=new
                              referencia.Service1(procs[i].direccion);

comprobadorPot.ComprobarPotenciaWS();
```

Estos tiempos devueltos son los tiempos que tardan los servidores en hacer unos cálculos cuya única finalidad es medir el tiempo que tardan en realizarlos. Se podría considerar como un benchmark del servidor: hacemos unos cálculos para determinar el rendimiento de los ordenadores y así determinar cuales son los más rápidos, que son los que ejecutarán mayor carga de trabajo.

Para cada llamada que hacemos al servicio web lanzamos un hilo, de esta manera se calcularán todos los tiempos de manera concurrente (Figura 7.1.1.2.1).

```
hiloPotencia[i] = new Thread(new ThreadStart(ComprobarPotenciaHilo));

hiloPotencia[i].Name= i.ToString();

hiloPotencia[i].Start();

hiloPotencia[i].Join();
```

Dentro del método `ComprobarPotenciaHilo` se realiza la llamada al servicio web que calcula los tiempos.

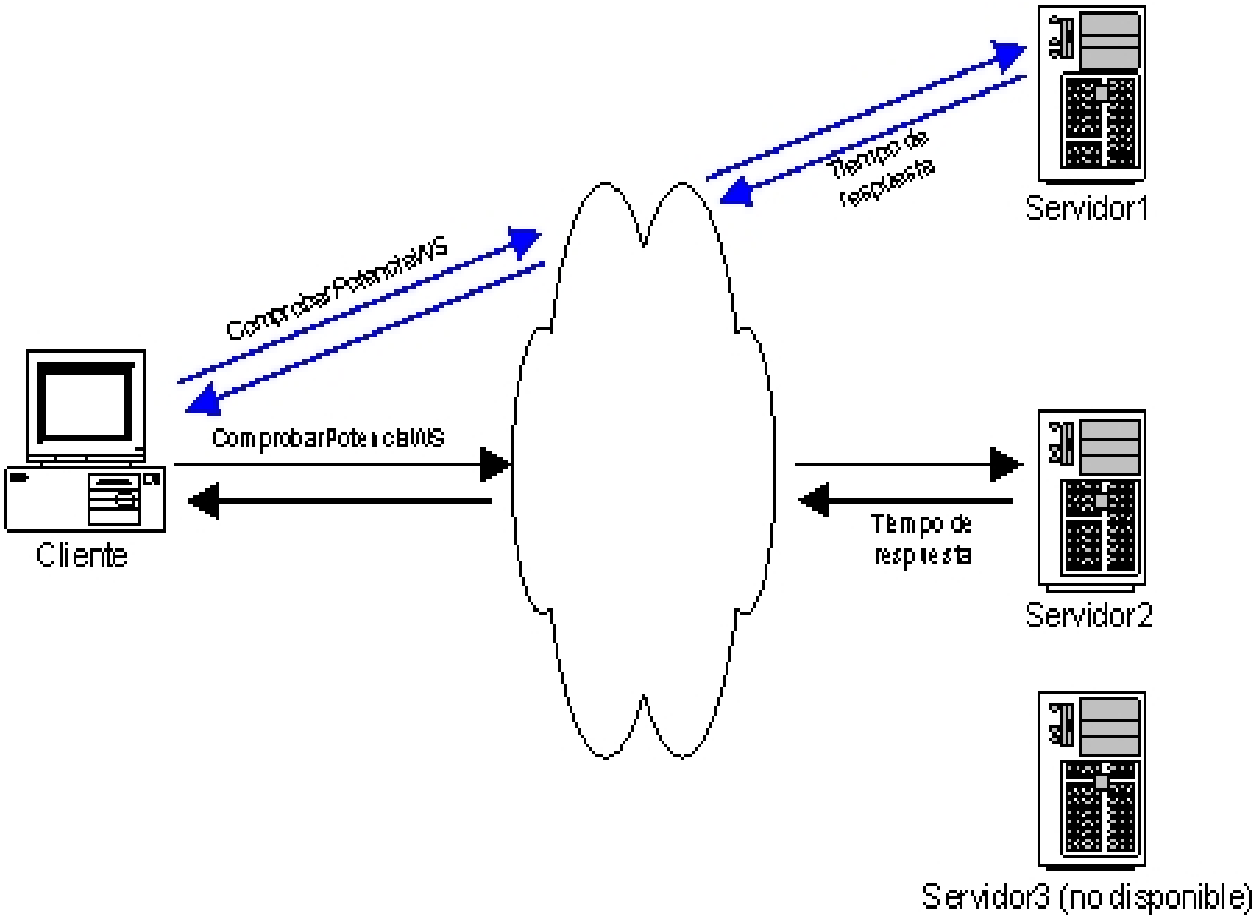


Figura 7.1.1.2.1

7.1.2. Tratamiento de imágenes

.NET ofrece, funcionalidades para el tratamiento de imágenes. Para ello Microsoft ha incluido en este entorno la tecnología **GDI+** (Graphic Device Interface) cuyo objetivo es abstraer las características del hardware a una API de alto nivel. GDI+ es un conjunto de clases base que son capaces de producir las instrucciones que deben ser enviadas a los controladores de dispositivos gráficos para producir la salida gráfica apropiada en el monitor (o impresa en papel).

Nosotros hemos usado estas funciones de .NET (que residen en el espacio de nombres System.Drawing) aunque no tan extensivamente como nos hubiera gustado, ya que al apoyarse sobre clases y objetos instanciados de estas clases no se consigue un rendimiento como el que obtendríamos con C++ si operamos a nivel de píxel, que es algo que tendremos que hacer.

Por lo tanto, para la manipulación de las imágenes a bajo nivel usamos secciones de código no seguras, es decir, bloques de código C++ delimitados por la palabra reservada **unsafe**.

La primera fase de la aplicación es comprobar que las imágenes de entrada (imágenes A y B) están en el formato que admite el algoritmo de enfriamiento simulado y si no lo están, transformarlas. El formato debe ser el de una imagen en blanco y negro (más concretamente en escala de grises) con formato de píxeles `Format8bppIndexed`.

En el siguiente bloque daremos a conocer algunos conocimientos previos necesarios para comprender lo que significan estos formatos y la manera en la que .NET encapsula las imágenes, que puede llegar a ser un poco confusa.

Dado que la manipulación de imágenes a bajo nivel a través de las funciones GetPixel y SetPixel no es realmente eficiente, accederemos a la memoria, como hemos mencionado antes, a través de código no seguro en C++.

En .NET hay 14 formatos de píxeles en imágenes. A nosotros nos interesa el formato **`Format8bppIndexed`**, ya que tiene una paleta de 256 colores codificando cada uno con 8 bits, de ahí su nombre. Es decir, tiene un array de 256 posiciones y en cada una guarda codificado (en 8 bits) un color, en nuestro caso será un gris. Esos 8 bits se dividen en 4 pares de dígitos hexadecimales (cada par es un byte) que indican los niveles de los 3 componentes espectrales de la luz (rojo, verde, azul) y el de opacidad o nivel alfa (este último indica el nivel de transparencia). De ahí el acrónimo **RGBA** (Red, Green, Blue, Alpha).

Esta manera de codificar los colores de la imagen nos permitirá guardarla o codificarla como un array de enteros, cuya manipulación será mucho más rápida y fácil.

Es importante no confundir los formatos de píxeles de imágenes con las extensiones de los achivos de imágenes en el sistema operativo. Nuestra aplicación acepta imágenes con las extensiones BMP, TIFF, GIF, PNG y JPEG.

Lo primero que debemos hacer es transformar la imagen original en una imagen con el formato de píxeles `Format8bppIndexed`, ya que será ese formato el que usará nuestro algoritmo de enfriamiento simulado.

Debido a que no se pueden presuponer algunos parámetros necesarios para la transformación puesto que si se hace se perdería información, no existe una función para convertir cualquier formato al que nosotros usaremos (`Format8bppIndexed`). Se podría pensar que lo más simple sería crear una función para cada uno de los 13 formatos restantes que convirtiera la foto a `Format8bppIndexed`. Sin embargo, aprovecharemos la ventaja de que se puede pasar desde cualquier formato al formato `Format24bppRgb`.

La idea entonces será pasar la imagen original a formato `Format24bppRgb` y hacer una única función que lo pase luego al formato `Format8bppIndexed`. En esta transformación se puede perder parte de la información de la imagen original, pero es información que no necesitaremos (por ejemplo los colores).

Aquí entramos en el proceso de pasar una imagen en formato `Format24bppRgb` (que puede estar en blanco y negro o en color) al formato objetivo `Format8bppIndexed`.

El siguiente fragmento de pseudocódigo ilustra el proceso que acabamos de explicar:

```
if (imagen <> Format8bppIndexed)
      PasarAFormat24bppRgb( imagen );

PasarAFormat8bppIndexed( imagen );
```

Hay que recordar que las imágenes con la paleta de colores codificada en formato RGBA tienen un byte para cada uno de sus 4 componentes, y hay que pasarlo a un gris que ocupará solamente un byte..

Esto se hace traduciendo el píxel del sistema RGB al sistema **YIQ**. Este último sistema es el usado en la codificación de imágenes en el formato NTSC de las televisiones a color, que aprovecha algunas características de la visión humana para maximizar el uso de la señal.

El formato YIQ tiene 3 componentes que son respectivamente la luminancia, la crominancia roja y la crominancia azul. Esta transformación es muy útil porque de aquí obtenemos la luminancia (Y) que es justamente el gris que buscamos.

Según la CIE (Comisión Internacional sobre Iluminación) la luminancia es la manera objetiva de medir la cantidad de brillo. Es la línea donde los tres componentes R, G y B son iguales, es decir, tienen el mismo valor siendo el mínimo 0 y el máximo 255.

Para pasar del sistema RGB a YIQ aplicamos a cada uno de los 3 componentes de color RGB unos coeficientes y los sumamos:

Y = (R x 0.299) + (G x 0.587) + (B x 0,114)

donde Y es un número de 0 a 255 (un gris). El 0 es el negro y a medida que aumenta el número se aclara hasta llegar al 255, el blanco.
Esta es la parte del programa en la que se emplea código no seguro, accediendo directamente a píxeles que están en la memoria y aplicándoles estos coeficientes.

La siguiente imagen muestra la relación entre los sistemas YIQ y RGB:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.528 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Figura 7.1.2.1

Ahora tenemos una imagen en escala de grises en formato `Format8bppIndexed`, y hemos guardado cada uno de los grises (representados por números enteros de 0 a 255) en un array de dos dimensiones (el ancho y alto de la foto). Por ejemplo, si tenemos una imagen de 800x600 píxeles, obtendremos un array de enteros de 600 filas y 800 columnas.

A partir de este momento, todo el proceso se hará usando este array, dado que las operaciones de acceso a los datos son mucho más rápidas.

Resumiendo, hemos pasado de tener dos imágenes en cualquier formato (y en uno de los 5 tipos de extensiones que admite nuestro programa) a tener dos arrays de dos dimensiones de enteros.

Esos dos arrays de enteros se los pasaremos a las máquinas que los vayan a procesar, que son el cliente y los servidores remotos disponibles.

7.1.3. Implementación de los servicios web

En este apartado se explicará con detalle la estructura del servicio web desarrollado para este proyecto, lo que se ha denominado en el apartado de desarrollo “aplicación del lado del servidor”.

El servicio es un servicio web ASP.NET, dentro de las plantillas de proyecto de ASP.NET, que esta incluido en el paquete de Visual Studio.NET 2003.

Resulta muy sencillo implementar un servicio web, ya que Visual Studio, por medio del VSDesigner reduce la complejidad y se encarga de todo lo relativo a las comunicaciones a través de Internet (el documento Dicovery y el WSDL), de tal manera que los desarrolladores del proyecto solo tienen que centrarse en la lógica de negocio.

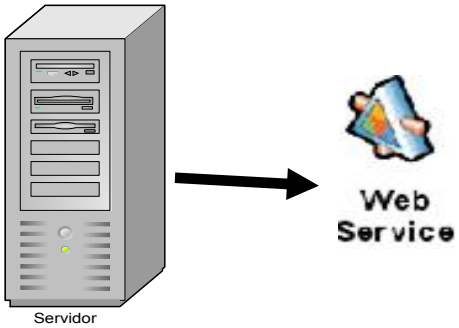


Figura 7.1.3.1

El cuerpo del servicio web está formado por 4 métodos que serán invocados desde la aplicación cliente. Cada uno de estos métodos se ha desarrollado atendiendo a las necesidades de la aplicación cliente, aunque pueden ser invocados por cualquier otra aplicación que necesite exactamente las funcionalidades que ofrecen. Seguidamente se explican los pormenores de cada uno de estos métodos:

- Método 1:

  La cabecera del método es la siguiente:

```
DameProcesadoresAUtilizar():String[]
```

  Este método devuelve la información sobre todos los posibles procesadores que se pueden utilizar. O mejor dicho, manda a la aplicación cliente todas las direcciones desde donde se puede ejecutar el servicio web. La lista de direcciones se devuelve como un array, en el que cada posición contiene una cadena de caracteres con una dirección.

  La idea de este método es que la aplicación cliente siempre tenga actualizada la lista de direcciones que puede utilizar. En un primer prototipo de la aplicación, este método no existía ya que las direcciones se incluían dentro de la aplicación. Pero más tarde, y con el objetivo de generalizarla lo máximo posible, se pensó que en cualquier momento se podría disponer de un "servidor más", es decir, un ordenador disponible más en la red donde estuviera instalado el servicio web. En el primer prototipo, habría que modificar el código de la aplicación cliente e incluir una línea más. Y así con todas las aplicaciones cliente que hubiera distribuidas.

  Por este motivo, se pensó que sería mejor que la aplicación cliente, al comenzar su ejecución, se conectara a algún servidor y se descargara la lista de todas las direcciones. De esta manera sólo se tendría que modificar este servicio web desde el servidor (y ni siquiera desde todos los servidores), y todas las aplicaciones clientes

quedarían exentas de modificaciones. Si el cliente no está conectado a la red, sólo usará su propio procesador.

- Método 2:

Que tiene la forma:

```
ComprobarExistenciaWS():string
```

Este método únicamente devuelve la cadena de caracteres "Hola".

Aunque parezca una simpleza, la aplicación cliente lo invoca y del resultado de la invocación se conocerá si el servidor esta disponible para ser usado, o si por el contrario ha habido algún problema con la conexión.

De esta manera, la aplicación podrá elaborar la verdadera lista de direcciones que se va a utilizar, haciendo la comprobación con cada una de las direcciones que ha obtenido gracias al Método 1.

- Método 3:

La cabecera tiene el siguiente formato:

```
ComprobarPotenciaWS():void
```

Hace una llamada al método `CompruebaCarga()` de la librería `AlgComprobarCarga`. Este cálculo se realiza con ánimo de cronometrar el tiempo que tarda cada procesador en ejecutar esa función. De esta manera, si se puede llevar a cabo una comparativa de tiempos. En resumidas cuentas, el propósito de este algoritmo se puede asemejar a la idea de benchmark, es decir, que sirve para evaluar el rendimiento del ordenador. Para más información, revisar la sección 7.1.1.2 (Estimación de tiempos de procesadores).

- Método 4

El método presenta la siguiente estructura:

```
public String[] CalculaEnfSimuladoWS(
    String[] smatrizA,String[] smatrizB,int numFilas,
   int numColumnas,int numFI,int numFF,int vueltas,
   int solapamiento)
```

Este método es uno de los más importantes, ya que es el que va a determinar las diferencias entre las imágenes. Hace una llamada a la librería AlgEnfriamientoSimulado y a la librería SerializaciónDatos.

Lo primero que hace esta función cuando recibe los datos es deshacer la serialización de los datos que recibe por entrada. Esto significa que traduce los datos que le llegan por la red al formato que necesita. El concepto de serialización viene explicado con detalle en el punto 7.1.5.

Una vez obtenido los datos en el formato adecuado, se hace una llamada al a librería de AlgEnfrimamientoSimulado para que calcule los datos correspondientes (mas información sobre esta librería en la sección 7.1.6). A continuación se vuelven a serializar los datos para poder enviarlos a la aplicación cliente.

El uso del servicio web requiere la creación de la clase Proxy que es la encargada de la comunicación entre la aplicación cliente y el servicio web. La aplicación cliente se comunica directamente con el Proxy para invocar las operaciones necesarias del servicio web. Así, al invocar las llamadas de las funciones a través del Proxy, el cliente cree que está llamando a una función local, cuando en realidad lo está haciendo a una máquina remota.

Se puede ver la información relativa a un servicio web o verificar la operatividad del servicio en la máquina sobre la que se ha desarrollado la aplicación (en este caso, golls2.esi.ucm.es). Para ello, basta con dirigir el navegador a la ubicación donde se encuentra el servicio web. Para hacer una prueba, se puede ver la información del servicio web que se ha implementado en este proyecto en esta dirección:

http://golls2.esi.ucm.es/Grupo2/ServicioWebEnfSimulado/Service1.asmx

La siguiente figura muestra la lista de todas las operaciones disponibles para el servicio web que se muestran al seguir este vínculo.

Service1

as operaciones siguientes son compatibles. Para una definición formal, revise la descripción de servicios.

- CalculaEnfSimuladoWS
- DameProcesadoresAUtilizar
- ComprobarPotenciaWS
- ComprobarExistenciaWS

iste servicio Web utiliza http://tempuri.org/ como espacio de nombres predeterminado.

ecomendación: cambiar el espacio de nombres predeterminado antes de hacer público el servicio Web XML.

ada servicio Web XML necesita un espacio de nombres único para que las aplicaciones de cliente puedan distinguir este servicio de otros servicios del Web. http://tempuri.org/ está disponible para servicios Web ML que están en desarrollo, pero los servicios Web XML publicados deberían utilizar un espacio de nombres más permanente.

ebe identificar su servicio Web XML con un espacio de nombres que controle. Por ejemplo, puede utilizar el nombre de dominio de Internet de su compañía como parte del espacio de nombres. Aunque muchos spacios de nombres de servicios Web XML parecen direcciones URL, éstos no pueden señalar a recursos reales en el Web. (Los espacios de nombres de los servicios Web XML son los URI.)

n los servicios Web XML que se crean con ASP.NET, se puede cambiar el espacio de nombres predeterminado utilizando la propiedad Namespace del atributo WebService. Este atributo es un atributo aplicado a la lase que contiene los métodos del servicio Web XML. A continuación se muestra un ejemplo de código que establece el espacio de nombres en "http://microsoft.com/webservices/":

#

```
[WebService(Namespace="http://microsoft.com/webservices/")]
public class MyWebService {
    // implementación
}
```

isual Basic.NET

```
<WebService(Namespace:="http://microsoft.com/webservices/")> Public Class MyWebService
    ' implementación
End Class
```

7.1.4. Razonamiento del reparto de filas de la imagen

Para poder realizar la ejecución del algoritmo del cálculo de las diferencias en varios servidores, es necesario hacer una división de las matrices que forman las imágenes de entrada. Cada división será una submatriz de la matriz imagen, y cada servidor hará el tratamiento de una de estas submatrices. Ya se ha comentado anteriormente (puntos 7.1.1.2 y 7.1.2) el formato de matriz bidimensional en el que hemos traducido las imágenes.

El reparto de la matriz se puede hacer de distintas formas, nosotros hemos elegido la que hemos creído más fácil de entender e implementar, es decir, por filas. Cada submatriz estará definida por su fila inicial, las filas intermedias y su fila final. Estas filas serán pasadas como parámetros a cada servidor junto con la matriz inicial de la imagen y así el servidor correspondiente hará los cálculos de la parte de la matriz que le es indicada a través de las filas inicial y final.

Lo primero es calcular el número de filas que va a ejecutar cada servidor. De este cáculo se encarga el método llamado `CalculaReparto`. Para realizar este cálculo se tiene en cuenta el tamaño de la matriz imagen de entrada y el tiempo obtenido de cada procesador al ejecutar un pequeño cálculo, (este método se ha explicado anteriormente en los puntos 7.1.1.2 y 7.1.3 de esta documentación).

La fórmula matemática para realizar este cálculo es la siguiente:

$$T / T_T = C$$

Donde T es la suma de los tiempos de respuesta de los servidores ($\sum TR_i$) dividido entre el tiempo de respuesta del servidor i (TR_i) y T_T es la suma de los T de todos los servidores.

El siguiente fragmento de código realiza la suma de los tiempos de respuesta de los servidores:

```
for (int i=0;i<procs.Length;i++)
{
      if (procs[i].disponible)
      {
            sumaTiempos += procs[i].tiempoEmpleado;
      }
}
```

Y el siguiente código calcula la variable T:

```
for (int i=0;i<procs.Length;i++)
{
      if (procs[i].disponible)
      {
            if (procs[i].tiempoEmpleado != 0)
            {
                  ts[i]=sumaTiempos/procs[i].tiempoEmpleado;

                  t += ts[i];
            }
            else
            {
                  ts[i]= sumaTiempos / (float)0.1;

                  t += ts[i];
            }
      }
}
```

Procs es un array donde se guardan los datos de los servidores (ver 7.1.1.2). Si un procesador ha sido muy rápido y su tiempo es cero se le asigna 0.1 a efectos prácticos, ya que es imposible que tarde cero milisegundos en responder, y asi evitan problemas de divisiones con resultados infinitos.

En el código, en la variable *t* llevamos la suma de los T (el T del procesador i va en la posición i del array *ts*).

Sólo se hacen los cálculos en los procesadores que están disponibles, los no disponibles no sirven para nada.

Para terminar C se multiplica por el número de filas (C*NumFil) y ese es el número de filas que va a ejecutar ese procesador. El siguiente código realiza este cálculo y el cálculo de C para cada procesador:

```
for (int i=0;i<procs.Length;i++)
{
      if (procs[i].disponible)
      {
            float c;

            c = ts[i]/t;

           procs[i].numFilas = Convert.ToInt32(numFilasImagen*c);

           numFilasRedondeadas += procs[i].numFilas;
      }
}
```

En la variable *numFilasRedondeadas* se guarda el número de filas totales que se van asignando. Esto se hace porque la estimación de filas asignadas a cada servidor pueden ser números decimales, y como las filas a ejecutar tienen que ser número enteros (ya que mandamos filas enteras a procesadores) convertimos ese número a un entero. Debido a este redondeo, puede que al final haya alguna fila sin asignar, y estas filas sin asignar se las añadimos al equipo local, el que contine la aplicación cliente:

```
int numFilasRest=numFilasImagen-numFilasRedondeadas;

if (procs[0].numFilas + numFilasRest>=0)
      procs[0].numFilas += numFilasRest;
else
      procs[0].numFilas=0;
```

Con el cálculo de T para cada servidor se consigue un número proporcionalmente inverso al tiempo que ha tardado en responder con respecto al resto de servidores, es decir, un servidor con un tiempo de respuesta mayor que otro tendrá un número T menor (cuanto más tiempo de respuesta menos columnas va a ejecutar.) Con el cálculo de C para cada procesador se normaliza el número T calculado anteriormente para que sea un número de 0 a 1, y por lo tanto, la suma de todos los C dará 1 como resultado. Al multiplicar estos C por el número de filas se obtendrá como resultado las filas que va utilizar cada procesador en los cálculos.

Después de calcular el número de filas que ejecutará cada procesador se procede fijar el intervalo de filas que debe realizar cada procesador. Para ello se fijan las filas inicial y final, necesarias para determinar la submatriz correspondiente. Para hacer el cálculo de estas filas es necesario saber el número de filas de solapamiento que tiene el algoritmo.

El solapamiento es un número que indica la cantidad de filas que se calcularán en dos procesadores a la vez. Esto se hace para mejorar el proceso de detección de cambios, ya que aumenta la probabilidad de encontrar diferencias (Ver punto 7.1.6 para más detalle). Definiremos una variable que llevará cuenta de la primera fila de la matriz imagen que no esta asignada. Esta variable se inicializa con la fila 0, que es la primera y se llamará *fila incial*.

Ahora, pondremos el ejemplo de cómo se calcula el intervalo de filas a ejecutar por el primer procesador disponible. Como no encontramos en la primera fila, no existirá solapamiento en la parte de arriba de la submatriz, por lo tanto se suman el número de filas a ejecutar por el primer servidor a la variable *fila inicial* y obtendremos la *fila final* sin solapamiento. Para hallar la fila final con solapamiento simplemente se suma el número de filas a solapar a la *fila final* calculada anteriormente. El número de filas a ejecutar por cada servidor son las que hemos calculado al principio de este apartado. El código siguiente muestra este proceso:

```
if(ultimaFilaAsignada==0) //por aquí entrará localhost
{

      procs[i].filaIni=ultimaFilaAsignada;

      ultimaFilaAsignada=ultimaFilaAsignada+procs[i].numFilas;
            //ahora, esta comprobación es por
               si el cliente es el único servidor disponible

      if (ultimaFilaAsignada==numFilasImagen)

         procs[i].filaFin=ultimaFilaAsignada-1;
      else

         procs[i].filaFin=ultimaFilaAsignada-1+solapamiento;
}
```

Si nos encontramos en una fila intermedia de la matriz, esto es, que *fila inicial* no sea 0, entonces se añade solapamiento por encima y por debado del intervalo de filas. Esto se hace, por arriba restando las filas a solapar a la *fila inicial*, y por debajo, que se hace sumando las filas a solapar a la fila final. La fila inicial será la fila siguiente a la fila final de la submatriz anterior, y la fila final se calcula sumando las filas que tiene que ejecutar el servidor a la fila inicial. El código siguiente muestra este proceso:

```
else if (ultimaFilaAsignada+procs[i].numFilas < numFilasImagen)
{
      procs[i].filaIni=ultimaFilaAsignada - solapamiento;

      ultimaFilaAsignada=ultimaFilaAsignada+procs[i].numFilas;

      procs[i].filaFin=ultimaFilaAsignada-1+solapamiento;
}
```

Si la fila final a la que hemos llegado es la última fila de la matriz entonces no se añadirá solapamiento por debajo, y habremos terminado con el calculo de las submatrices. El código siguiente muestra este proceso:

```
else if(ultimaFilaAsignada+procs[i].numFilas==numFilasImagen)
{
      procs[i].filaIni=ultimaFilaAsignada - solapamiento;

      ultimaFilaAsignada=ultimaFilaAsignada+procs[i].numFilas-1;

      procs[i].filaFin=ultimaFilaAsignada;
}
```

7.1.5. Serialización de datos

Para aclarar este concepto, aquí se da una definición formal del concepto de serialización de datos. “La serialización de datos es el proceso en el que se toman objetos y se convierte su información de estado en un formato que permita su transporte o su almacenamiento. La acción inversa consiste en deshacer la serialización y dejar el objeto con la misma forma que tenía inicialmente”. Dicho de una manera más simple, la serialización consiste en transformar datos que tienen una forma, a otra forma, pero manteniendo su significado.

La necesidad de incluir en este proyecto la “serialización de datos” se pone de manifiesto cuando se intenta enviar al servicio web la información necesaria para calcular las diferencias entre las imágenes.

Desde el inicio del proyecto, se tratan las imágenes como una array bidimensional, en el que cada posición contiene un número de entre 0 y 255. Por otro lado, la función que calcula los cambios entre las dos imágenes de entrada, necesita como parámetros de entrada, entre otros, las dos imágenes.

Entre los tipos de datos que se le pueden pasar a los métodos de servicios web y los tipos de datos que devuelven se encuentran: los tipos de datos primitivos y arrays unidimensionales de tipos primitivos, entre otros. Como se puede observar, en ningún caso se contempla los arrays de dos dimensiones.

Por esta razón, se ha desarrollado una función que transforma el *array* de *double* bidimensional en un *array* unidimensional del tipo primitivo *String*. De esta manera se puede enviar sin ningún problema la imagen como un array unidimensional de tipo String. Cuando las imágenes llegan al método del servicio web, se deshace la serialización y se puede trabajar con la imagen en formato array bidimensional.

Las funciones de serialización y deserialización de los datos, en este caso, se llaman PasaAMatriz y PasaAString, y tienen la estructura que sigue:

```
pasaAString(double[][]): String[]

pasaAMatriz(String[]):double[][]
```

La primera función es la que llamamos de serialización, que transforma el array bidimensional de double que se le pasa como parámetro en un array unidimensional de String. El mecanismo que sigue esta función es que lee cada elemento de una fila del array de double y lo va metiendo en un String. Cuando termina de leer esa fila, guarda el string resultante en la fila correspondiente al array de String. Para que se vea más claro, se incluye el fragmento de código de la función:

```
String s="";

int numFilas=matriz.Length;

int numColumnas=matriz[1].Length;

String [] salida=new String[numFilas];


for(int i=0;i<numFilas;i++)
{
      for (int j=0;j<numColumnas;j++)
      {
            s = s + matriz[i][j]+" ";
      }
      salida[i] = s;

      s="";
}
return salida;
```

Gráficamente:

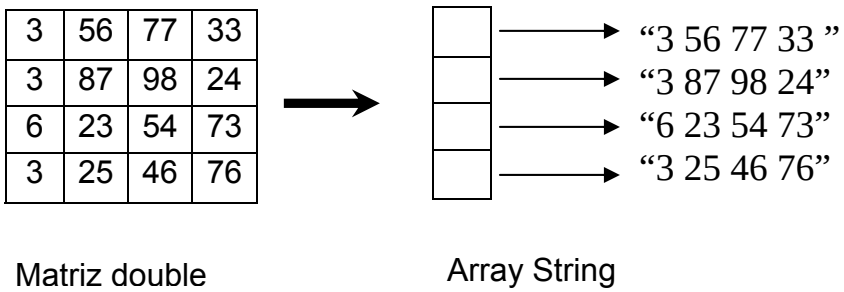


Figura 7.1.5.1

Y la segunda función es la que se viene denominando función de deserialización, que realiza la función inversa a la que se ha explicado. Recibe un array de String. Para cada fila, divide la cadena en tokens separados por espacios. Y guarda cada token en una posición de la matriz. Para una mejor comprensión, se puede revisar el código de la función:

```
int numFilas=lineas.Length;


double [][] matriz;//DECLARO LA MATRIZ

matriz=new double[numFilas][];//Inicializo la matriz (numero de
                                filas)

int numColum=0;

for (int i=0;i<numFilas;i
{
      string[] linea=lineas[i].Split(new Char[] {' '});//DIVIDO
                                                  LA FILA EN TOKENS
      numColum=linea.Length-1;

      matriz[i]=new double[numColum];     //inicializo la matriz
                                          (numero de columnas)
      for(int j=0;j<numColum;j++)
      {



            if (linea[j]!="")

                  matriz[i][j]=double.Parse(linea[j]);
      }
}
return matriz;
```

Gráficamente:

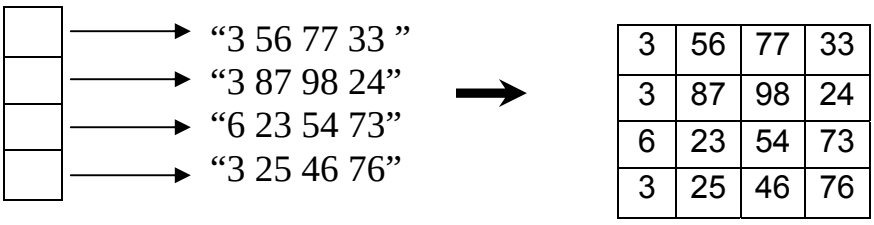


Figura 7.1.5.2

Estas funciones se encuentran integradas en la librería llamada: LibreriaSerializacion.dll, que están tanto en la aplicación del lado del cliente como en la del lado del servidor. Como es lógico, se tiene que encontrar en ambos sitios, ya que cuando la aplicación cliente envía datos al servicio web tiene que serializarlos. Cuando el servicio web los recibe los deserializa. En el momento en el que el servicio web culmina con su cometido, vuelve de nuevo a serializar los datos. El cliente los recibe y deserializa.

Se ha creado una librería con estas funciones con el fin de que puedan ser reutilizadas para cualquier otro proyecto.

7.1.6. Algoritmo de enfriamiento simulado

Antes de explicar cómo funciona el algoritmo de enfriamiento simulado hay que entender unas nociones básicas. Estado inicial: es el estado en el que comienza el problema. Espacio de estados: cada estado alcanzable a partir del estado inicial.

Este es un tipo de algoritmo de búsqueda por escalada. Estos algoritmos usan la información del estado actual del problema para hacer la búsqueda en el espacio de estados. A partir de la información del estado actual se intenta llegar a otro estado del proceso que sea mejor, y de esta forma nos vamos acercando a un estado solución óptimo. El estado solución puede que no sea el mejor de todos pero si uno lo suficientemente bueno. El estado suele reducirse a un número que nos indica como de lejos estamos de un estado solución del problema, cuanto más grande es el número más lejos estamos de la solución con lo que un estado mejor al actual sería aquel que tenga un número menor.

Los algoritmos de búsqueda por escalada tienen algunos problemas:

- Óptimo local: cuando se llega a un estado que es mejor que todos sus vecinos y descendientes, pero existe otro estado mejor en alguna parte.
- Meseta: todos los estados vecinos tienen el mismo valor, con lo cual no sabemos hacia donde ir.
- Puente: nos encontramos en un óptimo local y cerca tenemos un nodo solución mejor, sin embargo para llegar al óptimo global tendríamos que pasar por un nodo inaccesible lo que es imposible.

Este algoritmo se plantea como un problema de minimización de la función a optimizar (esta función es la que nos da el valor de cada estado). Existe una temperatura T que nos permite movernos a estados con valor peor al del estado actual, cuanto mayor es T más facilidad tendremos de hacer esto. Dicho de otra manera, al principio se pueden dar grandes saltos de un estado a otro (estados que están lejos entre sí) y según va disminuyendo T estos saltos se reducen (estados más cercanos). Al principio T es grande y según se avanza en el proceso va disminuyendo, al final es tan baja que el algoritmo se comporta como uno de escalada simple. Un algoritmo de escalada simple busca siempre un nodo mejor, y nunca tiene la opción de moverse a un estado peor que el actual.

Este enfriamiento permite que, si es lo suficientemente lento, lleguemos a un óptimo absoluto y no nos quedemos en un óptimo local del problema. En metalurgia, se sigue este proceso para templar metales y cristales calentándolos a una temperatura alta y luego enfriándolos gradualmente, para que el material se funda en un estado cristalino de energía baja.

El algoritmo tiene un esquema como el que sigue:

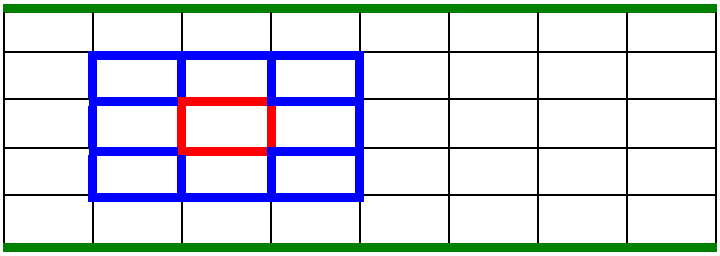
```
evaluar(INICIAL)

si INICIAL es solución entonces devolverlo y parar
si no
      ACTUAL := INICIAL
      MEJOR_HASTA_AHORA := ACTUAL
      T := TEMPERATURA_INICIAL
mientras haya operadores aplicables a ACTUAL y no se haya encontrado
solución   hacer
      seleccionar aleatoriamente operador no aplicado a ACTUAL
      {escoge movimiento aleatoriamente (no el mejor)}
      aplicar operador y obtener NUEVOESTADO
      calcular ΔE := evaluar(NUEVOESTADO) - evaluar(ACTUAL)
      si NUEVOESTADO es solución entonces devolverlo y parar
      si no
            si NUEVOESTADO mejor que ACTUAL {si mejora situación}
                  ACTUAL := NUEVOESTADO {se acepta el movimiento}
                  si NUEVOESTADO mejor que MEJOR_HASTA_AHORA
                  entonces MEJOR_HASTA_AHORA := NUEVOESTADO
            si no {si no mejora la situación, se acepta con prob<1}
                  calcular P':= e-ΔE/T
                  {probabilidad de pasar a un estado peor: se disminuye
                  exponencialmente con la "maldad" del movimiento, y
                  cuando la temperatura T baja}
                  obtener N {n° aleatorio en el intervalo [0,1]}
                  si N < P' {se acepta el movimiento}
                  entonces ACTUAL := NUEVOESTADO
      actualizar T de acuerdo con la planificación del enfriamiento
devolver MEJOR_HASTA_AHORA como solución
```

- **Solapamiento y ventana de ejecución**

Las partes en las que se dividen las imágenes para ser procesadas en procesadores remotos se componen de un número determinado de filas. A su vez, esas filas se componen de píxeles que determinarán su ancho.

El análisis de un píxel supone también el análisis de los que tiene alrededor. Esto es necesario para ver si ha habido cambios de una imagen con respecto a la otra.

El píxel que estamos analizando en un momento concreto más los que tiene alrededor (al menos ocho) componen lo que llamaremos una ventana de ejecución. La ventana mínima tendrá 3 píxeles de ancho y otros 3 de alto.



Supongamos que la matriz de la figura es la parte de la imagen que va a analizar un procesador. Se analizarán todos los píxeles, comenzando por la primera fila hasta la última y de izquierda a derecha. Si suponemos que el píxel

del borde rojo es el que estamos analizando en este momento, la ventana de ejecución serán los píxeles que están alrededor, en este caso, los de borde azul. Aquí la ventana de ejecución tiene un ancho y un largo de 3, pero este parámetro lo puede variar el usuario en las opciones del programa. Las ventanas de ejecución sólo pueden aumentar sus dimensiones de dos en dos, es decir, la siguiente ventana de ejecución más grande será de 5 píxeles y la siguiente de 7.

El problema viene en los límites superior e inferior de cada una de las partes en las que se ha dividido la imagen, delimitados en la figura por los bordes verdes.

¿Cómo se analizan los píxeles de la primera y de la última fila? Pues bien, además de su parte, analizará también la fila que hay por encima y la que hay por debajo. La solución se basa en que cada procesador recibe la imagen agrisada entera, no sólo su parte, y los límites (filas inicial y final) de la parte que le toca procesar. Así, tendrá a su disposición todas las filas por si las necesita.

Por ejemplo, si le toca analizar el rango de filas 4-9 (la 4 y la 9 incluidas), también analizará las filas 3 y 10. En este caso el solapamiento es de 1 píxel. Si el solapamiento fuese de 2 píxeles, analizaría también las filas 2 y 11. Éste es también un parámetro que elige el usuario en el menú opciones.

¿Por qué el solapamiento puede ser mayor que 1? El solapamiento es la cantidad de filas que va a analizar un procesador por encima y por debajo de la parte de la imagen que le toca procesar. Estas filas limítrofes serán pues, analizadas por 2 procesadores: al que corresponden esas filas y al que corresponden las filas de encima o debajo. Estas filas, al ser analizadas por 2 procesadores, se analizarán 2 veces, por tanto, con mayor probabilidad de descubrir diferencias.
Así, cuanto mayor sea el solapamiento, más tardará el procesamiento de las diferencias de las imágenes, pero mejor será el resultado.
La ventana de ejecución y el solapamiento están relacionados por la siguiente restricción:

Solapamiento es mayor o igual que **(Ventana** div **2)**

Esto es así por el problema que hemos explicado antes de los límites superior e inferior de cada una de las partes en las que se ha dividido la imagen, delimitados en la figura por los bordes verdes. Para que todos los píxeles sean analizados junto con los datos de los que los rodean (cuando éstos existen).

Por eso, el solapamiento mínimo (1) se deduce de la ventana mínima de ejecución (de tamaño 3).

- **Algoritmo para la Detección de Cambios en Imágenes**

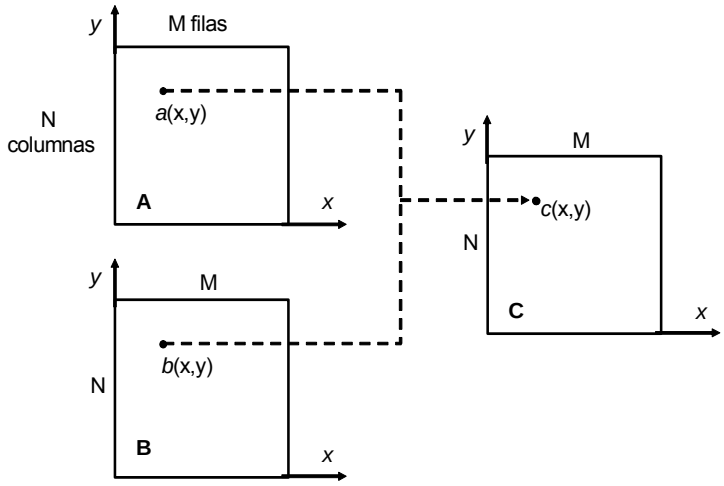
Entrada:
Dos imágenes A y B de dimensión variable filas *x* columnas, cada píxel de tamaño 1 byte en el rango [0..255].

Procedimiento:
1. Cálculo de la matriz inicial de cambios S
2. Cálculo de la matriz de probabilidades P
3. Proceso iterativo de enfriamiento simulado

Salida:
Una imagen R de las mismas dimensiones que las de entrada con valores binarios

1. Cálculo de la matriz inicial S

1. Dadas las imágenes de entrada A y B obtener una matriz cociente C tal que el píxel c(x,y) = a(x,y)/b(x,y). Para evitar posibles divisiones por cero sumar a la matriz B un cierto epsilon.

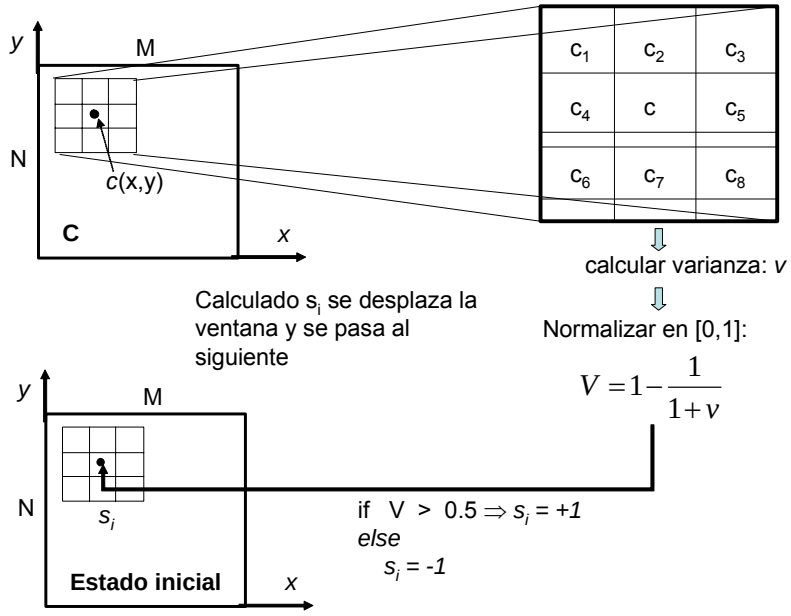


2. Definir una región de interés de tamaño 3 *x* 3 píxeles (por ejemplo) y recorrer toda la matriz C superponiendo esa ventana. Con los valores de C abarcados por dicha ventana calcular la media y la varianza de esa región.

3. Crear una nueva matriz S de las mismas dimensiones de las imágenes de entrada con todos sus valores a -1. Si la varianza supera un determinado valor cambiar el valor a +1.

4. Repetir los pasos 1 y 2 para la matriz cociente c(x,y) = b(x,y)/a(x,y). Si la varianza obtenida para cada píxel supera un determinado valor cambiar el valor correspondiente de S a +1.

Esta matriz S es el estado inicial del que parte el algoritmo de enfriamiento simulado (por tanto con valores −1 y +1,que representarán no cambio y cambio respectivamente)



2. Cálculo de la matriz de probabilidades P

1. Calcular la imagen diferencia en valor absoluto X = abs(A−B)

2. Obtener dos conjuntos que se denominan X_c y X_n, que se corresponden con valores de cambios y no cambios.

3. Obtención de X_c y X_n

Calcular la desviación típica σ de la matriz X. Ahora para cada valor de X en la posición (x,y),

if $X(x,y) > T^* \sigma \Rightarrow X_c(k) = X(x,y)$

else
$X_n(k) = X(x,y)$

4. Calcular las medias y varianzas para los datos X_c y X_n: mc, sc y mn, sn

5. Para cada valor de la matriz X(*x,y*) calcular una probabilidad como sigue

$$pXc(x,y) = \frac{1}{\sqrt{2\pi}sc}\exp\left[-0.5\left(\frac{X(x,y)-mc}{sc}\right)^2\right]$$

$$pXn(x,y) = \frac{1}{\sqrt{2\pi}sn}\exp\left[-0.5\left(\frac{X(x,y)-mn}{sn}\right)^2\right]$$

$$pX(x,y) = pXc(x,y) + pXn(x,y)$$

6. Normalizar los valores de $pX(x,y)$ al rango [0,1], como sigue: calcular el máximo (Max) y el mínimo (min) de todos los valores y obtener:

$$pX(x,y) = \frac{pX(x,y) - \min}{Max - \min}$$

El resultado final es una matriz bidimensional de probabilidades P en las localizaciones (x,y), a la que habrá que recurrir después durante el proceso de enfriamiento simulado.

3. Enfriamiento simulado

Inicializar la temperatura T(k), k = 0 (iteración)

do k = k + 1

do recorrer toda la matriz de estados inicial E píxel a píxel, considerando que su estado es s_i. Definir una región de vecindad de ese píxel de dimensión 3*x*3 *o* 5*x*5 de forma que los estados de esa región serán s_j. Tomando como referencia la matriz de probabilidades P y las posiciones *i* y *j*, calcular w_{ij} de la siguiente forma:

en la localización i
if *pX*(x,y) < 0.05 then p_i = -1 else p_i = +1

en la localización j
if *pX*(x,y) < 0.05 then p_j = -1 else p_j = +1

$w_{ij} = p_i\, p_j$

Calcular $Ea = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}\, s_i\, s_j$ (para una vecindad de 3x3 por ejemplo)

E*b* = - E*a*

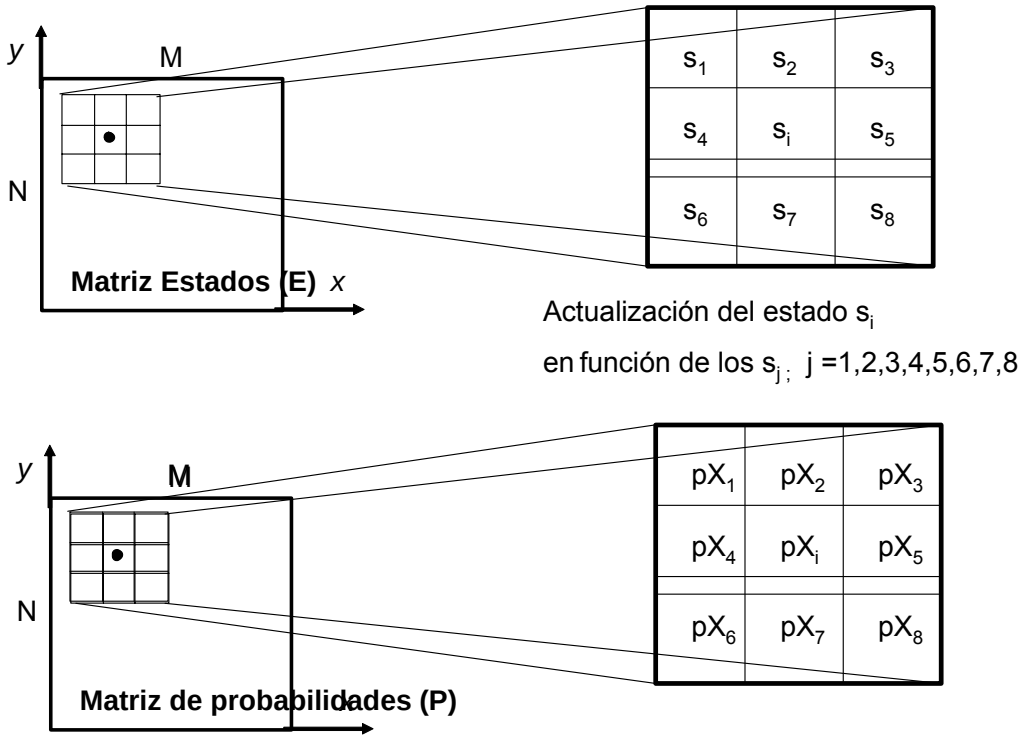
if E*b* < E*a*
then $s_i = -s_i$

else if $\exp-\left(\frac{E_b - E_a}{T(k)}\right) > random(0,1)$

then $s_i = -s_i$

until todos los nodos han sido visitados

until k = k_{max} (máximo numero de iteraciones o la energía se estabiliza)



7.1.7. Recomposición parcial de la imagen

Recordemos que a cada servidor le hemos indicado que calcule sólo una parte de las matrices, por lo tanto devolverá una parte de la solución. Cada servidor devuelve una matriz solución de tamaño igual a las imágenes iniciales, pero sólo son válidas las filas que han sido calculadas entre las filas inicial y final que hemos pasado a cada servidor como parámetro. Es necesario recoger todas estas matrices solución parciales devueltas por los servidores e insertar en la matriz solución final las filas válidas de cada uno.

Para hacer el cálculo de las matrices soluciones parciales hemos lanzado un hilo por cada servidor que ejecuta el algoritmo de enfriamiento simulado, de esta manera el proceso se ejecuta en los servidores de forma concurrente. Para ejecutar el algoritmo en un servidor se llama al método `LanzaHilos` que llama, a su vez, al método `calcular` del servicio web que contiene el algoritmo de enfriamiento simulado. Antes hemos pasado como parámetros las matrices de las imágenes así como las filas inicial y final que va a usar cada procesador, además de otros parámetros necesarios para el correcto funcionamiento del algoritmo.

