Sistemas Informáticos
Curso 2002-03

Arquitectura de un Sistema de Optimización accesible a través de Servicios Web XML

José Luis Peinado Moreno
Cristina Iglesias Suárez
María Nieves Fraile Herrero

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

Facultad de Informática
Universidad Complutense de Madrid

Se autoriza a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Fdo. José Luis Peinado Moreno

Fdo. Cristina Iglesias Suárez

Fdo. María Nieves Fraile Herrero

Índice

1.- Resúmenes

1.1.- Resumen

En este proyecto hemos desarrollado el diseño y la implementación de un sistema de optimización con restricciones en el que se integran tres alternativas algorítmicas distintas: *propagación de consistencia sobre dominios finitos, algoritmos genéticos, y bifurcación y acotación.*

El acceso al sistema se puede realizar de tres formas diferentes: desde una interfaz nativa sobre el servidor que soporta el sistema, desde un computador remoto conectado a Internet utilizando un *browser* convencional, y desde una interfaz nativa sobre un computador remoto conectado a Internet utilizando Servicios Web XML.

Todo el sistema se ha desarrollado dentro de la plataforma Microsoft .NET, y por tanto requiere para ejecutarse en el lado del servidor el Framework de .NET en las tres alternativas de acceso, y en el lado del cliente sólo en la alternativa de acceso mediante interfaz nativa.

Por cada alternativa de optimización del sistema se han implementado 3 módulos operativos, uno para cada modo de acceso. Cada uno de los módulos se corresponde con un tipo de aplicación .NET diferente. Una aplicación Windows que se ejecuta en el servidor desde un terminal del propio servidor, una aplicación Web que se ejecuta en el servidor desde un *browser* ubicado en un computador remoto (cliente), y una aplicación Windows que se ejecuta en el cliente y que se conecta al servidor mediante un Servicio Web.

1.2.- Summary

In this project we have developed the design and implementation of a constraints optimization system in which three different algoritmic alternatives have been integrated: consistency propagation on finite domains, genetic algorithms, and branch and bound.

The system can be accessed using three different methods: from the server that supports the system using a native interface, from a remote computer connected to Internet using a conventional browser and, finally, from a native interface on a remote computer connected to Internet using Web Services XML.

The whole system has been developed within the Microsoft.NET platform, so to be executed, it requires the Framework .NET software on the server side, and on the client side only for Web Services access.

For each optimization algorithmic three operative modules have been implemented, one for each access way. Each module corresponds with a different type of .NET application. A Windows application that is executed on the server, a Web application that is executed on the server controled by a browser located in a remote computer (client), and a Web services application that is executed on the server contorled by a windows application located on a client computer.

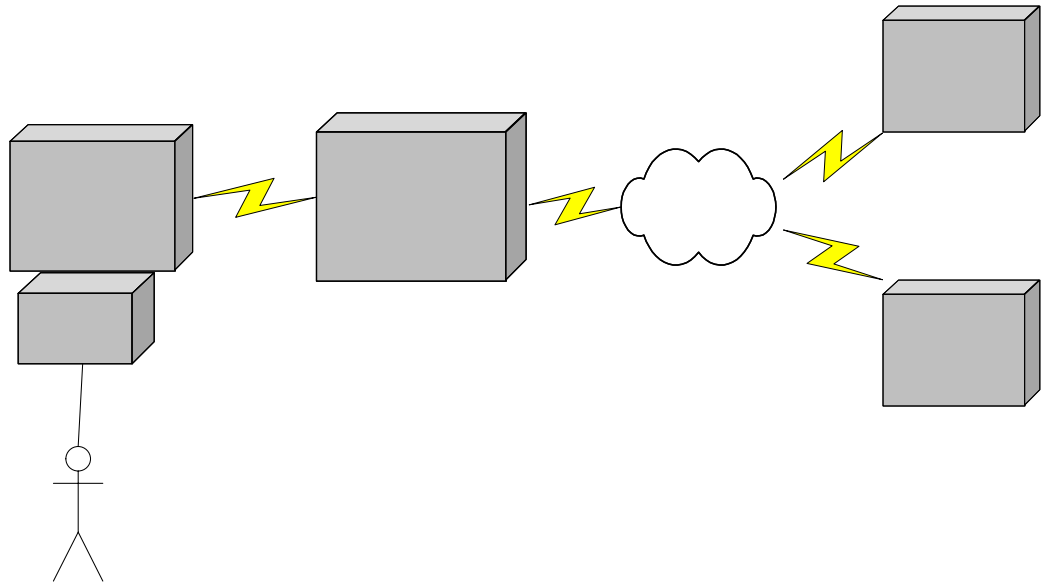
2.- Palabras clave

- Restricción
- Función objetivo
- Propagación
- Dominio finito
- Genético
- Evolutivo
- Población
- Individuo
- Genoma, genes
- Operador de selección
- Operador de cruce
- Operador de mutación
- Vuelta atrás, Backtracking
- Ramificación, Branch
- Poda, Bound
- C#
- Visual Studio .NET
- .NET Framework
- XML
- Servicio Web, Web Service
- UDDI
- SOAP
- ASP.NET
- IIS, Internet Information Server

3.- **Introducción**

En este proyecto hemos desarrollado el diseño y la implementación de un sistema de optimización con restricciones en el que se integran tres alternativas algorítmicas distintas: *propagación de consistencia sobre dominios finitos, algoritmos genéticos, y bifurcación y acotación.*

El acceso al sistema se puede realizar de tres formas diferentes: desde una interfaz nativa sobre el servidor que soporta el sistema, desde un computador remoto conectado a Internet utilizando un navegador (*browser*) convencional, y desde una interfaz nativa sobre un computador remoto conectado a Internet utilizando Servicios Web XML.



Todo el sistema se ha desarrollado dentro de la plataforma Microsoft .NET, y por tanto requiere para ejecutarse en el lado del servidor el Framework de .NET en las tres alternativas de acceso, y en el lado del cliente sólo en la alternativa de acceso mediante interfaz nativa.

Por cada alternativa de optimización del sistema se han implementado 3 módulos operativos, uno para cada modo de acceso. Cada uno de los módulos se corresponde con un tipo de aplicación .NET diferente. Una aplicación Windows que se ejecuta en el servidor desde un terminal del propio servidor, una aplicación Web que se ejecuta en el servidor desde un *browser* ubicado en un computador remoto (cliente), y una aplicación Windows que se ejecuta en el cliente y que se conecta al servidor mediante un Servicio Web.

La memoria del proyecto está organizada de la siguiente manera. En primer lugar haremos una descripción de los algoritmos de optimización implementados en el sistema. En segundo lugar analizaremos los componentes de Internet que permiten dar soporte al sistema, es decir, la Páginas Activas y los Servicios Web XML. En tercer lugar estudiaremos la forma de implementación de estos componentes dentro de la plataforma .NET, es decir, IIS (Internet Information Server) y ASP.NET. En cuarto lugar daremos una descripción detallada de la implementación de cada uno de los métodos de optimización. Finalmente, en quinto lugar, describiremos la forma y uso del sistema con algunos ejemplos prácticos.

4.- Algoritmos de Optimización

En los apartados siguientes daremos una breve descripción de los métodos de optimización utilizados en el sistema.

4.1.- Optimización con Restricciones sobre Dominios Finitos

4.1.1.- Introducción

La Programación con Restricciones tiene unos sólidos fundamentos teóricos y se utiliza cada vez más en el campo comercial, hasta el punto que se está convirtiendo en el método preferido para modelar problemas de optimización.

La Programación con Restricciones (CP), es el resultado de investigaciones desarrolladas por la comunidad de Inteligencia Artificial en el área de la Programación Lógica y la Satisfacción de Restricciones. Los métodos CP utilizan la propagación de consistencia a través de las restricciones como motor de inferencia. En cada nodo, la propagación de restricciones, reduce el dominio de las variables.

Los dominios pueden ser continuos, discretos, boleamos etc. Cuando la propagación produce un domino vacío, se elimina el nodo y se produce vuelta atrás (*backtracking*). La bifurcación se realiza siempre que el dominio de una variable conste de más de un elemento (dominios discretos o boleamos), o los límites no estén dentro de una cierta tolerancia (dominios continuos). La programación con restricciones, se desarrolló originariamente para resolver problemas de satisfacibilidad, y después se extendió a problemas de optimización. Para ello se resuelve un problema de satisfacibilidad en el que la función objetivo del problema, se rescribe como una restricción que obliga a ser igual a una nueva variable, el dominio de esta nueva variable, da los límites inferior y superior de los valores de la función objetivo. Siempre que se obtiene una solución factible durante la búsqueda, se imponen restricciones adicionales que limitan los valores de la función objetivo. La búsqueda termina cuando se han eliminado todos los nodos.

La eficiencia de los métodos CP depende fundamentalmente de los algoritmos de propagación de consistencia que se utilizan para reducir el dominio de las variables. La ventaja de los métodos CP radica pues, en el uso activo de las restricciones para podar *a priori* el espacio de búsqueda, eliminando combinaciones de valores que no pueden aparecer juntas en una solución.

