Sistemas Informáticos
Curso 2004 / 2005

Entorno de desarrollo para aplicaciones distribuidas sobre redes de servicios Web/Grid

Realizado por :

Mª Carmen Guardiola Palomino
Ana Mellado López
Mercedes Muñoz Muñoz

Dirigido por :

Prof. José Jaime Ruz Ortiz
Dpto. Arquitectura de Computadores y Automática

Facultad de Informática
Universidad Complutense de Madrid

ÍNDICE GENERAL

1. AUTORIZACIÓN A LA UCM.

Los autores de este proyecto, Mª Carmen Guardiola Palomino, Mercedes Muñoz Muñoz y Ana Mellado López, autorizamos a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos (no comerciales) tanto la memoria como el código y el prototipo desarrollado.

Fdo. Mª Carmen Guardiola Palomino

Fdo. Mercedes Muñoz Muñoz

Fdo. Ana Mellado López

2. LISTA DE PALABRAS CLAVE

2.1. ALGORITMO GENÉTICO

2.2. SERVICIO WEB

2.3. INTERCAMBIO

2.4. ESTADO DE UN SERVICIO WEB

2.5. MULTIHILO

2.6. PARALELO

2.7. C#

2.8. PLATAFORMAS .NET

2.9. SINCRONIZACIÓN

2.10. REFLEXIÓN

3. UBICACIÓN Y METODOLOGÍA.

3.1. Ubicación.

Actualmente, el uso de procesadores para el estudio de fenómenos científicos requiere cada vez más una mayor potencia de cálculo para poder llevar a cabo el estudio del fenómeno al nivel de detalle que el propio fenómeno requiere.

Por ejemplo, ciencias como

- aeronáutica / aerodinámica — Disminuye costes
- centrales nucleares — Investigación más segura
- astrofísica — Sistemas muy grandes
- dinámica molecular — Sistemas muy pequeños
- dinámica de partículas — Escala microscópica
- *sist. descritos por PDE´s* — Escala macroscópica

exigen una dinámica de cálculo muy variada y de profundidad diversa.

La solución clásica que se había desarrollado hasta ahora consistía en la "computación centralizada basada en servidor", es decir, era un único equipo (más grande y más potente) el que realizaba todos los cálculos del sistema.

Con posterioridad se desarrolla una "alternativa económica a la adquisición de un sistema multiprocesador" que consiste en un conjunto de computadores que interactúan entre sí (trabajando de manera paralela) para conseguir un objetivo común. La ganancia que esta forma de uso consigue es mucho menor debido a las partes secuenciales de los programas, la sobrecarga debida a las comunicaciones-sincronizaciones y la desigualdad de carga. No obstante, la ventaja predominante de este tipo de sistema reside en que nos permite aprovechar los recursos infrautilizados de otros equipos para nuestro propio uso mientras estos no están siendo usados por sus propias aplicaciones. Además el utilizar un grupo de computadores que trabajan conjuntamente con un objetivo común nos permite reducir los costes de los equipos y su mantenimiento y mejorar considerablemente la relación coste-rendimiento.

3.2. Metodología.

En primer lugar se ha desarrollado la aplicación mediante un "paralelismo simulado" o MULTIHILO, el que las diversas instancias del algoritmo se ejecutan todas en la misma máquina, cada una en un hilo. De esta forma antes de probar la aplicación con un "paralelismo real" se depuran los errores de programación y se comprueba el buen funcionamiento del algoritmo.

En segundo lugar, una vez que se ha comprobado que el algoritmo funciona correctamente, se prueba la aplicación en PARALELISMO real en WEB donde se podrán observar realmente los beneficios de una ejecución paralela. En

este caso, cada instancia del algoritmo de minimización se lanza a una máquina diferente a través de la web (servicios web en .NET). En nuestro caso, además del uso normal de los Servicios Web .NET tenemos que implementar el concepto adicional de "estado de un Servicio Web" puesto que las distintas instancias del algoritmo (que se ejecutan en distintas máquinas) tienen que compartir información para optimizar su resultado, por lo que esta información (≡ "estado del Servicio Web") debe conservarse entre las distintas llamadas al Servicio Web.

Es importante destacar la importancia del uso de esta metodología. El desarrollo de la aplicación en multihilo es un paso que nos permite asegurar el llevar a cabo una depuración completa de la aplicación para que cuando la probemos con paralelismo real en web estemos seguros de su correcto funcionamiento. Además, al desarrollar la aplicación en estas dos versiones, podremos observar realmente los beneficios que una ejecución paralela nos aporta frente a una ejecución multihilo.

3.2.1. Algoritmo.

Se ha elegido como instancia de algoritmo un algoritmo genético. Se escogió éste por las facilidades que ofrecía el tener una estructura adaptable al proceso de paralelización, ya que es un algoritmo multipoblación en el que cada población puede residir en una máquina mientras interactúan. A esta adaptabilidad a la paralelización que el algoritmo genético tiene, en nuestro caso debemos añadir la conexión entre las distintas poblaciones debido a los recursos compartidos entre ellas.

4. OBJETIVO DEL PROYECTO

El objetivo del proyecto ha sido implementar a través de las facilidades aportadas por recursos de la plataforma .NET una aplicación basada en aprovechar el paralelismo que ofrecen los Servicios Web, con una estructura de programación orientada a objetos.

Para poder hacer una investigación más real sobre la ganancia de paralelo respecto a multihilo, se ha desarrollado una aplicación que no es totalmente independiente, ya que comparten ciertos recursos. Este tipo de aplicaciones se denominan aplicaciones débilmente acopladas y son las que presentan mayores facilidades para el desarrollo de aplicaciones paralelas.

Nuestro proyecto se centra en las aplicaciones débilmente acopladas a través de servicios Web en los que, debido a ese acoplamiento, el estado del servicio Web se conserva de forma que las llamadas a los servicios no son independientes unas de otras (como es normal en los servicios Web), sino que los datos calculados o guardados en la llamada anterior están disponibles para llamadas futuras al servicio Web.

Este uso de los servicios Web es muy útil en las aplicaciones distribuidas en las que sea necesario guardar cierta información de los datos para usos futuros.

The aim of the Project has been to implement through the facilities provided by resources of .NET platform, an aplication based on using paralelism offering by Web Services with an Objects Oriented Programming structure.

To make a more real research on the improvement of paralel execution over Multi-thread one, a not independent application has been developed, because they share some resources. This kind of applications are called strongly connected applications.

Our project is concerned on this strongly connected applications through Web services in wich, because of this connection, the state of the web service is kept in a way the calling of the services are not independent among others (as usually happens in a Web Service), but calculated or kept in the calling before data are available for future calling to web services.

This way to use web services is usefull in distributed applications , where is necesary to guard a piece of information to future uses.

5. DESARROLLO DEL PROYECTO

Este capítulo se centra en la descripción del desarrollo de las partes en las que ha consistido el proyecto. Por un lado, y como se mentó en el apartado de objetivo de la práctica, Para construir una implementación fiel de nuestro objetivo, recordemos que era sacar partido a las facilidades de paralelización que ofrecen los servicios Web/Grid (para más información léase Apartado *4.3., Tecnologías/ Redes Web/Grid*) el desarrollo se ha centrado en la implementación de un algoritmo genético multipoblación que comunica sus subpoblaciones a través de un buffer en el que se intercambian los individuos, Además para la posible comparativa sobre la aportación de una aplicación paralelizada frente a una que no lo sea, La aplicación tiene dos formas distintas de ejecución. En multihilo, es decir toda la aplicación se ejecuta en la misma máquina y Paralelo, la aplicación parte la carga gracias a la utilización de servicios Web/Grid.

5.1. Algoritmo Genético e Intercambio.

Para hacer buen uso de nuestro objetivo de la práctica se ha implementado un algoritmo genético de minimización mejorado según se propone en a teoría de Algoritmos Genéticos. Una de las posibles mejoras propuestas sobre un algoritmo genético simple que desarrolla todo su ciclo (inicio, selección, cruce y mutación) en una única población, es la simulación de este comportamiento en múltiples poblaciones, es decir que se ejecuten varios algoritmos genético y cada uno trabaje con una subpoblación, esto es lo que denominamos (Algoritmo Genético Multipoblación) y que es la parte a la que hemos explotado el paralelismo, ya que las subpoblaciones con las que trabaja cada algoritmo genético son totalmente independientes unas de otras.

Otra de las posibles mejoras es, partiendo de un algoritmo genético multipoblación, implementar una migración entre las subpoblaciones con las que trabaja cada algoritmo genético. Esta parte no es paralelizable ya que debe haber una comunicación entre los algoritmos genéticos. Esto es lo que nosotros denominamos como INTERCAMBIO.

El intercambio es un proceso que hace uso de un buffer (recurso compartido) y una cola de comunicaciones, En el buffer estarán guardados los mejores individuos de cada subpoblación, cada cierto tiempo los algoritmos genéticos colocan en la cola de comunicaciones su mejor individuo y recogen de la misma el individuo mejor del buffer (si es mejor que el que ellos colocaron). La filosofía con la que se ha implementado esta comunicación está basada en el PRODUCTOR-CONSUMIDOR, de forma que el algoritmo genético cuando coloca su mejor individuo en la cola se comporta como productor, cuando el proceso intercambio esté libre actuará como consumidor de la cola de la que recogerá el individuo, operará con él y lo colocará en el buffer y en la cola siendo esta vez productor de la cola, el algoritmo genético, cuando vea que el

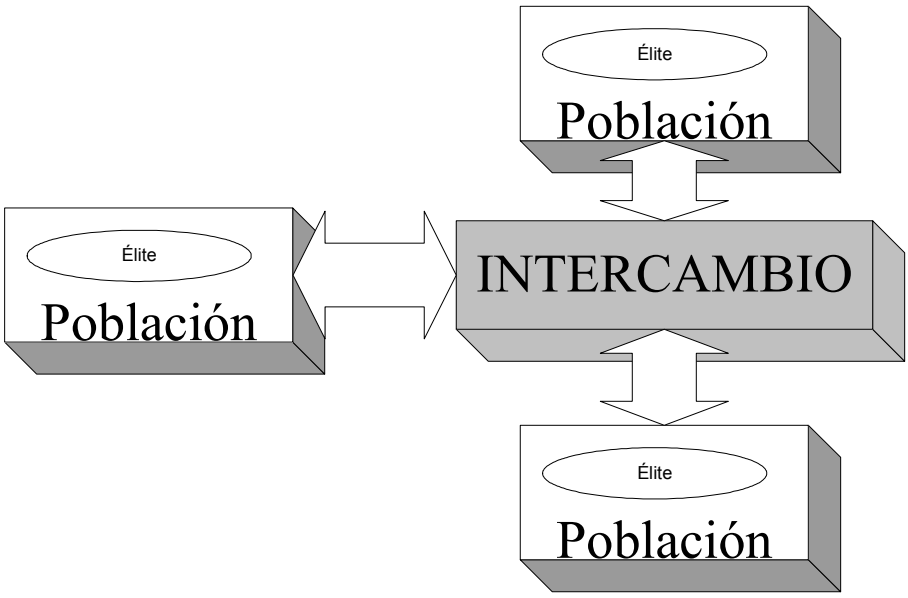
intercambio ya se ha realizado, actuará como consumidor de la cola, recogerá el individuo y lo sustituirá por un individuo de su población.
Se debe destacar que gracias a esta filosofía de productor-consumidor, los algoritmos genéticos siguen ejecutándose con normalidad sin necesidad de esperar a que el Intercambio termine sus operaciones y devuelva el individuo.
También se debe destacar que el proceso intercambio está sincronizado por un monitor de forma que sólo opera con un individuo cada vez, las peticiones que le llegan y tienen que esperar se guardan en la cola de comunicaciones, y cuando el intercambio esté disponible, seguirá operando con los individuos de la cola, uno por uno.

Por otra parte el algoritmo genético también implementa otra mejora como es el ELITISMO, que consiste en ir guardando los mejores individuos generados en cada generación, Esta mejora fue implementada ya que nos es muy útil para el comportamiento que hemos desarrollado ya que si los individuos mejores están guardados en la élite, cada vez que se quiere intercambiar el mejor individuo no hay que buscarlo en toda la población sino sólo entre los individuos que conforman la élite.

De gran importancia para el correcto funcionamiento del algoritmo es la elección de la representación de los individuos, así como de la función de fitness para la correcta evaluación y la elección del criterio de parada del algoritmo, es decir, cuando considera la población que no tiene que seguir su ciclo y que ya se ha obtenido una solución satisfactoria. Esto se ha implementado con un umbral de mejora, de forma que si en la generación correspondiente al estudio no ha mejorado en es umbral la puntuación de algún individuo de la élite, el algoritmo genético finaliza. (para más información sobre los Algoritmos genéticos léase el *Anexo 1. Algoritmos genéticos*).

Para hacer más general la aplicación, el algoritmo genético está totalmente parametrizado es decir, la función de optimización del algoritmo genético, así como las variables de la función de optimización, las restricciones de las variables, el tamaño de la población y el criterio de parada del algoritmo genético los introduce el propio usuario.

Para la implementación de esta posibilidad se ha usado un recurso que esta más desarrollado en C# que en otros lenguajes de programación, la Reflexión. Que consiste en la generación de código en tiempo de ejecución, gracias a esta facilidad, el algoritmo genético puede funcionar para cualquier función de optimización sin necesidad de cambiar el código del algoritmo genético (para más información sobre Reflexión léase Anexo 2. Reflexión). Por tanto cualquier usuario que necesite un algoritmo genético para una función de optimización determinada podrá usar nuestro algoritmo genético a través del Servicio Web/Grid simplemente llamando al Servicio Web con la función de optimización deseada, el tamaño de la población, las variables y sus rangos y el criterio de parada.
El diagrama del comportamiento es el siguiente:



El algoritmo genético se ha implementado como una biblioteca de clases de forma que está disponible para toda la aplicación con sólo referenciarla. Ésta es una de las facilidades que ofrece la plataforma .NET y el Visual Studio Project 2003.

También se ha implementado una biblioteca de clases virtual de forma que la biblioteca que implementa el algoritmo genético hereda de ella, ya que las dos posibles formas de ejecución de la aplicación tienen en común todo excepto la forma del intercambio, por tanto existe una biblioteca de clases común que es la "InterfazBibliotecaAG.dll"y dos bibliotecas que sólo contienen el algoritmo genético en sí, que heredan de ella y son "BibliotecaAG.dll" y "BibliotecaAGBuffer.dll".

5.2. Multihilo.

En el caso de Multihilo la parte paralelizable de la aplicación, es decir la ejecución de cada algoritmo genético que trabaja con una subpoblación, se ejecuta en un hilo dentro de la misma máquina, además el proceso intercambio está también en otro hilo para poder implementar la filosofía del productor-consumidor citada anteriormente.

El usuario selecciona el número de hilos que desea ejecutar, este número representa el número se algoritmos genéticos que se van a lanzar y que van a intercambiar entre ellos.

Los hilos son lanzados a través de la clase "PadreMultihilo" que es la encargada de lanzar los hilos y esperar a que acaben. Si no ocurre ningún error, el padre multihilo devuelve el resultado a la interfaz gráfica donde el usuario podrá ver el mejor resultado que han calculado los hilos. Si ocurre algún error, el padre Multihilo se encarga de detener la ejecución del hilo que tuvo el problema, y devuelve a la interfaz gráfica el resultado mejor de entre los hilos que se han ejecutado correctamente.