```
AlgEnfriamientoSimulado.Class1 algorit = new
          AlgEnfriamientoSimulado.Class1(mA,mB,numFilasImagen,
          numColumnasImagen,fInicial,fFinal,vueltasIter,tamVentana);

algorit.calcular(sol);

soluciones[i]=sol;

RecomponeSolucion(Thread.CurrentThread);
```

Cuando termina el servicio web de hacer los cálculos nos devuelve la matriz con la solución parcial que ha calculado ese servidor. Lo último que hace el hilo que hemos lanzado para el cálculo de una parte de la matriz, es llamar al método `RecomponeSolucion` que se encargara de rellenar las filas que ha calculado en la matriz solución final. Este método recorre las filas, delimitadas por las filas inicial y final que ha usado ese servidor, de la matriz solución final y las va rellenando celda a celda con los valores de esas mismas filas en la matriz solución parcial devuelta por el servicio web. Si hay solapamiento entonces habrá filas repetidas en, al menos, dos matrices solución parciales; para rellenar estas filas se hace una función OR de las filas iguales en cada matriz solución parcial. La función OR lo que hace es que si una celda de la matriz es diferencia para una solución parcial y no es diferencia para otra, entonces nosotros si lo tomamos como diferencia. A continuación se ve un ejemplo en el que hay solapamiento en la fila 3, y cómo se realiza la OR en esa fila:

Parte de la solución calculada por el servidor1:

1	0	0	Fila1
0	1	1	Fila2
1	0	1	Fila3

Parte de la solución calculada por el servidor2:

0	0	1	Fila3
1	1	0	Fila4
0	0	0	Fila5

Solución final:

1	0	0	Fila1
0	1	1	Fila2
1	0	1	Fila3
1	1	0	Fila4
0	0	0	Fila5

Este código muestra como se hace este cálculo:

```
int numero=int.Parse(hilo.Name);

double[][] s = soluciones[numero];

int fIni = procs[numero].filaIni;

int fFin = procs[numero].filaFin;



//recorremos las filas

mut.WaitOne();
//aquí solo puede entrar un hilo cada vez

for (int j=fIni;j<=fFin;j++)
{
      //recorremos las columnas
      for (int k=0;k<numColumnasImagen;k++)
      {
            if ((s[j][k]==1)||(solucion[j][k]==1))
            {
                  solucion[j][k]= 1;

                  solucionS[j][k]= "1";
            }
            else
            {
                  solucion[j][k]= 0;

                  solucionS[j][k]= "0";
            }
      }
}
if ( (fFin == (fIni+ solapamiento*2 -1)) || (fFin==fIni))  //en
                  este caso, este procesador ha analizado 0 filas

      formulario.Escribe("La llamada a " +
           procs[numero].direccion + " ha terminado. Filas 0");
```

```
        else
              formulario.Escribe("La llamada a " +
                        procs[numero].direccion + " ha terminado. Filas "
                        + fIni + " - " + fFin);

        MostrarPartesImagen();  //muestro aquí la parte de la imagen
                          //correspondiente para que no haya que esperar
                           //a los hilos por orden para visualizar las
partes