4.1.2.- Restricciones de dominios finitos

Son los dominios de restricciones en los que los posibles valores que pueden tomar las variables están restringidos a un conjunto finito. Tanto las restricciones booleanas como las enteras son ejemplo de restricciones de dominios finitos, en las booleanas los valores que pueden tomar las variables serán *Verdadero o Falso,* y en las enteras, está restringida a permanecer en un rango finito de números enteros. Ejemplos de aplicaciones reales de este método podrían ser el *timetabling*, *scheduling* etc.…, en los que se debe elegir entre un número finito de posibilidades.

Se han desarrollado distintos métodos para la resolución de problemas de restricciones de dominio finito, técnicas de nodo y arco consistencia, desarrolladas por la comunidad de Inteligencia Artificial para resolver problemas de satisfacción de restricciones y técnicas de propagación de límites, desarrolladas por la comunidad de Programación de Restricciones.

4.1.2.1.- Resolutor mediante *backtracking* simple

Para resolver cualquier problema de satisfacción de restricciones, siempre podemos recurrir al método de *backtracking*, en el que se explora por completo el espacio de soluciones. El problema es que computacionalmente es muy caro (exponencial), al ser la resolución de satisfacción de restricciones un problema NP-Duro.

4.1.2.2.- Nodo y Arco Consistencia

A continuación veremos unos resolutores de problemas de satisfacción de restricciones, cuya complejidad es polinómica en el caso peor. Estos resolutores, están basados en el hecho de que si un dominio de una de las variables queda vacío, el problema es insatisfacible. La idea subyacente es que el problema original se transforma en otro equivalente, en el que los dominios de las variables se han reducido, así si uno de los dominios llega a quedar vacío, entonces el nuevo problema y por tanto el original son insatisfacibles.

Estos resolutotes funcionan de la siguiente manera, toman cada restricción del problema, y utilizan la información del dominio de cada variable de la restricción para eliminar valores de los dominios del resto de variables. Propagan información de los valores de los dominios permitidos, de una variable a otra, hasta que los dominios son consistentes con la restricción, por eso se llaman también, resolutotes basados en consistencia.

Al final puede haber dos resultados, o bien, que haya algún domino vacío, en cuyo caso el problema es insatisfacible, o bien, que se llegue a que cada dominio tenga sólo un valor, y que por tanto será una solución al problema.

Una restricción c es nodo consistente con el dominio D si, o bien, $|vars(c)| \neq 1$ o, si $vars(c) = \{x\}$, y para cada $d \in D(x)$, $\{x \rightarrow d\}$ es una solución de c.
Un problema de satisfacción de restricciones con restricciones $c1 \wedge \ldots \wedge cn$ y dominio D, es nodo consistente si cada restricción ci es nodo consistente con D para $1 \leq i \leq n$. (Donde *vars(c)* es el conjunto de variables de la restricción c)

Una restricción c es arco consistente con el dominio D si, o bien, $|vars(c)| \neq 2$ o, si $vars(c) = \{x,y\}$, y para cada $dx \in D(x)$, hay algún $dy \in D(y)$, tal que $\{x \rightarrow dx, y \rightarrow dy\}$ es una solución de c y para cada $dy \in D(y)$, hay algún $dx \in D(x)$ tal que $\{x \rightarrow dx, y \rightarrow dy\}$ es una solución de c.
Un problema de satisfacción de restricciones con restricciones $c1 \wedge \ldots \wedge cn$ y dominio D, es arco consistente si cada restricción ci es nodo consistente con D para $1 \leq i \leq n$. (Donde *vars(c)* es el conjunto de variables de la restricción c)

4.1.2.3.- Consistencia de límites

La consistencia de arcos y nodos funcionan bien si lo que tenemos son restricciones de a lo sumo dos variables, ya que si tuviesen más de dos, sería ignoradas en la comprobación de consistencia. La límite consistencia extiende la arco consistencia para poder trabajar con restricciones de cualquier número de variables.

Un problema de satisfacción de restricciones es aritmético si el dominio de cada variable es un dominio finito de enteros, y las restricciones son aritméticas. Es posible convertir un problema no aritmético en uno que si lo es, como por ejemplo el problema del coloreado de mapas, en el que asemejamos cada color a un número entero. Con los problemas de restricciones aritméticos, aparece un nuevo concepto, la límite consistencia.

Una restricción primitiva *r* es límite consistente con dominio *D* si para cada variable *x* en *vars(r)* existen números reales *d1,..., dk* para el resto de variables *x1,..., xk* tal que *{x→min(D,x), x1→d1,…, xk→dk}* es una solución de *r,* y análogamente para *{x→max(D,x)}.* (Donde *vars(c)* es el conjunto de variables de la restricción *c*)

Un problema de satisfacción de restricciones aritmético es límite consistente si lo son todas sus restricciones primitivas

Dado un dominio de valores para cada variable, hay métodos eficaces para calcular el nuevo rango de valores de la variable de la restricción, para que sea límite consistente, estos métodos se llaman *reglas de propagación.*

Las reglas de propagación para algunas restricciones son las siguientes:

- Restricción ***X = Y + Z*** *,* que es equivalente a X=Y+Z, Y=X-Z y Z=X-Y, con lo que se obtienen las siguientes reglas de propagación:

  X>=min(D,Y)+min(D,Z) X<=max(D,Y)+max(D,Z)
  Y>=min(D,X)-max(D,Z) Y<=max(D,X)-min(D,Z)
  Z>=min(D,X)-max(D,Y) Z<=max(D,X)-max(D,Y)

- Restricción ***4W+3P+2C <= 9***, con lo que se obtienen las siguientes reglas de propagación:

  W<=9/4-3/4*min(D,P)-2/4*min(D,C)
  P<=9/3-4/3*min(D,W)-2/3*min(D,C)
  C<=9/2-4/2*min(D,W)-3/2*min(D,P)

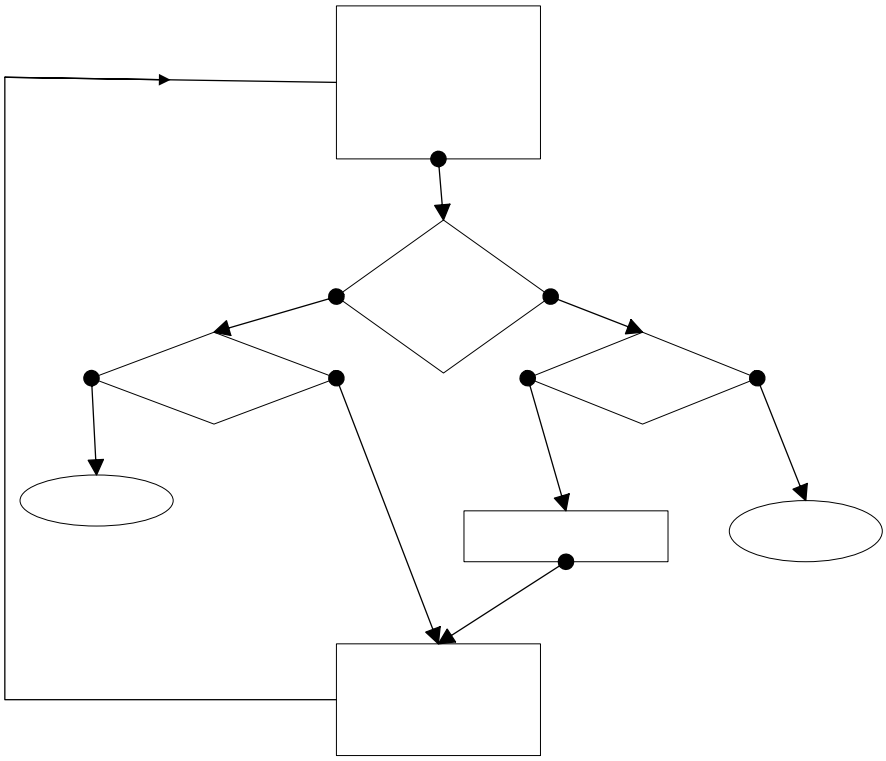
  Con las reglas obtenidas para la anterior restricción y la regla para el operador = podemos extrapolar para obtener las de los operadores <, >= y >.

- Inecuaciones (**Y<>Z)**, que proporcionan una propagación débil. Sólo hay propagación cuando un miembro toma un valor fijo e igual al mínimo o máximo del otro miembro.