En la Figura 1, que se muestra a continuación, se puede ver el comportamiento externo de la aplicación para el caso Multihilo:

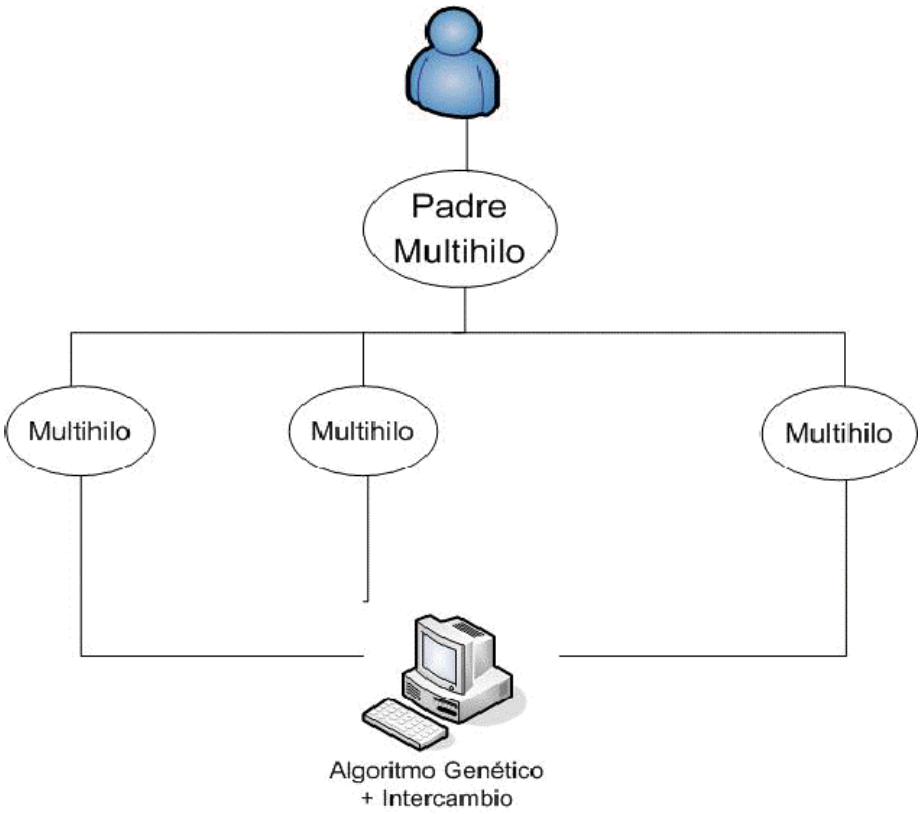


Figura 1. Esquema externo Multihilo

Internamente el comportamiento de cada hilo es la ejecución del Algoritmo Genético implementado en la "BibliotecaAGBuffer.dll" la cual referencia a otra biblioteca de clases dónde está implementado el proceso intercambio, ésta biblioteca se llama "BibliotecaBuffer.dll". Cada cierto tiempo los algoritmos genéticos llaman en otro hilo a la ejecución del intercambio, y cuando esté disponible recogerá el individuo intercambiado.

En la Figura 2 puede verse un diagrama del comportamiento interno de la aplicación en su modo de uso multihilo.

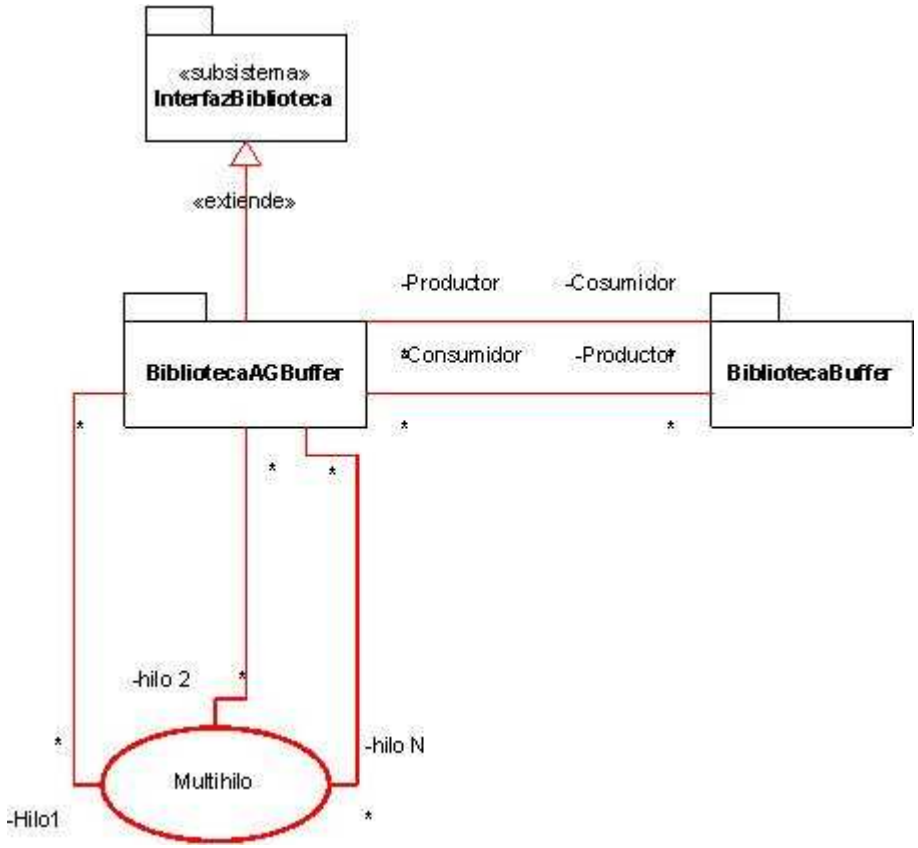


Figura 2. Esquema interno Multihilo

5.3. Paralelo.

Con la paralelización vamos un paso más allá ya que al repartir las poblaciones entre distintas máquinas a partir de sus IP's lo que estamos consiguiendo es que los hilos no tengan que competir por el tiempo del procesador, por lo que se consigue el mismo efecto pero con una importante optimización en el tiempo. Esta mejora se aprecia siempre que el tiempo que tarde el algoritmo genético en ejecutarse sea superior al tiempo que tardan en establecerse las comunicaciones.

Haciendo uso de los Servicios Web que son clases que se publican en un servidor Web con soporte para ASP.NET. Estos servicios presentan la particularidad de que sus métodos pueden ser llamados remotamente. Estas clases pueden escribirse en la mayoría de los lenguajes adaptados a .NET (Para más información sobre servicios Web, léase apartado *4.2. Tecnologías/servicios Web*).

El comportamiento de la aplicación es similar al caso de Multihilo, sólo que en este caso las poblaciones residirán en servicios Web independientes en el espacio, pero que se comunican con un servicio Web centralizador que es el que realiza las migraciones entre las poblaciones.

Cabe destacar que la naturaleza de los servicios Web es que cada llamada a un método de un servicio, crea un nuevo objeto de la clase a la que se hace la petición (Servicio Web) el objeto atiende esta petición y es destruido tras ello. Este tipo de implementación de Servicios Web es válida para cada algoritmo genético, por que son independientes unos de otros. Pero en la parte no paralelizable de la aplicación, es decir en el Intercambio las llamadas a este servicio Web no son independientes, por tanto es necesario que el Servicio Web que implementa el intercambio sea único y además que conserve el estado para que cada llamada al este servicio no sea independiente de la anterior. Este problema se solventa utilizando un tipo implementado en C# que permite guardar en un contenedor los valores de cada llamada y recuperarlos en la llamada siguiente (Para más información sobre la conservación del estado, léase el Apartado *4.2.5. Tecnologías/ Servicio Web/ Mantenimiento del estado.).*

Cuando el usuario selecciona el modo de aplicación paralelo, Debe elegir de entre las máquinas que están disponibles, aquellas en las que desee que se ejecute el algoritmo genético.

Las máquinas disponibles son aquellas en las que el Servicio Web que implementa el Algoritmo Genético, está instalado y es accesible. Para saber si una máquina está disponible se hace una llamada a un método del servicio Web que ejecuta un bucle si la llamada nos devuelve una respuesta, el servicio Web está disponible. Si no nos devuelve respuesta en un tiempo determinado consideramos que ese servicio web no está disponible y por tanto la máquina tampoco.

Una vez seleccionadas las máquinas se crea un hilo que lanza a "Padre Paralelo" que es el encargado de inicializar el “ServicioWebBuffer” que es donde está implementado el intercambio. Este servicio web debe inicializarse ya que como conserva el estado, cada vez que se lance de nuevo la aplicación se debe inicializar el estado.

Por otro lado la clase “PadreParalelo” crea instancias de la clase "Paralelo", en los cuales se crean objetos de “ServicioWebAG” a los que se le pasa la única instancia de “ServicioWebBuffer” que se declara (y que existe).

La estructura general de la aplicación en su modo de paralelo puede verse a continuación en la Figura 3:

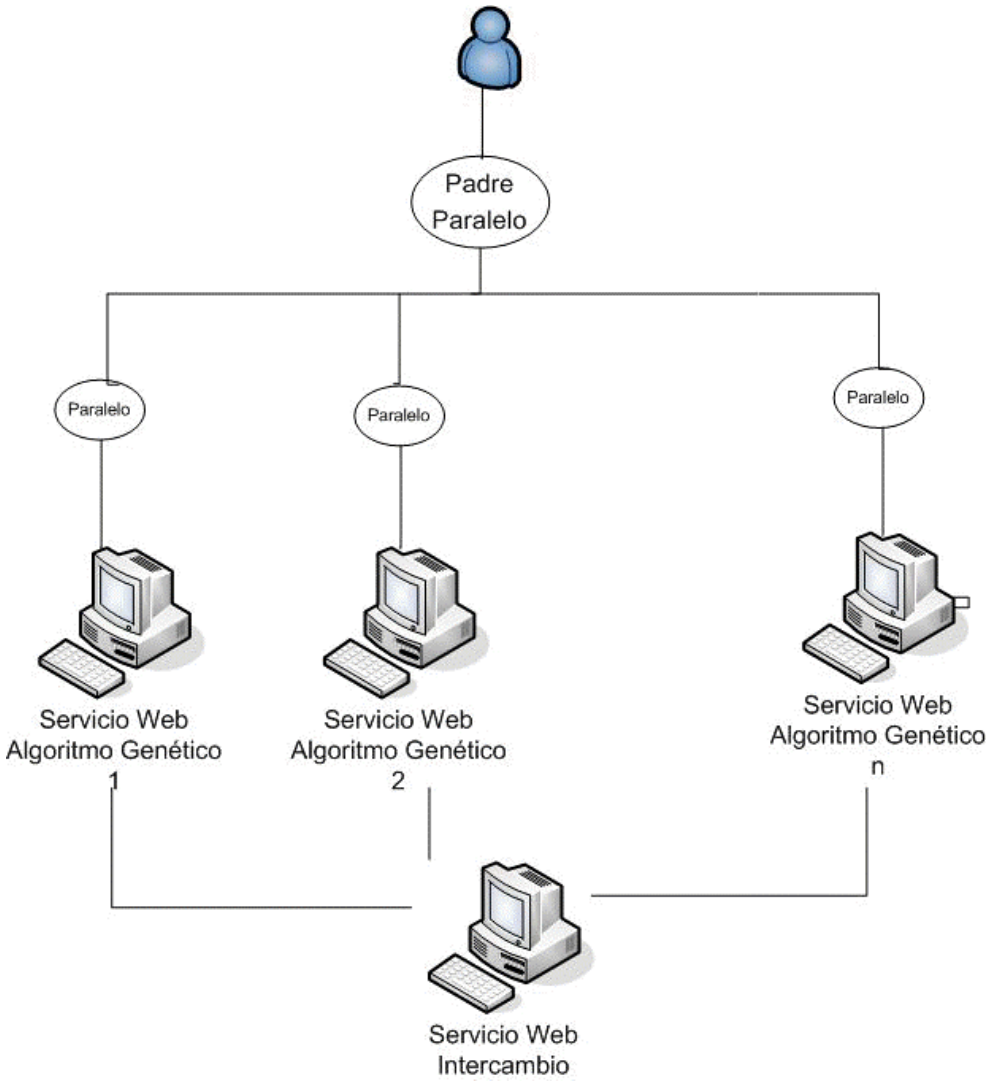


Figura 3. Esquema externo de Paralelo

El comportamiento de la aplicación en su caso de paralelo, es análogo al caso de multihilo. Así padre Paralelo también espera a que los Servicios Web terminen su ejecución y devolver el resultado a la interfaz gráfica para que el usuario pueda ver el mejor resultado.

El servicio Web que ejecuta el algoritmo genético "ServicioWebAG" tiene referenciada la biblioteca de clases "BibliotecaAG.dll" dónde está implementado el Algoritmo genético, este a su vez tiene referenciado el "ServicioWebBuffer" para ejecutar los intercambios.

A continuación se muestra la Figura 4 con el esquema interno del modo de aplicación paralelo:

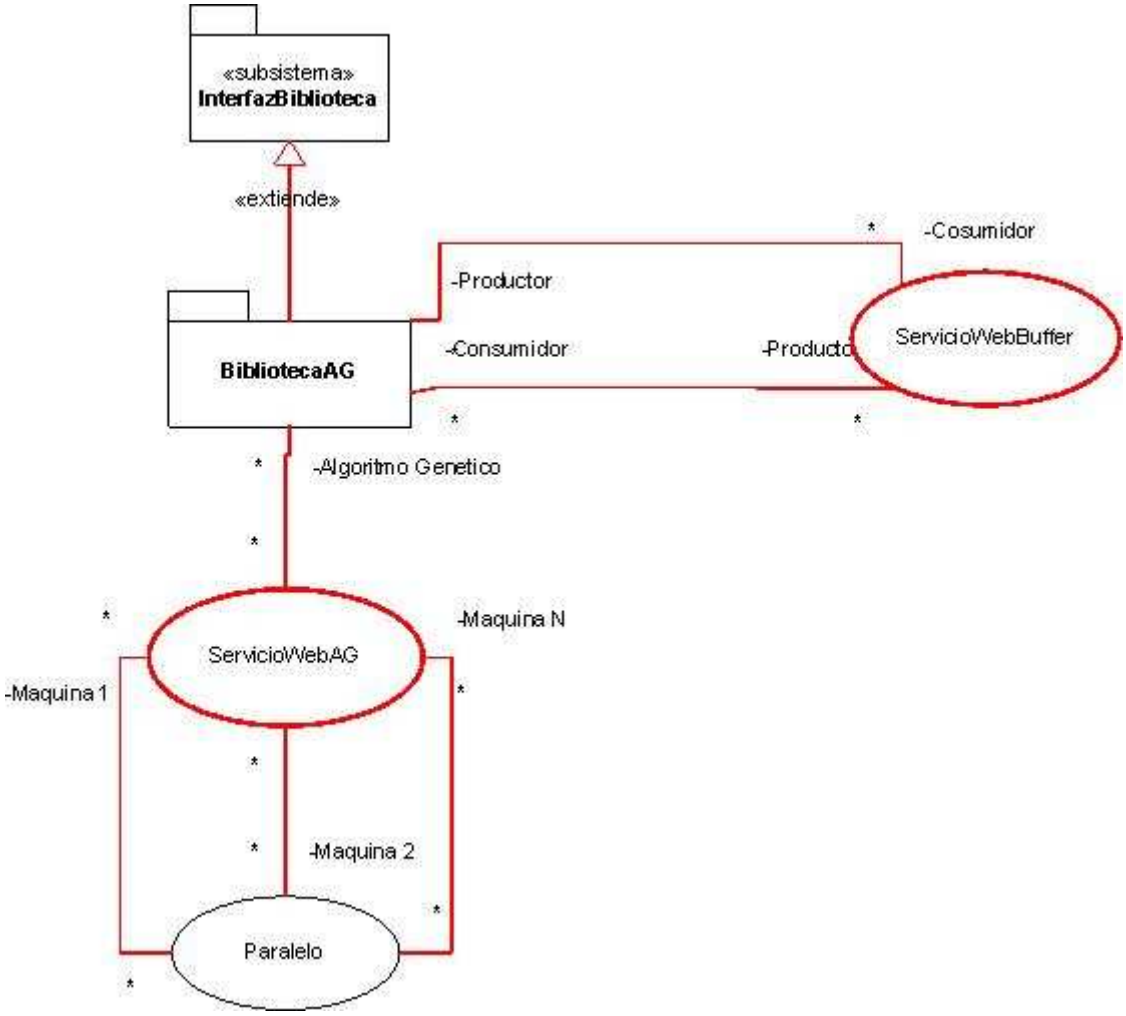
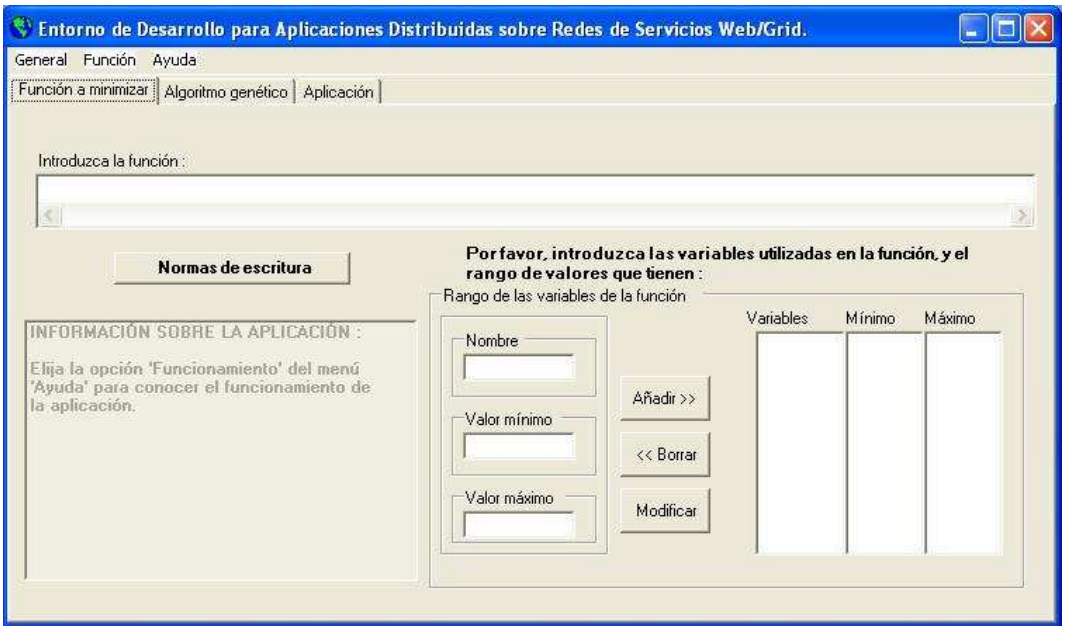