        mut.ReleaseMutex();
```

Este código es el del método `RecomponeSolucion`. Se puede observar que usamos un semáforo (mutex) para que dos hilos que están recomponiendo su parte de la solución no puedan modificar a la vez la matriz solución final y así evitar posibles errores al rellenar las filas.

El rellenar las filas de la solución final dentro de cada hilo se hace con la finalidad de mostrar la parte que ha resuelto cada hilo al terminar, sin esperar a los demás. Los hilos no terminan en un orden predeterminado, por lo que si rellenásemos las filas de la imagen al terminar todos los hilos, quizá un hilo que ha acabado tuviera que esperar hasta que acabase los demás para poder rellenar sus filas, es decir, se irían rellenando las filas de cada servidor en un orden establecido sin tener en cuenta el orden real en el que terminan. De esta última manera habría fragmentos de imagen calculados que no se mostrarían hasta que no llegase su turno.

Puede suceder que mientras un servidor esta ejecutando el algoritmo falle y se quede no operativo en ese momento. En caso de que ocurra esto, esa parte de la imagen asingada a este procesador no puede quedarse sin calcular, ya que sino nuestra aplicación no sería robusta. Para impedir que parte de la imagen se quede sin calcular, se reasignará a la máquina cliente.

La forma de saber si un servidor sigue operativo, es asignando al hilo que se lanze contra el un tiempo que creemos prudecial. Si finalizado este tiempo no se ha obtenido la solución de este servidor, lo consideraremos no disponible y pasaremos su carga de trabajo a la máquina cliente, para asegurarnos de que el algoritmo termina y de que obtenemos una solución. Este método hará aumentar el tiempo en que obtenemos la solución, pero de esta manera se garantiza la obtención de una solución en cualquiera de los casos.

Para hacer la estimación de este tiempo, se tiene en cuenta el tamaño de la imagen y el tiempo que ha tardado en responder cada procesador. Se hace una comparación con los mismos datos tomados en un ejemplo real.

7.2. **Resultados**

7.2.1. Imágenes resultado

La imagen que se obtiene como resultado es una imagen en blanco y negro que muestra las diferencias entre las imágenes iniciales. Las diferencias se muestran en blanco sobre un fondo negro. La zona negra significa que no hay diferencias entre las imágenes.

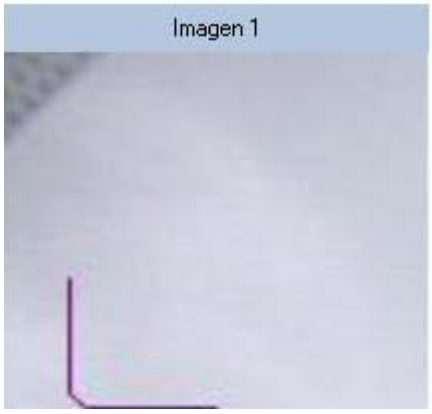


Figura 7.2.1

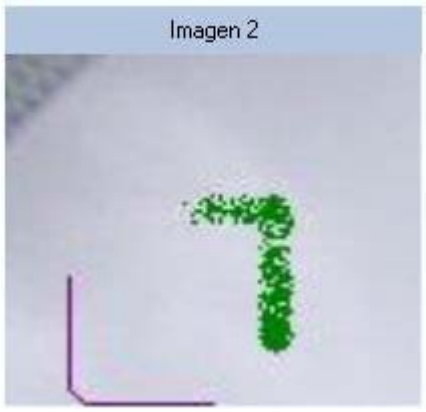


Figura 7.2.1

Con las imágenes de arriba (Figuras 7.2.1 y 7.2.2) se obtendría la siguiente solución:

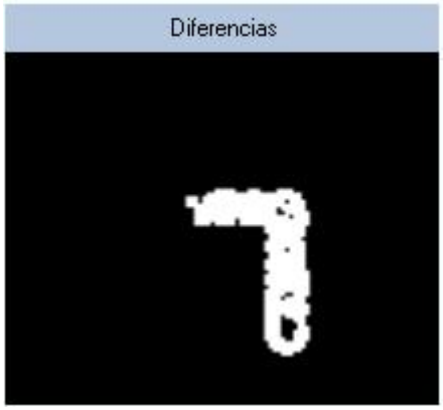


Figura 7.2.3

La imagen solución se guarda en el directorio donde se encuentran las imágenes de entrada con el nombre de “salida.bmp”. Las versiones en blanco y negro intermedias de las imágenes iniciales se guardan también en la carpeta con extensión .bmp (fotoByNA.bmp y fotoByNB.bmp), se usa ese formato (bmp) para no perder calidad en las imágenes.

Durante el proceso se ve como la imagen solución se va rellenando por partes. Cada parte que se va rellenando es la que ha calculado uno de los

servidores. Las líneas verdes indican una delimitación entre las zonas que va a calcular cada servidor, sin tener en cuenta el solapamiento (Figura 7.2.4).

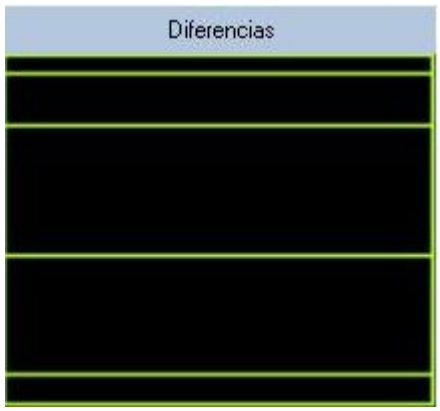


Figura 7.2.4

En la imagen que aparece a continuación (Figura 7.1.8.5) se puede comprobar como se van mostrando las filas de la imagen solución según van terminando los hilos de los distintos servidores, sin mostrarse la imagen entera al finalizar los cálculos:

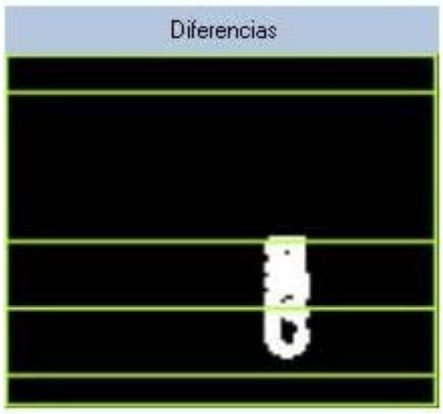


Figura 7.2.5

Mostraremos ahora ejemplos de las imágenes que devolvería como resultado la aplicación.

Imagen 1

Imagen 2

Imagen salida

Figura 7.2.6

Imagen 1

Imagen 2

Imagen salida

Figura 7.2.7

Imagen 1

Imagen 2

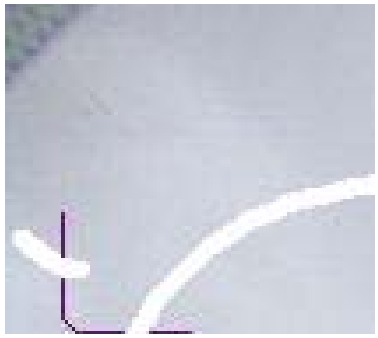


Imagen salida

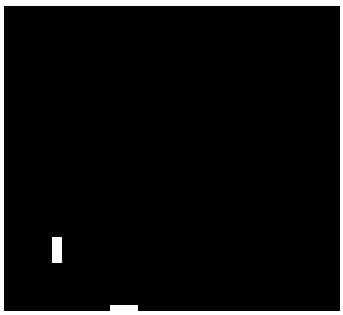


Figura 7.2.8

Las fotos de las figuras 7.2.6 y 7.2.7 son idénticas. Sólo se ha retocado un pequeño matiz. Como resultado se obtiene una imagen mostrando en blanco la zona que ha cambiado de una foto a otra.

Es curioso el resultado de la figura 7.2.8. Como se puede observar la diferencia entre las dos imágenes de entrada es que en la imagen 2 aparecen dos líneas blancas. Sin embargo, en la imagen resultado sólo aparecen como diferencias los trazos de la línea rosa que coinciden con las líneas blancas.

Este comportamiento se debe al algoritmo de enfriamiento simulado. Cualquier algoritmo que detecte cambios en las imágenes que hagan la comparación simple píxel a píxel, mostrarían como diferencia las dos líneas blancas enteras. Pero el algoritmo que se ha utilizado en este proyecto es más eficiente. Este algoritmo se da cuenta de que los dos colores (el blanco de la línea y el del fondo de la imagen) son tan parecidos que obvia la diferencia.

En un ejemplo real, supongamos que tenemos dos imágenes sacadas desde el mismo sitio, en diferentes horas del día. En una de ellas hay mayor nivel de luminosidad y existen zonas donde hay sombras, y otra imagen tomada cuando se esta poniendo el sol. Este algoritmo sería capaz de darse cuenta de que se trata de la misma imagen, aunque una tenga un factor de luminosidad diferente a la otra. Y también obviaría las sombras producidas en la imagen de forma similar a como ha hecho en la figura 7.2.8.

7.2.2. Evaluación de rendimiento

Ahora se tratará de mostrar los resultados en cuanto a tiempo se refiere. A continuación se muestra el material utilizado para realizar las pruebas

- Imágenes utilizadas

Las imágenes que se han utilizado para recoger tiempos de ejecución tienen un tamaño de **885 X 543.** Aunque no son imágenes excesivamente grandes, bastan para plasmar los objetivos que se perseguían con la realización del proyecto.

Imagen 1

Imagen 2

- Ordenadores utilizados

Primero, se debe hacer una distinción entre los equipos que se han utilizado para ejecutar la aplicación, que llamaremos de ahora en adelante máquinas cliente, y los equipos que contienen el servicio web, que recibirán el nombre de servidores.

Para llevar a cabo las pruebas de ejecución del programa y recopilar los tiempos para realizar el estudio, se han utilizado dos máquinas cliente. Cada una tiene unas características diferentes que se muestran a continuación:

- "Pc1": Pentium III 667MHz 256 MB de RAM //IBM
- "Pc2": Pentium IV 1700MHz y 256 MB de RAM //P23

Como servidores se han utilizado tres máquinas, en las que se ha instalado el mismo servicio web. Un detalle importante a destacar es que aunque son tres máquinas, se utilizan 4 procesadores, ya que una de ella cuenta con dos procesadores.

- "GOLS": Pentium II 400MHz y 256 Mb de RAM.
- "GOLLS2": xeon 2´4 GHz y 2 GB de RAM (Esta máquina es la que contiene dos procesadores de las mismas características).
- "ServidorPortatil": Procesador Intel ® Pentium (R) M 1,6 GHz y 1024 MB de RAM.

Los motivos por los que se han elegido estas máquinas para hacer las pruebas son sencillos. Por un lado, se han utilizado porque son ordenadores a los que tenemos acceso. Por otro lado, porque son máquinas que tienen diferentes características con las que se pueden obtener datos importantes en este estudio.

Se debe hacer énfasis en lo siguiente:

- Los procesadores de "GOLLS2" son muy potentes.
- El servidor GOLS es lento, incluso más, que las dos máquinas clientes utilizadas.

A continuación se enumerarán las pruebas realizadas, comentando la máquina cliente utilizada, los servidores implicados y los tiempos obtenidos. Todas estas pruebas (excepto la 7 y 8) se han realizado sin modificar la carga que el programa estima que tiene que calcular cada procesador.

Prueba 1

Máquina cliente: PC1
Servidores: Localhost (procesador de la máquina cliente)

Tiempo invertido: 463 segundos.

Prueba 2

Máquina cliente: PC1
Servidores: Localhost (procesador de la máquina cliente),
Golls2
Golls2
Gols
ServidorPortatil

Tiempo invertido: 270 segundos.

Prueba 3

Máquina cliente: PC2
Servidores: Localhost (procesador de la máquina cliente)

Tiempo invertido: 336 segundos.

Prueba 4

Máquina cliente: PC2
Servidores: Localhost (procesador de la máquina cliente)
Golls2

Tiempo invertido: 263 segundos.

Prueba 5

Máquina cliente: PC2
Servidores: Localhost (procesador de la máquina cliente)
Golls2
Golls2

Tiempo invertido: 198 segundos.

Prueba 6

Máquina cliente: PC2
Servidores: Localhost (procesador de la máquina cliente)
Golls2
Golls2
Gols
ServidorPortatil

Tiempo invertido: 172 segundos.

Prueba 7

Máquina cliente: PC2
Servidores: Localhost (procesador de la máquina cliente)
Golls2
Gols2
Gols
Gols
ServidorPortatil

Tiempo invertido: 252 segundos.

Prueba 8

Máquina cliente: PC2
Servidores: Localhost (procesador de la máquina cliente)
Golls2
Golls2
Gols
ServidorPortatil

Tiempo invertido: 260 segundos.

Es importante mencionar la relación que existe en el tiempo que los procesadores tardan en devolver la solución. Para las pruebas 1, 2, 3, 4, 5 y 6, se podría afirmar que el tiempo que pasa desde que termina el primer procesador implicado en el cálculo, hasta que lo hace el último, hay un intervalo que ronda, a lo sumo, los 20 segundos. Este valor es una media que se ha calculado a partir de todas las pruebas realizadas.

Esta “sincronización” se debe a que el programa realiza una pequeña estimación sobre la rapidez de cada uno de los procesadores. Y en base a este dato, el programa reparte la imagen de forma equilibra teniendo en cuenta la potencia de cada uno de los procesadores. Por ejemplo, si vamos a utilizar dos procesadores, y uno es el doble de rápido que el otro, entonces el programa le asignará el doble de trabajo al más rápido (balance de carga).

Para resumir, se muestra un cuadro que recoge los datos de estas pruebas:

SERVIDORES / MAQUINAS CLIENTE	LOCALHOST	GOLLS2	GOLLS2	GOLS	SERV. PORT	TIEMPOS
PC1	X					463
	X	X	X	X	X	270
PC2	X					336
	X	X				263
	X	X	X			198
	X	X	X	X	X	172
	X	X	X	X X	X	252
	X	X	X	X	X	260

- Carga desequilibrada

Cada línea de la tabla se corresponde con una prueba.

En la primera prueba se utiliza sólo el procesador de la máquina cliente PC1. Como se puede observar, tarda 463 segundos. Cuando se utilizan todos los procesadores con la carga equilibrada (prueba 2) se tarda sólo 270 segundos. Con lo que se gana algo más de tres minutos de tiempo, es decir, se ha logrado conseguir la misma solución un 40% menos de tiempo.

Hablemos ahora en términos de mejora de rendimiento o *speedup (*tiempo de ejecución de un programa en un sistema uniprocesador respecto a la medida de tiempo de ejecución del programa en un sistema multiprocesador). El speedup se calcula como:

$$speedup = \frac{t_1}{t_j}$$

Considerando que $t_1$es el tiempo que tarda en ejecutarse la aplicación en el procesador de la máquina cliente PC1, y que t_j es el tiempo que tarda en ejecutarse la aplicación utilizando todos los procesadores que tenemos disponibles (prueba 2), la mejora de rendimiento en este caso es de 1,7.

A partir de la prueba 3, se utiliza la máquina cliente PC2.

Para la prueba 3, se ha utilizado sólo el procesador de la máquina PC2, que ha tardado 336 segundos. Este tiempo se tomará como referencia para calcular el speedup del resto de las pruebas, ya que se corresponde con el t_1 de la fórmula anterior.

En la prueba 4, utilizando un procesador de Golls2 y la máquina cliente, se ha conseguido un tiempo de 263 segundos, 73 segundos menos que si utilizásemos solo el procesador de la máquina cliente.

En la prueba 5, se han ganado 138 segundos con respecto de la prueba número 3.

En la prueba 6 se han ganado sólo 164 segundos.

Como se puede observar, a medida que se introducen más servidores, disminuye el tiempo en que obtenemos solución. Pero también hay que destacar que cuando se incluyen servidores más lentos, la mejora obtenida es menos cuantiosa.

Ahora estudiaremos un poco más a fondo los resultados obtenidos en las pruebas 7 y 8.

En la prueba 7, se han utilizado todos los procesadores, con la salvedad de que al servidor más lento (Gols), que es más lento incluso que la máquina cliente, se le han asignado dos hilos. El tiempo que ha disminuido la ejecución con respecto a utilizar solo la máquina cliente es de 84 segundos. Pese a que sí se ha mejorado tiempo en comparación con la prueba 3, se ha empeorado en comparación con la prueba 6.

En la prueba 8, se han utilizado todos los procesadores que tenemos a nuestra disposición, pero se ha modificado el reparto de carga que ha estimado el programa. La modificación ha consistido en a que los procesadores más lentos le hemos asignado el trabajo que le correspondería al más rápido y viceversa.

Como se puede observar el la tabla, al hacer este cambio, el tiempo que se tarda en tener la imagen con las diferencias es prácticamente igual que el que se tardaría si solo se utilizara Golls2 con la carga equilibrada.

La siguiente tabla muestra el speedup correspondiente a las pruebas 3, 4, 5, 6, 7 y 8.

	Localhost	Golls2	Golls2	Gols	ServPortatil	Speedup
Prueba 3	X					1
Prueba 4	X	X				1,3
Prueba 5	X	X	X			1,7
Prueba 6	X	X	X	X	X	2
Prueba 7	X	X	X	X X	X	1,3
Prueba 8 *	X	X	X	X	X	1,3

* Carga desequilibrada.

Tras realizar las pruebas, se han llegado a las siguientes conclusiones:

- Si se modifica la cantidad de filas que el programa asigna a cada procesador, se pierde la "sincronización" de las respuestas de los servidores.

- Si se pierde la sincronización en las respuestas, se incrementa el tiempo que tarda el programa en proporcionar la imagen solución.

- Si se envía más trabajo a los servidores que son más lentos, se obtendrá un speedup menor que si se mandara al más rápido.

- En resumen, para obtener el tiempo óptimo de respuesta, hay que hacer el reparto de la imagen a calcular, en proporción a la potencia de cada servidor. En este caso, conseguiremos el tiempo mínimo de respuesta. Este reparto proporcional, es el que calcula el programa por defecto, gracias a las estimaciones que realiza.

- También hay que destacar que las imágenes que se han elegido eran iguales salvo una pequeña modificación. Por tanto las condiciones luminosas de las imágenes eran las mismas. Esto significa que el algoritmo del enfriamiento simulado ha tenido que dar menos vueltas, y por tanto ha mostrado antes las soluciones. Además, las imágenes que hemos utilizado son muy pequeñas. Por eso, la mejora de rendimiento que se muestra en la tabla es muy bajo. Si las imágnes tuvieran diferente factor luminoso, o fueran más grandes o se hubiera aumentado el número de vueltas que da el algoritmo de enfriamiento simulado, la carga a procesar sería mayor y se hubiera obtenido mayor mejora del rendimiento. De este modo, la mayor parte del tiempo invertido es debido al tiempo que se tardan en comunicar los procesadores.
  Por eso decimos, que en el caso de que el cálculo fuera tan grande que el tiempo invertido en las comunicaciones se considerara despreciable, el speedup que conseguiríamos sería mayor y sería lineal.

En conclusión, con el sistema paralelo Web/Grid desarrollado, hemos conseguido optimizar, de forma considerable, el tiempo en que el algoritmo de enfriamiento simulado compara dos imágenes. Pero que esta optimización depende de la potencia de los procesadores involucrados en el cálculo, y en mayor medida depende del balance de la carga.

7.3. **Manual del usuario**

Este documento trata de ser una guía útil en la que se aprenda de forma rápida y fácil el uso y extensiones del programa.

7.3.1 ¿Qué es?

- Introducción