El alcance del algoritmo de propagación de restricciones en esta implementación, se limita a variables con un solo dominio entero, y los propagadores para las operaciones, igual, distinto, menor o igual, menor, mayor o igual y mayor. Estos propagadores, como veremos más adelante, irán eliminando enteros de los límites de los dominios, y nunca enteros centrales del dominio, con el fin de que el dominio resulte límite consistente**.**

El algoritmo de propagación de restricciones, se basa en la aplicación de propagadores sobre los dominios iniciales, con el fin de restringirlos, para eliminar, los enteros, que nunca podrán formar parte de la solución, por la propia naturaleza de las restricciones que hay que aplicar a las variables, y así, reducir considerablemente, el número de búsquedas que hay que realizar para encontrar las soluciones. Una vez que no se puedan reducir más los dominios con los propagadores de consistencia, se empezará una fase de búsqueda, en la que se fija un valor de una variable (*labeling*), y suponiendo que el dominio de esa variable es sólo ese número fijado, se empieza una propagación de nuevo. Si el dominio de alguna variable queda vacío, se hace *backtracking* y se fija otro valor de la variable, si por el contrario, se llega a un estado en el cuál, cada dominio de cada variable, tiene un solo valor, la tupla de dichos valores, será una solución. El etiquetado de variables, se hará para todas las variables del problema.

El algoritmo general es el siguiente:



4.2.- Ramificación y Poda con Backtracking

4.2.1.- Introducción

Esta técnica de diseño, cuyo nombre en castellano proviene del término inglés *Branch and Bound*, se aplica normalmente para resolver problemas de optimización. Ramificación y Poda, al igual que el diseño Vuelta Atrás (*backtracking*), realiza una enumeración parcial del espacio de soluciones basándose en la generación de un árbol de expansión.

Una característica que le hace diferente al diseño de Vuelta Atrás es la posibilidad de generar nodos siguiendo distintas estrategias. Recordemos que el diseño Vuelta Atrás realiza la generación de descendientes de una manera sistemática y de la misma forma para todos los problemas, haciendo un recorrido en profundidad del árbol que representa el espacio de soluciones. El diseño Ramificación y Poda en su versión más sencilla puede seguir un recorrido en anchura (estrategia LIFO) o en profundidad (estrategia FIFO), o utilizando el cálculo de funciones de coste para seleccionar el nodo que en principio parece más prometedor (estrategia de mínimo coste o LC).

Como veremos a continuación, además de estas estrategias, la técnica de Ramificación y Poda utiliza cotas para podar aquellas ramas del árbol que no conducen a la solución óptima. Para ello calcula en cada nodo una cota del posible valor de aquellas soluciones alcanzables desde ése. Si la cota muestra que cualquiera de estas soluciones tiene que ser necesariamente peor que la mejor solución hallada hasta el momento no necesitamos seguir explorando por esa rama del árbol, lo que permite realizar el proceso de poda.

En consecuencia, y a la vista de todo esto, podemos afirmar que lo que le da valor a esta técnica es la posibilidad de disponer de distintas estrategias de exploración del árbol y de acotar la búsqueda de la solución, que en definitiva se traduce en eficiencia. La dificultad está en encontrar una buena función de coste para el problema, buena en el sentido de que garantice la poda y que su cálculo no sea muy costoso. Si es demasiado simple probablemente pocas ramas puedan ser excluidas. Dependiendo de cómo ajustemos la función de coste mejor algoritmo se deriva.

4.2.2.- Conceptos básicos

El algoritmo de Ramificación y Poda (*branch & bound*) es una técnica basada en el recorrido de un árbol de soluciones que:

- Realiza un **recorrido sistemático** en un árbol de soluciones
- El recorrido no tiene por qué ser necesariamente en profundidad sino que seguirá una **estrategia de ramificación**, guiada por estimaciones del beneficio, que se realizarán para cada nodo
- Se usan **técnicas de poda** para eliminar nodos que no lleven a la solución óptima
- La poda se realiza **estimando** en cada nodo las **cotas de beneficio** que se pueden obtener a partir del mismo

Un concepto fundamental para entender el algoritmo de ramificación y poda es el de ***nodo vivo.*** Nodo vivo del árbol de expansión es un nodo con posibilidades de ser ramificado, es decir, un nodo que no ha sido podado. Para determinar en cada momento que nodo va a ser expandido y dependiendo de la estrategia de búsqueda seleccionada, necesitaremos almacenar todos los nodos vivos en alguna estructura que podamos recorrer.

4.2.3.- Metodología de resolución

El **método general** a aplicar para la resolución de un problema mediante este algoritmo es el siguiente:

1. Para cada nodo **i**, tendremos:
   - Cota superior (CS(i)) y Cota inferior (CI(i)) del beneficio (o coste) óptimo que se podrían alcanzar a partir de ese nodo. Estas cotas determinarán el momento en el que se podrá realizar una poda.
   - Estimación del beneficio (o coste) óptimo que se puede obtener a partir de ese nodo. Esta estimación se puede calcular como una media de las cotas anteriores y puede ayudar a decidir qué parte del árbol se explorará primero.

2. **Estrategia de poda**:
   - Supongamos un problema de maximización de la función objetivo en un punto intermedio del recorrido del árbol de soluciones en el que, además, nos podemos encontrar algún nodo a partir del cual no se pueda llegar a ninguna solución válida.
   - Se han recorrido varios nodos 1..n, estimando para cada uno la cota superior y la cota inferior para todo nodo j entre 1 y n.
   - Se podrá podar un nodo **i**, si:

     CS (i) <= Beneficio (j),

     para algún j, solución final (factible).

3. **Estrategia de ramificación**:
   - De forma implícita, se tendrá en cuenta a la hora de llevar a cabo la ramificación del árbol de soluciones, la lista de **nodos vivos** en cada momento.
   - La lista de nodos vivos contiene la lista de nodos que ya han sido generados pero que todavía no han sido explorados, para cada variable del problema a resolver. Son los nodos pendientes de tratar por el algoritmo.
   - Las posibles estrategias de ramificación a emplear son las siguientes:
     - Estrategia FIFO (*first in first out*): la lista de nodos vivos es una cola y, por lo tanto, el recorrido del árbol se realiza en **anchura**.
     - Estrategia LIFO (*last in first out*): la lista de nodos vivos es una pila y, por lo tanto, el recorrido del árbol se realiza en **profundidad**.
     - Estrategia LC (*least cost*): se selecciona de toda la lista de nodos vivos aquel con el que se obtenga un mayor beneficio (o menor coste) para explorar a continuación.
   - Dadas las características del problema a resolver en nuestro caso, la estrategia de ramificación empleada ha sido la **estrategia LC**.

4.2.4.- Algoritmo general

Básicamente, en un algoritmo de Ramificación y Poda básico, se realizan tres etapas: La primera de ellas, denominada de *Selección*, se encarga de extraer un nodo de entre el conjunto de los nodos vivos. La forma de escogerlo va a depender directamente de la estrategia de búsqueda que decidamos para el algoritmo. En la segunda etapa, la *Ramificación*, se construyen los posibles nodos hijos del nodo seleccionado en el paso anterior. Por último se realiza la tercera etapa, la *Poda*, en la que se eliminan algunos de los nodos creados en la etapa anterior. Esto contribuye a disminuir en lo posible el espacio de búsqueda y así atenuar la complejidad de estos algoritmos basados en la exploración de un árbol de posibilidades. Aquellos nodos no podados pasan a formar parte del conjunto de nodos vivos, y se comienza de nuevo por el proceso de selección. El algoritmo finaliza cuando encuentra la solución, o bien cuando se agota el conjunto de nodos vivos.

El algoritmo general es el siguiente:

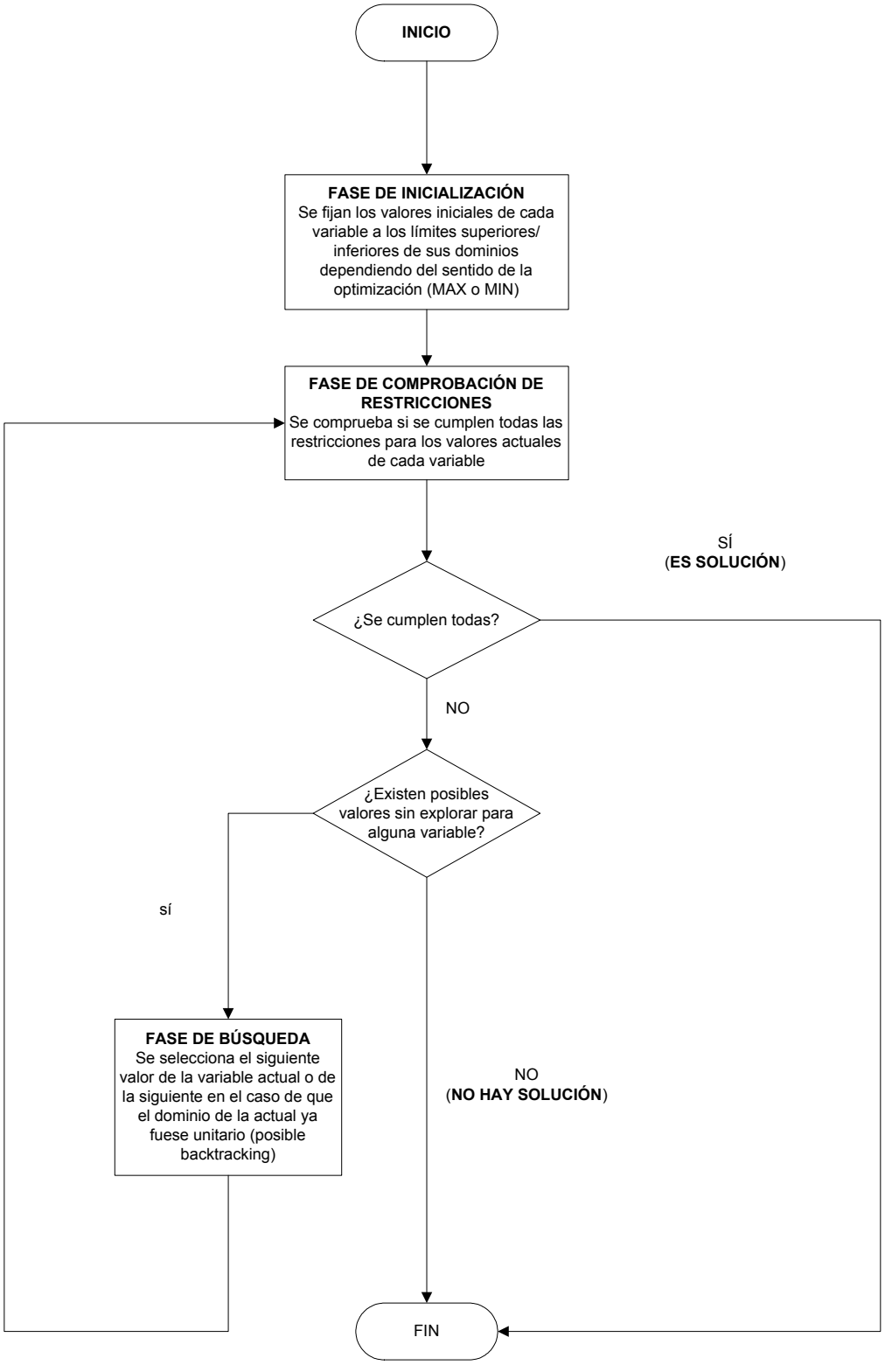
```
repetir
  expandir el nodo vivo más prometedor;
  generar todos sus hijos;
  una vez generados, el padre se mata;
  para cada hijo hacer
    si tiene un coste esperado peor que
       el de la mejor solución en curso
    entonces
            se mata;
    sino
       si tiene un coste esperado mejor que el
          de la mejor solución en curso y no es
          solución
       entonces
             se pasa a la lista de nodos vivos;
       sino
         pasa a ser la mejor solución en curso;
         se revisa toda la lista de nodos
         vivos, eliminando los que prometen
         algo peor de lo conseguido;
       fsi
     fsi
  fpara
hasta que la lista esté vacía;
```

4.2.5.- Backtracking

Además, para optimizar los resultados obtenidos por la implementación realizada del algoritmo de ramificación y poda para la resolución del problema que nos ocupa, se ha añadido el ***backtracking***. Con la aplicación de la técnica de *backtracking* (Vuelta Atrás), conseguimos la exploración exhaustiva de todos los nodos y, por tanto, que la solución obtenida, en la mayoría de los casos, sea la **solución óptima** del problema planteado.

4.2.6.- Esquema del algoritmo implementado

El esquema del algoritmo implementado es el siguiente:



4.3.- Algoritmo Genético

4.3.1.- Introducción y conceptos

Los algoritmos genéticos nacieron con el planteamiento de crear un algoritmo con la misma filosofía que emplea la naturaleza en la selección natural. La idea básica es la siguiente: generemos un conjunto con algunas de las posibles soluciones. Cada una va a ser llamada *individuo*, y a dicho conjunto se le denominará *población*.

Cada individuo tiene una información asociada a él. A dicha información se la va a denominar *código genético*. En un problema de optimización corresponde a las variables libres, es decir, aquellas a las que el algoritmo tiene que asignar un valor para que una función sea mínima o máxima para esos valores. Esta función se denominará *función de adaptación* y determina el grado de adaptación de un individuo.

Los algoritmos genéticos son métodos estocásticos de búsqueda ciega de soluciones cuasióptimas.

4.3.2.- Características de los algoritmos genéticos

Algunas de las características de los algoritmos genéticos son:

- Son *algoritmos estocásticos*. Dos ejecuciones distintas pueden dar dos soluciones distintas.
- Son *algoritmos de búsqueda múltiple*, luego dan varias soluciones.
- Son *algoritmos que hacen una barrida mayor al subespacio de posibles soluciones válidas*, y con diferencia.
- Tienen algún *elemento aleatorio*, por lo que dan soluciones aproximadas, no exactas.
- *La optimización es función de la representación de los datos*. Este es el concepto clave dentro de los algoritmos genéticos, ya que una buena codificación puede hacer la programación y la resolución muy sencillas, mientras que una codificación errada nos va a obligar a estudiar que el nuevo genoma cumple las restricciones del problema, y en muchos problemas tendremos que *abortar* los que no cumplan las restricciones, por ser estas demasiado complejas. Además, la velocidad de convergencia va a estar fuertemente influenciada por la representación
- *Es una búsqueda paramétricamente robusta*. Eso quiere decir que hemos de escoger realmente mal los parámetros del algoritmo para que no converja.
- Por último, *los algoritmos genéticos son intrínsecamente paralelos*. Esto significa que, independientemente de que lo hayamos implementado de forma paralela o no, buscan en distintos puntos del espacio de soluciones de forma paralela.

4.3.3.- Decisiones para implementar un algoritmo genético

Las decisiones que hay que tomar para implementar un algoritmo genético son:

- *Criterio de codificación*. Como se va a almacenar la información en el genoma. La codificación del genoma es fundamental en un problema de algoritmos genéticos. En nuestro algoritmo genético, el número de genes es igual al número de variables declaradas en el problema de restricciones. Cada gen codificará el valor de una variable, y la posición del gen dentro del genoma se corresponderá con el orden de declaración de la variable que representa.

  Ejemplo: Supongamos que tenemos cuatro variables declaradas, A, B, C y D, con valores 4, 8, 20 y 12 respectivamente. El genoma que codifica esta asignación de valores a las variables es:

4	8	20	12
Gen 0	Gen 1	Gen 2	Gen 3

- *Criterio de tratamiento de individuos no factibles*. Como se van a tratar a los individuos que no cumplan las restricciones. En nuestro algoritmo genético, se permite que individuos no factibles pertenezcan a la población.
- *Criterio de inicialización*. Cómo se va a construir la población inicial del algoritmo genético. En nuestro algoritmo genético, la población inicial se genera de forma aleatoria. Para ello se generan aleatoriamente tantos individuos como indique el tamaño de la población. Generar aleatoriamente un individuo significa que su generase genera aleatoriamente, para cada gen se genera un número entero aleatorio entre el límite inferior y el límite superior de la variable a la que representa ese gen.

  Ejemplo: Supongamos que tenemos una variable declarada A cuyo límite inferior es –5 y cuyo límite superior es 5. Un valor aleatorio posible del gen que la representa en un genoma de un individuo de la población será por ejemplo –3, pero nunca será por ejemplo 8.

- *Criterio de parada*. Determina cuándo el algoritmo ha llegado a una solución aceptable. En nuestro algoritmo evolutivo, el criterio de parada es el número de generaciones.
- *Función de adaptación*. Corresponde a la función de costo de la investigación operativa tradicional.
- *Operadores genéticos*. Se emplean para determinar cómo va a ser la nueva generación. Básicamente son los operadores de cruce y mutación, aunque pueden ser empleados otros adicionales -muerte, aborto, envejecimiento...-. Tanto cruce como mutación pueden ser realizados de muchas formas distintas. En nuestro algoritmo genético, implementamos tanto cruce como mutación.
- *Criterios de reemplazo*. Los criterios que determinan quiénes se van a cruzar. No tienen que ser obligatoriamente los mismos que los criterios de selección de los padres.
- *Parámetros de funcionamiento*. Determinados parámetros que, sin poder ser englobados en ninguno de los anteriores, son fundamentales para el funcionamiento de un algoritmo genético. En nuestro algoritmo genético son el tamaño de la población, el número de generaciones, la probabilidad de la aplicación de los operadores de cruce y de mutación.