Figura 4. Esquema interno Paralelo

5.4. Interfaz Grafica

El objetivo del desarrollo de la interfaz gráfica ha sido crear una interfaz gráfica lo más amigable posible al usuario. Para ello además de la posibilidad de introducir el tamaño de la población, el número de generaciones, el umbral del criterio de parada etc. El usuario debe introducir en la aplicación la función de optimización del algoritmo genético así como las variables y sus rangos. Para ello el usuario debe cumplir unas normas de escritura de la función para facilitar el procesamiento del analizador sintáctico que comprueba que la función está correctamente escrita. (para más información sobre el analizador sintáctico, léase el *Anexo 3. Analizador Sintáctico*)

Para facilitar al usuario la introducción de la función de optimización y sus variables, existe la posibilidad de que éstos parámetros una vez escritos en la interfaz, puedan ser guardados en un archivo con extensión .txt de forma que el usuario no tenga que escribir la función cada vez que ejecute la aplicación o desee cambiar la función de optimización, sino que simplemente puede cargarlas del fichero en el que lo guardó. La escritura de estos ficheros también tiene unas normas de escritura que pueden verse en el apartado *6. Funcionamiento paso a paso.*



6. TECNOLOGÍAS UTILIZADAS

6.1. Plataforma.NET y C#

6.1.1. Introducción.

C# es un lenguaje (totalmente) nuevo desarrollado por Microsoft integrado en la plataforma .NET. Ésta es un conjunto de nuevas tecnologías que tienen como objetivo obtener una plataforma sencilla y potente para distribuir el software en forma de servicios (llamados servicios web) que puedan ser suministrados remotamente y que puedan comunicarse y combinarse unos con otros de manera totalmente independiente de la plataforma, lenguaje de programación y modelo de componentes con los que hayan sido desarrollados.

La plataforma .NET tiene capacidad para aceptar prácticamente cualquier lenguaje

¿Porqué se eligió C# en lugar de cualquier otro lenguaje? La razón fundamental es que C# es el único que se diseñó específicamente para la plataforma .NET y es capaz de utilizar todo su potencial. Por esta razón, se suele decir que C# es el lenguaje nativo de .NET. Además es un lenguaje "limpio" ya que carece de elementos heredados innecesarios en .NET al haber sido diseñado "desde cero" sin tener que proporcionar compatibilidad hacia atrás, por lo que se ha tenido más libertad en el diseño y se ha puesto especial hincapié en la simplicidad.

La sintaxis y estructuración de C# es muy similar a la C++, ya que la intención de Microsoft con C# es facilitar la migración de códigos escritos en estos lenguajes a C# y facilitar su aprendizaje a los desarrolladores habituados a ellos. Sin embargo, su sencillez y el alto nivel de productividad son equiparables a los de Visual Basic.

En resumen, C# es un lenguaje de programación que toma las mejores características de lenguajes preexistentes como Visual Basic, Java o C++ y las combina en uno solo.

De C++ toma :

- sintaxis (casi todos sus operadores, palabras reservadas y expresiones),
- sistema de tipos,
- posibilidad de trabajar con punteros (aunque el código C# que los utilice queda marcado como "no seguro" y no se ejecuta en entornos en los que no tiene permisos).

De Java toma :

- orientación a objetos
- ejecución en una máquina virtual (compilación a código intermedio)
- gestión automática de memoria
- renuncia a la herencia múltiple de clases presente en C++

A todos ellos añade :

- En cuanto al sistema de tipos :

- C# está más fuertemente tipado que C++, presenta un sistema unificado de tipos, en el que todos los tipos, incluso los primitivos, derivan de un tipo objeto común, a la vez que permite el uso de optimizaciones para tipos primitivos.
- Todos los tipos usados derivan en última instancia el tipo 'object',
- Muchos tipos se usan de forma distinta.

- Referido a clases
  - Implementa 'propiedades' del tipo de las que existen en Visual Basic, y los métodos de las clases son accedidos mediante '.' en lugar de '::'.

6.1.2. Características generales de C#.

Los creadores del C# (Anders Hejlsberg, Scott Wiltamuth y Peter Golde) describieron el lenguaje como "...simple, moderno, orientado a objetos, de tipado seguro y con una fuerte herencia de C/C++".

Las principales características de C# además de las ya mencionadas, son :

- Portabilidad :
  Ejecución en máquina virtual, generación de código intermedio, tamaño de los tipos de datos básicos es fijo e independiente del compilador, sistema operativo o máquina para quienes se compile.

- Sencillez :
  C# elimina muchos elementos que otros lenguajes incluyen y que son innecesarios en .NET. Por ejemplo:
  - El código escrito en C# es **autocontenido**, lo que significa que no necesita de ficheros adicionales al propio fuente tales como ficheros de cabecera o ficheros IDL
  - El tamaño de los tipos de datos básicos es fijo e independiente del compilador, sistema operativo o máquina para quienes se compile (no como en C++), lo que facilita la portabilidad del código.
  - No se incluyen elementos poco útiles de lenguajes como C++ tales como macros, herencia múltiple o la necesidad de un operador diferente del punto (`.`) acceder a miembros de espacios de nombres (`::`)

- Orientación a objetos :
  C# no admite ni funciones ni variables globales sino que todo el código y datos han de definirse dentro de definiciones de tipos de datos, lo que reduce problemas por conflictos de nombres y facilita la legibilidad del código. C# soporta todas las características propias del paradigma de programación orientada a objetos: **encapsulación**, **herencia** y **polimorfismo**.

- Orientación a componentes :
  La sintaxis de C# permite definir cómodamente **propiedades** (similares a campos de acceso controlado), **eventos** (asociación controlada de funciones de respuesta a notificaciones) o **atributos** (información sobre un tipo o sus miembros)

- Sistema de Tipos :
  - Seguridad de tipos : C# incluye mecanismos que permiten asegurar que los accesos a tipos de datos siempre se realicen correctamente, lo que permite evitar que se produzcan errores difíciles de detectar por acceso a memoria no perteneciente a ningún objeto y es especialmente necesario en un entorno gestionado por un recolector de basura. Para ello se toman medidas del tipo:
    - Sólo se admiten **conversiones entre tipos compatibles**. Esto es, entre un tipo y antecesores suyos, entre tipos para los que explícitamente se haya definido un operador de conversión, y entre un tipo y un tipo hijo suyo del que un objeto del primero almacenase una referencia del segundo (**downcasting**).
    - No se pueden usar **variables no inicializadas**. El compilador da a los campos un valor por defecto consistente en ponerlos a cero y controla mediante análisis del flujo de control del fuente que no se lea ninguna variable local sin que se le haya asignado previamente algún valor.
    - Se comprueba que todo **acceso a los elementos de una tabla** se realice con índices que se encuentren dentro del rango de la misma.
    - Se puede controlar la **producción de desbordamientos** en operaciones aritméticas, informándose de ello con una excepción cuando ocurra. Sin embargo, para conseguirse un mayor rendimiento en la aritmética estas comprobaciones no se hacen por defecto al operar con variables sino sólo con constantes (se pueden detectar en tiempo de compilación)
    - A diferencia de Java, C# incluye **delegados**, que son similares a los punteros a funciones de C++ pero siguen un enfoque orientado a objetos, pueden almacenar referencias a varios métodos simultáneamente, y se comprueba que los métodos a los que apunten tengan parámetros y valor de retorno del tipo indicado al definirlos.

      Pueden definirse métodos que admitan un número indefinido de parámetros de un cierto tipo, y a diferencia lenguajes como C/C++, en C# siempre se comprueba que los valores que se les pasen en cada llamada sean de los tipos apropiados.

  - Sistema de tipos unificado : A diferencia de C++, en C# todos los tipos de datos que se definan siempre derivarán, aunque sea de manera implícita, de una clase base común llamada **System.Object**, por lo que dispondrán de todos los miembros definidos en ésta clase (es decir, serán "objetos")

    A diferencia de Java, en C# esto también es aplicable a los tipos de datos básicos Además, para conseguir que ello no tenga una repercusión negativa en su nivel de rendimiento, se ha incluido un mecanismo transparente de **boxing** y **unboxing** con el que se consigue que sólo sean tratados como objetos cuando la situación lo requiera, y mientras tanto puede aplicárseles optimizaciones específicas.

    El hecho de que todos los tipos del lenguaje deriven de una clase común facilita enormemente el diseño de colecciones genéricas que puedan almacenar objetos de cualquier tipo.

  - Extensibilidad de tipos básicos : C# permite definir, a través de **estructuras**, tipos de datos para los que se apliquen las mismas optimizaciones que para los tipos de datos básicos.

- Extensibilidad de operadores : Para facilitar la legibilidad del código y conseguir que los nuevos tipos de datos básicos que se definan a través de las estructuras estén al mismo nivel que los básicos predefinidos en el lenguaje, al igual que C++ y a diferencia de Java, C# permite redefinir el significado de la mayoría de los operadores -incluidos los de conversión, tanto para conversiones implícitas como explícitas- cuando se apliquen a diferentes tipos de objetos.

  También se da la posibilidad, a través del concepto de **indizador**, de redefinir el significado del operador `[]` para los tipos de dato definidos por el usuario, con lo que se consigue que se pueda acceder al mismo como si fuese una tabla. Esto es muy útil para trabajar con tipos que actúen como colecciones de objetos.

- Reflexión : (*léase Anexo 2 : Reflexión*)

- Versionable :

  C# incluye una **política de versionado** que permite crear nuevas versiones de tipos sin temor a que la introducción de nuevos miembros provoquen errores difíciles de detectar en tipos hijos previamente desarrollados y ya extendidos con miembros de igual nombre a los recién introducidos. En C# se toman dos medidas:
  - Se obliga a que toda redefinición deba incluir el modificador **`override`**, con lo que la versión de la clase hija nunca sería considerada como una redefinición de la versión de miembro en la clase padre ya que no incluiría **`override`**. Para evitar que por accidente un programador incluya este modificador, sólo se permite incluirlo en miembros que tengan la misma signatura que miembros marcados como redefinibles mediante el modificador **`virtual`**. Así además se evita el error tan frecuente en Java de creerse haber redefinido un miembro, pues si el miembro con **`override`** no existe en la clase padre se producirá un error de compilación.
  - Si no se considera redefinición, entonces se considera que lo que se desea es ocultar el método de la clase padre, de modo que para la clase hija sea como si nunca hubiese existido. El compilador avisará de esta decisión a través de un mensaje de aviso que puede suprimirse incluyendo el modificador **`new`** en la definición del miembro en la clase hija para así indicarle explícitamente la intención de ocultación.

- Eficiente :

  En principio, en C# todo el código incluye numerosas restricciones para asegurar su seguridad y no permite el uso de punteros. Sin embargo, y a diferencia de Java, en C# es posible saltarse dichas restricciones manipulando objetos a través de punteros. Para ello basta marcar regiones de código como inseguras (modificador **`unsafe`**) y podrán usarse en ellas punteros de forma similar a cómo se hace en C++, lo que puede resultar vital para situaciones donde se necesite una eficiencia y velocidad procesamiento muy grandes.

- Compatible :

  Para facilitar la migración de programadores, C# no sólo mantiene una sintaxis muy similar a C, C++ o Java que permite incluir directamente en código escrito en C# fragmentos de código escrito en estos lenguajes, sino que el CLR también ofrece, a través de los llamados **Platform Invocation Services** (**PInvoke**), la

posibilidad de acceder a código nativo escrito como funciones sueltas no orientadas a objetos tales como las DLLs de la API Win32. Nótese que la capacidad de usar punteros en código inseguro permite que se pueda acceder con facilidad a este tipo de funciones, ya que éstas muchas veces esperan recibir o devuelven punteros.

También es posible acceder desde código escrito en C# a objetos COM. Para facilitar esto, el *.NET Framework SDK* incluye una herramientas llamadas **`tlbimp`** y **`regasm`** mediante las que es posible generar automáticamente clases proxy que permitan, respectivamente, usar objetos COM desde .NET como si de objetos .NET se tratase y registrar objetos .NET para su uso desde COM.

Finalmente, también se da la posibilidad de usar controles ActiveX desde código .NET y viceversa. Para lo primero se utiliza la utilidad **aximp**, mientras que para lo segundo se usa la ya mencionada **regasm**.

Librería del lenguaje : Contrariamente a la mayoría de lenguajes, C# no incluye una librería específica, sino que utiliza la librería de clases de la plataforma .NET para todas sus necesidades, desde utilización de la consola hasta la programación multihilo o el cifrado de seguridad.

Estandarización : Además de los méritos técnicos, uno de las razones del éxito de C# y la plataforma .NET ha sido por el proceso de estandarización que Microsoft ha seguido (y que ha sorprendido a más de uno). Microsoft, en lugar de reservarse todos los derechos sobre el lenguaje y la plataforma, ha publicado las especificaciones del lenguaje y de la plataforma, que han sido posteriormente revisadas y ratificadas por la Asociación Europea de Fabricantes de Computadoras (ECMA). Esta especificación (que se puede descargar libremente en el http://www.ecma-international.org/publications/standards/Ecma-334.htm) permite la implementación del lenguaje C# y de la plataforma .NET por terceros, incluso en entornos distintos de Windows.

6.1.3. Algunas características específicas de C#.

Espacios de nombres :

Del mismo modo que los ficheros se organizan en directorios, los tipos de datos se organizan en **espacios de nombres**.

Por un lado estos espacios permiten tener más organizados los tipos de datos, lo que facilita su localización. De hecho, esta es la forma en que se encuentra organizada la BCL, de modo que todas las clases más comúnmente usadas en cualquier aplicación pertenecen al espacio de nombres llamado **`System`**, las de acceso a bases de datos en **`System.Data`**, las de realización de operaciones de entrada/salida en **`System.IO`**, etc

Por otro lado, los espacios de nombres también permiten poder usar en un mismo programa varias clases con igual nombre si pertenecen a espacios diferentes. La idea es que cada fabricante defina sus tipos dentro de un espacio de nombres propio para que así no haya conflictos si varios fabricantes definen clases

con el mismo nombre y se quieren usar a la vez en un mismo programa. Obviamente para que esto funcione no han de coincidir los nombres los espacios de cada fabricante, y una forma de conseguirlo es dándoles el nombre de la empresa fabricante, o su nombre de dominio en Internet, etc.