Este programa es una aplicación que compara dos imágenes del mismo objetivo focal y devuelve en una tercera los cambios encontrados.

- Características de la aplicación

La importancia de este programa radica en:

  - el uso eficaz de recursos
  - su rapidez de cálculo
  - su fácil manejo
  - su interfaz amistosa para los usuarios
  - su gran capacidad de configuración de opciones

- Interfaz

Como se ha comentado anteriormente, la interfaz es muy sencilla, útil y cómoda, a la vez que amigable. Cualquier usuario que utilice la aplicación no tendrá problemas, ya que es muy intuitiva.

A continuación se muestra una captura de imagen de la ventana principal del programa:

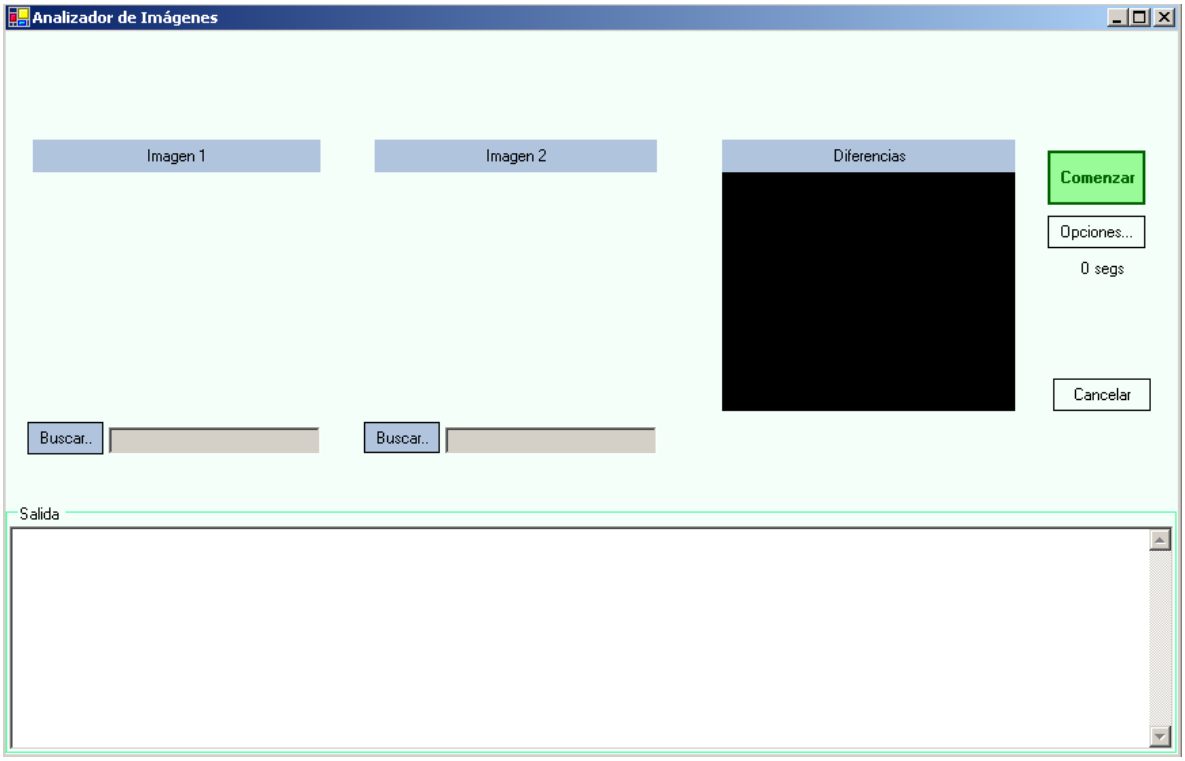


Figura 7.3.1

7.3.2 Entorno requerido

Lo que se necesita para ejecutar el programa:

- Windows 98, 2000, XP ó superior.
- Para ejecutar una aplicación escrita en C# en una plataforma que no tenga instalado el paquete mínimo de .NET se debe incrustar el runtime o motor de ejecución .NET en el ejecutable.

7.3.3 Instalación

La aplicación podría ser perfectamente instalada por un usuario que no tuviera nociones avanzadas a cerca del mundo de la informática. Basta con descargar el archivo (véase el apartado 7.4).

El archivo que se descarga se llama Programa.rar, como este que se muestra:


Programa.rar

Figura 7.3.2

El contenido de esta carpeta se extraerá a aquella dirección donde se quiera instalar, por ejemplo, en C. Para descomprimir este archivo es necesario tener un descompresor de archivos .rar.

Cuando se extrae el contenido de este archivo, se obtiene una carpeta llamada Programa con los siguientes archivos:

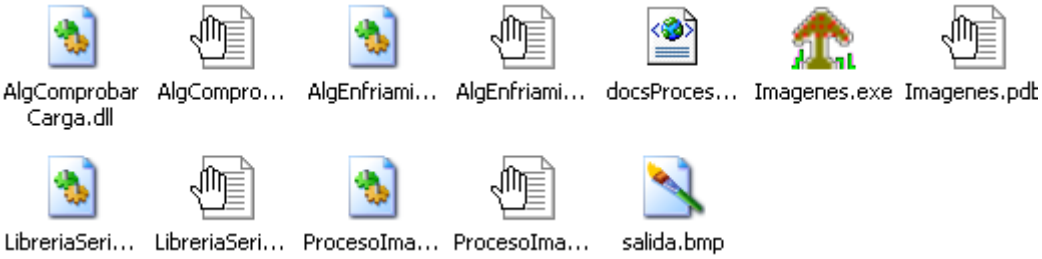


Figura 7.3.3

Estos son todos los pasos necesarios para instalar el programa.

7.3.4 Comenzando y finalizando

- Comenzando

Para comenzar a utilizar el programa, hacer doble clic sobre el archivo Imágenes.exe.

Figura 7.3.4

- Finalizando

Para terminar la ejecución de la aplicación, simplemente se cierra la ventana.

7.3.5 Diálogos y vistas

Este programa, debido a su sencillez, muestra muy pocas opciones al usuario.

Básicamente, sólo existen tres formularios con los que el usuario puede interactuar. Uno de ellos es el que aparece al iniciar la ejecución:

- ***Formulario nº1***

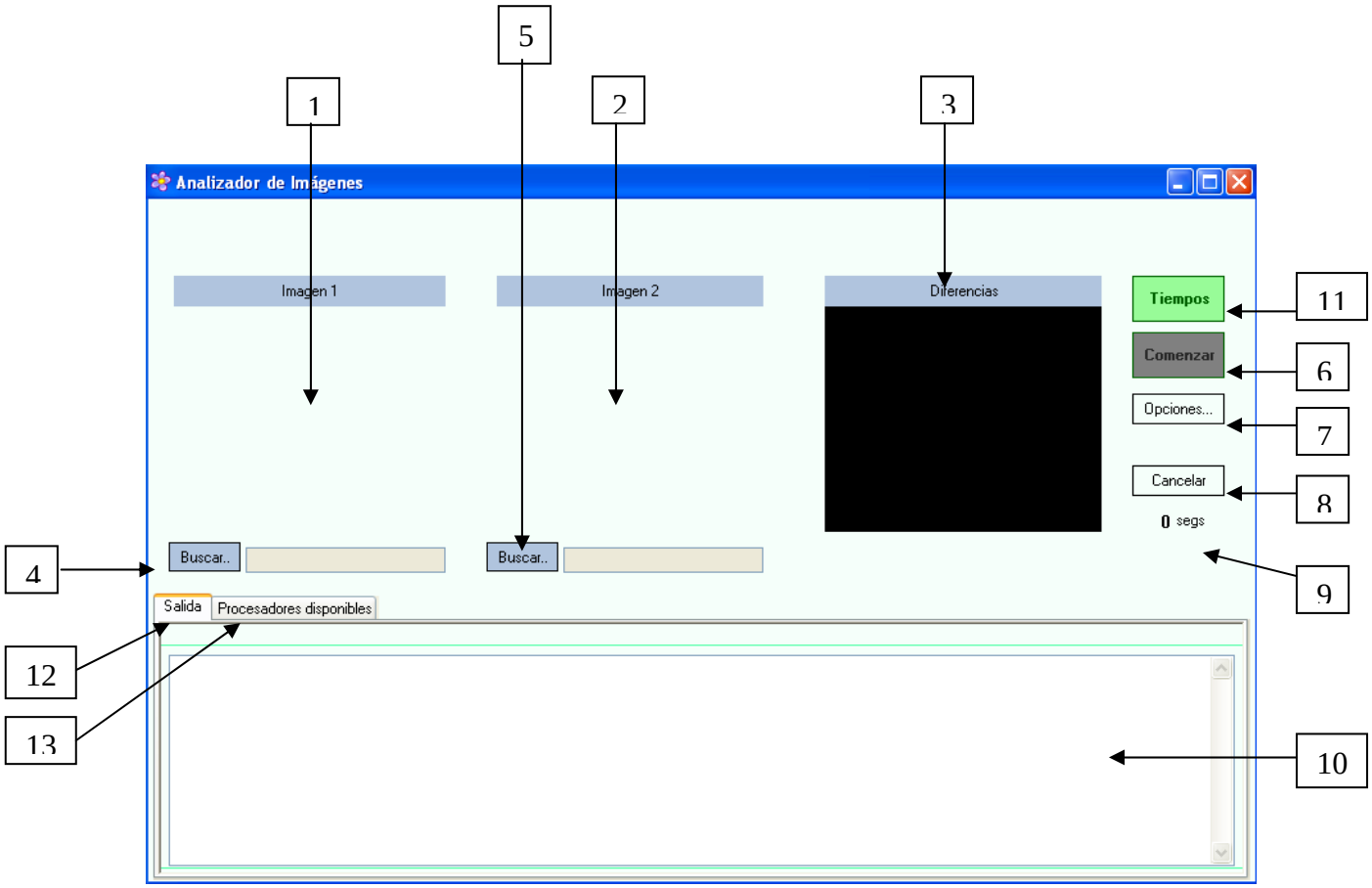


Figura 7.3.5

En este formulario el usuario debe introducir las imágenes que desee comparar.

Gracias a los botones Buscar, se podrá navegar a través de las carpetas del sistema operativo para seleccionar las imágenes a comparar (números 4 y 5 en la figura).

Una vez seleccionadas las imágenes, aparecerán en las posiciones que muestran los números 1 y 2 de la figura.

La imagen fruto de la comparativa se mostrará en la posición que muestra el número 3.

El botón Comenzar (número 6 en la figura) inicia la ejecución de la aplicación.

El botón opciones (número 7 de la figura), despliega una ventana en la que se pueden modificar determinados atributos que afectan al cálculo que realiza el algoritmo. Este formulario se explica más adelante (figura 7.3.6).

El botón Cancelar (número 8 de la figura), cancela la ejecución en curso, y deja el sistema en el estado inicial, esto es, de la misma manera a como se encuentra cuando se arranca la aplicación.

La referencia 9 a la figura 7.3.5, es un reloj que se inicia cuando se pulsa el botón Comenzar y que se detiene cuando ya se ha calculado la imagen solución final. . De esta manera se puede cronometrar el tiempo total que tarda la aplicación en completar todo su cálculo.

El botón de Tiempos (número 11) muestra en la caja de texto, número 10 de la figura, el tiempo que han tardado los procesadores en realizar un pequeño cálculo. Y en la pestaña de Procesadores disponibles, (número 12 de la figura), mostraría el nombre de los procesadores disponibles, en ese momento.

A través del cuadro de texto del formulario (referencia 10 de la figura 7.3.5), se pueden observar los diferentes pasos que va realizando el algoritmo. Todos los mensajes que aparecen en este cuadro se explican en el siguiente punto de esta documentación.

- ***Formulario nº 2***

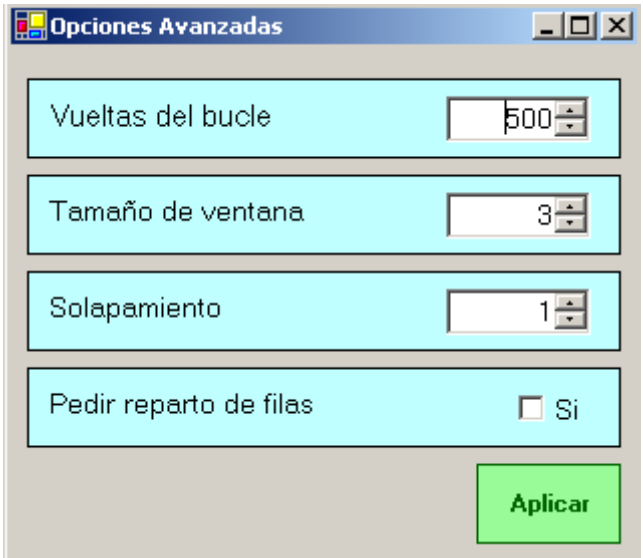


Figura 7.3.6

Este formulario permite al usuario modificar determinados valores que tienen que ver directamente con el algoritmo de detección de diferencias.

Si aumentamos los valores de Tamaño Ventana y Solapamiento, haremos que el cálculo sea más preciso. Por defecto, el algoritmo tomará los valores de 3 y 1 respectivamente. Al mismo tiempo que aumenta la eficiencia del algoritmo, aumenta también el tiempo que tarda en obtener una respuesta.

Las vueltas del bucle, consiguen modificar la imagen que se pasa por entrada, con el fin de eliminar brillos o excesiva luminosidad. Como no queda muy claro, se explica un ejemplo a continuación.

Imaginemos que tenemos dos imágenes de entrada. Son exactamente iguales, ya que se han sacado desde un mismo objetivo focal, y que no se ha variado nada de la imagen. Sin embargo, se han obtenido a diferentes horas del día, por lo que una de ellas tendrá mayor nivel de luminosidad. Las imágenes son idénticas, pero cualquier algoritmo que buscara las diferencias comparando píxel a píxel encontraría que todos los píxeles son diferentes. Luego como resultado mostraría que toda la imagen es diferente.

Con la variable Vueltas del bucle, por tanto, podemos suavizar la imagen de entrada, y así el algoritmo respondería que no hay diferencias entre las imágenes

Pero con este número hay que tener mucho cuidado, porque con un número muy elevado de Vueltas del bucle se podría distorsionar la imagen demasiado.

Y, y por último, si marcamos la opción “Pedir reparto filas”, cuando el algoritmo va a comenzar el cálculo, permite al usuario que modifique las filas a tratar por cada procesador disponible.

- ***Formulario nº 3***

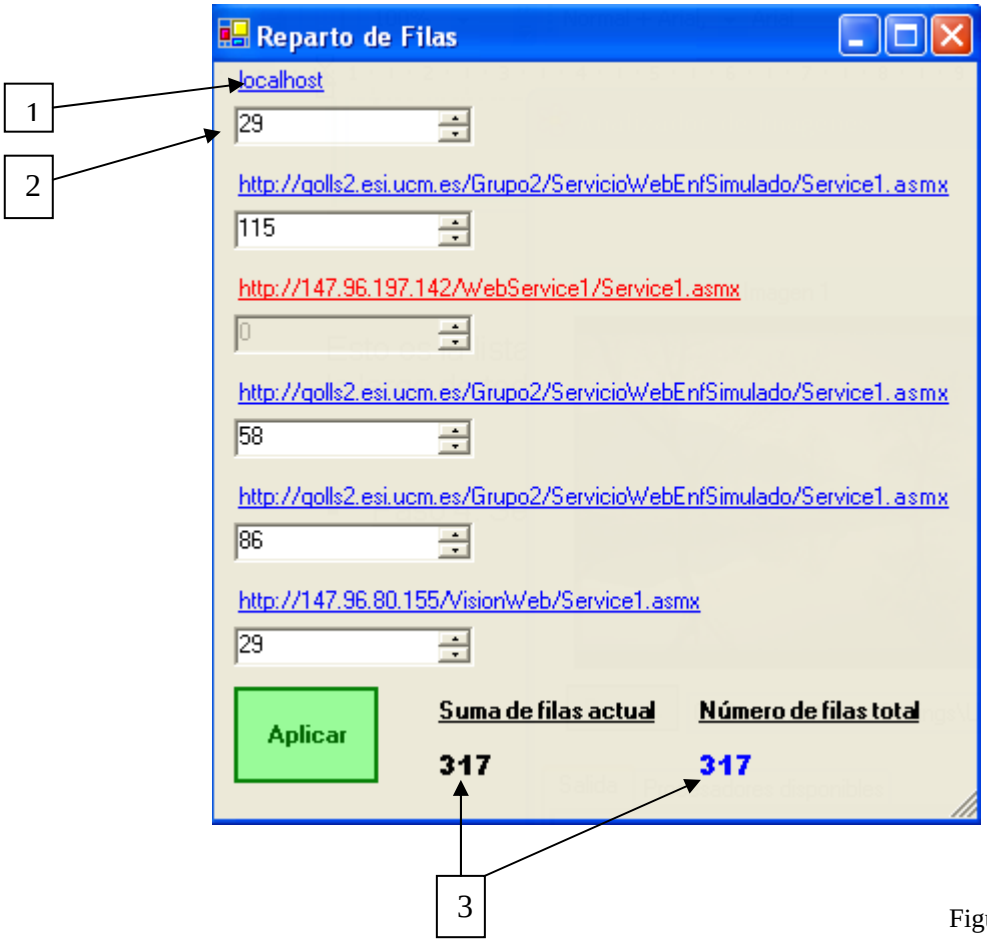


Figura 7.3.7

Este es el formulario que aparece cuando se solicita explícitamente “Pedir reparto filas” en el formulario nº 2.

El número 1 de la figura, indica el nombre del procesador, y el número 2, el número de filas que la aplicación ha calculado que debe procesar cada uno. Este número se puede incrementar y decrementar.

Cuando se modifican los números de filas, se modifica el número correspondiente a “Suma de filas actual”.

Lo correcto es que, después de modificar los valores, tanto “Suma de filas actual” como “Numero de filas total”, tengan el mismo número. En caso contrario, al pulsar el botón “Aplicar“ para que se apliquen estos cambios y se continúe con la ejecución del programa, aparecerá el mensaje de de la figura 7.3.10 que se muestra en el siguiente apartado.

7.3.6 Viendo mensajes

En este apartado se mostrarán los mensajes que pueden aparecer cuando se trabaja con la aplicación.

- Si no se ha seleccionado aún alguna de las imágenes a comparar, aparecerá una ventana como la siguiente:

Figura 7.3.8

- Si se seleccionan imágenes de las mismas características (mismo tamaño), se mostrará la frame que muestra la figura 7.3.8.

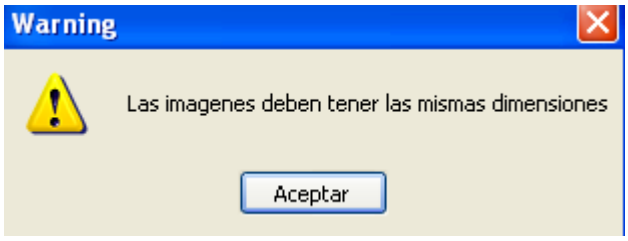


Figura 7.3.9

- El siguiente formulario aparecerá en caso de que al modificar el número de filas que debe calcular cada procesador, la suma actual no coincida con la suma total de filas (número de filas que tiene la imagen)

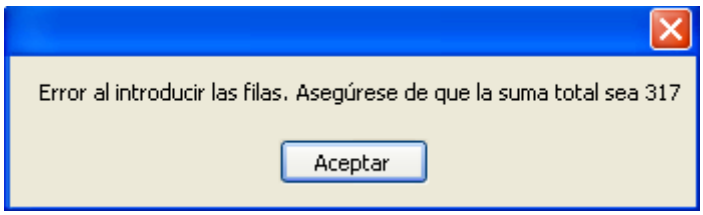


Figura 7.3.10

7.3.7 Ejemplo de uso

Para comprender mejor el funcionamiento de la aplicación, pondremos un ejemplo del programa en funcionamiento y los pasos a dar.

Para ello se ejecuta la aplicación, como se mostró en el apartado de Comenzando que se comentó anteriormente.

Y en la pantalla aparecerá un formulario como el siguiente:

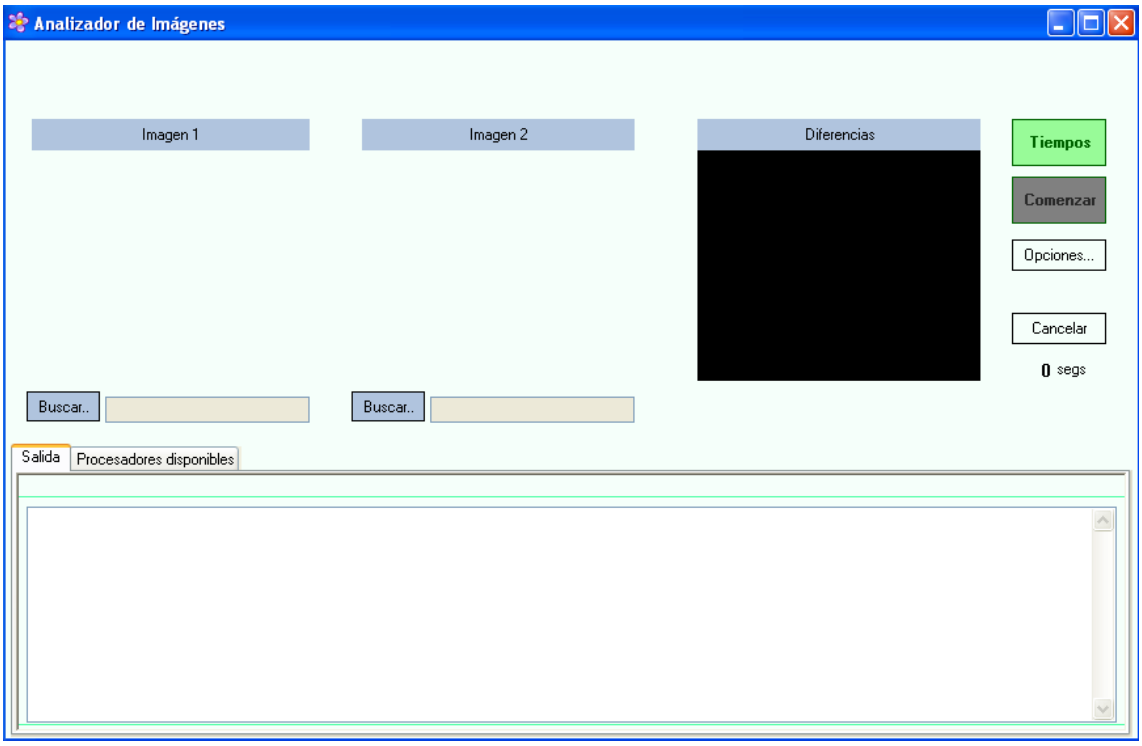


Figura 7.3.11

- Paso 1. Lo primero que se debe hacer es seleccionar las imágenes a comparar (si no se hace de esta manera, aparecerá el mensaje de de la figura 7.3.7). Para seleccionar la primera imagen se pulsa el botón Buscar y se abrirá una ventana como la que muestra la figura 7.3.10. Gracias a esta ventana se podrá navegar a través del sistema de archivos hasta encontrar la imagen deseada. Una vez que se tenga, se pulsa el botón abrir y la imagen aparecerá en el formulario, como aparece en la figura 7.3.11.

La segunda imagen se seleccionará de manera análoga.

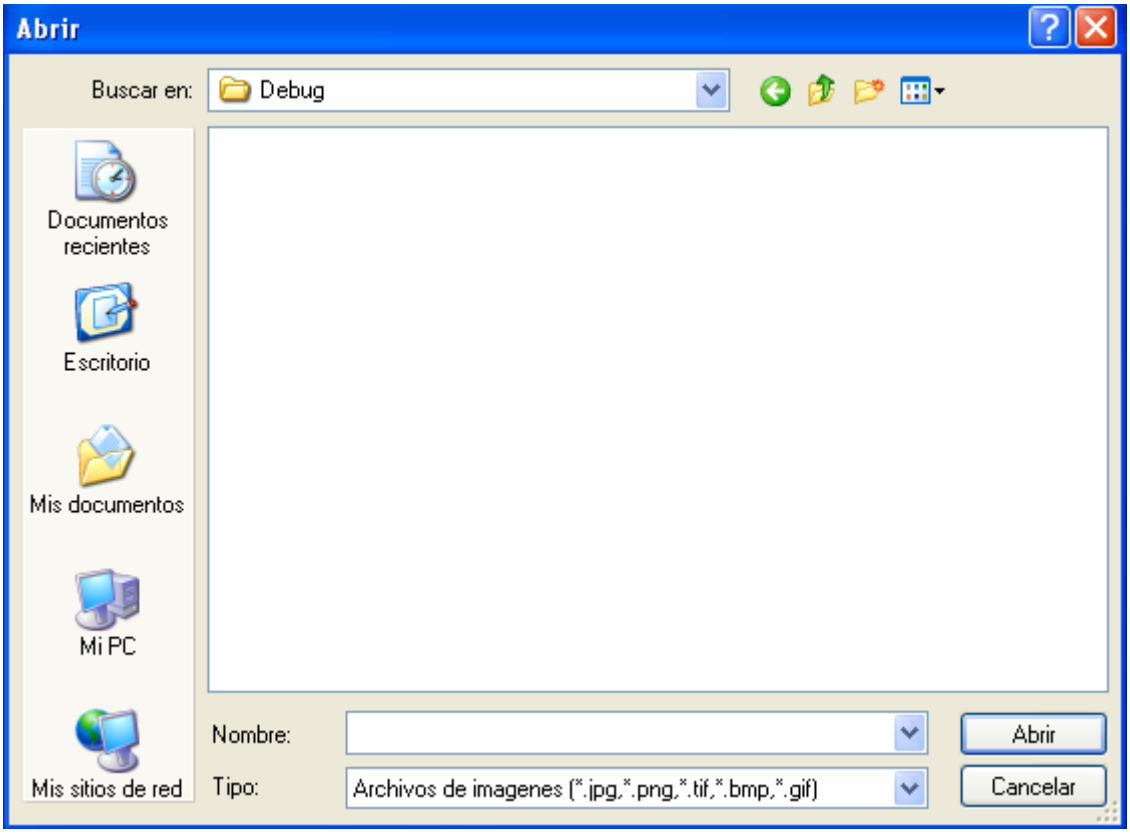


Figura 7.3.12

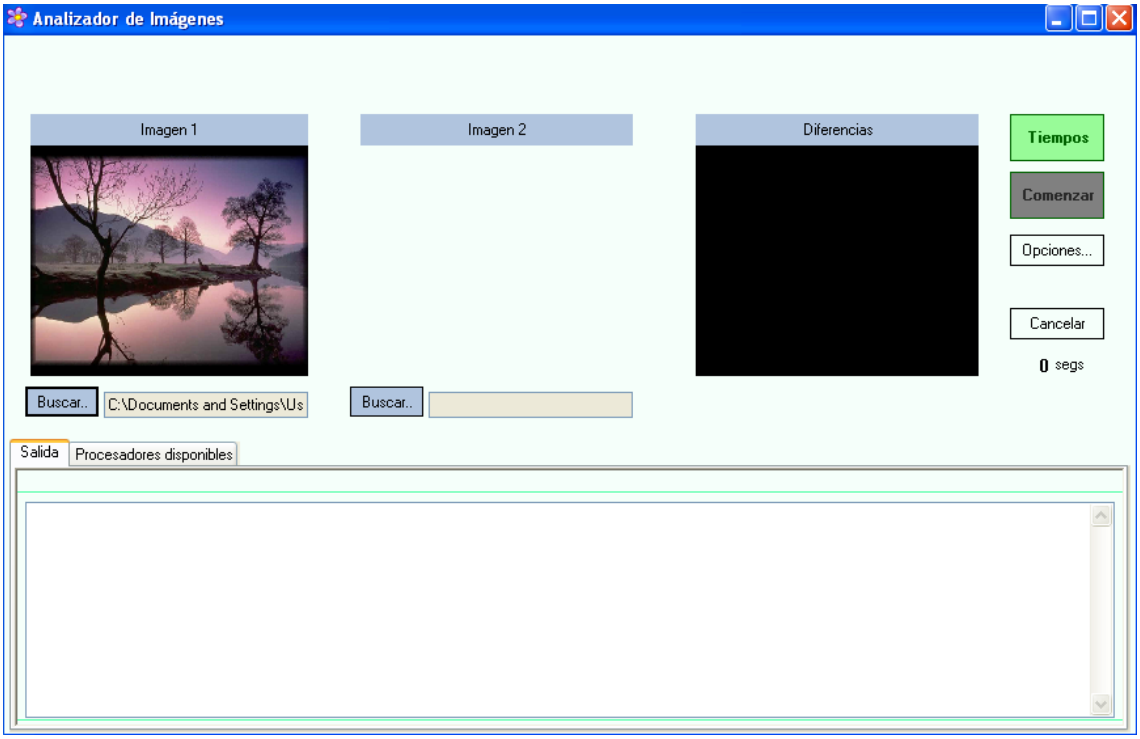


Figura 7.3.13

- Paso 2. Supóngase ahora, que se quiere ver cómo el algoritmo ha repartido la carga (filas, en este caso) entre los diferentes computadores y posiblemente quiera modificar las líneas repartidas. Para hacer esto, marcaremos la opción “Pedir reparto de filas” del formulario 2.

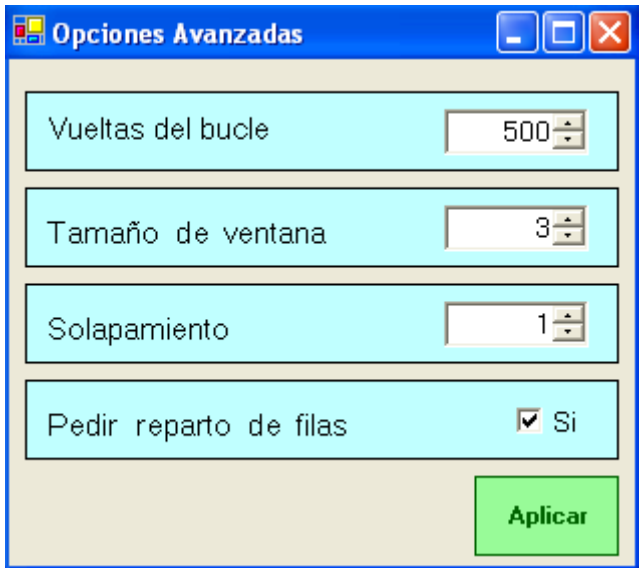


Figura 7.3.14

- Paso 3. Se pulsa el botón tiempos para ver el tiempo que cada procesador ha tardado en realizar un pequeño cálculo (ni que decir tiene que el procesador que menor tiempo ha obtenido, es el más rápido).

En la pestaña de Salida (número 12 de la figura 7.3.5) aparecerá:

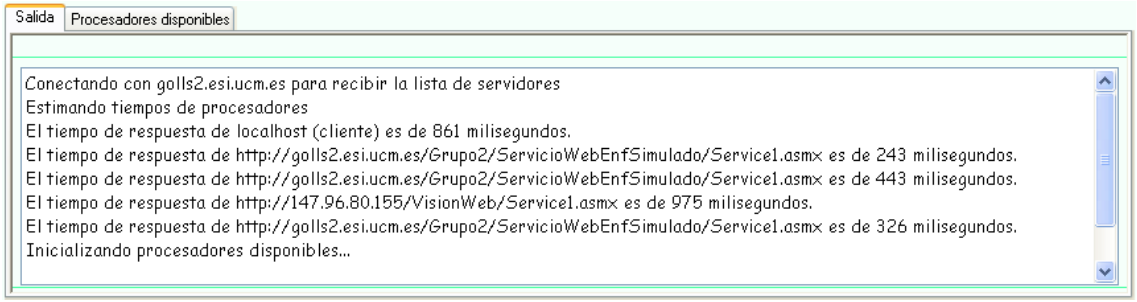


Figura 7.3.15

Esto una lista de los procesadores que están disponibles en ese instante, y el tiempo que han tardado en realizar el cálculo.

En la pestaña de Salida (número 13 de la figura 7.3.5) aparecerá:

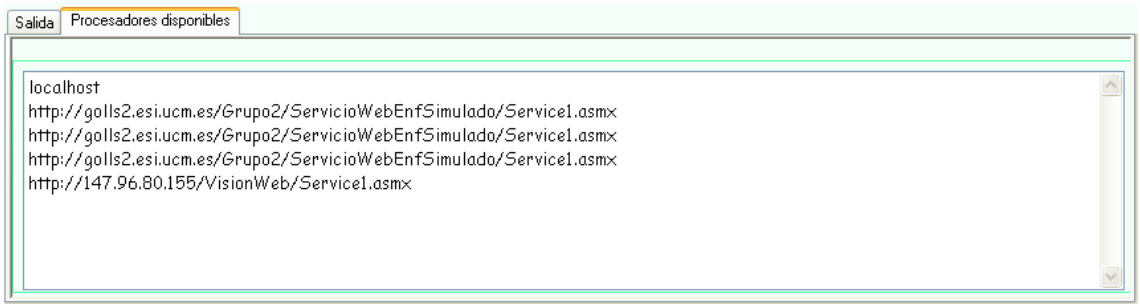


Figura 7.3.16

Esto es la lista de los procesadores disponibles, y que podremos consultar a lo largo de toda la ejecución del programa.

- Paso 4. Se pulsa el botón Comenzar, para iniciar el cálculo. Inmediatamente, se aparece el frame de la figura 7.3.15.

Figura 7.3.17

En este caso particular se puede interpretar que el procesador que aparece en rojo no esta disponible en este momento. Y por tanto no se puede modificar su número de filas.

Se puede probar a meter diferentes cantidades de filas a cada procesador, pero al final se debe pulsar el botón Aplicar sólo cuando la Suma de filas actual y Número de filas total tengan el mismo número.

Para este ejemplo, no se modificarán las cantidades del formulario.

- Paso 5. A partir de ahora el usuario deja de interactuar con la aplicación. Es ella la que va realizando todos los pasos. Para empezar, muestra la imagen de las diferencias troceada. Cada trozo corresponde a lo que tiene que calcular un procesador.

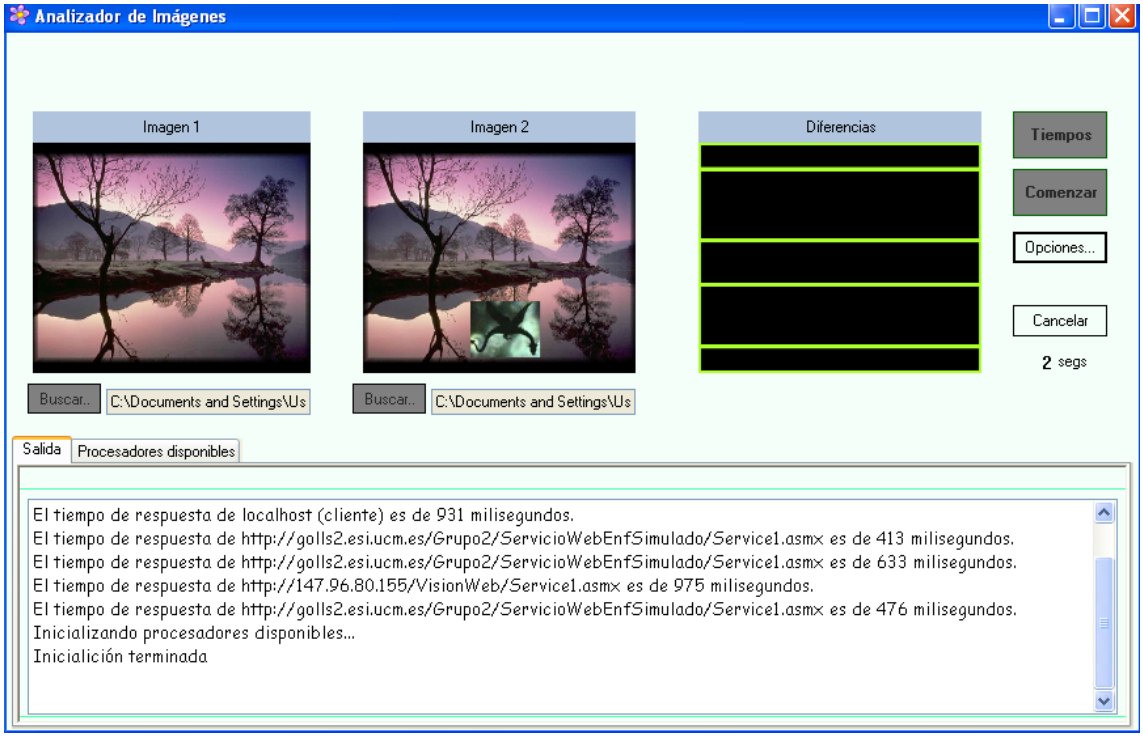


Figura 7.3.18

Por definición, el trozo de imagen de arriba lo realiza la máquina cliente (localhost). Se puede conocer la parte de la imagen que realiza cada procesador. Para ello, se puede pulsar la pestaña de Procesadores disponibles (figura 7.3.16). De tal manera que el primer trozo de la imagen lo realiza el primer procesador de la lista de la pestaña de procesadores disponibles. El segundo trozo de la imagen los realizará el segundo procesador de la lista y así sucesivamente.

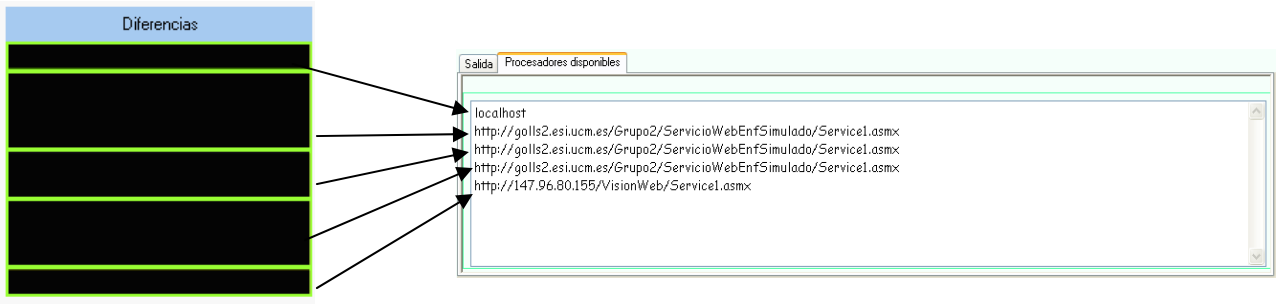


Figura 7.3.19

Cuando ya se ha terminado de realizar el cálculo de las diferencias de una de las partes de la imagen, se muestra por pantalla. Y en el trozo de imagen correspondiente aparecen las diferencias correspondientes. Además, en la caja de texto de la pestaña Salida aparece la información sobre el procesador que ha terminado.

Ahora solo queda esperar a que todas las partes sean procesadas.

Cuando todas las partes han sido procesadas, aparecerá en el la parte de diferencias, la imagen resultado. Donde las zonas oscuras indican que no hay cambio, y las zonas claras que si se ha producido cambio.

Las diferencias entre las imágenes que hemos comparado son las que se muestran en el apartado Diferencias de la figura 7.3.20.

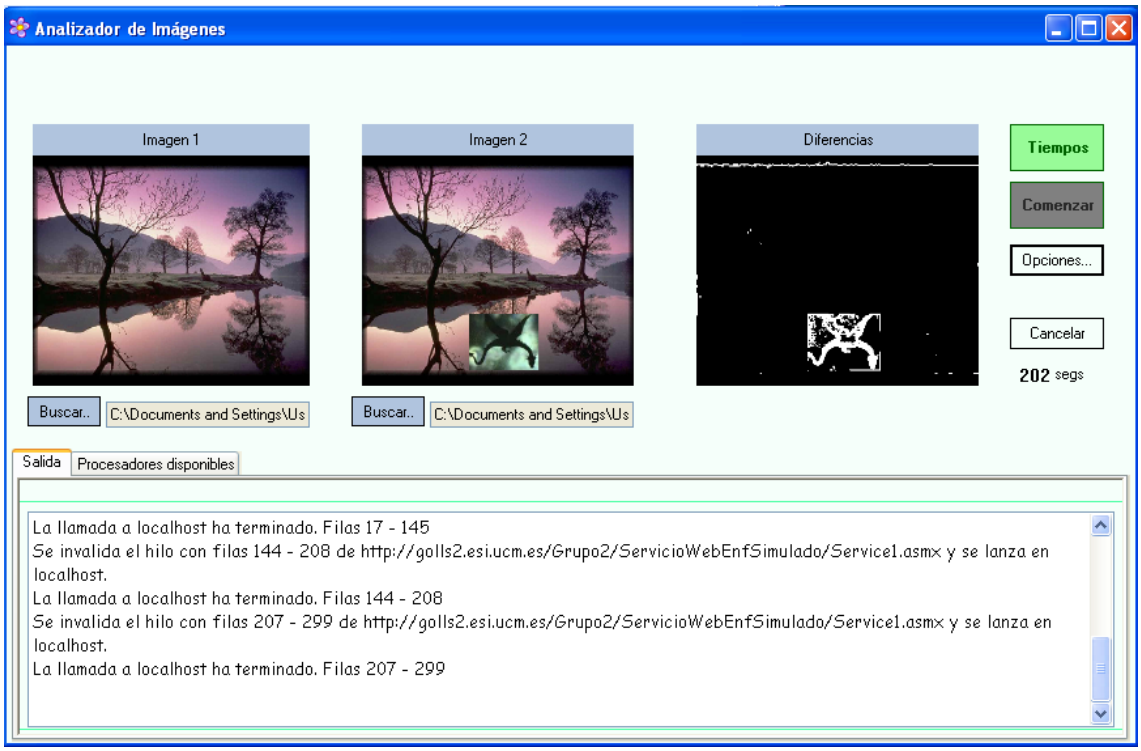


Figura 7.3.20

Además de mostrar la imagen que contiene las diferencias en el formulario de la aplicación, esta imagen resultado se guarda como un archivo con extensión .bmp, y nombre Salida. La ubicación por defecto de este archivo es el directorio donde se encuentra el ejecutable de la aplicación.

Siempre se guarda en la misma carpeta y con el mismo nombre, por lo que si accedemos a la carpeta donde se encuentra el ejecutable de la aplicación y abrimos el fichero Salida.bmp, se verá la última imagen de diferencia que se haya calculado.

7.3.8 FAQ´s

- ¿Se pueden comparar imágenes de diferentes formatos?

Si se puede, mientras que las imágenes tengan el mismo tamaño. Es decir, se puede comparar una imagen con extensión .gif y otro con extensión .jpg y la aplicación mostraría las diferencias siempre que tengan los mismos píxeles de alto y de ancho.

- Si se toman como entrada dos imágenes en blanco y negro, ¿también se comparan?

Si, también acepta imágenes en blanco y negro.

- ¿Qué tipos de formato de imágenes permite comparar?

Los formatos que pueden tener las imágenes que se comparan son: bmp, jpeg, tiff, png y gif.

- ¿Se guarda la imagen de salida de la aplicación en algún archivo, para que pueda ser utilizado?

Si, se guarda. Este archivo se salva en el mismo directorio donde se encuentran las imágenes de entrada que se le pasan a la aplicación, y se llama Salida.bmp.

Por tanto, este archivo contendrá las diferencias de la última comparación que se haya realizado.

- ¿Por qué los botones varían de color?

El color de los botones indica cuando está activo y se puede pulsar (color verde) o cuando está deshabilitado (color gris).

- Cuando ejecuto el algoritmo, además de guaradarme en el directorio correspondiente el archivo Salida.bmp, aparecen también dos archivos mas llamados fotoByNA.bmp y fotoByNB.bmp, ¿qué contienen estos archivos?

Contienen las versiones en blanco y negro intermedias de las imágenes iniciales de entrada.

7.4. **Descargar el proyecto**

La aplicación se podrá descargar desde la página web del proyecto. La direccion es:

http://golls2.esi.ucm.es/grupo2/DocWeb/inici0.htm

8. Apéndices

8.1. Diagramas UML

Los diagramas que a continuación se muestran, han sido desarrollados con el programa Rational Rose.

El lenguaje UML es específico para diseñar programas orientados a objetos. En esta aplicación se ha construido una aplicación web con hilos concurrentes que realizan llamadas a servicios web remotos, y por tanto es muy complicado modelarla con UML.

Aun así, aquí se presentan algunos de los diagramas que se han podido realizar sin dificultad.

Diagrama de casos de uso:

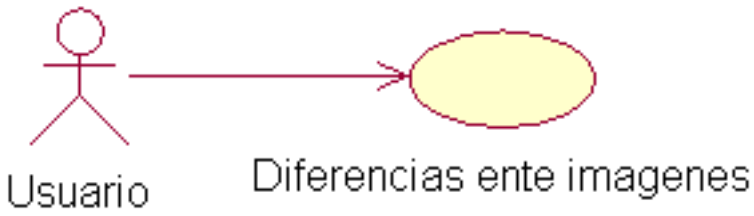


Figura 8.1.1

En este caso, el diagrama es muy sencillo ya que sólo hay un caso de uso.

Diagrama de actividades:

Para que resulte más cómodo revisar este diagrama, se ha dividido en tres partes.

- Diagrama inicial: donde comienza la actividad.

- Diagrama para calcular diferencias, que es una de las actividades del diagrama anterior.

- Diagrama para Calcular1, que es una de las actividades del diagrama anterior.

- Diagrama inicial:

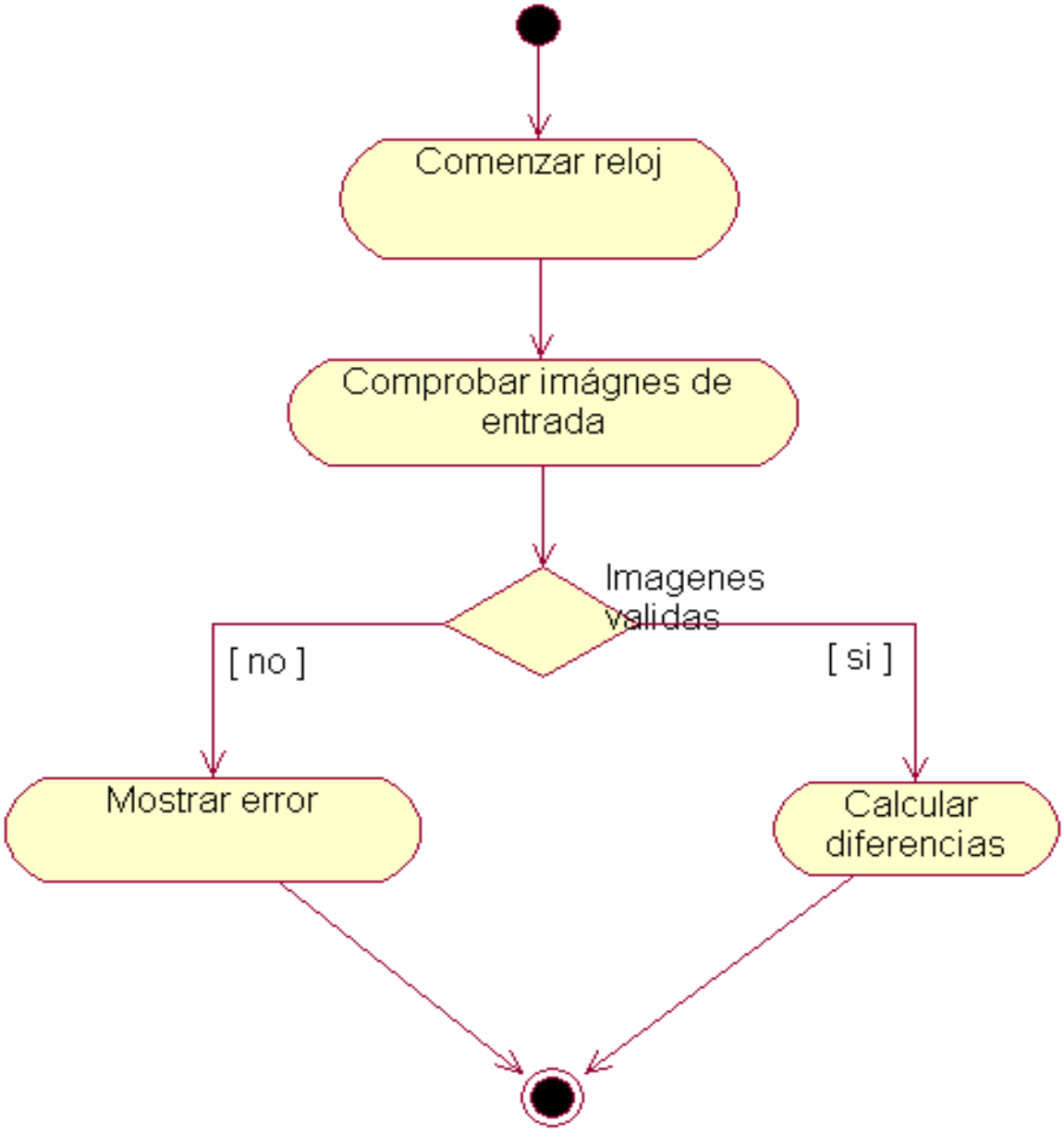


Figura 8.1.2

- Diagrama para calcular diferencias:

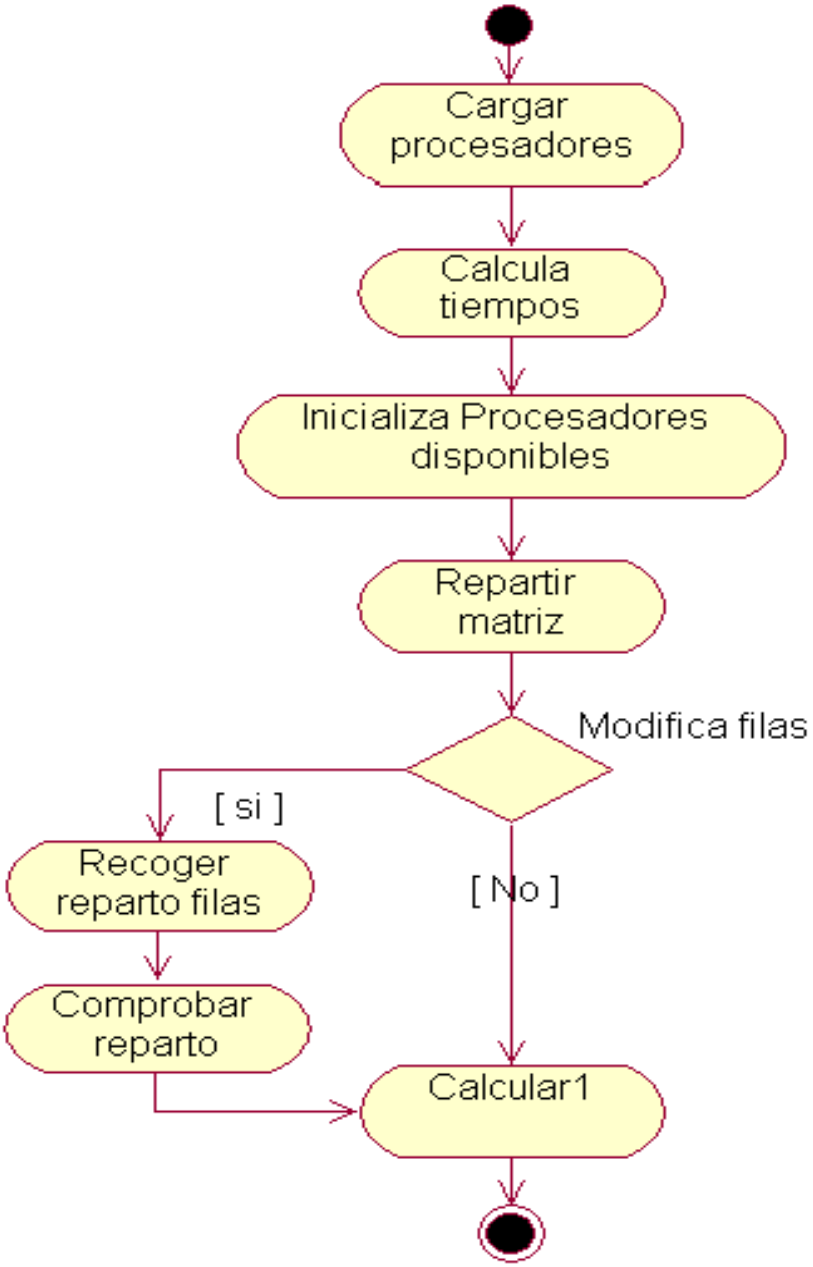


Figura 8.1.2

- Diagrama para Calcular1

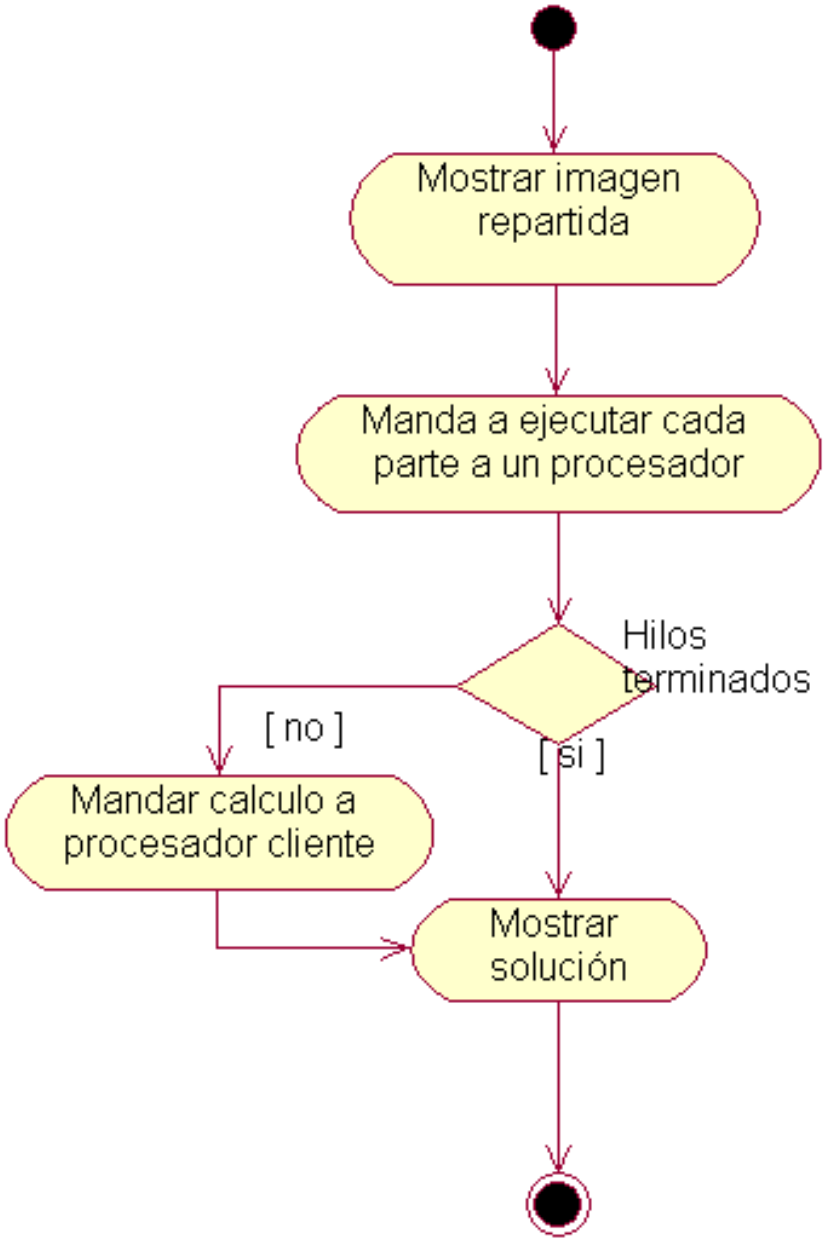


Figura 8.1.3

Diagrama de componentes

Por último, se van a mostrar los diagramas de componentes.

Este diagrama se ha descompuesto en un diagrama para los componentes del lado de cliente, y por otro lado, para las componentes del lado del servidor.

- Diagrama de componentes del lado del cliente:

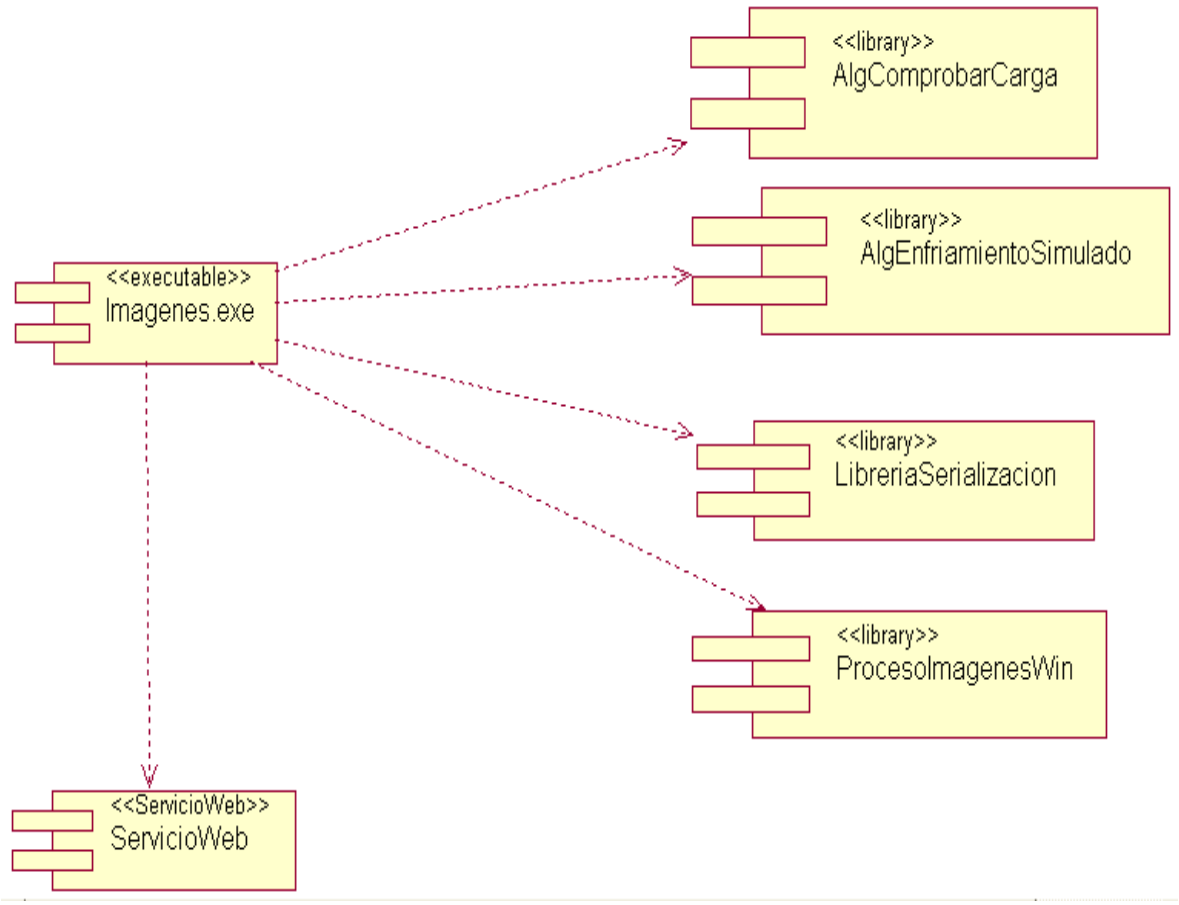


Figura 8.1.4

- Diagrama de componentes del lado del servidor:

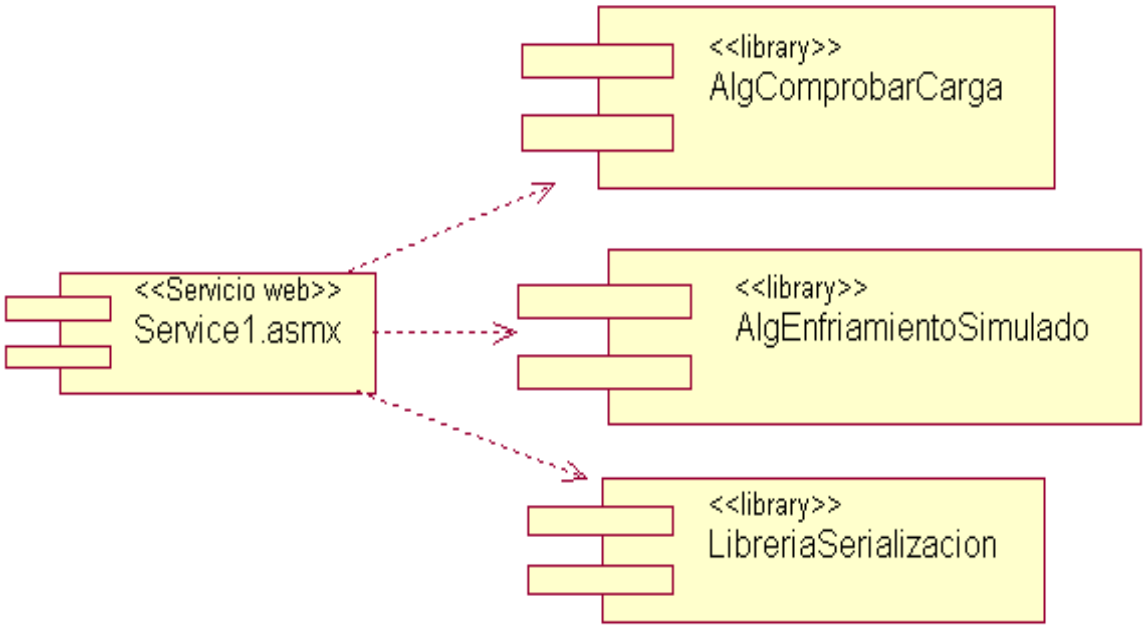


Figura 8.1.5

8.2. Código

Librerias

- AlgComprobarCarga