4.3.4.- El operador de selección

Para aplicar los operadores genéticos tendremos que seleccionar un subconjunto de la población. Algunas de las técnicas que disponemos son:

- *Selección directa*: toma elementos de acuerdo a un criterio objetivo, como son «los x mejores», «los x peores»... los del tipo «el cuarto individuo a partir del último escogido» son empleados con mucha frecuencia cuando se quieren seleccionar dos individuos distintos, y se selecciona el primero por un método aleatorio o estocástico.
- *Selección aleatoria*: puede ser realizado por *selección equiprobable* o *selección estocástica*.
    - *Selección equiprobable*: todos tienen la misma probabilidad de ser escogidos. Por ejemplo, en nuestro algoritmo la madre en el cruce es escogida con probabilidad equiprobable.
    - *Selección estocástica*: la probabilidad de que un individuo sea escogido depende de una heurística. Los distintos procedimientos estocásticos son:
        - *Selección por sorteo*: cada individuo de la población tiene asignado un rango proporcional -o inversamente proporcional- a su adaptación. Se escoge un número aleatorio dentro del rango global, y el escogido es aquel que tenga dicho número dentro de su rango. La probabilidad de ser escogido es proporcional/inversamente proporcional al grado de adaptación del individuo.
        - *Selección por escaños*: se divide el rango del número aleatorio en un número predeterminado de escaños. Los escaños se reparten de acuerdo con la ley d'Hont, tomando como «puntuación» para repartir los escaños el grado de adaptación. Observamos que es más probable escoger un elemento de baja probabilidad por este método que en el de selección por sorteo.
        - *Selección por restos estocásticos*: igual que el método de selección de escaños, sólo que los escaños no asignados directamente -es decir, aquellos en que se aplica directamente la ley d'Hont- se asignan de forma aleatoria. La probabilidad de escoger un elemento de muy baja probabilidad es más alta que en el de selección por escaños.
        - *Por ruleta*: definimos un rango con las características de la selección por sorteo. El número al azar será un número aleatorio forzosamente menor que el tamaño del rango. El elemento escogido será aquel en cuyo rango esté el número resultante de sumar el número aleatorio con el resultado total que sirvió para escoger el elemento anterior. El comportamiento es similar al de una ruleta, donde se define un avance cada tirada a partir de la posición actual. Tiene la ventaja de que no es posible escoger dos veces consecutivas el mismo elemento, y que puede ser forzado a que sea alta la probabilidad de que no sean elementos próximos en la población -esto último no es una ventaja de por sí; salvo que algunos de los otros operadores genéticos emplee un método de selección directa basado en la posición relativa de los individuos de la población-. En la bibliografía más antigua se emplea este término para definir lo que aquí hemos definido como selección por sorteo.

- *Por torneo*: escoge un subconjunto de individuos de acuerdo con una de las técnicas anteriores -habitualmente, aleatoria o estocástica- y de entre ellos selecciona el más adecuado por otra técnica -habitualmente, determinística de tipo «el mejor» o «el peor»-. Esta técnica tiene la ventaja de que permite un cierto grado de elitismo -el mejor nunca va a morir, y los mejores tienen más probabilidad de reproducirse y de emigrar que los peores- pero sin producir una convergencia genética prematura, si la población es, al menos, un orden de magnitud superior al del número de elementos involucrados en el torneo. En caso de que la diferencia sea menor no hemos observado mucha diferencia entre emplear el torneo o no. La selección por torneo ha sido la técnica empleada en nuestro algoritmo para decidir tanto el padre -en unión con el criterio «el mejor»-, como quién va a emigrar -en unión con el criterio «el mejor»- y quién va a morir -en unión con el criterio «el peor»-.

En nuestro algoritmo genético hemos implementado el operador de selección utilizando la técnica de selección por sorteo.

4.3.5.- El operador de cruce

Se denomina *operador de cruce* a la forma de calcular el genoma del nuevo individuo en función del genoma del padre y de la madre. El operador de cruce es fuertemente responsable de las propiedades del algoritmo genético, y determinará en gran medida la evolución de la población.

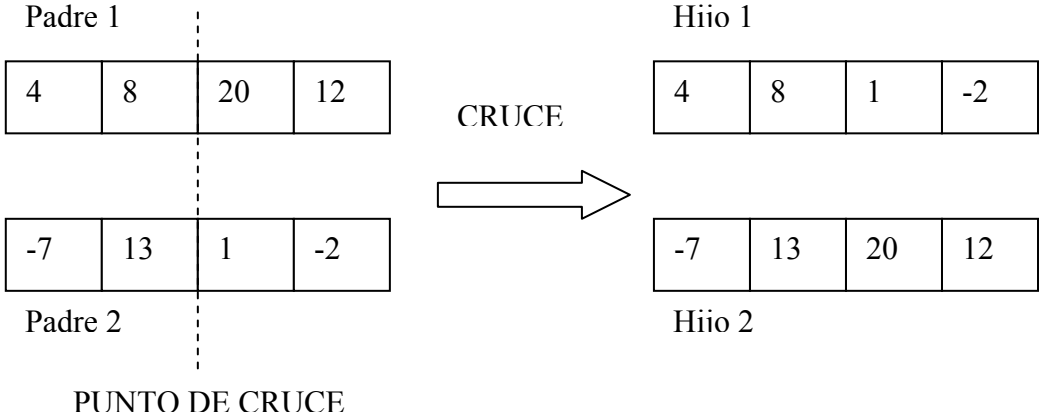
Existen gran cantidad de técnicas de cruce. Las técnicas básicas son:

- *Cruce básico*: se selecciona un punto al azar de la cadena. La parte anterior del punto es copiada del genoma del padre y la posterior del de la madre.
- *Cruce multipunto*: igual que el cruce básico, sólo que estableciendo más de un punto de cruce.
- *Cruce segmentado*: existe una probabilidad de que un cromosoma sea punto de un cruce. Conforme se va formando la nueva cadena del descendiente, para cada gen, se verifica si ahí se va producir un cruce.
- *Cruce uniforme*: para cada gen de la cadena del descendiente existe una probabilidad de que el gen pertenezca al padre, y otra de que pertenezca a la madre.
- *Cruces para permutación*: Existe una familia de cruces específicas para los problemas de permutación, siendo algunos de ellos:
    - *Cruce con correspondencia parcial*: toma una subsecuencia del genoma del padre y procura preservar el orden absoluto de los fenotipos -es decir, orden y posición en el genoma- del resto del genoma lo más parecido posible de la madre. Aparece también en la bibliografía como *PMX*.
    - *Cruce de orden*: toma una subsecuencia del genoma del padre y procura preservar el orden relativo de los fenotipos del resto del genoma lo más parecido posible de la madre. Lo podemos encontrar en la bibliografía como *OX*.

- *Cruce de ciclo*: Tomamos el primer gen del genoma del padre, poniéndolo en la primera posición del hijo, y el primer gen del genoma de la madre, poniéndolo dentro del genoma del hijo en la posición que ocupe en el genoma del padre. El fenotipo que está en la posición que ocupa el gen del genoma del padre igual al primer gen del genoma de la madre se va a colocar en la posición que ocupe en el genoma del padre, y así hasta rellenar el genoma del hijo. Este método también es conocido en la bibliografía como *CX*.

Es una buena idea que, tanto la codificación como la técnica de cruce, se hagan de manera que las características buenas se hereden; o, al menos, no sea mucho peor que el peor de los padres. En problemas en los que, por ejemplo, la adaptación es función de los pares de genes colaterales, el resultante del cruce uniforme tiene una adaptación completamente aleatoria.

En nuestro algoritmo genético hemos implementado el operador de cruce utilizando la técnica de cruce básico. Ejemplo:



4.3.6.- El operador de mutación

Se define *mutación* como una variación de las informaciones contenidas en el código genético -habitualmente, un cambio de un gen a otro producido por algún factor exterior al algoritmo genético-. En Biología se definen dos tipos de mutaciones: las generativas, que se heredan y las somáticas, que no se heredan. En los algoritmos genéticos sólo nos serán interesantes las mutaciones generativas. Mas, ¿por qué puede interesar que incorporemos este mecanismo aleatorio?

Algunas de las razones que pueden motivar a incorporar son:

- *Desbloqueo del algoritmo*. Si el algoritmo se bloqueó en un mínimo parcial, una mutación puede sacarlo al incorporar nuevos fenotipos de otras zonas del espacio.
- *Acabar con poblaciones degeneradas*. Puede ocurrir que, bien por haber un cuasi-mínimo, bien porque en pasos iniciales apareció un individuo demasiado bueno que acabó con la diversidad genética, la población tenga los mismos fenotipos. Si se ha llegado a una población degenerada, es preciso que las mutaciones introduzcan nuevos genomas.

- *Incrementar el número de saltos evolutivos*. Los saltos evolutivos -aparición de un fenotipo especialmente valioso, o, dicho de otra forma, salida de un mínimo local- son muy poco probables en un genético *puro* para un problema genérico. La mutación permite explorar nuevos subespacios de soluciones, por lo que, si el subespacio es bueno en términos de adaptación, se producirá un salto evolutivo después de la mutación que se expandirá de forma exponencial por la población.
- *Enriquecer la diversidad genética*. Es un caso más *suave* que el de una población degenerada -por ejemplo, que la población tenga una diversidad genética pobre-, la mutación es un mecanismo de prevención de las poblaciones degeneradas.