El verdadero nombre de una clase, al que se denomina **nombre completamente calificado**, es el nombre que le demos al declararla prefijado por la concatenación de todos los espacios de nombres a los que pertenece ordenados del más externo al más interno y seguido cada uno de ellos por un punto (carácter **.**)

Si no definimos una clase dentro de una definición de espacio de nombres, se considera que ésta pertenece al denominado **espacio de nombres global** y su nombre completamente calificado coincidirá con el nombre que le demos al definirla.

Aparte de definiciones de tipo, también es posible incluir como miembros de un espacio de nombres a otros espacios de nombres.

Delegados y eventos :

Un **delegado** es un tipo especial de clase cuyos objetos pueden almacenar referencias a uno o más métodos de tal manera que a través del objeto sea posible solicitar la ejecución en cadena de todos ellos.

Los delegados son muy útiles ya que permiten disponer de objetos cuyos métodos puedan ser modificados dinámicamente durante la ejecución de un programa. De hecho, son el mecanismo básico en el que se basa la escritura de aplicaciones de ventanas en la plataforma .NET. Por ejemplo, si en los objetos de una clase `Button` que represente a los botones estándar de Windows definimos un campo de tipo delegado, podemos conseguir que cada botón que se cree ejecute un código diferente al ser pulsado sin más que almacenar el código a ejecutar por cada botón en su campo de tipo delegado y luego solicitar la ejecución todo este código almacenado cada vez que se pulse el botón.

Sin embargo, también son útiles para muchísimas otras cosas tales como asociación de código a la carga y descarga de ensamblados, a cambios en bases de datos, a cambios en el sistema de archivos, a la finalización de operaciones asíncronas, la ordenación de conjuntos de elementos, etc. En general, son útiles en todos aquellos casos en que interese pasar métodos como parámetros de otros métodos.

Además, los delegados proporcionan un mecanismos mediante el cual unos objetos pueden solicitar a otros que se les notifique cuando ocurran ciertos sucesos. Para ello, bastaría seguir el patrón consistente en hacer que los objetos notificadores dispongan de algún campo de tipo delegado y hacer que los objetos interesados almacenen métodos suyos en dichos campos de modo que cuando ocurra el suceso apropiado el objeto notificador simule la notificación ejecutando todos los métodos así asociados a él.

Un **evento** es una variante de las propiedades para los campos cuyos tipos sean delegados. Es decir, permiten controlar la forman en que se accede a los campos delegados y dan la posibilidad de asociar código a ejecutar cada vez que se añada o elimine un método de un campo delegado.

6.2.Servicios Web con .NET

6.2.1. Introducción al Servicio Web

Un servicio Web es una clase que se publica en un servidor Web con soporte para ASP.NET. Presenta la particularidad de que sus métodos pueden ser llamados remotamente. Estas clases pueden escribirse en la mayoría de los lenguajes adaptados a .NET.

Para acceder a un servicio Web se pueden utilizar varios protocolos Web estándar, como HTTP GET o HTTP POST, aunque el específicamente diseñado para ello por Microsoft es el Protocolo de Acceso a Objetos Simple (SOAP), que se basa en la utilización del protocolo HTTP para el transporte de mensajes y el lenguaje XML para la escritura del cuerpo de estos mensajes.

Una de las grandes ventajas de los servicios Web es que pueden ser accedidos desde cualquier aplicación que sea capaz de generar mensajes e interpretar mensajes escritos en SOAP, aún cuando ésta no esté diseñada para la plataforma .NET. Es más, las aplicaciones que usen estos servicios no necesitan conocer cuál es la plataforma ni cuál es el modelo de objetos ni cuál es el lenguaje utilizado para implementar estos servicios. Otra gran ventaja es su sencillez de escritura. También hay que tener en cuenta la facilidad y transparencia con la que aquellos clientes del servicio escritos bajo .NET pueden acceder al mismo, gracias a las utilidades generadoras de proxys.

Cada WebService tiene dos archivos asociados: uno con extensión "asmx" y otro con extensión "cs", en lenguaje C#.

En la arquitectura .NET, WebServices siempre se implementa en una clase derivada de "System.Web.Services.WebService". En esta clase agregamos las funciones (métodos) que serán llamadas mediante SOAP. La diferencia entre WebMethod y un método común, es la presencia de un "atributo WebMethod", una especie de política de compilación.

6.2.2. Soap

SOAP es una parte importante de la arquitectura .NET de Microsoft y cuenta con amplio soporte en Visual Studio.NET. Se trata de un protocolo de comunicaciones para los servicios XML Servicios Web XML

Es el protocolo diseñado por Microsoft para el acceso a los servicios Web. Para la comunicación con los servicios Web podríamos utilizar mensajes escritos en este protocolo, interpretando sus respuestas. Para evitar este trabajo se trabaja con un mecanismo mucho más sencillo y transparente basado en el uso de proxys.

Sólo define el formato XML para mensajes. Existen otras partes en la especificación SOAP que describen cómo representar los datos de un programa

como XML y cómo utilizar SOAP para realizar llamadas a procedimiento remoto (RPC). Estas partes opcionales de la especificación se utilizan para implementar aplicaciones estilo RPC en las que el cliente envía un mensaje SOAP que contiene una función a la que se puede llamar, además de los parámetros para pasar a la función. El servidor, por su parte, devuelve un mensaje con los resultados de la función ejecutada.

Una parte opcional de la especificación SOAP define los contenidos de un mensaje HTTP que contiene un mensaje SOAP. HTTP es un buen soporte porque casi todos los sistemas operativos actuales admiten HTTP. Pero HTTP no es necesariamente el soporte para una comunicación con SOAP. Existen implementaciones que admiten protocolos tan dispares como MSMQ, MQ Series, SMTP o TCP/IP. Pero la principal ventaja de HTTP, frente a los demás es que las empresas tienen ya las infraestructuras de redes y el personal necesario para gestionarlo. Como añadido, HTTP en la actualidad dispone ya de vías para controlar la seguridad, el control y el balance de cargas entre servidores, por lo que se vuelve un protocolo muy versátil.

6.2.3. Proxy

Un proxy es una clase que ofrece la misma interfaz que el servicio Web al que se pretende acceder pero que está almacenada en la máquina local y no contiene la verdadera implementación de los métodos ofrecidos por el servicio, sino que en su lugar contiene código encargado de redirigir las llamadas que hagamos a sus métodos al verdadero servicio Web. Gracias a los proxys, conseguimos comunicarnos con el servicio remoto como si fuese una clase común , y el proxy el encargado de intercambiar los mensajes SOAP necesarios para la comunicación remota.

Para generar automáticamente el proxy es necesario contar con una especificación del servicio (métodos ofrecidos, signatura de estos, protocolos soportados, etc.) que sea fácil de interpretar por una máquina. Para ello se utiliza un fichero XML escrito en el llamado “Lenguaje Descriptor del Servicio Web”(WSDL) Para obtener este fichero sólo hay que acceder al servicio Web concatenado la cadena “¿WSDL “ al final del mismo.

Gracias a este tipo de ficheros, podemos generar el proxy de manera automática utilizando la utilidad wsdl.exe incluida en el .NET SDK.

A partir del proxy la escritura de una aplicación que haga el uso del servicio es sencilla. Sólo hay que usar el proxy como si se tratase de una clase remota.

6.2.4. WSDL

Es el lenguaje que permite la definición de la interfaz con un servicio Web. Con él ponemos de manifiesto un contrato que define el modo en que podemos interactuar con el Servicio. Aunque en el ámbito que nos ocupa está ligado a SOAP, en realidad puede expresar enlaces a protocolos diferentes. WSDL es un protocolo fundamental en las arquitecturas actuales de Servicios Web, y es de

hecho un estándar que admiten todos los fabricantes que se han interesado por la plataforma.

6.2.5. Mantenimiento del estado

Por defecto un servicio Web carece de estado, ya que por cada llamada a un método de un servicio se crea un nuevo objeto de la clase a la que se hace la petición; objeto que la atiende y es destruido tras ello. Por consiguiente, no se guarda información sobre llamadas previas, y por tanto los objetos de un servicio Web escrito así no son verdaderos objetos, ya que lo que realmente el cliente hace son llamadas sueltas que no tienen relación entre sí.

Para conseguir almacenar información sobre llamadas previas, la clase WebService de la que todo servicio Web hereda proporciona unos objetos llamados Application y Session, de clase httpApplicationState y HttpSessionState respectivamente, que actúan como depósitos de información sobre llamadas previas. El primero de estos objetos, que es tratado en el trabajo, permite almacenar información que será compartida por todas las aplicaciones clientes del servicio, mientras que la información almacenada en el segundo sólo es compartida entre sucesivas llamadas de un mismo cliente.

6.3. Redes Web/Grid

6.3.1. Introducción a redes Web/Grid

La informática Grid está basada en un tipo de conectividad de redes que, a diferencia de las convencionales, aprovecha los ciclos de proceso que no usan los diferentes dispositivos informáticos para llevar a cabo operaciones que requieren de una gran potencia de procesamiento e involucran una ingente cantidad de información. Básicamente, las redes Grid tratan de conectar múltiples sistemas heterogéneos de forma que los recursos e información almacenada en los mismos se puedan compartir. Así, se consigue optimizar los recursos al ponerlos en común para cargas de trabajo de gran capacidad y facilita la colaboración de empresas con socios comerciales y proveedores, así como la participación de diferentes organizaciones dispersas geográficamente en un mismo proyecto.

Las redes Web/Grid conforman una nueva generación de Internet donde los recursos están permanentemente disponibles y la información se intercambia en tiempo real gracias a la conexión de múltiples ordenadores. Las redes Grid se conciben como un cluster distribuido a gran escala que actúa como una nueva forma de informática paralela para entornos distribuidos. Este entramado está dotado de una serie de funciones de control capacitadas para comprender las necesidades de los recursos, identificar dinámicamente aquellos que están disponibles y localizarlos dentro de la red.

Web/Grid tiene que ver con la coordinada compartición de recursos y resolución de problemas en organizaciones virtuales dinámicas y multiinstitucionales, entendiéndose por compartición no sólo intercambio de ficheros sino también acceso a ordenadores, datos y otros recursos, de manera altamente controlada. El conjunto de instituciones y usuarios que comparten estos recursos y se someten a las reglas de compartición forman una organización virtual.

6.3.2. Aspectos indispensables de una red Web/Grid

La idea de Grid es que sean arquitecturas fiables, consistentes y accesibles. Por ello, cinco características indispensables están presentes en ellas:

- Uniformidad: el usuario de Web/Grid ha de ver los diferentes datos y recurso como un único recurso.
- Transparencia: los datos en los diferentes formatos y tipos de ficheros han de ser integrados, de manera que el usuario pueda manejarlos con independencia de la fuente de que provengan.
- Fiabilidad: disponibilidad permanente, que exige especial cuidado en la tolerancia a fallos y la redundancia.
- Ubicuidad: recursos disponibles para la mayor cantidad posible de usuarios, lo que implica un software de comunicaciones que resida físicamente

en el cliente remoto y en un servidor de comunicaciones localizado entre cliente y servidor de aplicaciones, que esté preparado para soportar una amplia diversidad de plataformas y sistemas operativos.

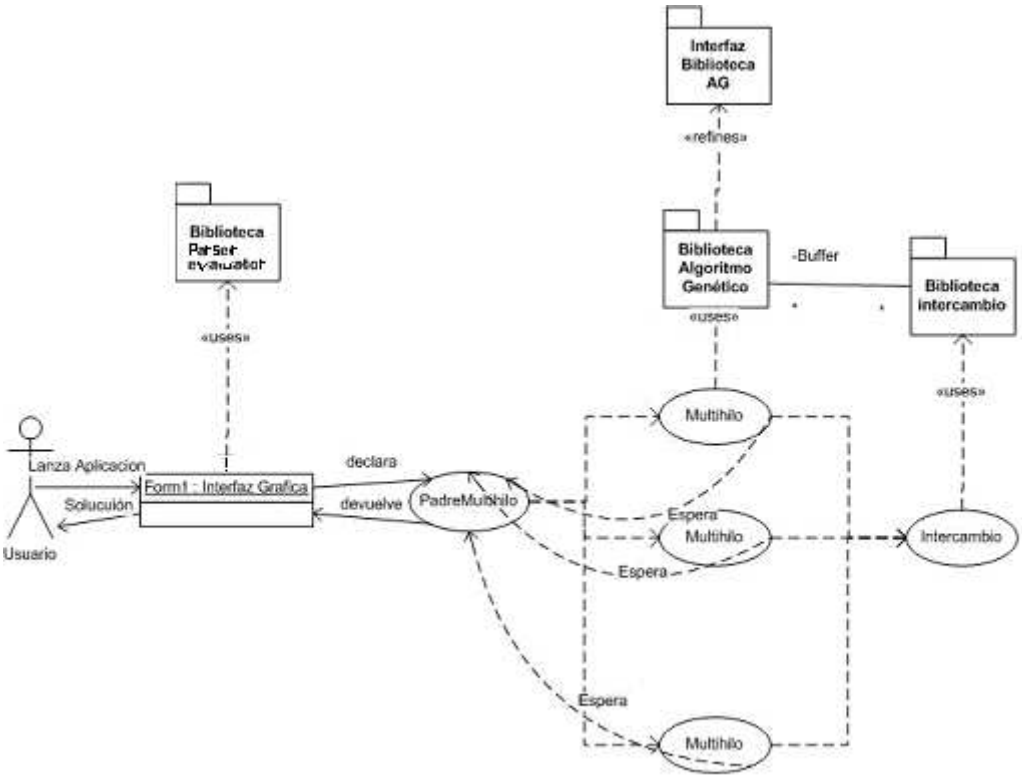
- Seguridad: es fundamental tanto en los datos como en el acceso a los recursos compartidos.

7. DISEÑO

7.1. Diagrama de Casos de Uso

7.1.1. Caso de Uso : Multihilo

A través del siguiente diagrama intuitivo se puede ver esquemáticamente el comportamiento de la aplicación en el caso de Multihilo, teniendo presente las bibliotecas donde se definen el algoritmo genético y el buffer de intercambio, así como la biblioteca del Parser donde se encuentran todas las clases necesarias para hacer el analizador sintáctico (l*éase ANEXO 3 : Analizador Sintáctico*).



El usuario después de introducir los datos del algoritmo genético, así como la función y sus restricciones, selecciona el número de hilos que desea ejecutar. El primer paso de la aplicación es analizar si la función a minimizar es correcta sintácticamente, así como la corrección de las restricciones introducidas. De ello se encarga el parser evaluador. Si los datos han sido introducidos correctamente se ejecutarán los hilos. Estos son lanzados a través del "PadreMultihilo" que es el encargado de lanzar los hilos y esperar a que acaben, si no ocurre ningún error. Cada uno de estos hilos desarrolla todo el algoritmo genético, por la Biblioteca Algoritmo Genético (*léase ANEXO 1 : Algoritmo genético*), donde se encuentran todas las clases necesarias para el desarrollo de éste algoritmo. Cabe destacar

dentro de esta biblioteca la utilización de la clase Evaluator, que tiene un comportamiento muy particular. Esto es debido a que es el encargado de evaluar la función para cada individuo, y por ello debe crear la clase en tiempo de ejecución (*léase ANEXO 2 : Reflexión*).

Vemos que Biblioteca Algoritmo Genético redefine la interfaz Biblioteca AG. Esto se debe a que el comportamiento del Algoritmo genético es muy similar tanto en la ejecución Paralela como en la Multihilo. Pero, sin embargo presentan ciertas peculiaridades, por lo que a la hora del diseño nos pareció una forma óptima tanto por su claridad como por el funcionamiento.

El padre multihilo devuelve el resultado a la interfaz gráfica donde el usuario podrá ver el mejor resultado que han calculado los hilos. Si ocurre algún error, el padre Multihilo se encarga de detener la ejecución del hilo que tuvo el problema, y devuelve a la interfaz gráfica el resultado mejor de entre los hilos que se han ejecutado correctamente y el tiempo que ha tardado en ejecutarse, así como el mejor resultado de cada uno de los hilos.