```
using System;

namespace AlgComprobarCarga
{
	/// <summary>
	/// Descripción breve de Class1.
	/// </summary>
	public class Class1
	{
		public Class1(){}
		public void CompruebaCarga()
		{
			DateTime antes = DateTime.Now;
			long suma=3;
			for (int i=1;i<=2000000;i++)
			{
				suma+=(long)Math.Pow(suma,i);
				suma*=3;
				int temp=0;
				while (temp<30000000)
				{
					long suma2=45;
					for (int j=1;i<=2000000;i++)
					{
						suma2+=(long)Math.Pow(suma,i);
						suma2*=3;
						int temp2=0;
						while (temp2<30000000)
						{
							temp2+=temp2+1;
						}
					}
					temp+=temp+1;
				}
			}
		}
	}
}
```

- AlgEnfriamientoSimulado

```
using System;
using System.IO;
//using System.Collections;

namespace AlgEnfriamientoSimulado
{
```

```
/// <summary>
/// Descripción breve de Class1.
/// </summary>
public class Class1
{     //Esta clase calcula el algoritmo simulado.
      static double TU = 1;   // valor umbral de la desviacion
      tipica en el calculo de la matriz de probabilidades (T*)
      static double epsilon = 0.0001;
// valor para evitar las divisiones por cero
      int VUELTAS = 500;
      int m=500;//1200;numero de filas total de la matriz.
      int numFilaIni;// numero de fila inicial a tratar
      int numFilaFin;// numero de fila final a tratar
      int n =500;//1600;      // numero de columnas.
      int r = 3;  //Tamaño de la ventana  // region de 3 X 3


      double[][] A = new double[0][]// matriz de puntos de la
                                          imagen  A
      double[][] B = new double[0][];// matriz de puntos de la
                                          imagen  B
      double[][] C = new double[0][];// matriz cociente
      double[][] M = new double[0][];// matriz de medias
      double[][] V = new double[0][];// matriz de varianzas
      double[][] S = new double[0][];// matriz de estado
      double[][] X = new double[0][];

      double mX = 0;    // media de la matriz X
      double dX = 0;    // desviacion tipica de la matriz X
      double mc = 0;    // media de Xc
      double sc = 0;    // varianza de Xc
      double mn = 0;    // media de Xn
      double sn = 0;    // varianza de Xn
      double[][] pX = new double[0][];// matriz probabilidad de X

      //Constructor
      public Class1(){}
      //Constructor con parámetros
      public Class1(double[][] matrizA,double[][] matrizB,int
            numFilas,int numColumnas, int numFI,int numFF,int
            vueltasIter,int tamVentana)
      {
            numFilaIni=numFI;
            numFilaFin=numFF;
            m=numFilas;
            n=numColumnas;
            VUELTAS=vueltasIter;
            r=tamVentana;

            //PARA INICIALIZAR LAS MATRICES
            A = new double[m][];//matriz de puntos de la imagen A
            B = new double[m][];//matriz de puntos de la imagen B
            C = new double[m][];// matriz cociente
            M = new double[m][];// matriz de medias
            V = new double[m][];// matriz de varianzas
            S = new double[m][];// matriz de estado
            X = new double[m][];
            pX = new double[m][];
```

```
            for(int i=0;i<m;i++)
            {
                  A[i]= new double[n];
                  B[i]= new double[n];
                  C[i]=new double [n];
                  M[i]=new double [n];
                  V[i]=new double [n];
                  S[i]=new double [n];
                  X[i]=new double [n];
                  pX[i]=new double [n];
            }


            A=matrizA;
            B=matrizB;
      }


      //public void calcular(ref double[][] S)
      public void calcular( double[][] solucion)
      {
            MatrizProbabilidad();

            EstadoInicial();

            for (int k = 1; k <= VUELTAS; k++)
            {
                  double T = 100;
                  Iteracion(T/k*k);
            }


            for (int i=0;i<S.Length ;i++)
                  for (int j=0; j<S[i].Length;j++)
                  {
                        solucion[i][j]=S[i][j];
                  }
      }

      private void EstadoInicial()
      {
            // Inicializacion de la matriz S
            for (int i = 0; i <= m-1; i++)
                  for (int j = 0; j <= n-1; j++)
                  {
                        S[i][j] = -1;
                  }
            SemiEstadoInicial(A,B);

            SemiEstadoInicial(B,A);
      }

      private void SemiEstadoInicial(double [][] a, double[][] b)
      {
            // Inicializacion de la matriz M
            for (int i = 0; i <= m-1; i++)
                  for (int j = 0; j <= n-1; j++)
                  {
                        M[i][j] = 0;
                  }
            // Inicializacion de la matriz V
```

```
for (int i = 0; i <= m-1; i++)
      for (int j = 0; j <= n-1; j++)
      {
            V[i][j] = 0;
      }

// Calculo de la matriz C (cociente)
for (int i = 0; i <= m-1; i++)
      for (int j = 0; j <= n-1; j++)
      {
            C[i][j] = a[i][j]/(b[i][j]+ epsilon);
      }

// Calculo de la matriz M (media)
for (int i = 0; i <= m-1; i++)
      for (int j = 0; j <= n-1; j++)
      {
            int nn = 0;
            for (int k = -(r-1)/2; k <= (r-1)/2; k++)
                  for (int l = -(r-1)/2; l <= (r-
                                          1)/2; l++)
                        if (i+k < 0 || i+k > m-1 ||
                              j+l < 0 || j+l > n-1)
                        {
                              M[i][j] = M[i][j]+ 0;
                        }
                        else
                        {
                              M[i][j] = M[i][j]+
                                          C[i+k][j+l]
                                    ;
                              nn = nn + 1;
                        }
            M[i][j] = M[i][j]/nn;

      }

// Calculo de la matriz V (varianza)
for (int i = 0; i <= m-1 ;i++)
      for (int j = 0; j <= n-1 ;j++)
      {
            int nn = 0;
            for (int k = -(r-1)/2; k <= (r-1)/2; k++)
                  for (int l = -(r-1)/2; l <= (r-
                                          1)/2; l++)
                        if (i+k < 0 || i+k > m-1 ||
                              j+l < 0 || j+l > n-1)
                        {
                              V[i][j] = V[i][j]+ 0;
                        }
                        else
                        {
                              V[i][j] = V[i][j] +
                        Math.Pow(C[i+k][j+l],2);
                              nn = nn + 1;
                        }


            V[i][j] = 1 - 1/(1+((V[i][j]/nn) -
                              Math.Pow(M[i][j],2)));
      }
```

```
		// Calculo de la matriz S (Estado Inicial)
		for (int i = 0; i <= m-1; i++)
			for (int j = 0; j <= n-1; j++)
				if (V[i][j] > 0.1)
				{
					S[i][j] = 1;
				}
	}

	private void MatrizProbabilidad()
	{

		//Calculo de la matriz X (diferencia)
		for (int i = 0; i <= m-1; i++)
			for (int j = 0; j <= n-1; j++)
			{

				double a=A[i][j];
				double b=B[i][j];
				double c=Math.Abs(a-b);
				X[i][j] = Math.Abs(a-b);

			}

		// Calculo de la media de X
		for (int i = 0; i <= m-1; i++)
			for (int j = 0; j <= n-1; j++)
			{
				mX = mX + X[i][j];
			}
		mX = mX / (m*n);


		// Calculo de la desviacion tipica de X
		for (int i = 0; i <= m-1; i++)
			for (int j = 0; j <= n-1; j++)
			{
				dX = dX + Math.Pow((X[i][j]- mX),2);
			}
		dX = Math.Pow((dX / (m*n)),0.5);


		// Calculo de mc y mn =  medias de Xc y Xn
		int k = 0;
		for (int i = 0; i <= m-1; i++)
			for (int j = 0; j <= n-1; j++)
			{
				if (X[i][j] > TU*dX)
				{
					mc = mc + X[i][j];
					k = k +1;
				}
				else
				{
					mn = mn + X[i][j];
				}
			}
		mc = mc / k ;
		mn = mn / ((m*n)-k);

		// Calculo de sc y sn =  varianzas de Xc y Xn
```

```
                k = 0;
                for (int i = 0; i <= m-1; i++)
                      for (int j = 0; j <= n-1; j++)
                      {
                            if (X[i][j] > dX)
                            {
                                  sc = sc + Math.Pow((X[i][j]-
                                              mX),2);
                                  k = k +1;
                            }
                            else
                            {
                                  sn = sn +  Math.Pow((X[i][j]-
                                              mX),2);
                            }
                      }
                sc = sc / k ;
                sn = sn / ((m*n)-k);


                // Calculo de pX = probabilidad de X
                double pXc;
                double pXn;
                double maxX = 0;
                double minX = 99999999999;
                for (int i = 0; i <= m-1; i++)
                      for (int j = 0; j <= n-1; j++)
                      {
                            pXc = (1/(Math.Pow(2*Math.PI,0.5)*sc))*
                                  Math.Exp(-0.5*Math.Pow(((X[i][j]-
                                  mc)/sc),2));
                            pXn = (1/(Math.Pow(2*Math.PI,0.5)*sn))*
                                  Math.Exp(-0.5*Math.Pow(((X[i][j]-
                                  mn)/sn),2));