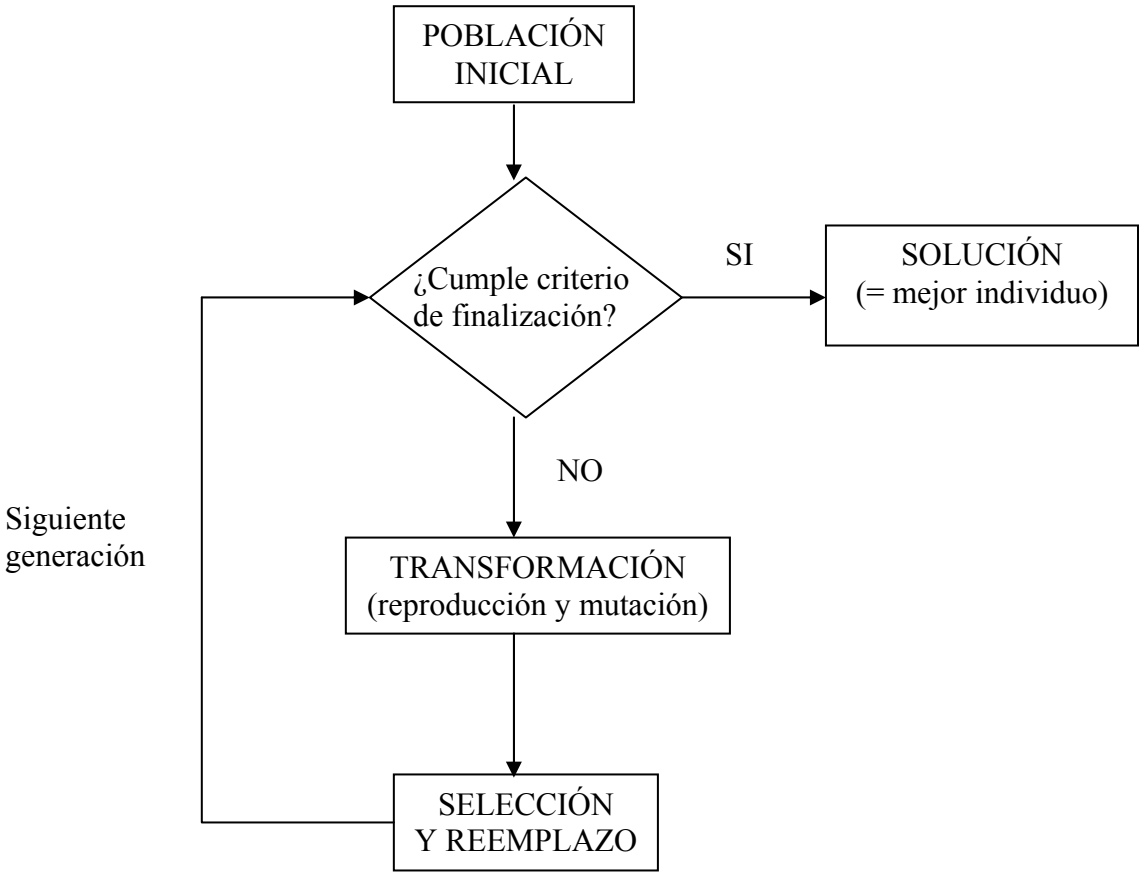
Sin embargo, si la tasa de mutación es excesivamente alta tendremos la ya conocida deriva genética. Una estrategia muy empleada es una tasa de mutación alta al inicio del algoritmo, para aumentar la diversidad genética, y una tasa de mutación baja al final del algoritmo, para conseguir que converja.

Existen varias técnicas distintas de mutación. Algunas de éstas son:

- *Mutación de bit*: existe una única probabilidad de que se produzca una mutación de algún bit. De producirse, el algoritmo toma aleatoriamente un bit, y lo invierte.
- *Mutación multibit*: cada bit tiene una probabilidad de mutarse o no, que es calculada en cada pasada del operador de mutación multibit.
- *Mutación de gen*: igual que la mutación de bit, solamente que, en vez de cambiar un bit, cambia un gen completo. Puede sumar un valor aleatorio, un valor constante, o introducir un gen aleatorio nuevo.
- *Mutación multigen*: igual que la mutación de multibit, solamente que, en vez de cambiar un conjunto de bits, cambia un conjunto de genes. Puede sumar un valor aleatorio, un valor constante, o introducir un gen aleatorio nuevo.
- *Mutación de intercambio*: existe una probabilidad de que se produzca una mutación. De producirse, toma dos bits/genes aleatoriamente y los intercambia.
- *Mutación de barajado*: existe una probabilidad de que se produzca una mutación. De producirse, toma dos bits o dos genes aleatoriamente y baraja de forma aleatoria los bits -o genes, según hubiéramos escogido- comprendidos entre los dos.

En nuestro algoritmo genético hemos implementado el operador de mutación utilizando la técnica de mutación multigen.

4.3.7.- Esquema general del algoritmo genético



5.- Recursos de Internet utilizados en el sistema

5.1.- Páginas estáticas

En los primeros tiempos de Internet sólo se podía acceder a los servidores Web para solicitar páginas multimedia, totalmente pasivas, que el servidor enviaba al cliente (*navegador del usuario*) en formato HTML, utilizando el protocolo HTTP. El navegador del cliente transformaba el archivo HTML en información legible en pantalla. En este marco, el usuario sólo se limitaba a indicarle al servidor la página que quería solicitar.

5.2.- Páginas activas

Las páginas activas son páginas HTML que integran código fuente que se ejecuta en el servidor cada vez que la página es solicitada por el cliente. Al realizar la petición de la página activa, se ejecuta el código fuente, y genera un resultado que junto con el código HTML forman una salida visualizada en el navegador del usuario.

Al permitir ejecutar código fuente en el servidor, se aprovecha la capacidad del mismo para realizar cálculos pesados, descargando así a la máquina cliente de dicho trabajo, dedicándose así esta máquina sólo a mostrar los resultados de la ejecución en el navegador del usuario.

El rendimiento por tanto de la página activa será menor, puesto que hay una ejecución previa de todo el código de *script* de servidor de la página, que junto con el código HTML estático que existía, formarán un código HTML final distinto en cada ejecución de la página, puesto que ésta normalmente dependerá de parámetros que establece el usuario en formularios u otras formas de entrada. Así por ejemplo la petición de una página activa que visualiza vuelos disponibles, generará un resultado según el origen y el destino del vuelvo que pida el usuario, aunque la página a la que se accede sea la misma, y sólo cambien los parámetros de entrada.

El aspecto de una página activa que visualiza la frase "hola mundo", sería la siguiente (el lenguaje de *script* será pseudocódigo para dicha página sea independiente de la plataforma):

```
<html>
<body>
<h1>Hola mundo 1</h1><br>
<%
        holamundo="Hola mundo 2"
%>
<%= holamundo%>
</body>
</html>
```

Cuando el navegador hace una petición a esta página al servidor, éste detecta que es una página activa, y realiza la ejecución del código entre <%%>, que en este caso define una variable *holamundo* a la que se le asigna el texto "Hola mundo 2", después en la siguiente línea se muestra el valor de la variable. Esta ejecución se hace antes de enviar nada al cliente.

A la hora de enviar la página generada al navegador del cliente, se empieza a procesar la página de arriba abajo, y todo lo que es *html* se envía como tal, y cuando se encuentra con los símbolos <%%>, sustituye todo lo que hay dentro por el resultado de la ejecución realizada previamente, por tanto el resultado total en el navegador del usuario será:

Hola mundo 1
Hola mundo 2

Con la diferencia de que "Hola mundo 1" lo genera el navegador interpretando el código html <h1>Hola mundo 1</h1>, y "Hola mundo 2" se genera en el servidor.

Existe una multitud de lenguajes concebidos o no para Internet. Cada uno de ellos explota más a fondo ciertas características que lo hacen más o menos útiles para desarrollar distintas aplicaciones. En el dominio de la red, los lenguajes de lado servidor más ampliamente utilizados para el desarrollo de páginas dinámicas son el ASP, JSP, PHP y PERL.

El ASP (Active Server Pages) es un lenguaje derivado del Visual Basic desarrollado por Microsoft. Evidentemente su empleo se realiza sobre plataformas funcionando bajo sistema Windows NT/2000. Más tarde hablaremos de la plataforma más reciente de Microsoft para ASP, llamada ASP.NET.

JSP (Java Server Pages) es igual que ASP, excepto en el lenguaje utilizado, que en este caso es Java.

El PHP podría ser considerado como el lenguaje análogo al ASP utilizado en plataformas Unix y Linux.

Estos dos lenguajes resultan bastante útiles para la explotación de bases de datos y su aprendizaje resulta accesible para una persona profana de la programación. Cualquiera de ellos resultaría la opción ideal a la hora de hacer evolucionar un sitio Web realizado en HTML.

Por otra parte, el PERL es un lenguaje más rápido y potente que requiere obviamente un aprendizaje más largo y resulta más reservado para personas ya familiarizadas con la verdadera programación.