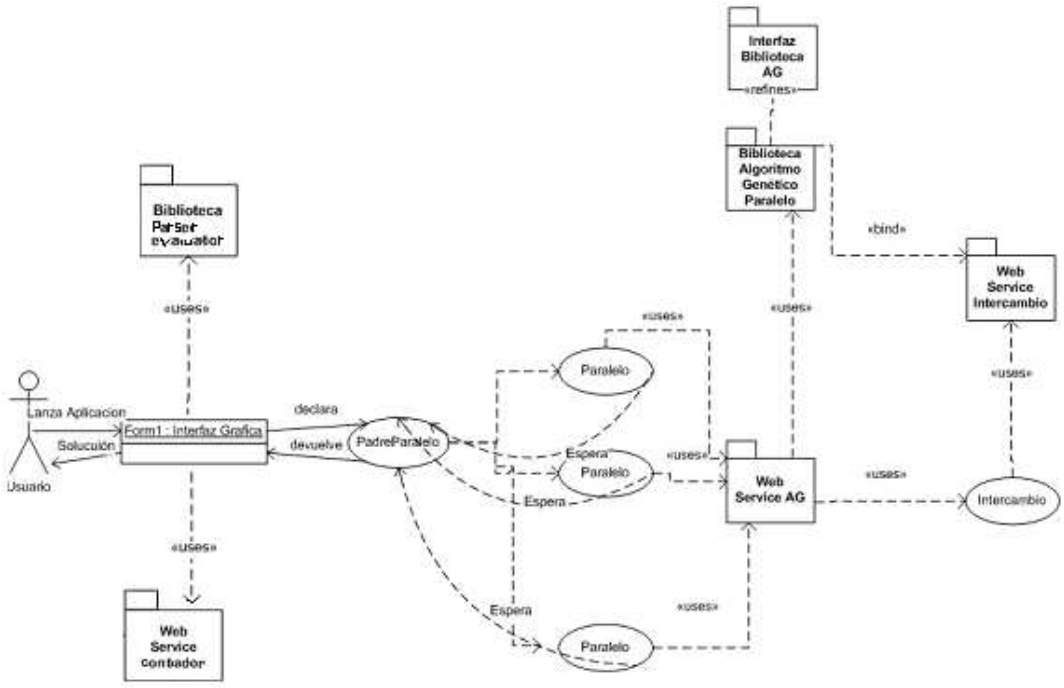
Cada algoritmo genético lanzado en un hilo, cada cierto tiempo intercambia su mejor individuo con el mejor de un buffer dónde cada algoritmo genético pone su mejor individuo, los algoritmos en otro hilo lanzan el intercambio, y con filosofía de productor consumidor, recogerán el dato cuando esté disponible, pero no esperando a que se realice el intercambio sino comprobando cada cierto tiempo que ya pueden leer del buffer.

7.1.2. Caso de Uso : Paralelo

Para este caso también se tiene un diagrama intuitivo donde puede observarse a grandes rasgos el comportamiento de la aplicación. En él apreciamos la similitud con el caso Multihilo y se ve cómo es necesario que la Interfaz Gráfica llame a Web Service Contador para estudiar cuáles son las máquinas disponibles en ése momento.

Se aprecia como sigue el mismo patrón de funcionamiento, así como la utilidad de la utilización del Interfaz del algoritmo Genético en relación con el funcionamiento de la aplicación en Multihilo.

También es destacable como cada uno de los Algoritmos genéticos se desarrolla en un Web Service AG (l*éase TECNOLOGÍAS UTILIZADAS: 4.2 WebServices*).



Cuando el usuario selecciona el modo de aplicación paralelo, Debe elegir de entre las máquinas que están disponibles, aquellas en las que desee que se ejecute el algoritmo genético.

Las máquinas disponibles son aquellas en las que el Servicio Web AG (Algoritmo Genético) está instalado y es accesible. Para mirar estas máquinas disponibles se vale de Web Service Contador. Al igual que en el caso Multihilo el

siguiente paso a dar es la comprobación de que la función introducida sea correcta y que exista una coherencia entre ésta y las variables introducidas así como sus restricciones.

Una vez seleccionadas las máquinas se crea un hilo que lanza a "Padre Paralelo" que es el encargado de inicializar el Servicio Web Buffer (Intercambio), que recordemos conserva el estado y por tanto es necesaria su inicialización cada vez que ejecutamos la aplicación, además este Servicio Web sólo debe estar instalado en una máquina ya que sólo se necesita una instancia del Servicio Web.

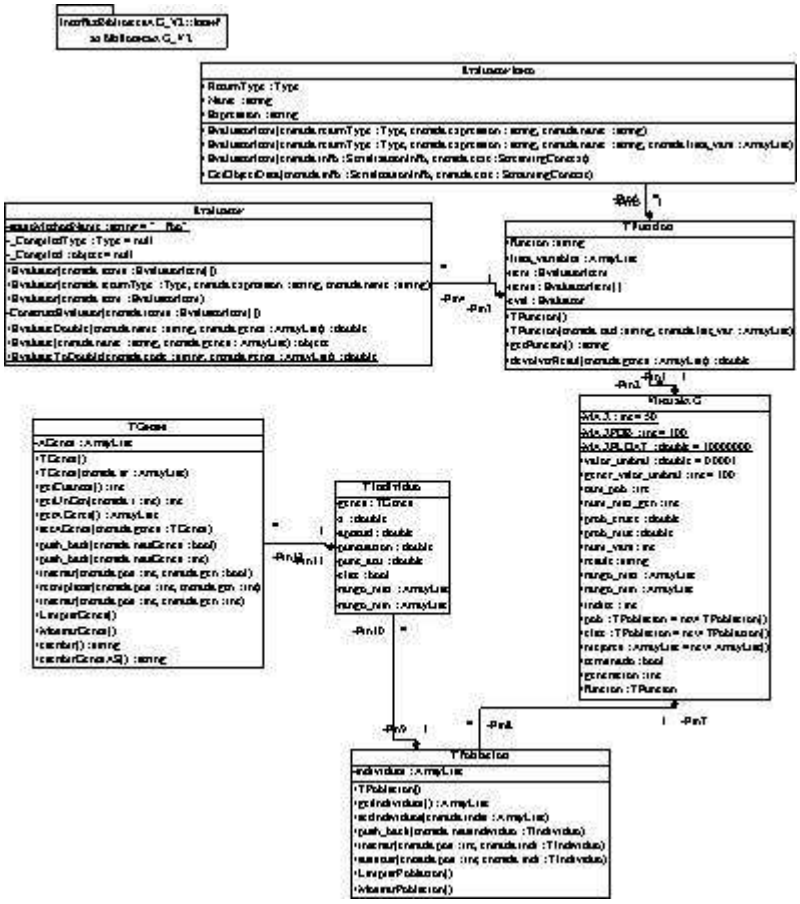
Por otro lado padre paralelo crea instancias de "Paralelo", en los cuales se crean objetos de Servicio Web AG donde se le pasa la única instancia de Servicio Web Buffer que se declara (y que existe). Por tanto todos los Servicios Web AG llaman al mismo Web Service Intercambio.

El comportamiento de la aplicación en su caso de paralelo, es análogo al caso de multihilo. Por lo tanto padre Paralelo también espera a que los Servicios Web terminen su ejecución y devolver el resultado a la interfaz gráfica para que el usuario pueda ver el mejor resultado, así como el tiempo empleado en la ejecución de la aplicación. Al igual que en el caso anterior, si el usuario lo desea puede ver los resultados obtenidos en cada una de las subpoblaciones.

7.2. Diagrama de Clases

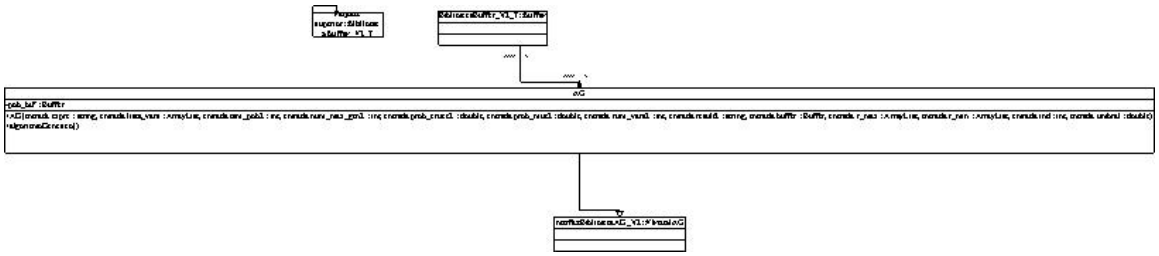
Proyecto principal :

Interfaz de la biblioteca algoritmo genético para las dos formas de ejecución de la aplicación :



En la ejecución Multihilo :

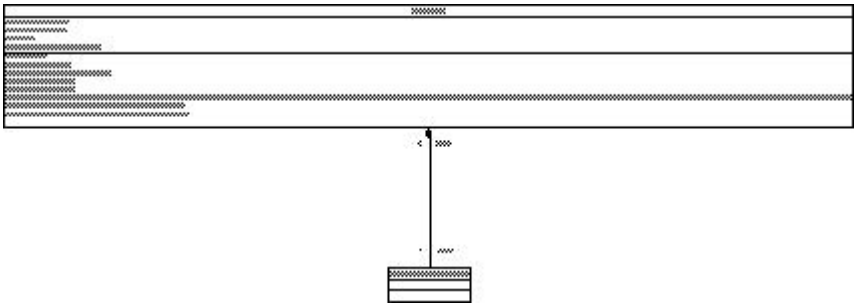
- BibliotecaAGBuffer :



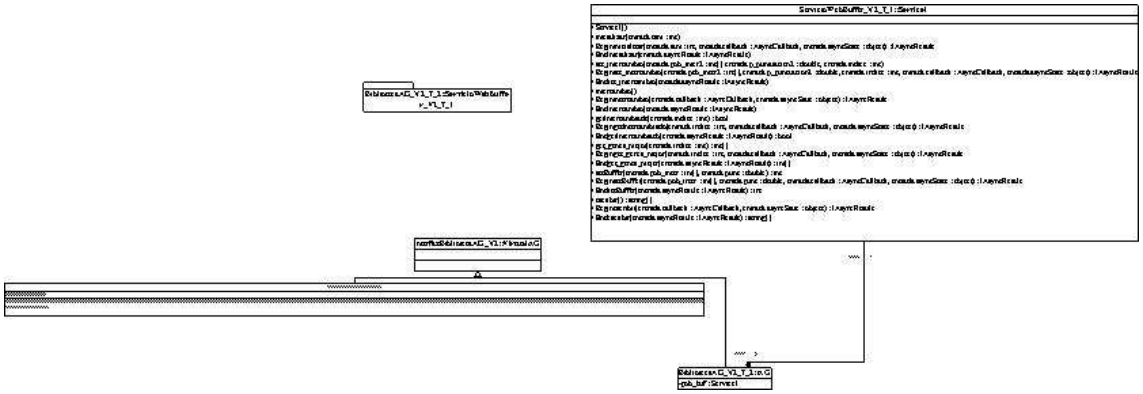
- BibliotecaBuffer :

En la ejecución Paralela :

- ServicioWebAG (Algoritmo Genético) :



- BibliotecaAG :



- ServicioWebBuffer :

8. FUNCIONAMIENTO PASO A PASO

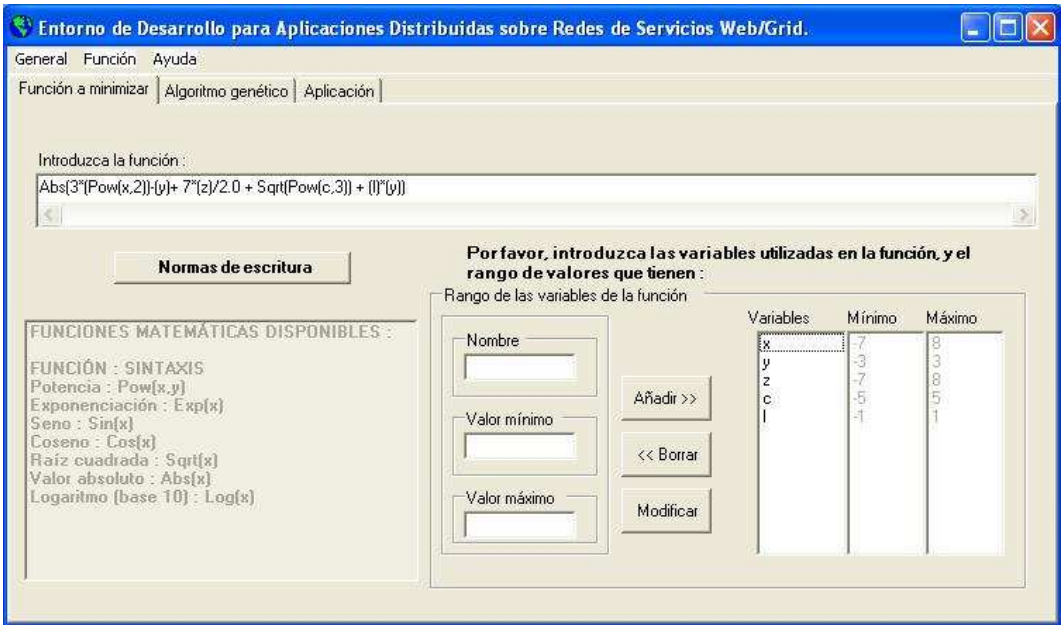
8.1. Introducción de parámetros

Al iniciar la aplicación se muestra un formulario en el que hay tres áreas claramente diferenciadas mediante pestañas, y un menú muy simple.

En la pestaña "Función a minimizar" el usuario debe introducir manualmente la función a minimizar así como las variables que en ella intervienen y el rango de valores que toman dichas variables:

La función debe cumplir ciertas normas de escritura que se citan a continuación:

```
FUNCIONES MATEMÁTICAS DISPONIBLES: NORMAS DE ESCRITURA
    FUNCIÓN: SINTAXIS
    Potencia  Pow(x,y)
    Exponenciación: Exp(x)
    Seno: Sin(x)
    Coseno: Cos(x)
    Raíz cuadrada: Sqrt(x)
    Valor absoluto: Abs(x)
    Logaritmo (base 10): Log(x)
```



Además, para una mayor comodidad se pueden leer todos los datos de la función a minimizar desde un fichero a través del menú "Función->Cargar" y también guardarlos para un uso posterior (menú "Función->Salvar"). Los ficheros que contengan los datos de la función a minimizar deben estar escritos en un archivo .txt con un formato determinado que puede ser consultado en el menú "Función->Formato en ficheros".

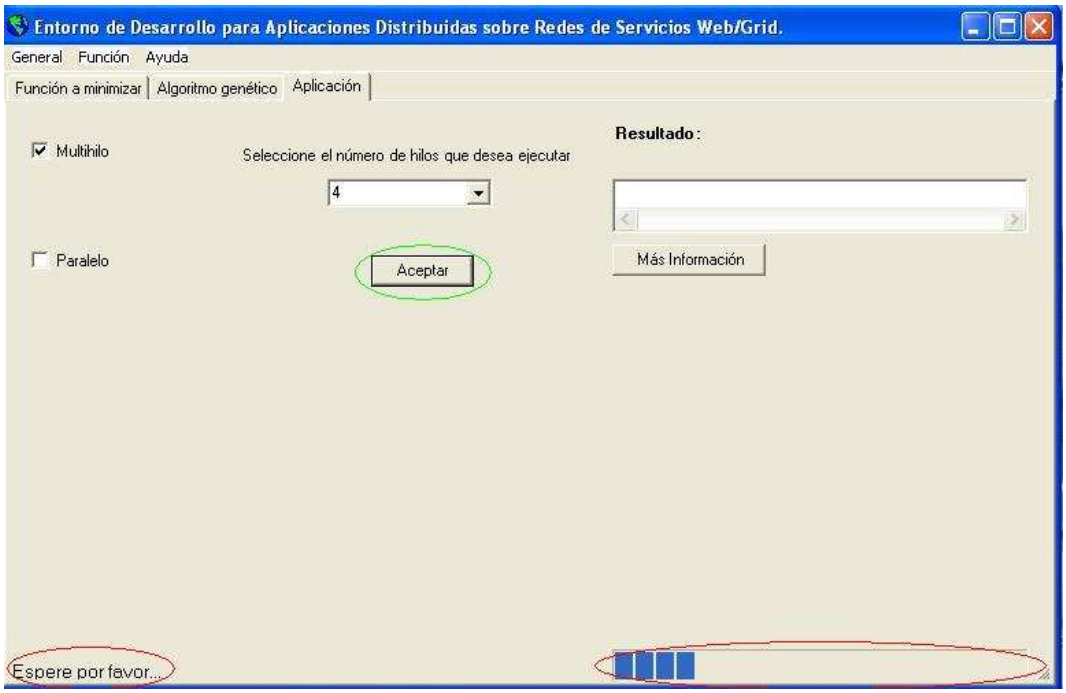
El usuario debe rellenar todos los datos de la función para poder continuar con la ejecución de la aplicación.