                            pX[i][j] = pXc + pXn;
                            maxX = Math.Max(maxX,pX[i][j]);
                            minX = Math.Min(minX,pX[i][j]);
                      }
                for (int i = 0; i <= m-1; i++)
                      for (int j = 0; j <= n-1; j++)
                      {
                            pX[i][j] = (pX[i][j] - minX)/(maxX-minX);
                      }
          }

          private void Iteracion(double T)
          {
                int pi;
                int pj;
                int wij;
                double Ea = 0;
                double Eb = 0;
                Random R = new Random();
                      //En este punto es donde se hace que solo se
    calcule las filas correspondientes de la matriz

                for (int i =  numFilaIni; i <= numFilaFin; i++)
                //Desde la fila inicial a la fila final
                      for (int j = 0; j <= n-1; j++)
```

```
                        {
                              if (pX[i][j] < 0.05)
                              {
                                    pi = -1;
                              }
                              else
                              {
                                    pi = 1;
                              }
                              for (int k = -(r-1)/2; k <= (r-1)/2; k++)
                                    for (int l = -(r-1)/2; l <= (r-
                                                      1)/2; l++)
                                    {
                                          if (((i+k) >= 0) && ((i+k) <=
                                                (m-1)) && ((j+l) >= 0)
                                                && ((j+l) <= (n-1)))


                                          {
                                                if (pX[i+k][j+l]< 0.05)
                                                {
                                                      pj = -1;
                                                }
                                                else
                                                {
                                                      pj = 1;
                                                }
                                                if (i == j)
                                                      wij = 0;
                                                else
                                                      wij = pi * pj;
                                                Ea = Ea +
                                                      wij*S[i][j]*
                                                      S[i+k] [j+l];
                                          }
                                    }
                              Ea = -Ea * 0.5;
                              Eb = -Ea;
                              if (Eb < Ea)
                              {
                                    S[i][j] = -S[i][j];
                              }
                              else
                              {
                                    if (Math.Exp(-(Eb-Ea)/T) >
                                                R.Next(1))
                                    {
                                          S[i][j] = -S[i][j];
                                    }
                              }
                        }
                  }


            }
      }
```

- LibreriaSerializacion

```
using System;


namespace LibreriaSerializacion
{
	/// <summary>
	/// Descripción breve de Class1.
	/// </summary>
	public class Serializacion
	{
		public Serializacion(){}
/**SERIALIZACION DE DATOS DE MATRIZ A UN ARRAY DE STRING******/
//ESTA FUNCION TRANSFORMA EL ARRAY DE STRING QUE SE LE PASA, EN UNA
MATRIZ DOUBLE[][]

		public double[][] pasaAMatriz(String[] lineas)
		{

			//TRATAMIENTO DEL STRING

			//string[] lineas = s.Split(new Char[] {'\n'});
			//DIVIDO EL TEXTO EN FILAS
			int numFilas=lineas.Length;


			double [][] matriz;//DECLARO LA MATRIZ
			matriz=new double[numFilas][];//Inicializo la matriz
							(numero de filas)

			int numColum=0;

			for (int i=0;i<numFilas;i++)//DE LINEAS NOS SIRVE
							TODO MENOS LA ULTIMA
			{

				string[] linea=lineas[i].Split(new Char[]
						{' '});//DIVIDO LA FILA EN TOKENS
				numColum=linea.Length-1;
				matriz[i]=new double[numColum];
	//inicializo la matriz (numero de columnas	)
				for(int j=0;j<numColum;j++)
				{
					try
					{
						if (linea[j]!="")

							matriz[i][j]=double.Parse(
								linea[j]);
					}
					catch(System.FormatException ex)
					{

					}
				}
			}
			return matriz;
```

```
            }

            //ESTA FUNCION CONVIERTE LA MATRIZ QUE SE LE PASA POR
            PARAMETRO A STRING

            public String[] pasaAString(double[][] matriz)
            {
                  String s="";
                  int numFilas=matriz.Length;
                  int numColumnas=matriz[1].Length;
                  String [] salida=new String[numFilas];
                  for(int i=0;i<numFilas;i++)
                  {
                        for (int j=0;j<numColumnas;j++)
                        {
                              s = s + matriz[i][j]+" ";
                        }
                        salida[i] = s;
                        s="";
                  }
                  return salida;
            }
      }
}
```

Aplicación cliente

```
using System;
using System.Threading;
using System.Windows.Forms;//Para que muestre el MessageBox
using ProcesoImagenesWin;
using System.Drawing;

namespace Imagenes
{
      /// <summary>
      /// Contiene todos los datos referentes a las imagenes y
            procesadores disponibles. Además es el encargado
      /// de realizar las llamadas a los servicios web
      /// </summary>
      public class Controlador
      {
            #region Variables
            //Esta estructura contiene toda la informacion que vamos a
            necesitar de cada servidor
            //Reservamos la posicion 0 para la informacion de la
            maquina cliente
            public struct Processor
            {
                  public string direccion;//Contiene el nombre del
                                                servidor
                  public bool disponible;//Indicara si el servidor esta
                                                activo o no
                  public int tiempoEmpleado;//Tiempo que emplea en
                                                realizar
                  public int filaIni;//Fila desde la que va a empezar a
                        calcular la solucion (si no existiera solap)
                  public int filaFin;//Fila en que va a terminar el
                        calculo de la solucion (si no exisitiera solap)
                  public int numFilas; //filaFin-filaIni
            }
```

```
            int anchoA,anchoB,altoA,altoB;
            private string archivoA,archivoB;
            private String[] respuesta;
            private int timeout=200000; //200 segundos
            private bool[] consigueComunicar;
            private bool pideFilas;
            private int vueltasIter=500;
            private int solapamiento=1;
            private int tamVentana=3;//el tamaño de la ventana (matriz
                  de tamñano tamVentanaXtamVentana) debe ser impar
            private Thread[]
hiloExistencia,hiloPotencia,hilosCalculaTiempos;
            private bool[] responde;
            private int[] tiempo;   //tiempo de cada procesador en la
                              prueba de carga, en milisegundos
            private int numProcs; //el número de procesadores
            Processor[] procs=null;//procs es un array de tipo
                        Processor que contiene una posicion para
            //cada servidor de que disponemos
            double[][][] soluciones;//aqui guardamos las soluciones
                                    devueltas por cada servidor
            double[][] solucion;//esta es la solucion final recompuesta
            String[][] solucionS;//esta es la solucion final
                              recompuesta para mostrarla en la interfaz
            private Thread[] procsDisp;
            int sumaTiempos; //tiempo total que tardan los procesadores
                              en la prueba de carga, en milisegundos
            FormularioPrincipal formulario;
            private int numFilasImagen;//Este es el numero de filas
                              total que contiene la imagen
            private int numColumnasImagen;//Este es el numero de
                              columnas total que contiene la imagen
            private double[][] mA=new double[0][],mB=new double[0][];
            private int tiempoPrudencial=5; //el máximo tiempo en que
                  permitimos q respondan los servidores, en segundos
            private static Mutex mut = new Mutex();
            private LA transformador;
            private referencia.Service1[] comprobador;
            string archSal = "salida";
            string formatoSal = "bmp";
            private FormRepartoFilas frf;

            #endregion

            #region Constructores


            /// <summary>
            /// Constructor, se crea a partir del formulario principal,
                  dos archivos que son las imagenes, una
            /// variable booleana que indica si las fotos son del mismo
                  tamaño, el número de vueltas que dará
            /// el algoritmo de enfriamiento simulado, el solapamiento
                  y una variable que indica si se va a mostrar
            /// el formulario que muestra las filas que ejecutará cada
                  procesador
            /// </summary>
            /// <param name="f">formulario principal</param>
            /// <param name="archivoA">nombre del archivo de una de las
                  imagenes</param>
            /// <param name="archivoB">nombre del archivo de una de las
```

```
        imagenes</param>
/// <param name="validos">indica si las imagenes tienen el
        mismo tamaño</param>
/// <param name="vueltas">es el número de vueltas que dará
        el algoritmo de enfriamiento simulado</param>
/// <param name="solap">indica el tamaño del
        solapamiento</param>
/// <param name="pideFilas">indica si vamos a mostrar el
        formulario que muestra las filas que ejecutará cada
/// procesador</param>
public Controlador(FormularioPrincipal f,string
        archivoA,string archivoB,ref bool validos,
        int vueltas,int solap,bool pideFilas)//Constructor
{
        this.vueltasIter=vueltas;
        this.pideFilas = pideFilas;
        this.archivoA = archivoA;
        this.archivoB = archivoB;
        formulario= f;

        transformador = new LA(archivoA,mA,archivoB,mB);

        LeeFotos();
        this.MA=transformador.A;
        this.MB=transformador.B;
        anchoA = transformador.ImagenA.Width;
        altoA = transformador.ImagenA.Height;
        anchoB = transformador.ImagenB.Width;
        altoB = transformador.ImagenB.Height;
        if (ComprobarMedidasImagenes(solap))  //para
        comprobar que las imagenes tengan las mismas medidas
        {
              validos=true;
        }
        numFilasImagen=altoA;
        numColumnasImagen=anchoA;
        //el solapamiento no puede ser mayor que el número de
              filas de la imagen
        if (solapamiento > numFilasImagen)
        {
              this.solapamiento = numFilasImagen;
        }
        soluciones=new double [numProcs][][];
        solucion=new double[altoA][];
        solucionS=new string[altoA][];
        for (int h=0;h<altoA;h++)
        {
              solucion[h]=new double[anchoA];
              solucionS[h]=new String[anchoA];
        }
}
#endregion

#region Calcular

/// <summary>
/// Realiza las llamadas necesarias para la inicialización
de los procesadores disponibles, así como de sus datos
/// </summary>
```

```
public void PreCalculo()
{
	formulario.VaciaTextBoxProcs();
	CargarProcesadores();//Inicializamos el array que
		contiene la informacion de cada procesador


	//Debemos saber los servidores que tenemos
	disponibles
	CalculaTiempos();

	//Para tener en cuenta los que están disponibles
	InicializaProcsDisp();
	for (int i=0;i<procs.Length;i++)
	{
		if (procs[i].disponible)
		{
			string proces="";
			proces=procs[i].direccion;
			formulario.Escribe2(proces);
		}
	}
	formulario.ValidarBotonTiempos();
	formulario.ValidarBotonComenzar();
}


//esta es la función ppal. Va a llamar a todos los metodos
	y servicios web debidamente modularizados
/// <summary>
/// Realiza las llamadas para hacer el reparto de filas de
	las matrices de las imagenes, y muestra
/// el formulario FormRepartoFilas si es necesario. También
	llama a Calcular1
/// </summary>
public void Calcular()
{
	try
	{
		//repartimos las filas entre los procesadores
		(sin decidir aun en qué orden)
		CalculaReparto();

		//Hacemos los calculos para repartir la matriz
		en filas para hacer los cálculos
		int[] array = new int[procs.Length];
		for (int i=0; i<procs.Length;i++)
		{
			array[i] = procs[i].numFilas;
		}
		RepartoMatriz(array);

		if (pideFilas==true)
		{
			MuestraFormularioRepartoFilas();
		}

		Calcular1();
	}
	catch (System.DivideByZeroException dvz)
	{
```

```
                        MessageBox.Show("Error en el reparto de las
                              filas. Tiempo total de los procesadores:
                              0", "Warning",
                              MessageBoxButtons.OK,
                              MessageBoxIcon.Warning );
                  }
                  catch (Exception exc)
                  {
                        string mensaje= "Ha habido un error en la
                              operación: " + exc.Message;
                        MessageBox.Show( mensaje , "Warning",
                              MessageBoxButtons.OK,
                              MessageBoxIcon.Warning );
                  }
            }

            /// <summary>
            /// Realiza las llamadas para lanzar los hilos para
                  ejecutar los servicios web en los distintos
            /// servidores disponibles, y muestra la imagen solución en
                  el formulario
            /// </summary>
            public void Calcular1()
            {
                  //muestra el reparto de las partes de la imagen
                  MostrarPartesImagen();
                  //Mandamos a ejecucion para que cada procesador
                  disponible(incluido el de la maquina cliente)
                  ejecuten
                  //su parte de codigo
                  CreaHilosEnfriamiento();
                  //mostrar la imagen, parar el reloj y validar el
                  botón Comenzar
                  formulario.PararReloj();
                  formulario.MostrarImg( ImagenSolucion()
                                          ,"pictureBoxS");


                  formulario.ValidarBotonTiempos();
                  formulario.ValidarBotonA();
                  formulario.ValidarBotonB();
            }

            #endregion

            #region CargarProcesadores
            //Guarda la informacion de todos los servidores que tenemos
            disponibles
            public void CargarProcesadores()
            {
                  //Carga los procesadores disponibles, para ello se
                  conecta a un servidor para que proporcione de las
                  direcciones
                  bool dameProcesadores=false;
                  int i=0;
                  referencia.Service1 servicio=null;
                  String[] procesadores=null;

                  while ( !dameProcesadores && i<2)
                  {
                        //Por defecto se conecta solo a dos
```

```
                  procesadores
            if (i==0)
            {
                  try
                  {
                        formulario.Escribe("Conectando con
                              golls2.esi.ucm.es para
                              recibir la lista de
                              servidores");
                        servicio=new
                              referencia.Service1("http://g
                              olls2.esi.ucm.es/Grupo2/Servi
                              cioWebEnfSimulado/Service1.as
                              mx");

                        procesadores
            =servicio.DameProcesadoresAUtilizar ();
                        dameProcesadores=true;
                        numProcs=procesadores.Length;//El
                        número de procesadores total con
                        los que podemos contar.Incluyendo
                        la maquina cliente.
                  }
                  catch (System.Net.WebException
                        ex){numProcs=1;}
            }
            else
            {
                  try
                  {
                        formulario.Escribe("Conectando con
      147.96.188.72 para recibir la lista de servidores");
                        servicio=new
      referencia.Service1("http://147.96.188.72/WebService1
      /Service1.asmx");
                        procesadores
      =servicio.DameProcesadoresAUtilizar ();
                        dameProcesadores=true;

                        numProcs=procesadores.Length;//El
      número de procesadores total con los que podemos
      contar.Incluyendo la maquina cliente.
                  }
                  catch (System.Net.WebException
                        ex){numProcs=1;}
            }
            i++;
      }


      procs = new Processor[numProcs];
      consigueComunicar= new bool[numProcs];
      hiloExistencia= new Thread[numProcs];
      soluciones=new double [numProcs][][];
      comprobador = new referencia.Service1 [numProcs];
      hiloPotencia=new Thread[numProcs];
      hilosCalculaTiempos= new Thread[numProcs];
      responde= new bool[numProcs];
      tiempo = new int[numProcs];
      respuesta = new String[numProcs];
```

```
                AddProcessor(0,"localhost",true);//Este siempre esta
                      disponible

                if (numProcs>1)//Se tiene que hacer esta comprobacion
                      porque hay algún proc NO disponible
                {
                      for (int j=1; j<procesadores.Length;j++)
                      {
                            AddProcessor(j,procesadores[j],false);
                            //En principio no está disponible
                      }
                }

          }
          public void InicializaProcsDisp()
          {
                formulario.Escribe("Inicializando procesadores
disponibles...");
                int[] v = new int[numProcs]; //este vector lleva 1 si
                      el procesador de esa posicion esta disponible y
                      0 si no lo esta
                int v1 = 0; //numero de procesadores no disponibles,
                      se usa para restarlo al numero total de procs y
                      asi hacer un array de thread de tamaño menor
                /*calculo de los procesadores disponibles*/
                for (int i=0;i<numProcs;i++)
                {
                      if (procs[i].disponible)
                      {
                            v[i]=1;
                      }
                      else
                      {
                            v[i]=0;
                            v1++;
                      }
                }
                int v2=0;//aqui se lleva el calculo de la posicion
                      del vector de threads actual
                procsDisp = new Thread[numProcs-v1];//el numero de
                      threads es el numero total de procs menos los
                que no estan disponibles
                //en i se lleva el nombre del proc actual, si esta no
                disponible saltamos al siguiente y no se modifica v2
                for (int i=0;i<numProcs;i++)
                {
                      if (v[i]==1)
                      {
                            procsDisp[v2] = new Thread(new
                                  ThreadStart(LanzaHilos));
                            procsDisp[v2].Name = i.ToString();//el
                            nombre del thread es el del proc, es el
                            valor i NO el valor v2
                            //ya que puede que haya algun proc no
                            disponible y dejan de coincidir i y v2
                            v2++;//sumo uno para pasar a la siguiente
                            posicion del array de threads, notese que
                            NO coincidira el
                            //valor de la i con el de v2 en el
                            momento en el que algun proc no este
                            disponible
```

```
                        }
                  }
                  formulario.Escribe("Inicialición terminada");
            }


            private void AddProcessor(int numero, string nombre,bool
disponible)
            {
                  procs[numero].direccion =nombre;
                  procs[numero].disponible =disponible;
            }


            #endregion

            #region Tiempos
            /// <summary>
            /// Crea un hilo por servidor para hacer las pruebas de
                  existencia y tiempos
            /// </summary>
            private void CalculaTiempos()
            {
                  formulario.Escribe("Estimando tiempos de
                  procesadores");
                  //lanzamos el cálculo de tiempos del cliente
                  hilosCalculaTiempos[0]= new Thread(new
                        ThreadStart(CalculaTiemposCliente));
                  hilosCalculaTiempos[0].Start();

                  for (int i=1;i<numProcs;i++) //Haremos un hilo por
                              cada procesador
                  {
                        hilosCalculaTiempos[i]= new Thread(new
                              ThreadStart(CalculaTiemposHilos));
                        hilosCalculaTiempos[i].Name= i.ToString();
                        hilosCalculaTiempos[i].Start();

                  }

                  for ( int h=1; h<hilosCalculaTiempos.Length; h++)
                  {
                        try
                        {
                              hilosCalculaTiempos[h].Join();
                        }
                        catch (System.NullReferenceException e)
                        {
                              MessageBox.Show("Fallo al calcular los
                                    tiempos de los servidores remotos",
                                    "Warning",
                                    MessageBoxButtons.OK,
                                          MessageBoxIcon.Warning );
                        }
                  }
                  //recogemos el hilo lanzado para el cliente
                  hilosCalculaTiempos[0].Join();

            } //CalculaTiempos

            /// <summary>
```

```
        /// Calcula la existencia (disponibilidad) de un servidor y
              su tiempo de cálculo
        /// </summary>

        private void CalculaTiemposHilos()
        {
              int i= Convert.ToInt16( Thread.CurrentThread.Name );
              try
              {
                    consigueComunicar[i]=true;
                    int tiempoExistencia = tiempoPrudencial * 1000;
        //cogemos un tiempo prudencial

                    //creamos un hilo para comprobar que el
                    servidor existe y no está caido

                    hiloExistencia[i] = new Thread(new
                          ThreadStart(ComprobarExistenciaHilo));
                    hiloExistencia[i].Name= i.ToString();
                    hiloExistencia[i].Start();


responde[i]=hiloExistencia[i].Join(tiempoExistencia);//Esperamos
                    a que termine y le damos un tiempo prudencial

                    /*"responde" significa que el hilo ha terminado
                    en menos de "tiempoExistencia". Pero podría
                    devolver true porque se produzca una excepción,
                    por ejemplo si el servidor está caido.
                    Por eso ponemos "consigueComunicar, que nos
                    dice si se ha producido alguna excepción"*/

                    if (responde[i] &&
                          consigueComunicar[i])//comprobamos cuánto
                    tiempo tarda en ejecutar una simple tarea.
                    {
                          hiloPotencia[i] = new Thread(new
                                ThreadStart(ComprobarPotenciaHilo))
                          ;
                          hiloPotencia[i].Name= i.ToString();

                          hiloPotencia[i].Start();
                          hiloPotencia[i].Join();
                          if (consigueComunicar[i])//por si ha
                                fallado la conexión justo
                                comprobando la carga
                          {
                                procs[i].disponible=true;
                                procs[i].tiempoEmpleado=tiempo[i];
                                formulario.Escribe("El tiempo de
                                      respuesta de " +
                                      procs[i].direccion + " es de
                                      " + tiempo[i] + "
                                      milisegundos.");
                          }
                          else
                          {
                                procs[i].disponible=false;
                          }
                    }
                    else
```

```
                {  // no responde antes de tiempoRespuesta
                    procs[i].disponible=false;

                }

            }
            catch (System.NullReferenceException e){throw new
                Exception("fallo en CalculaTiemposHilos()");}
        }

        private void CalculaTiemposCliente()
        {
            AlgComprobarCarga.Class1 comprobar= new
                        AlgComprobarCarga.Class1();
            DateTime antes = DateTime.Now;
            comprobar.CompruebaCarga();
            DateTime despues = DateTime.Now;
            TimeSpan ts=despues-antes;
            procs[0].tiempoEmpleado = ts.Milliseconds;
            formulario.Escribe("El tiempo de respuesta de
                localhost (cliente) es de " + ts.Milliseconds +
                " milisegundos.");
        }


        public int EstimaTiempos()
        {
            int numP = 0;
            int totTmp = 0;
            int dev = 0;
            double dev1 = 0;
            //aqui se hace la estimacion del tiempo que tendran
            que esperar como maximo los hilos. Se calculara la
            estimacion para todos los procesadores disponibles y
            devolvera el maximo valor de todos.
            for (int i=0;i<numProcs;i++)
            {
                if (procs[i].disponible)
                {
                    totTmp+=procs[i].tiempoEmpleado;
                    numP++;
                }
            }
            double normT=0;//media de lo que tardaria un
                            procesador en responder
            normT = totTmp/numP;

            double norm=0;//una estimacion de lo que podria
                        tardar el procesador anterior en calcular
                        esas filas
            norm = normT*numFilasImagen*numColumnasImagen;

            dev1 = (norm*6000)/8254312;//regla de tres con un
                                        ejemplo real
            /*2 procesadores. sus tiempos en responder 812 y 843.
            (812+843)/2 = 827.5. Hay 95 filas y 105 columnas.
            827.5*95*105 = 8254312.5
             * con esos datos tardo el algoritmo unos 6 seg, que
                 son 6000 miliseg
             * REGLA DE TRES:
             * 8254312.5 ----------- 6000
```

```
            * norm ---------------- dev1*/

            dev = (int)Math.Round(dev1);
            return dev;
      }


      #endregion

      #region Reparto
      /// <summary>

      /// Reparte la matriz entre los procesadores disponibles
      según su tiempo de respuesta y potencia de cálculo
      /// </summary>
      public void RepartoMatriz(int[] array)
      {

            //int delta=numFilasTotal/sumaTiempos;
            int ultimaFilaAsignada=0;
            //int numFilas=0;
            for (int i =0;i<procs.Length ;i++)
            {
                  if (procs[i].disponible)
                  {
                        procs[i].numFilas= array[i];
                        if(ultimaFilaAsignada==0)  //por aquí
                                                pasará localhost
                        {

                              procs[i].filaIni=ultimaFilaAsignada
                              ;

                              ultimaFilaAsignada=ultimaFilaAsigna
                              da+procs[i].numFilas;
                              //ahora, esta comprobación es por
            si el cliente es el único servidor disponible
                              if
                              (ultimaFilaAsignada==numFilasImagen
                              )

                              procs[i].filaFin=ultimaFilaAsignada
                              -1;
                              else

                              procs[i].filaFin=ultimaFilaAsignada
                                          -1+solapamiento;
                        }
                        else
                        if(ultimaFilaAsignada+procs[i].numFilas==
                        numFilasImagen)
                        {
                              procs[i].filaIni=ultimaFilaAsignada
                                          - solapamiento;


                              ultimaFilaAsignada=

                                    ultimaFilaAsignada+
                                    procs[i].numFilas-1;
```

```
                        procs[i].filaFin=

                              ultimaFilaAsignada;
                        }
                        else if(
                        (ultimaFilaAsignada+procs[i].numFilas==
                        numFilasImagen-1) &&
                        (procs[i].numFilas==0) )
                        { //caso especial para cuando se le
                  obliga al último procesador analizar 0 filas

                              procs[i].filaIni=ultimaFilaAsignada
                              ;

                        //ultimaFilaAsignada=numFilasImagen-1;

                              procs[i].filaFin=ultimaFilaAsignada
                              ;
                        }

                        else if
                        (ultimaFilaAsignada+procs[i].numFilas <
                        numFilasImagen)
                        {
                              procs[i].filaIni=ultimaFilaAsignada
                                    - solapamiento;