5.3.- Servicios Web

El World Wide Web ha pasado de ser un medio para la publicación de información y contenidos, a convertirse en una plataforma para el diseño y desarrollo de aplicaciones informáticas distribuidas

Se ha popularizado durante el año 2000 un nuevo paradigma en el diseño de aplicaciones informáticas para la Web: los llamados *Web services* (servicios Web).
En el modelo de aplicación Web "tradicional" encontramos una importante limitación: la interacción comienza y termina en dos puntos claramente definidos: la petición del usuario y la respuesta de la aplicación informática.

Únicamente son dos los "interlocutores" que participan en este proceso. En cada intercambio de información que se produce, la aplicación informática debe "construir" una página resultado en formato HTML para presentar la información al usuario.

Un servicio Web se suele definir como una unidad de aplicación capaz de ofrecer datos o servicios de procesamiento a otras aplicaciones informáticas. Así, una aplicación podría ofrecer distintos servicios a otras aplicaciones. Las características de estos servicios son:

- Que se solicitan a través del Web
- Que los resultados de su ejecución también se devuelven a la aplicación peticionaria a través del Web
- Que se tramitan según un modelo "estandarizado".

La principal diferencia de los Servicios Web con respecto a otras tecnologías Web hasta ahora desarrolladas, es que los resultados de la ejecución de un servicio Web se integran directamente con los lenguajes de programación tradicionales, pudiendo desde una aplicación nativa, realizar una llamada a un método de un Servicio Web con total transparencia para el desarrollador, como si de cualquier componente nativo se tratase, con la diferencia de que dicho componente se está ejecutando en una máquina remota en cualquier lugar del mundo.

El concepto de servicio Web está arropado por una serie de estándares publicados por el W3C y apoyados por los principales fabricantes de tecnología (IBM, Microsoft, etc.), estos estándares señalan cómo se deben cursar las peticiones de servicio a servidores remotos, la forma en la cual éstos deben enviar los resultados, y cómo se deben publicar o dar a conocer los servicios que están accesibles a través de un servidor Web.

Ninguno de estos estándares trata la forma en la que debe implementarse o programarse el servicio en sí mismo. En este punto, se deja libertad absoluta a los fabricantes y proveedores para elegir el lenguaje de programación que deseen utilizar.

Un servicio Web consiste en una función disponible en un servidor conectado al Web. Esta función puede consistir en cualquier cosa:

- Realizar un simple cálculo con unos datos que se le envían como parámetro,
- Acceder a una base de datos para recuperar un conjunto de registros,
- Validar la corrección de una información o contrastarla frente a otros datos, etc.

El servicio Web podrá ser solicitado desde otro programa informático que se ejecute en un ordenador conectado al Web. Junto a la solicitud de la ejecución, se pueden enviar al ordenador que ofrece el servicio unos parámetros que el servicio Web remoto tomará como base para el cálculo o la función.

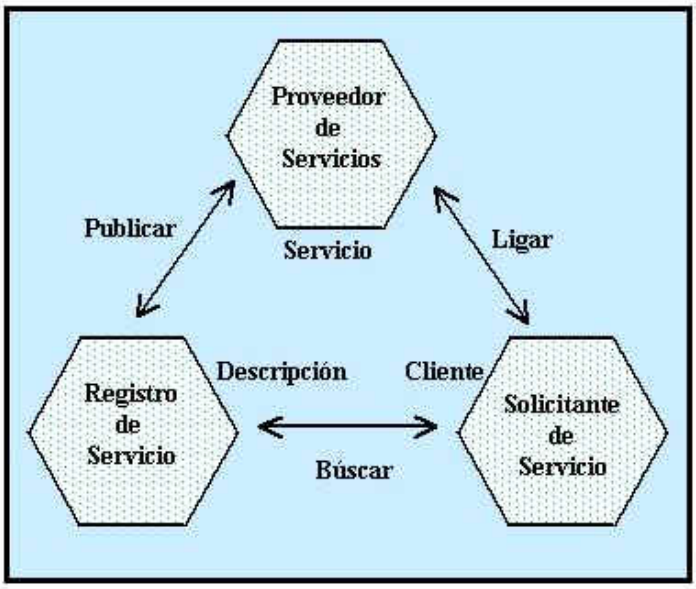
La aplicación que actúa como cliente debe conocer:

- La URL del servidor remoto que ofrece el servicio,
- El nombre del servicio que se solicita, y
- Los parámetros que se deben enviar junto con la llamada al servicio.

Estos datos se enviarán mediante *http*, el servidor que ofrece el servicio Web leerá los parámetros que se le han enviado, llamará a un componente o programa encargado de implementar el servicio, y los resultados que se obtengan de su ejecución serán devueltos al servidor que solicitó la ejecución del servicio.

Las **componentes** de los servicios Web son:

- **Servicio**. La aplicación es ofrecida para ser utilizada por solicitantes que llenan los requisitos especificados por el proveedor de servicios. La implementación se realiza sobre una plataforma accesible en la red. El servicio se describe a través de un leguaje de descripción de servicio. Tanto la descripción como las políticas de uso han sido publicadas de antemano en un registro.
- **Proveedor de Servicio**. Desde el punto de vista comercial, es quien presta el servicio. Desde el punto de vista de arquitectura, es la plataforma que provee el servicio.
- **Registro de Servicios**. Es un depósito de descripciones de servicios que puede ser consultado, donde los proveedores de servicios publican sus servicios y los solicitantes encuentran los servicios y detalles para utilizar dichos servicios.
- **Solicitante de servicios**. Desde el punto de vista comercial, la empresa que requiere cierto servicio. Desde el punto de vista de la arquitectura, la aplicación o cliente que busca e invoca un servicio.



Entre las razones por las cuales los servicios Web jugarán un rol principal en la siguiente generación de sistemas distribuidos, están:

- *Interoperabilidad.* Cualquier servicio Web puede interactuar con cualquier otro servicio Web. El protocolo estándar *SOAP* permite que cualquier servicio pueda ser ofrecido o utilizado independientemente del lenguaje o ambiente en que se haya desarrollado.
- *Omnipresencia.* Los servicios Web se comunican utilizando *HTTP* y *XML.* Cualquier dispositivo que trabaje con éstas tecnologías puede ser huésped y acceder a los servicios Web. Por ejemplo, pronto serán utilizados en teléfonos, automóviles o aún en máquinas vendedoras de refrescos. Por ejemplo, una máquina de venta de refrescos puede comunicarse vía inalámbrica con el servicio Web de un proveedor local y ordenar un pedido de suministro.

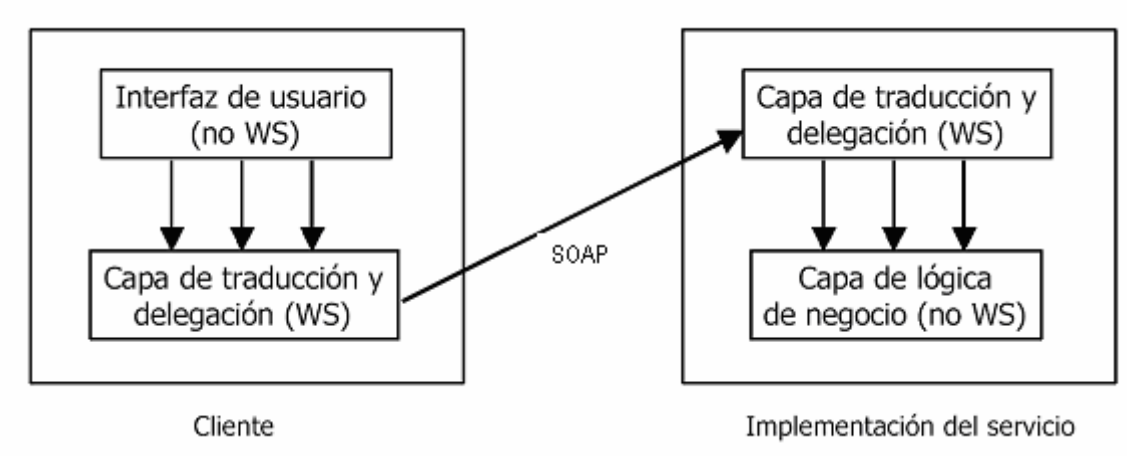
- *Barrera mínima de participación.* Los conceptos detrás de los servicios de Web son fáciles de comprender y se ofrecen Herramientas de Desarrollo (*ToolKits*) por IBM, Sun Microsystems, Microsoft etc.. que permiten a los desarrolladores crear e implementar rápidamente servicios de Web.

- *Apoyo de las Industrias.* Todas las compañías apoyan el protocolo *SOAP* y la tecnología derivada de los servicios Web.