En la siguiente pestaña "Algoritmo genético" el usuario tiene la posibilidad de dar valores a los parámetros del algoritmo genético que se usará para la minimización. No obstante para facilitar trabajo al usuario, por defecto estos parámetros tendrán unos valores predeterminados.

Por último, y principal, la pestaña "Aplicación" permite al usuario elegir entre las dos formas de : multihilo y paralelo.

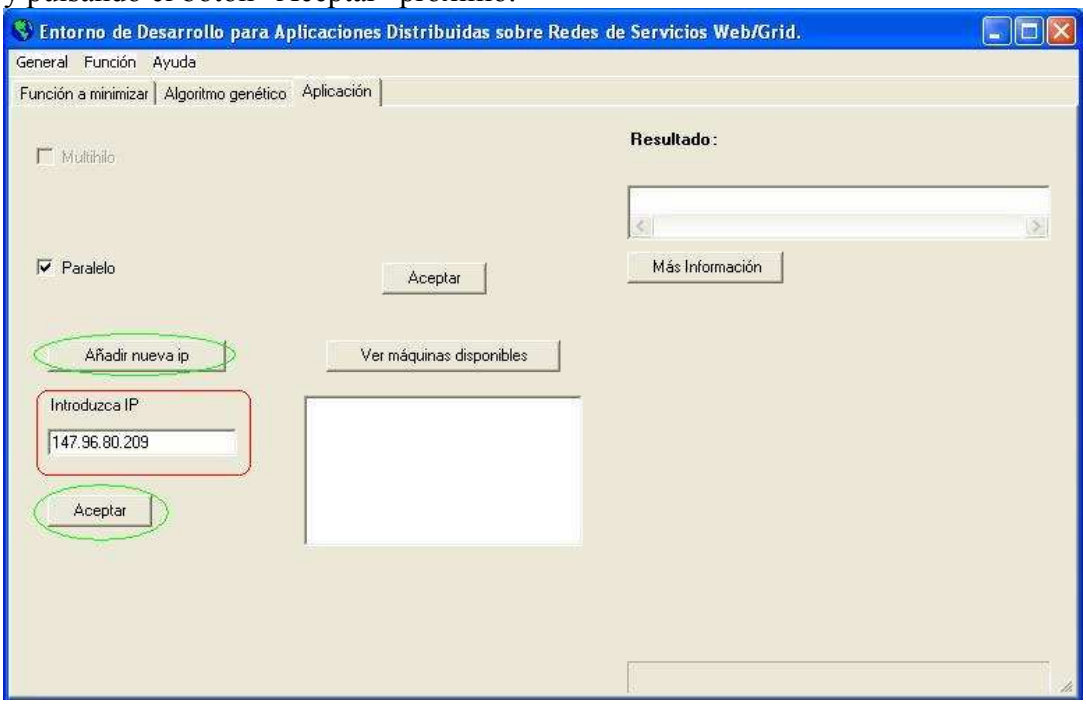
8.2.Multihilo

En **Multihilo** el usuario debe elegir el número de hilos sobre los que lanzar el algoritmo de genético de minimización, y a continuación pulsar el botón "Aceptar", momento en el cuál comenzará el proceso de minimización.

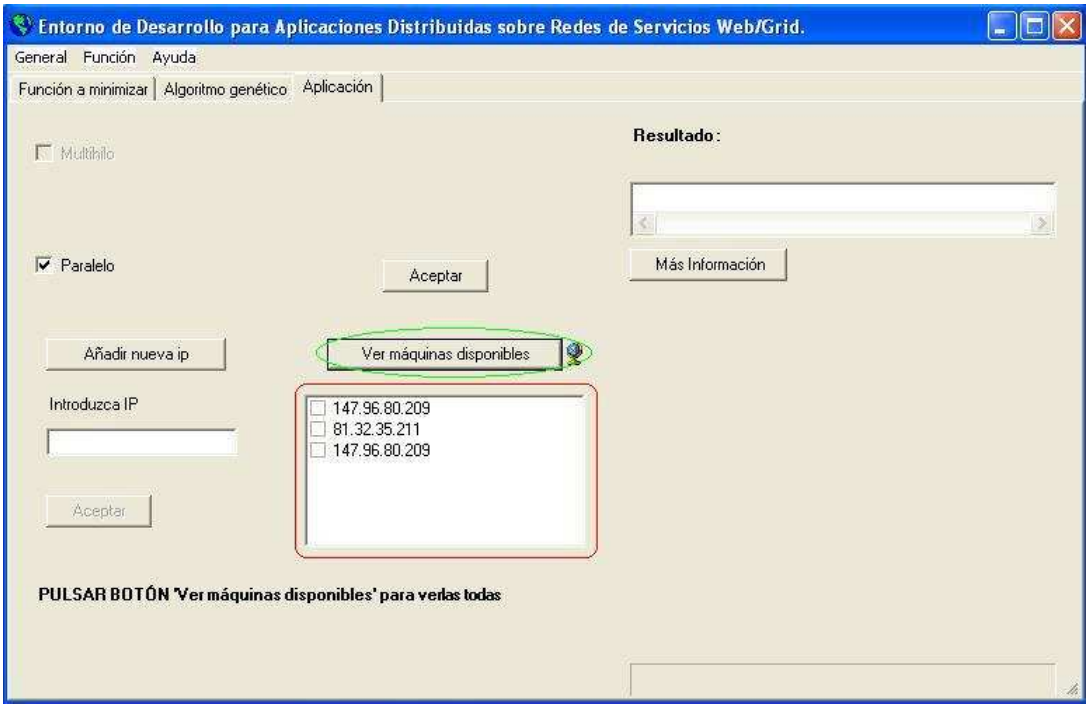


8.3. Paralelo

En **Paralelo** el usuario puede introducir las direcciones IP sobre las que desea lanzar la aplicación mediante el botón "Añadir nueva ip", escribiendo la IP y pulsando el botón "Aceptar" próximo.

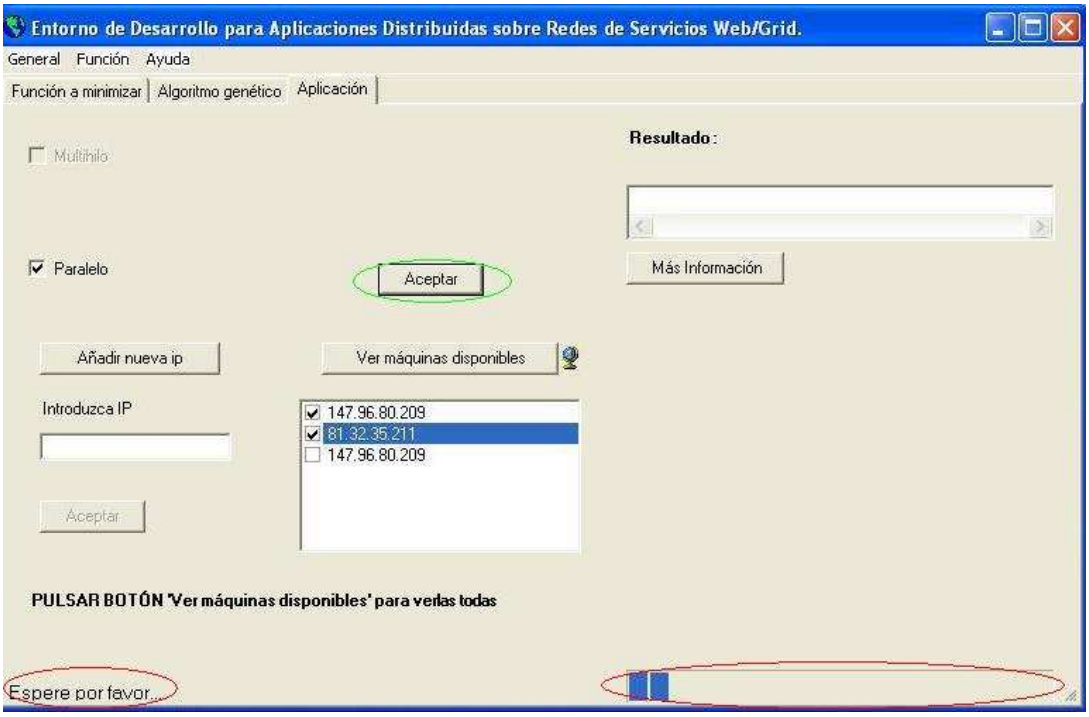


Además, puede ver las máquinas que hay disponibles para lanzar la aplicación pulsando el botón "Ver máquinas disponibles".



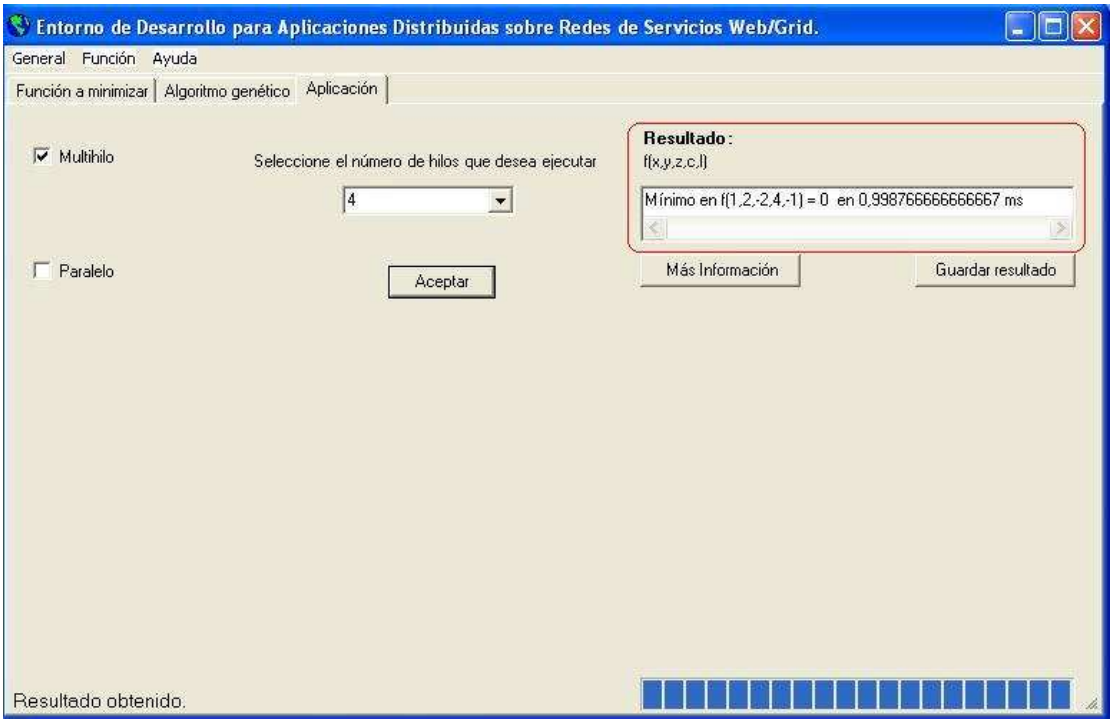
Si introdujo una IP que no está operativa en el momento de lanzar la aplicación, no se mostrará en la lista de máquinas disponibles.

Sobre la lista de máquinas disponibles el usuario puede seleccionar aquellas sobre las que desea lanzar la aplicación, y a continuación pulsar el botón "Aceptar", momento en el cuál comenzará el proceso de minimización.

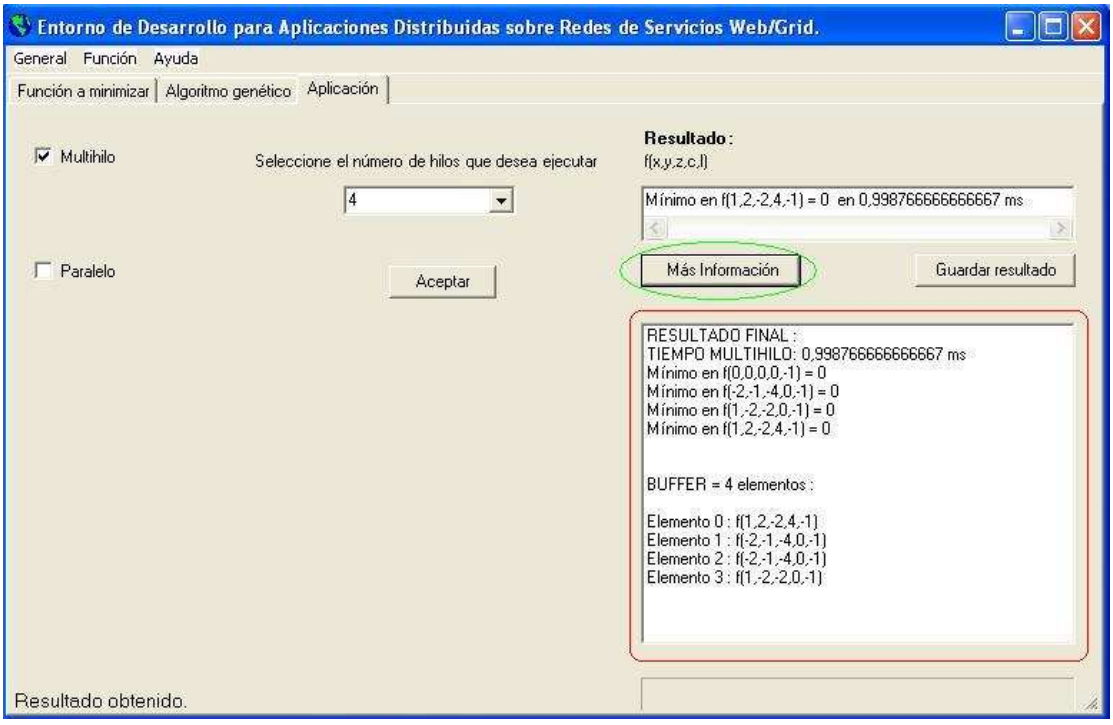


8.4.Solución de la aplicación

Una vez que la aplicación ha calculado el mínimo que buscaba, tanto en multihilo como en paralelo, el usuario puede verlo en la pestaña aplicación :



Y además si desea más información sobre los resultados intermedios que el algoritmo ha ido calculando, puede visualizarla pulsando el botón "Más información" :



9. CONCLUSIÓN Y PRUEBAS

9.1. Características de las máquinas.

La plataforma computacional para la ejecución Multihilo corresponde a un procesador Mobile Intel Celeron a 2,4 GHz con 256 MB memoria RAM DDR. Fue compilado con sistema operativo Windows XP. La ejecución Multihilo se basó en el lanzamiento de tres hilos en esta máquina.

Para la ejecución Paralela además de a este equipo, se envió otro hilo a un Servicio Web de la máquina con procesador Xeon a 2,4GHz con 2GB de RAM (encontrándose en este equipo el Servicio Web de Intercambio) y otro con procesador Pentium III a 1 Ghz con 256 MB de RAM. Destacamos que el proceso centralizador se lleva acabo en la máquina más rápida.