                              ultimaFilaAsignada=
                                    ultimaFilaAsignada+
                                    procs[i].numFilas;

                              procs[i].filaFin=ultimaFilaAsignada
                                    -1+solapamiento;
                        }
                        else
                        {
                              throw new Exception("Reparto de
                              filas de matriz mal hecho");
                        }
                  }
            }
            CorregirSolapamiento();
      }

      /// <summary>
      /// Corrige las filas asignadas si el solapamiento es
            demasiado grande y se asignan filas no correctas.
      /// </summary>
      private void CorregirSolapamiento()
      {
            for (int i =0;i<procs.Length ;i++)
            {
                  if (procs[i].disponible)
                  {
                        if (procs[i].filaIni < 0)
                              procs[i].filaIni = 0;
                        if (procs[i].filaFin >
                              this.numFilasImagen)
                              procs[i].filaFin =
                                    this.numFilasImagen-1;
```

```
                }
            }
        }



        public void CalculaReparto()
        {
            int numFilasRedondeadas = 0;
            sumaTiempos=0;
            float t=0;
            float[] ts = new float[procs.Length];

            for (int i=0;i<procs.Length;i++)
            {
                if (procs[i].disponible)
                {
                    sumaTiempos += procs[i].tiempoEmpleado;
                }
            }
            for (int i=0;i<procs.Length;i++)
            {
                if (procs[i].disponible)
                {
                    if (procs[i].tiempoEmpleado != 0)
                    {
                        ts[i]= sumaTiempos /
                            procs[i].tiempoEmpleado;
                        t += ts[i];
                    }
                    else
                    {
                        ts[i]= sumaTiempos / (float)0.1;
                        t += ts[i];
                    }
                }
            }
            for (int i=0;i<procs.Length;i++)
            {

                if (procs[i].disponible)
                {
                    float c;
                    c = ts[i]/t;
                    procs[i].numFilas =
Convert.ToInt32(numFilasImagen*c);
                    numFilasRedondeadas += procs[i].numFilas;

                }
            }

            int numFilasRest=numFilasImagen-numFilasRedondeadas;
            //Esta instruccion es para que el cliente ejecute las
                filas restantes que no han sido
            //asignadas a ningun procesador. esto sucede porque
                redondeamos los decimales y
            //sin esto, podría ser que el calculo fuera
                incorrecto.
            //Ojo, que el númFilasRest puede ser negativo. En
```

```
                ese caso se las restamos al cliente
            //pensaré eso de que puede dar menos de 0 en el
                cliente si se le restan filas

            //Esto seria si el numero de filas fuera negativo
            if (procs[0].numFilas + numFilasRest>=0)
                procs[0].numFilas += numFilasRest;
            else
                procs[0].numFilas=0;
        }

        public void MuestraFormularioRepartoFilas()
        {
            frf= new FormRepartoFilas(formulario,this);

            //Aquí se paran todos los hilos hasta q cierras esta
            ventana modal.
            frf.ShowDialog();
            formulario.ContinuarReloj();
        }

        #endregion

        #region Procesamiento
        public void CreaHilosEnfriamiento()
        {
            int espera=0;
            espera = EstimaTiempos();//llamamos a la funcion que
                estima el tiempo de espera de los hilos
            for (int j=0; j<procsDisp.Length;j++)
            {
                procsDisp[j].Start();
            }
            //esperamos a que todos terminen
            foreach( Thread hilo in procsDisp)
            {
                bool ok=false;

                if (hilo.Name == "0")
                {
                    hilo.Join();
                    ok = true;
                }
                else
                {
                    ok = hilo.Join(espera);
                }

                /*aqui se manda al cliente lo que no ha podido
                ejecutar un servidor si ha transcurrido el
                tiempo * estimado que aparece en el join()*/
                if (!ok)
                {
                    double[][] sol;
                    sol=new double [numFilasImagen][];
                    for (int h=0;h<numFilasImagen;h++)
                    {
                        sol[h]=new
                            double[numColumnasImagen];
                    }
```

```
                        int fInicial,fFinal;
                        int i= Convert.ToInt16(hilo.Name);
                        fInicial = procs[i].filaIni;
                        fFinal = procs[i].filaFin;

                        //invalidamos el hilo viejo y creamos uno
                              nuevo que se ejecuta en localhost
                        formulario.Escribe("Se invalida el hilo
                              con filas " + fInicial + " - " +
                              fFinal +
                              " de " + procs[i].direccion + " y
                              se lanza en localhost.");
                        procs[i].direccion = "localhost";
                        hilo.Suspend();
                        try
                        {
                              AlgEnfriamientoSimulado.Class1
                              algorit = new
                                  AlgEnfriamientoSimulado.Class1(
                                    mA,mB,numFilasImagen,
                                    numColumnasImagen,fInicial,
                                    fFinal,vueltasIter,
                                    tamVentana);
                              algorit.calcular(sol);
                        }
                        catch(Exception excepcion)
                        {
                              throw new Exception("Error en la
                              llamada a " + procs[i].direccion);
                        }
                        soluciones[i]=sol;

                        //Aqui cada hilo rellenara su parte de la
                              matriz solucion
                        //Mando el hilo actual a
                              recomponesolucion para despues
                              saber el nombre

                        RecomponeSolucion(hilo);//segun pone en
                              la ayuda hilo aun no ha terminado
                              si ha transcurrido el tiempo
                        //por lo que lo podemos usar para mostrar
                              la solucion con su nombre
                  }
            }//en este punto ya esta creada la matriz solucion

      }


      public void LanzaHilos()
      {

            /*Aqui se miraria el nombre del hilo y dependiendo de
                  cual sea se haria una llamada
             * al servicio web correspondiente del servidor que
                  sea(determinamos un nombre
             * de hilo para cada servidor que tengamos)*/
            int i= Convert.ToInt16( Thread.CurrentThread.Name );
            int fInicial,fFinal;
            double[][] sol;
            sol=new double [numFilasImagen][];
```

```
for (int h=0;h<numFilasImagen;h++)
{
      sol[h]=new double[numColumnasImagen];
}

fInicial = procs[i].filaIni;
fFinal = procs[i].filaFin;

String[] smA,smB,ssol;

LibreriaSerializacion.Serializacion serial= new
      LibreriaSerializacion.Serializacion();
smA=serial.pasaAString(mA);
smB=serial.pasaAString(mB);
try
{
      if (procs[i].disponible)
      {
            if (i==0)
            {
                  AlgEnfriamientoSimulado.Class1
                  algorit = new
                     AlgEnfriamientoSimulado.Class1(
                     mA,mB,numFilasImagen,
                     numColumnasImagen,fInicial,
                     fFinal,vueltasIter,tamVentana);
                  algorit.calcular(sol);
                  soluciones[i]=sol;

                  //Aqui cada hilo rellenara su parte
                  de la matriz solucion
                  //Mando el hilo actual a
                  recomponesolucion para despues
                  saber el nombre

            RecomponeSolucion(Thread.CurrentThread);
            }
            else //if (i==1)
            {
                  referencia.Service1 algoritmo = new
                        referencia.Service1
                        (procs[i].direccion);
                  algoritmo.Timeout= timeout;
                  ssol =
                      algoritmo.CalculaEnfSimuladoWS(
                        smA,smB,numFilasImagen,
                        numColumnasImagen,
                        fInicial,fFinal,vueltasIter,
                        tamVentana);
                  sol=serial.pasaAMatriz(ssol);

                  soluciones[i]=sol;

                  RecomponeSolucion(
                              Thread.CurrentThread);
            }
      }
}
catch(Exception excepcion)
{
      throw new Exception("Error en la llamada a " +
```

```
                        procs[i].direccion);
                }

        }


        public void RecomponeSolucion(Thread hilo)
        {
                //miramos el numero del hilo y dependiendo del que
                        sea tomaremos los datos para rellenar
                //su parte de la matriz solucion
                int numero=int.Parse(hilo.Name);

                double[][] s = soluciones[numero];
                int fIni = procs[numero].filaIni;
                int fFin = procs[numero].filaFin;
                //recorremos las filas
                mut.WaitOne();
                for (int j=fIni;j<=fFin;j++)
                {
                        //recorremos las columnas
                        for (int k=0;k<numColumnasImagen;k++)
                        {
                                //aqui solo puede entrar un hilo cada vez
                                //mut.WaitOne();
                                if ((s[j][k]==1)||(solucion[j][k]==1))
                                {
                                        solucion[j][k]= 1;
                                        solucionS[j][k]= "1";
                                }
                                else
                                {
                                        solucion[j][k]= 0;
                                        solucionS[j][k]= "0";

                                }

                        }
                }
                if ( (fFin == (fIni+ solapamiento*2 -1)) ||
                                (fFin==fIni))  //en este caso, este
                        procesador ha analizado 0 filas
                        formulario.Escribe("La llamada a " +
                                procs[numero].direccion + " ha terminado.
                                Filas 0");
                else
                        formulario.Escribe("La llamada a " +
                                procs[numero].direccion + " ha terminado.
                                Filas " + fIni + " - " + fFin);
                MostrarPartesImagen();  //muestro aqui la parte de la
                imagen correspondiente para que no haya que esperar
                //a los hilos por orden para visualizar las partes
                mut.ReleaseMutex();
        }

        #endregion


        #region ComprobarTiempos
        private void ComprobarExistenciaHilo()
        {
```

```
            int i= Convert.ToInt16( Thread.CurrentThread.Name );

            try
            {
                  comprobador[i] = new
                        referencia.Service1(procs[i].direccion);
                  comprobador[i].Timeout = timeout;

                  respuesta[i]=
                        comprobador[i].ComprobarExistenciaWS();//
                        El metodo devuelve un string
            }
            catch (System.Net.WebException ex)
            {
                  consigueComunicar[i]=false;
            }
      }


      private void ComprobarPotenciaHilo()
      {
            int i= Convert.ToInt16( Thread.CurrentThread.Name );
            try
            {
                  referencia.Service1 comprobadorPot=new
                        referencia.Service1(procs[i].direccion);
                  comprobadorPot.Timeout = timeout;
                  DateTime antes = DateTime.Now;
                  comprobadorPot.ComprobarPotenciaWS();
                  DateTime despues = DateTime.Now;
                  TimeSpan ts=despues-antes;
                  tiempo[i] = ts.Milliseconds;

            }
            catch (System.Net.WebException ex)
            {
                  consigueComunicar[i]=false;
            }
      }

      #endregion

      #region FuncionesImagenes

      public void MostrarPartesImagen()
      {
            int [,] array = new int[procsDisp.Length,2];
            for (int i=0;i<procsDisp.Length;i++)
            {
                  int numProcesador =
                        int.Parse(procsDisp[i].Name);
                  if (i==0)
                  {
                        array[i,0] =
                              procs[numProcesador].filaIni;
                        array[i,1] = procs[numProcesador].filaFin
                              - solapamiento+1;
                  }
                  else if (i==procsDisp.Length-1)
                  {
                        array[i,0] = procs[numProcesador].filaIni
```

```
                        + solapamiento;
                    array[i,1] =
                        procs[numProcesador].filaFin;
                }
                else
                {
                    array[i,0] = procs[numProcesador].filaIni
                        + solapamiento;
                    array[i,1] = procs[numProcesador].filaFin
                        - solapamiento+1;
                }
            }
            formulario.MostrarImg( transformador.DividirImagen
                (solucion,array,1,0) ,"pictureBoxS");
        }


        /// <summary>
        /// Devuelve la imagen solucion
        /// </summary>
        /// <param name="solucion">array de 2 dimensiones con
            double de 0 a 255 que contiene los datos</param>
        /// <param name="archSal">nombre del archivo con el que la
            imagen se guardará en el disco duro</param>
        /// <param name="formatoSal">formato del archivo con el que
            la imagen se guardará en el disco duro</param>
        /// <param name="1">es el número que identifica al blanco
            en la foto que se devuelve</param>
        /// <param name="0">es el número que identifica al negro en
            la foto que se devuelve</param>
        /// <returns>Bitmap</returns>
        public Bitmap ImagenSolucion()
        {
            return transformador.ImagenSolucion
                (solucion,archSal,formatoSal,1,0);
        }


        /// <summary>
        /// Comprueba que las dos imágenes de entrada tienen las
            mismas dimensiones. También ajusta el solapamiento
        /// si éste es más grande que el alto de la imagen.
            Devuelve true si tienen las mismas dimensiones.
        /// </summary>
        /// <param name="solap">el solapamiento elejido por el
            ususario</param>
        /// <returns>bool</returns>
        public bool ComprobarMedidasImagenes(int solap)
        {
            if ( (anchoA != anchoB) || (altoA != altoB) )
            {
                return false;
            }
            else
            {
                if (solap > altoA) //el solapamiento no puede
                    ser mayor que el alto de la imagen
                    this.solapamiento = altoA;
                else
                    this.solapamiento = solap;
                return true;
```

```
        }
    }

    /// <summary>
    /// Agrisa las imágene A y B y las guarda como imágenes en
          escala de grises en el disco duro.
    /// </summary>
    private void LeeFotos()
    {
          string formatoDstA = "bmp";
          string formatoDstB = "bmp";
          string archivoByNA = "fotoByNA";
          string archivoByNB = "fotoByNB";

          transformador.Agrisar("A");
          //transformador.Guarda(transformador.A,"fotoA.txt");

          transformador.Escribe(transformador.ImagenA,archivoBy
                NA,formatoDstA);  // crea la foto resultado.


          transformador.Agrisar("B");
          //transformador.Guarda(transformador.B,"fotoB.txt");

          transformador.Escribe(transformador.ImagenB,archivoBy
    NB,formatoDstB);  // crea la foto resultado.

    }

    #endregion

    #region Propiedades

    public double[][] MA
    {
          get{ return mA;}
          set{ mA=value;}
    }

    public double[][] MB
    {
          get{ return mB;}
          set{ mB=value;}
    }

    public string ArchivoA
    {
          get{ return archivoA;}
          set{ archivoA=value;}
    }

    public string ArchivoB
    {
          get{ return archivoB;}
          set{ archivoB=value;}
    }

    public LA Transformador
    {
          get{ return transformador;}
          set{ transformador=value;}
```

```
            }
            public string ArchSal
            {
                  get{ return archSal;}
                  set{ archSal=value;}
            }
            public string FormatoSal
            {
                  get{ return formatoSal;}
                  set{ formatoSal=value;}
            }
            public Processor[] Procs
            {
                  get{ return procs;}
                  set{procs=value;}
            }
            public int NumFilasImagen
            {
                  get{return numFilasImagen;}
            }
            public int Solapamiento
            {
                  get{return solapamiento;}
                  set{solapamiento=value;}
            }
            #endregion
      }
}
```

Servicio web

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;
using AlgEnfriamientoSimulado;
using AlgComprobarCarga;
using System.IO;


//147.96.80.209/Grupo2/ServicioWebEnfSimulado/Service1.asmx
//http://147.96.188.72/WebService1/Service1.asmx
namespace ServicioWebEnfSimulado
{
      /// <summary>
      /// Descripción breve de Service1.
      /// </summary>
      public class Service1 : System.Web.Services.WebService
      {

            public Service1()
            {
                  //CODEGEN: llamada necesaria para el Diseñador de
                  servicios Web ASP .NET
                  InitializeComponent();
```

```
}

#region Código generado por el Diseñador de componentes

//Requerido por el Diseñador de servicios Web
private IContainer components = null;

/// <summary>
/// Método necesario para admitir el Diseñador. No se puede
/// modificar el contenido del método con el editor de
/// código.
/// </summary>
private void InitializeComponent()
{
}

/// <summary>
/// Limpiar los recursos que se estén utilizando.
/// </summary>
protected override void Dispose( bool disposing )
{
      if(disposing && components != null)
      {
            components.Dispose();
      }
      base.Dispose(disposing);
}

#endregion


[WebMethod]
public String[] DameProcesadoresAUtilizar()
{
      //La posicion 0 no la inicializo por convenio,porque
      siempre la considero como localhost
      String[] procs=new String[4];

      procs[1]="http://golls2.esi.ucm.es/Grupo2/
            ServicioWebEnfSimulado/Service1.asmx";

      procs[2]="http://147.96.197.142/WebService1/
            Service1.asmx";

      procs[3]="http://golls2.esi.ucm.es/Grupo2/ServicioWeb
            EnfSimulado/Service1.asmx";

      return procs;
}


[WebMethod]
public string ComprobarExistenciaWS()
{
      return "hola";
}
[WebMethod]
public void ComprobarPotenciaWS()
{
      AlgComprobarCarga.Class1 comp=new
            AlgComprobarCarga.Class1();
```

```
            comp.CompruebaCarga();
        }


        [WebMethod]
            //Le pasamos las matrices double[][] como String[]
            porque el servicio web no acepta
            //parametros de entrada y de salida que sean
            double[][] (arrays de dos dimensiones, en definitiva)
        public String[] CalculaEnfSimuladoWS( String[]
            smatrizA,String[] smatrizB,int numFilas,int
            numColumnas,int numFI,int numFF,int vueltas,
            int solapamiento)

        {
            double[][] matrizA,matrizB;

            //Transformamos los datos que recibe el servicio web
                en datos que acepta la libreria que calcula
            //el enfriamiento simulado, llamando a la libreria
                que serializa los datos
            LibreriaSerializacion.Serializacion serial= new
                LibreriaSerializacion.Serializacion();
            matrizA=serial.pasaAMatriz(smatrizA);
            matrizB=serial.pasaAMatriz(smatrizB);


            //Inicializamos la matriz que va a devolver el
                resultado
            double[][] solucion=new double[numFilas][];//tamaño
                    de las filas
            for(int i=0;i<numFilas;i++)//tamaño de las filas
            {
                solucion[i]=new double [numColumnas];//tamaño
                                            de las columnas
            }

            //Creamos objeto de la clase de
                LibreriaEnfriamientoSimulado
            AlgEnfriamientoSimulado.Class1  alg= new
                AlgEnfriamientoSimulado.Class1(matrizA,matrizB,
                numFilas,numColumnas,numFI,numFF,vueltas,
                solapamiento);
            //Llamamos a la funcion de la clase
                LibreriaEnfriamientoSimulado que calcula la
                matriz solucion
            alg.calcular(solucion);

            //Transformamos los datos que devuelve la libreria
                que calcula el enfriamiento simulado
            //en datos que el servicio web puede devolver
            String[] ssolucion=serial.pasaAString(solucion);

            return ssolucion;

        }
    }
}
```

9. **Glosario**

- **Arquitectura cliente-servidor**: sistema distribuido donde un cliente se encarga de realizar peticiones a un sistema denominado *servidor* que se encarga de darles respuesta (cliente y servidor pueden coincidir físicamente en la misma máquina).

- **Aplicación**

    - Del lado del cliente: aplicación de un nodo de un sistema distribuido que se encarga de realizar peticiones a otros nodos del sistema.

    - Del lado del servidor: aplicación de un nodo de un sistema distribuido que se encarga de responder las peticiones que le llegan de otros nodos del sistema.

- **C#**: es el nuevo lenguaje de propósito general orientado a objetos creado por Microsoft para su nueva plataforma .NET.

- **Deserialización de datos**: consiste en deshacer la serialización y dejar el objeto con la misma forma que tenía inicialmente

- **Enfriamiento simulado**: este algoritmo está basado en la analogía entre el proceso de aprendizaje, que intenta minimizar la función de error, y la evolución de un sistema físico, que tiende a disminuir su energía cuando se decrementa la temperatura.

- **Hilos**: son procesos que representan una secuencia simple de instrucciones ejecutada en paralelo con otras secuencias.

- **Programación concurrente**: es una técnica de programación basada en la ejecución simultánea de procesos en un ordenador (con uno o varios procesadores).

- **Programación paralela**: es una técnica de programación basada en la ejecución simultánea de procesos en un ordenador (con uno o varios procesadores).

- **Programación distribuida**: es una técnica de programación basada en la ejecución simultánea de procesos en varios ordenadores.

- **Serialización de datos**: es el proceso en el que se toman objetos y se convierte su información de estado en un formato que permita su transporte o su almacenamiento.

- **Servicios Web**: constituyen el siguiente paso en la evolución de la tecnología orientada a objetos, y representan una revolución al alejarse de las arquitecturas tradicionales tipo *cliente-servidor* a nuevas arquitecturas distribuidas tipo *igual-a-igual* (*peer-to-peer).* Estos servicios consisten de un conjunto de estándares que permiten a los desarrolladores implementar aplicaciones distribuidas, utilizando herramientas muy distintas para crear aplicaciones que utilizan una combinación de módulos de software que son llamados desde diversos sistemas distribuidos en regiones geográficas distintas.

- **Simulated Annealing**: Estos algoritmos están basados en la analogía entre el proceso de aprendizaje, que intenta minimizar la función de error, y la evolución de un sistema físico, que tiende a disminuir su energía cuando se decrementa la temperatura.

- **SOAP (Simple Object Access Protocol)** es un protocolo para iniciar las conversaciones con un servicio UDDI. El SOAP simplifica el acceso a los objetos, permitiendo a las aplicaciones invocar métodos objeto o funciones, que residen en sistemas remotos. Una aplicación SOAP crea una un petición bloque en XML proporcionando los datos necesarios para el método remoto así como la ubicación misma del objeto remoto.

- **UDDI (Universal Description, Discovery and Integration)**, es un protocolo para describir los componentes disponibles de servicios Web. Este estándar permite a las empresas registrarse en un tipo de directorio sección amarilla de Internet que les ayuda anunciar sus servicios, de tal forma que que las compañías se puedan encontrarse unas a otras y realizar transacciones en la Web. El proceso de registro y consultas se realiza utilizando mecanismos basados en XML y HTTP(S). En el proyecto UDDI se trabaja para proveer un metodo de acceso comun a los metadatos necesarios para determinar su un elemento de código previamente elaborado es suficiente, y si lo es, cómo accederlo.

- **WSDL (Web Service Description Language)**, es el estándar propuesto para la descripción de los servicios Web, el cual consiste en un lenguaje de definición de interfaz (IDL - Interface Definition Language) de servicio basado en XML, que define la interfaz de servicio y sus características de implementación. El WSDL es apuntado en los registros UDDI y describe los mensajes SOAP que definen un servicio Web en particular.

- **XML (eXtensible Markup Language)**, se inició en Febrero de 1998 y ha revolucionado la forma en que estructuramos, describimos e intercambiamos información. Independientemente de múltiples formas en que utiliza hoy en día el XML, todas las tecnologías de servicios Web se basan en XML. El diseño de XML se deriva de dos fuentes

principales: SGML (Standard Generalized Markup Language) y de HTML (HyperText Markup Language).

- **Visual Studio .NET:** Visual Studio .NET es un **IDE** (Integrated Development Environment, entorno de programación) creado por Microsoft con el objetivo de hacer más fácil e intuitiva la programación. Provee a los programadores de un amplio conjunto de herramientas para construir aplicaciones distribuidas, para la web (servicios web con XML) y para dispositivos móviles.

10. Bibliografía

10.1. Básica

- C# Al descubierto.
  Editorial: Prentice Hall.Joseph Mayo. Traducción

- Profesional C# 2ª edición.
  Editorial Wrox. Autores: Simon Robinso,, K.Scott Allen, Ollie Cornes…

10.2. Complementaria

- Windows Form Programming in C#. Editorial Addison-Wesley. Chris Sells.

- http://msdn.microsoft.com/

- http://www.dcc.uchile.cl/~luguerre/cc61j/recursos/web-app.ppt#256,1,Diapositiva%201
- http://www.bobpowell.net/lockingbits.htm
- http://www.fisica.uson.mx/carlos/WebServices/WS_WSDL.htm
- http://msdn.microsoft.com/netframework/gettingstarted/default.aspx
- http://www.fisica.uson.mx/carlos/WebServices/WSRevolution.htm
- http://www.ibm.com/ar/desarrolladores/conozca.phtml
- http://www.clikear.com/manuales/csharp/index.asp
- http://geneura.ugr.es/~jmerelo/ws/
- http://java.oreilly.com/news/farley_0800.html