9.2. Datos utilizados para las pruebas.

Cabe destacar que para un correcto funcionamiento del algoritmo basta con utilizar un tamaño de población 200 y un rango de revisión umbral de 90 generaciones para la mayor parte de las funciones introducidas, pero para prolongar la duración de la ejecución y poder obtener unos resultados de los que se pudiera derivar una conclusiones, esto parámetros tuvieron que ser aumentados.

La elección de los nuevos valores para las pruebas se basó en la búsqueda de una función y unas restricciones para la misma que hicieran que la máquina tardara un tiempo apreciable. A partir de esta función con cierta complejidad y sabiendo lo que tenía que obtenerse de resultado, se dilató el valor umbral para la obtención de un resultado lo más ajustado posible. Después de diversas ejecuciones sobre esos datos se tomó el mayor de los tiempos de ejecución de cada uno de los casos.

Las funciones elegidas son todas funciones no lineales, con un número diverso de variables. Las restricciones introducidas para las mismas también se fueron aumentando para que la aplicación tardara más tiempo en encontrar un mínimo para la función determinada.

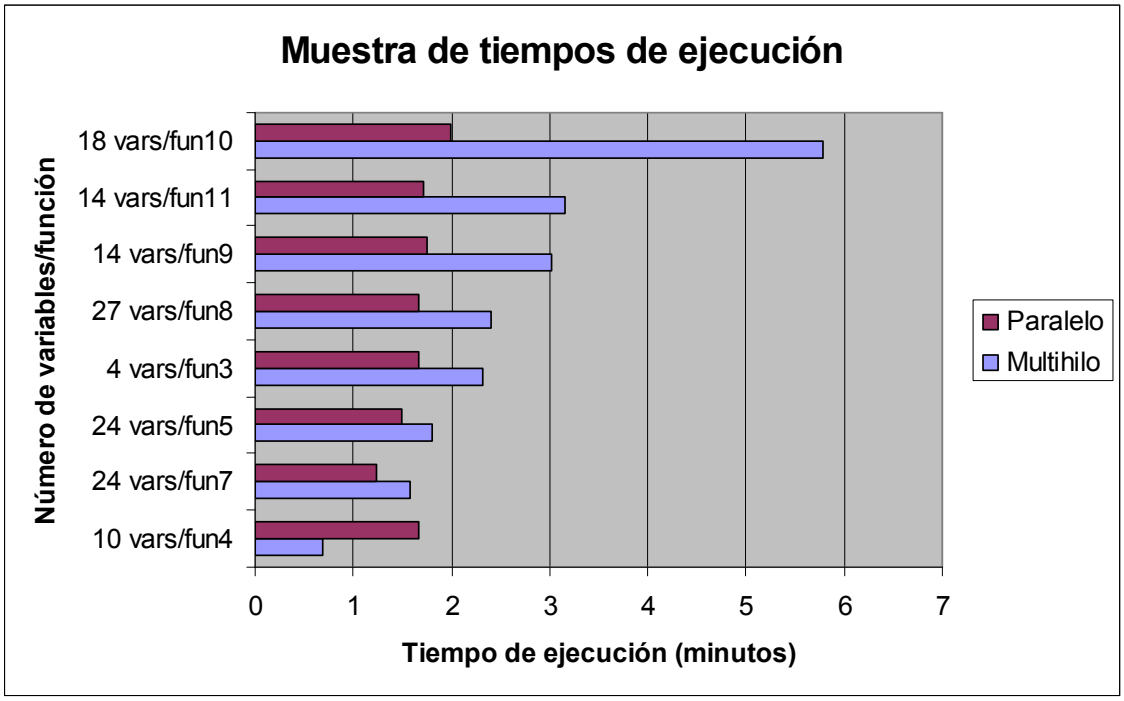
Los datos para las ejecuciones de la etapa tanto paralela como Multihilo son los siguientes:

- Tamaño de la población : 400 individuos.
- Rango de revisión umbral : 300 generaciones.
- Umbral : 90.
- Probabilidad de cruce : 0,9.
- Probabilidad de mutación : 0,1.

9.3. Conclusiones a partir de los datos.

Como primera observación del estudio desarrollado cabe destacar que aunque se aumente el número de variables de una función no se garantiza que aumente su tiempo de ejecución, por lo tanto para el estudio de estos tiempos se han utilizado funciones distintas tanto en su complejidad como en su número de variables.

La gráfica obtenida del estudio es la siguiente:



De aquí se deduce que para el caso de que una función tarde poco tiempo en ejecutarse en Multihilo, el tiempo que tarda en ejecutarse en Paralelo es superior o aproximadamente el mismo. Esto es debido a que en el caso de la ejecución en Paralelo tenemos tres poblaciones en tres máquinas distintas. Estas tres poblaciones están intercambiando información con el servicio Web de intercambio. Es decir que en este caso, el tiempo que tardan en ejecutarse los algoritmos genéticos es despreciable en comparación con el tiempo que tarda la información en transmitirse por la red. Por eso el tiempo que tarda en Multihilo es menor.

Sin embargo, se aprecia la mejora en el tiempo de ejecución de Paralelo en el momento en que la función tarda un tiempo en encontrar la solución. Se obtiene una optimización en el tiempo, ya que al tardar un tiempo mayor en ejecutarse un algoritmo, en el caso de Multihilo, los tres procesos compiten por los mismos recursos mientras que en el caso de paralelo cada población se desarrolla en una máquina distinta.

10. BIBLIOGRAFÍA

- http://www.di.uniovi.es/~cueva/asignaturas/extension/2003/01-Panorama/ServiciosWeb.pdf
- http://tejo.usal.es/~fgarcia/docencia/poo/02-03/trabajos/S3T4.pdf
- http://www.idg.es/iworld/articulo.asp?id=143763
- http://www.it.uc3m.es/~fvalera/ nt_red/trabajos/Grid
- http://www.monografias.com/trabajos11/compil/compil2.shtml
- http://www.msdn.microsoft.com/net
- http://msdn.microsoft.com/library/spa/
- http://www.ecma-international.org/publications/standards/Ecma-334.htm
- "Microsoft Visual C# .NET. Step By Step: Version 2003" 2003
  John Sharp, Jon Jagger
  Ed. Microsoft Press
- "Profesional C#" 2ª ed. 2002. Simon Robinsons
  Colección: De programador a programador
  Ed. Danysoft
- "Visual C# .net" 2002. Francisco Charte Ojeda
  Ed. Anaya Multimedia
- "El lenguaje de programación C#" 2002. Fco. Javier Ceballos Sierra
  Ed. Ra-Ma

11. GLOSARIO.

- **Microsoft .NET**

Plataforma multilenguaje desarrollada por Microsoft para distribuir software m forma de servicios suministrados remotamente que puedan comunicarse y combinarse independientemente de la plataforma, lenguaje de programación y modelo de componentes con los que hayan sido desarrollados.

- **CLR (Common Language Runtime)**

Núcleo de la plataforma .NET : motor encargado de gestionar la ejecución de las aplicaciones a las que ofrece numerosos servicios que simplifican su desarrollo y favorecen su fiabilidad y seguridad.

- **MSIL (Microsoft Intermediate Language)**

Lenguaje intermedio generado por los compiladores para la plataforma .NET

- **Metadatos**

Conjunto de datos organizados en forma de tablas que almacenan información sobre los tipos definidos en el módulo, los miembros de éstos y sobre cuáles son los tipos externos al módulo a los que se les referencia en el módulo.

- **Ensamblado**.

Agrupación lógica de uno o más módulos o ficheros de recursos (ficheros .GIF, .HTML, etc.) que se engloban bajo un nombre común

- **BCL (Librería de Clase Base**)

Librería incluida en el *.NET Framework* formada por cientos de tipos de datos que permiten acceder a los servicios ofrecidos por el CLR y a las funcionalidades más frecuentemente usadas a la hora de escribir programas.

- **CTS (Common Type System)**

(Sistema de Tipo Común) Conjunto de reglas que han de seguir las definiciones de tipos de datos para que el CLR las acepte.

- **CLS (Common Language Specification)**

(Especificación del Lenguaje Común) Conjunto de reglas que han de seguir las definiciones de tipos que se hagan usando un determinado lenguaje gestionado si se desea que sean accesibles desde cualquier otro lenguaje gestionado

- **.NET Framework SDK**

Kit de desarrollo de software que incluye las herramientas necesarias para el desarrollo, distribución y ejecución de la plataforma .NET.

- **Visual Studio .NET**

Entorno de programación que permite realizar todas las tareas del .NET Framework SDK desde una interfaz visual basada en ventanas

12. ANEXOS :

ANEXO 1: ALGORITMOS GENÉTICOS

1. Qué es un algoritmo genético

Las bases de los algoritmos genéticos las encontramos en la naturaleza. La naturaleza utiliza potentes medios para impulsar la evolución satisfactoria de los organismos. Los organismos que son poco aptos para un determinado ambiente mueren, en tanto que los que están bien adaptados para vivir, se **reproducen**.

Ocasionalmente **se producen mutaciones** al azar, y aunque implican la pronta muerte del individuo mutado, algunas mutaciones dan como resultado nuevas y satisfactorias especies.

Los Algoritmos Genéticos (AG) fueron introducidos por John Holland en 1970 inspirándose en el proceso observado en la evolución natural de los seres vivos. Se trata de crear inteligencia en el ordenador simulando la manera de en que lo hizo la naturaleza en nuestro planeta.

Los AG proporcionan un método de aprendizaje basado en la analogía con la evolución de las especies. Los AG generan un conjunto de hipótesis conocido como "población actual" se renueva reemplazando una proporción de esta población por los sucesores de las hipótesis más "adecuadas"(mediante el uso de una función de evaluación). Realmente solucionar el problema consistirá en encontrar la solución óptima, y por tanto, los AG son un método de búsqueda. En esta búsqueda se combina una búsqueda aleatoria, dada por las transformaciones de la población (en la mutación, así como en el cruce), con una búsqueda dirigida dada por la selección.

La popularidad de los AG se debe en parte a que la evolución es un método robusto y bien probado dentro de los sistemas biológicos naturales. Además son fácilmente paralelizables.

Aunque no se garantice encontrar la solución óptima, los AG generalmente encuentran soluciones con un alto grado de acierto.

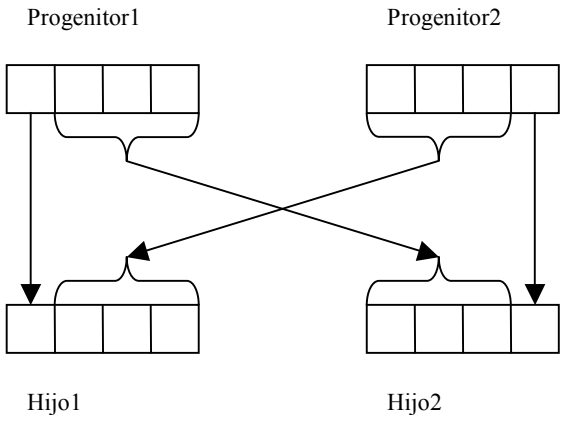
2. Partes principales del algoritmo genético

2.2.1 Representación genética de soluciones del problema

La elección de la representación de los individuos de la población es de gran importancia para la resolución de un problema. En este caso los cromosomas de cada individuo de la población serán un array de enteros que representarán los valores a tomar por cada una de las variables de la ecuación.

2.2.2 Líneas generales del algoritmo genético

• Generación de la población inicial. En ella se generan de forma aleatoria los cromosomas de cada individuo, es decir se generará un entero aleatoriamente dentro del rango introducido por parámetro, para cada uno de los genes de los que consta cada componente de la población.
• Evaluación: asignando un valor de peso o fitness a cada individuo en función de si es o no un mínimo de la ecuación. Para ello tenemos en cuenta no sólo la puntuación del individuo, sino su valor comparado con el resto de la población.
• Selección: elegiremos los individuos que se van a reproducir, así como los que van a sobrevivir para la siguiente generación. Depende del factor de selección pasado como parámetro por el usuario. La selección de los padres viene dada por probabilidades según su "aptitud". Mediante el procedimiento de "rueda" a cada individuo se le asocia una región de ruleta proporcional con su buena o mala aptitud.
• Reproducción: se generan nuevos individuos a partir del cruce de dos individuos originales, eligiendo el punto de cruce aleatoriamente. En este caso el operador de cruce que utilizaremos será el de la elección de un punto en cada uno de los padres produciéndose la mezcla de ambos progenitores a partir de ese punto para cada uno de sus hijos.



Otros operadores de Cruce que también son utilizados en los AG son:

De dos puntos: Se eligen dos puntos de ruptura al azar para intercambiar.

Uniforme: En cada bit se elige al azar un padre para que contribuya con su bit al del hijo, mientras que el segundo hijo recibe el bit del otro padre.

PMX, SEX: Son operadores más sofisticados fruto de mezclar y aleatorizar los anteriores.

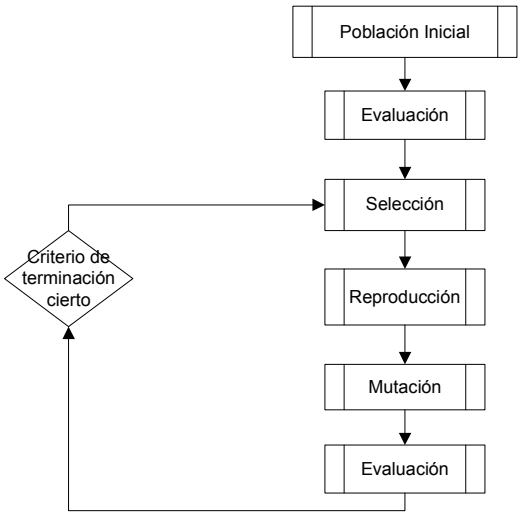
• Mutación: se muta aleatoriamente un gen del individuo, dependiendo del factor de mutación introducido. En el caso de que un individuo mute se revisa su adaptación.

- Criterio de parada: el algoritmo terminará cuando después de un número determinado de generaciones (que le será indicado por parámetro) no haya superado un valor umbral de mejora (siendo este valor umbral también introducido por parámetro por el usuario).

2.2.3 Normas del algoritmo genético

- La evolución opera a nivel de cromosoma, en lugar de en los individuos que representa.
- La selección escoge por sorteo los individuos que van a sobrevivir para la siguiente generación.
- En la reproducción tiene lugar la evolución mediante el paso de parte de los genes de cada uno de los progenitores, a los hijos. Y estos últimos sustituirán a sus padres en la población.
- Las mutaciones también alterarán los cromosomas de los individuos.
- En el AG sólo tendremos en cuenta la información de la generación inmediatamente anterior en cada momento. Por lo tanto podríamos decir que el algoritmo genético no tiene memoria.
- Convergencia del AG. En cada iteración (o generación) el AG opera con una sola población y se espera que el método converja en el infinito a un solo individuo.

El funcionamiento es el siguiente:

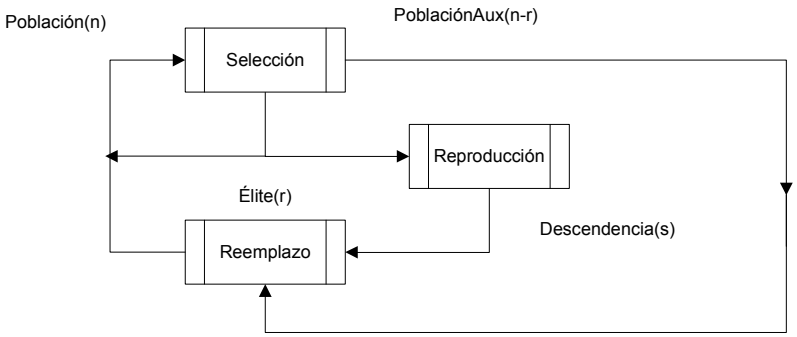


2.2.4 Mejoras del algoritmo genético

2.2.4.1 Elitismo

El proceso de selección es complementado con una estrategia elitista con la cual se asegura la convergencia hacia el individuo con el valor óptimo en cada generación. Una de sus posibles implementaciones es: en cada generación, después de haber ejecutado los operadores genéticos de cruce y mutación, se selecciona una muestra con los mejores individuos de la población llamado Elite, el cual reemplaza a los peores individuos en la generación siguiente. El tamaño de esta muestra dependerá del tamaño total de la población, así a una población mayor le corresponderá una élite con mayor número de individuos. Si existe un individuo mejor que el élite previo, entonces éste pasará a ser nuevo individuo de la élite. La idea fundamental es que el mejor individuo en cada generación prevalezca, para así asegurar la convergencia del Algoritmo Evolutivo Propuesto.

Añadimos el siguiente esquema al del algoritmo evolutivo anterior:



2.2.4.2 Paralelización del algoritmo evolutivo

Existen dos enfoques principales a la hora de paralelizar un algoritmo genético:

- **Paralelización Estándar o Global**: La evaluación de los cromosomas y/o la aplicación de los operadores genéticos se realiza en paralelo. Un procesador maestro supervisa la población y realiza la selección, procesadores esclavos reciben individuos para evaluarlos y aplicarles los operadores genéticos.
- **Paralelización por separación:** Toda la población se encuentra distribuida bien como subpoblaciones o bien como individuos.

1. Grano grueso.
   - Modelo Isla: Se permite enviar los emigrantes a cualquier isla.
   - Modelo Pasarela o Trampolín (*steeping stone*): Migración limitada. Sólo se permite a los emigrantes viajar a sus islas vecinas.
   - Modelo Centralizado
1. Grano grueso
2. Grano fino.
3. Híbridos (Combinación de los dos anteriores).

Se ha tomado el modelo de paralelismo grano grueso el cual toma una población global repartida entre los EPs migrando el mejor individuo, llamando subpoblación a cada una de las partes. Cada subpoblación evoluciona en un procesador por la ejecución del AE. Luego de un número de generaciones predeterminado, que se llama período de migración, se realiza una difusión o migración del mejor individuo de la población total en cada una de las subpoblaciones.

La estrategia para la migración consiste en centralizar los individuos mejor adaptados de cada subpoblación en uno de los procesadores (llamado Proceso Centralizador).

Dos tipos de comunicaciones entre nodos:

- **Síncronas**: El algoritmo evolutivo para y espera para recibir información de otro procesador antes de proceder con la siguiente población.
- **Asíncronas**: En este esquema el algoritmo evolutivo no se detiene para esperar a otros procesadores más lentos → Elimina los tiempos muertos producidos por las barreras de sincronización, que pueden ser extremadamente perjudiciales cuando se trabaja con una red de computadoras cuyo tráfico no se puede controlar totalmente y con máquinas heterogéneas cuyo balance de carga es sumamente difícil de optimizar

La elección de tipo de comunicación asíncrona para el proyecto se basó en la intención de solventar el inconveniente principal de este tipo de paralelización que es la aparición de tiempos de espera. Se solucionó este problema lanzando el intercambio con el proceso centralizador en un hilo. Esto permite que el AG continúe evolucionando sin esperar al intercambio, de manera que en la siguiente iteración en la que le toque intercambiar, primero se comprueba que terminó el anterior intercambio, se toma el individuo intercambiado (si es mejor que el mejor de la élite que haya en ese momento) y se hace una nueva llamada. Esto da lugar a tener en cuenta la sección crítica del intercambio, así como el lanzamiento de los hilos.

ANEXO 2: REFLEXIÓN

La capacidad de reflexión de la plataforma .NET (similar a la de la plataforma Java) nos permite explorar información sobre los tipos de los objetos en tiempo de ejecución.

Extensibilidad de modificadores: C# ofrece, a través del concepto de **atributos**, la posibilidad de añadir a los metadatos del módulo resultante de la compilación de cualquier fuente información adicional a la generada por el compilador que luego podrá ser consultada en tiempo ejecución a través de la librería de reflexión de .NET . Esto, que más bien es una característica propia de la plataforma .NET y no de C#, puede usarse como un mecanismo para definir nuevos modificadores.

Entre otras razones .NET supera a tecnologías de componentes precedentes por la transparencia con la que integra código y metadatos favoreciendo con ello la metaprogramación. Este trabajo muestra cómo usando reflexión (reflection) se define un evaluador de funciones que permite a partir de un objeto, en el que tenemos la función a evaluar y sus parámetros de entrada, obtener un objeto proxy equivalente en funcionalidad al original pero que garantiza ser subtipo de dicha interfaz. De este modo el objeto proxy puede ser utilizado como si estáticamente hubiese declarado que implementa el tipo evaluador. Con ello se facilita la adaptabilidad del software y la factorización de código ya existente para poder aprovechar una funcionalidad común.

La base de la reflexión la pone el espacio de nombres System.Reflection, que es el que permite la reflexión en el desarrollo .NET. La reflexión permite la inspección de metadatos en un fichero PE construyendo (en tiempo de ejecución) los tipos y sus miembros. El espacio de nombres System.Reflection define los siguientes tipos para analizar los metadatos del módulo de un ensamblado: Assembly, Module, Enum, ParameterInfo, MemberInfo, Type, MethodInfo, ConstructorInfo, FieldInfo, EventInfo, y PropertyInfo.

La clase System.Type es la clase principal de la reflexión. Es una clase abstracta y representa un tipo en el sistema de tipos común (CLR). Usando esta clase, podemos encontrar el nombre del tipo, los tipos usados en un módulo (un assembly puede contener uno o más módulos), y namespace, y ver si un tipo dado es por valor o por referencia… También permite consultas de tipos de ficheros, métodos, propiedades y eventos parseando las correspondientes tablas de metadatos. El mecanismo de Serialización de FCL usa reflexión para determinar que campos de tipo define. El formato de Serialización puede entonces obtener los valores de esos campos y escribirlos en un stream de bytes.

Más tarde los enlaces pueden ser obtenidos usando la reflexión. Por ejemplo, en algunas aplicaciones no sabemos que ensamblado cargar en tiempo de compilación por lo que pedimos al usuario que diga el nombre del ensamblador y el tipo en tiempo de ejecución y la aplicación puede cargarlo. Para este propósito, el tipo System.Reflection.Assembly ofrece tres métodos estáticos que permiten

especificar el ensamblado de carga: Load, LoadFrom, y LoadWithPartialName. Estos métodos son similares al LoadLibrary de la API Win32.

1. Ensamblado y metadatos

En la plataforma .NET se distinguen dos tipos de **módulos** de código compilado: **ejecutables** (extensión **.exe**) y **librerías de enlace dinámico** (extensión **.dll** generalmente) Ambos son ficheros que contienen definiciones de tipos de datos, y la diferencia entre ellos es que sólo los primeros disponen de un método especial que sirve de punto de entrada a partir del que es posible ejecutar el código que contienen haciendo una llamada desde la línea de comandos del sistema operativo. A ambos tipos de módulos se les suele llamar **ejecutables portables** (PE), ya que su código puede ejecutarse en cualquiera de los diferentes sistemas operativos de la familia Windows para los que existe alguna versión del CLR.

Un **ensamblado** es una agrupación lógica de uno o más módulos o ficheros de recursos (ficheros .GIF, .HTML, etc.) que se engloban bajo un nombre común. Un programa puede acceder a información o código almacenados en un ensamblado sin tener porqué saber cuál es el fichero en concreto donde se encuentran, por lo que los ensamblados nos permiten abstraernos de la ubicación física del código que ejecutemos o de los recursos que usemos. Por ejemplo, podemos incluir todos los tipos de una aplicación en un mismo ensamblado pero colocando los más frecuentemente usados en un cierto módulo y los menos usados en otro, de modo que sólo se descarguen de Internet los últimos si es que se van a usar.

Todo ensamblado contiene un **manifiesto**, que son metadatos con información sobre las características del ensamblado. Este manifiesto puede almacenarse cualquiera de los módulos que formen el ensamblado o en uno específicamente creado para ello, caso éste último necesario cuando es un **ensamblado satélite** (sólo contiene recursos)

El compilador .NET produce un fichero ejecutable portable PE para CLR con la extensión .exe o .dll. Este archivo PE está principalmente formado por metadatos comprimidos e IL (Intermediate Language). Los metadatos contienen un número de diferentes tablas; por ejemplo una tabla de definición de tipos, una tabla de definición de métodos… Parseando estas tablas se pueden obtener tipos de assembly y atributos. El espacio de nombres System.Reflection de los FCL soporta varios tipos de reflexión viendo o parseando estas tablas de metadatos.

PE (Portable Executable) = Metadata (bunch definition tables) + IL (Microsoft Intermediate Language) + Some other data which are not relevant to this article.

2. Módulos

El contenido de un módulo consta de MSIL y también de otras dos áreas muy importantes: la cabecera de CLR y los metadatos.

- Todos los compiladores que generan código para la plataforma .NET no generan código máquina para CPUs x86 ni para ningún otro tipo de CPU concreta, sino que generan código escrito en el lenguaje intermedio conocido como Microsoft Intermediate Lenguage (MSIL) El CLR da a las aplicaciones las sensación de que se están ejecutando sobre una máquina virtual, y precisamente MSIL es el código máquina de esa máquina virtual. Es decir, MSIL es el único código que es capaz de interpretar el CLR, y por tanto cuando se dice que un compilador genera código para la plataforma .NET lo que se está diciendo es que genera MSIL.
- La **cabecera de CLR** es un pequeño bloque de información que indica que se trata de un módulo gestionado e indica es la versión del CLR que necesita, cuál es su firma digital, cuál es su punto de entrada (si es un ejecutable), etc.
- Los **metadatos** son un conjunto de datos organizados en forma de tablas que almacenan información sobre los tipos definidos en el módulo, los miembros de éstos y sobre cuáles son los tipos externos al módulo a los que se les referencia en el módulo. Los metadatos de cada modulo los genera automáticamente el compilador al crearlo, y entre sus tablas se incluyen:

Tabla	**Descripción**
ModuleDef	Define las características del módulo. Consta de un único elemento que almacena un identificador de versión de módulo (GUID creado por el compilador) y el nombre de fichero que se dio al módulo al compilarlo (así este nombre siempre estará disponible, aunque se renombre el fichero)
TypeDef	Define las características de los tipos definidos en el módulo. De cada tipo se almacena su nombre, su tipo padre, sus modificadores de acceso y referencias a los elementos de las tablas de miembros correspondientes a sus miembros.
MethodDef	Define las características de los métodos definidos en el módulo. De cada método se guarda su nombre, signatura (por cada parámetro se incluye una referencia al elemento apropiado en la tabla ParamDef), modificadores y posición del módulo donde comienza el código MSIL de su cuerpo.
ParamDef	Define las características de los parámetros definidos en el módulo. De cada parámetro se guarda su nombre y modificadores.
FieldDef	Define las características de los campos definidos en el módulo. De cada uno se almacena información sobre cuál es su nombre, tipo y modificadores.
PropertyDef	Define las características de las propiedades definidas en el módulo. De cada una se indica su nombre, tipo, modificadores y referencias a los

	elementos de la tabla MethodDef correspondientes a sus métodos set/get.
EventDef	Define las características de los eventos definidos en el módulo. De cada uno se indica su nombre, tipo, modificadores. y referencias a los elementos de la tabla MethodDef correspondientes a sus métodos add/remove.
AssemblyRef	Indica cuáles son los ensamblados externos a los que se referencia en el módulo. De cada uno se indica cuál es su nombre de fichero (sin extensión), versión, idioma y marca de clave pública.
ModuleRef	Indica cuáles son los otros módulos del mismo ensamblado a los que referencia el módulo. De cada uno se indica cuál es su nombre de fichero.
TypeRef	Indica cuáles son los tipos externos a los que se referencia en el módulo. De cada uno se indica cuál es su nombre y, según donde estén definidos, una referencia a la posición adecuada en la tabla AssemblyRef o en la tabla ModuleRef.
MemberRef	Indican cuáles son los miembros definidos externamente a los que se referencia en el módulo. Estos miembros pueden ser campos, métodos, propiedades o eventos; y de cada uno de ellos se almacena información sobre su nombre y signatura, así como una referencia a la posición de la tabla TypeRef donde se almacena información relativa al tipo del que es miembro.

3. Utilización de la reflexión en el proyecto

Cada individuo de la población, (en cada generación), debe evaluar la función introducida, dando el valor de sus cromosomas a las variables. Para poder hacerlo de una forma no demasiado costosa, para cada Algoritmo genético existe una clase Evaluador que se crea en tiempo de ejecución. Cada individuo llamará a esta clase con sus genes, de forma que se evalúa la función.

Nuestra clase función se compone de un array de EvaluatorItem, en el que actualmente únicamente existe un único ítem, ya que únicamente necesitamos una función evaluador. Pero al ser un array de Ítem sería fácilmente extensible con otras funciones. En el ítem introducimos el string de la función.

Se usa el CodeDOM para crear un simple ensamblado con una única clase en él. Se transforma cada uno de los EvaluatorItems (en este caso uno sólo) en un método de la clase. Cuando se llama a "EvaluateDouble", se usa la reflexión para obtener un objeto "MethodInfo" y llamar al método.

Para compilar la expresión, se crea un CSharpCodeProvider y se pone los atributos de Compilador (como las adding referentes, etc). Entonces se crea la clase en la que añado el método. Finalmente se compila (buscando errores) y se salva el objeto.

El evaluador se crea pasándole como parámetro este array de Ítem. Se crea entonces una clase en tiempo de ejecución, en la que uno de sus métodos, introduciendo el ArrayList de genes evalúa la expresión.

Cuando se llama al objeto "_Compiled", se usa el objeto MethodInfo para invocar la llamada y devolver el resultado al individuo concreto de la población. Al hacer el "Invoke" le metemos como parámetros el array de genes, que estarán metidos en un array de "Object". Esta llamada se hará para cada individuo y en cada generación de cada uno de los algoritmos genéticos.

ANEXO 3: ANALIZADOR SINTÁCTICO

1. Introducción

Analizador o **parser** (término inglés para analizador gramatical), nombre dado a un conjunto de programas capaces de verificar la validez gramatical y eventualmente producir información sobre la estructura de una emisión lingüística determinada.

La parte del análisis divide al programa fuente en sus elementos componentes y crea una representación intermedia. Es decir, divide el string de entrada en tokens (clase léxica implementada como "Tokenizer.cs").

Durante el análisis se determina las operaciones que implica la expresión. Toma una serie de tokens del analizador léxico y se registran en una estructura jerárquica llamada árbol sintáctico abstracto (AST). Este es un parser muy simple implementado como "AParser.cs".

2. Funcionamiento del Analizador

Las operaciones que soporta este analizador sintáctico son:

- Operadores binarios: +, -, *, /
- Operadores monarios: +, -
- Agrupación de paréntesis
- Construcción de funciones: sin, cos, abs, pow, sqrt, exp, log (es fácilmente ampliable añadiendo más funciones a "Evaluator.cs").

Realmente se trata de un intérprete: En lugar de producir un programa objeto como resultado de una traducción, realiza las operaciones que implica la expresión. Toma el AST y lo evalúa como expresión. Este analizador puede construir un árbol, y después efectuar las operaciones de los nodos conforme "recorre" el árbol. Por ejemplo: en la raíz descubriría que tiene que realizar una suma, y llamaría a una rutina para evaluar la expresión de la derecha y después almacenaría el valor resultante en la localidad de memoria asignada con el identificador posición. En el hijo derecho de la raíz, la rutina descubriría que tiene que calcular la suma de dos expresiones. Se llamaría a sí misma de manera recursiva para calcular el valor.

El analizador léxico lee los caracteres uno a uno desde la entrada y va formando grupos de caracteres con alguna relación entre sí (tokens), que constituirán la entrada para la siguiente etapa del analizador. Cada token representa una secuencia de caracteres que son tratados como una única entidad.

El analizador sintáctico, también llamado parser, recibe como entrada los tokens que le pasa el Analizador Léxico (el analizador sintáctico no maneja directamente caracteres) y comprueba si esos tokens van llegando en el orden correcto (orden permitido por el lenguaje). La salida "teórica" de la fase de análisis sintáctico sería un árbol sintáctico.

Análisis Semántico: En definitiva, comprobará que el significado de lo que se va leyendo es válido.