Para que los servicios Web tengan éxito, se requieren vencer algunos retos técnicos, entre los cuales se encuentran:

- *Descubrimiento.* ¿Cómo se anuncia un servicio Web para ser descubierto por otros servicios? ¿Qué sucede si el servicio es modificado o se cambia una vez que ha sido anunciado?. Existen dos estándares nuevos que están diseñados para ello, el *WSDL* (Web Services Definition Language) y el *UDDI* (Universal Description, Discovery and Integration).
- *Confiabilidad*. Algunos sistemas huésped de servicios Web serán más confiables que otros. ¿Cómo se puede medir ésta confiabilidad y ser comunicada? ¿Qué sucede cuando un huésped de servicio temporalmente se sale de línea? ¿Cómo se localiza o utiliza un servicio alterno hospedado en otra compañía, o se pone en espera de que regrese el servicio original? ¿Cómo se sabe en que otra compañía se puede confiar?.
- *Seguridad.* Algunos servicios Web estarán públicamente disponibles y con poca seguridad, pero la mayoría de los servicios comerciales utilizarán comunicaciones encriptadas con autentificación. Es probable que el protocolo HTTP sobre SSL proveerá la seguridad básica, pero los servicios individuales requerirán de un mayor nivel de especificidad. ¿Cómo autentifica un servicio Web a sus usuarios? ¿Cómo distingue un servicio los niveles de privilegios entre los diversos usuarios?.
- *Transacciones.* Los sistemas tradicionales de transacciones utilizan un método de compromiso de dos fases - se recolectan todos los recursos participantes y se aseguran estos hasta que se lleva a cabo la transacción completa. Terminada la transacción se liberan los recursos. Este método puede funcionar bien en ambientes cerrados donde las transacciones son de corta duración, pero no trabaja bien en ambientes abiertos donde las transacciones pueden tomar horas, si no es que días.
- *Administración*. ¿Qué tipos de mecanismos se requieren para administrar un sistema altamente distribuido? ¿Es posible delegar la administración de algunos servicios Web a otros?
- *Contabilidad.* ¿Cómo se define qué tanto tiempo puede un usuario acceder y ejecutar un servicio Web? ¿Cómo se pueden cobrar los servicios Web? ¿Cómo será la comercialización del servicio, bajo suscripción o pago por evento?.
- *Pruebas.* ¿Cómo se puede depurar un servicio Web que proviene de diferentes compañías, es hospedado en diferentes ambientes y en diversos sistemas operativos?.

Diagrama de interacción general Cliente y Servicio:



Los servicios Web se definen a partir de las siguientes especificaciones:

- *SOAP (Simple Object Access Protocol)*
- *WSDL (Web Services Description Language)*
- *UDDI (Universal Description, Discovery and Integration)*

Del mantenimiento de las dos primeras, *SOAP* y *WSDL* se encarga el W3C. En el caso de *UDDI*, se trata de un proyecto en el que participan distintas empresas, el lenguaje XML constituye la base de todos ellos.

5.3.1.- SOAP (Simple Object Access Protocol)

SOAP es un protocolo ligero basado en XML, para el intercambio de información en un ambiente descentralizado y distribuido. SOAP permite la intercomunicación entre objetos de cualquier tipo - sobre cualquier plataforma, en cualquier lenguaje.

SOAP es un protocolo que opera en un contexto más amplio, el de los servicios Web, aparejado con el estándar UDDI (Universal Description, Discovery, and Integration) para proporcionar servicios de registro y mensajería entre los negocios.

5.3.1.1.- Capas de Red

En la evolución de servicios de Web han resaltado tres capas de red: TCP/IP, HTTP/HTML, y XML. Estas capas están colocadas una arriba de otra, y permanecen de esta manera hoy en día.

El primer nivel, el protocolo TCP/IP, se encarga primordialmente de pasar los paquetes de datos por el cable. El TCP/IP es un protocolo que garantiza la transmisión a través de redes públicas, enfatizando en confiabilidad del transporte de datos y la conectividad física.

El segundo nivel, HTML sobre HTTP, es una capa de presentación y se ocupa de búsquedas, recuperación y compartición de información basadas en navegadores. El énfasis aquí es la navegación a través de una interfaz gráfica y la manipulación de formatos de presentación. Los ambiente de trabajo en red, cargados de sistemas operativos propietarios y software dependiente de plataforma, están cambiando y abriendo camino a los sistemas abiertos de cómputo en Internet que son basados en estándares.

Al frente de este nuevo mundo basado en estándares se encuentra XML, que es el tercer nivel y quizás el factor de cohesión más importante de Internet. XML, es un formato de intercambio de datos fuertemente estructurados, que proporciona una nueva dimensión al nivel HTTP/HTML, en el cual la comunicación máquina-a-máquina se realiza por la interfaces estándar. Esta capa permite a los programas intercambiar datos formateados en forma independiente de plataforma y presentación. Se pueden agregar hojas de estilo XSLT opcionales como componente de presentación y/o transformación.

5.3.1.2.- XML, la clave en la descripción de servicios Web

La clave para hacer todo esto posible es la comunicación máquina-a-máquina, área en la cual el XML se especializa. XML permite construir todo tipo de definiciones de sintaxis de datos y permite que la información sea manipulada por los programas, eliminando todo tipo de adivinanza en las comunicaciones. Se pueden acordar las etiquetas a utilizar, definir las interfaces, y el procesamiento de datos puede ser estandarizado. Los servicios Web son programas componentes reutilizables que usan XML como marco de referencia de comunicación extensible estándar para facilitar este tipo de comunicación de máquina-a-máquina.

Dado que el XML es textual y legible por los humanos, es ideal como marco de referencia de transporte para otros servicios de Web débilmente acoplados. SOAP es una tecnología derivada de estándares anteriores basados en XML (XML-RPC), y de algún modo, apunta hacia un estándar emergente *ebXML*. El establecimiento de *ebXML* se encuentra en proceso de desarrollo, intentando proveer una definición amplia de mensajes compartidos de negocios entre socios comerciales. SOAP es más modesto en alcance y menos complejo de implementar.

5.3.1.3.- Anatomía de una petición/respuesta de SOAP

En principio una petición/respuesta de SOAP puede viajar sobre cualquier protocolo de transporte (HTTP, SMTP, etc.). Los encabezados diferirán de protocolo a protocolo, pero el contenido de SOAP es el mismo. La petición de SOAP en XML consiste de tres partes:

- El *sobre* ó *envoltura* de SOAP define un marco de referencia general para expresar *qué* hay en el mensaje, *quién* debe de atenderlo, y *si* es opcional u obligatorio. .
- Las *reglas de codificación* de SOAP definen un mecanismo de serialización que puede ser utilizado para intercambiar instancias de tipos de datos definidos por la aplicación.

- La representación RPC de SOAP define una *convención* que puede ser utilizada para representar los llamados y respuestas de procedimientos remotos.

La respuesta SOAP/HTTP es regresada como un documento XML dentro de una respuesta HTTP estándar. El documento XML está estructurado como cualquier respuesta excepto que el Cuerpo contiene inmerso el resultado método.

Dado que SOAP utiliza XML para codificar mensajes, es relativamente sencillo procesar los mensajes en cada paso del proceso de llamado. Además, la facilidad de depuración de mensajes SOAP permite la convergencia rápida de las diversas implementaciones de SOAP, la cual es importante en la interoperabilidad a gran escala.

La especificación SOAP define dos modelos de mensajes:

- Un mensaje que se enviará desde la aplicación cliente a la aplicación servidor, solicitando la ejecución de un método al que se pasan una serie de parámetros.
- Un mensaje que se enviará desde la aplicación servidor al cliente, y que contendrá datos XML con los resultados de la ejecución del método solicitado.

Ejemplo de petición de un libro, buscando por su ISBN:

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV=”http://schemas.xmlsoap.org/soap/encoding/">
        <SOAP-ENV:Header>
        </SOAP-ENV:Header>
        <SOAP-ENV:Body>
                <catalogo:buscaIsbn xmlns:catalogo="http://catalogo.org/cat">
                        <catalogo:isbn>
                                84-4553-3334-2X
                        </catalogo:isbn>
                </catalogo:buscaIsbn>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Ejemplo de respuesta:

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV=”http://schemas.xmlsoap.org/soap/encoding/">
        <SOAP-ENV:Header>
        </SOAP-ENV:Header>
        <SOAP-ENV:Body>
        <catalogo:buscaIsbnResponse xmlns:catalogo="http://catalogo.org/cat">
                <catalogo:titulo>
                        Catalogar materiales especiales
                </catalogo:titulo>
                <catalogo:autor>Marta de Juanes</catalogo:autor>
                </catalogo:buscaIsbnResponse>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Para facilitar la creación y el formateo de los mensajes *SOAP* que deben intercambiar las aplicaciones Web, los programadores disponen de distintas utilidades y aplicaciones, estas implementaciones incluyen utilidades para generar los mensajes a partir de componentes, actualmente existen ciertas "diferencias" en las implementaciones de los distintos fabricantes.

5.3.2.- WSDL (Web Service Description Language)

El lenguaje de descripción de servicios Web WSDL es un lenguaje XML que contiene información acerca de la interfaz, semántica, y administración de una llamada a un servicio Web.

Una vez que se ha desarrollado un servicio Web, se publica su descripción y se construye un puntero en un repositorio UDDI (*Universal Description, Discovery and Integration*) para que los usuarios potenciales lo puedan utilizar. Cuando alguien piensa en utilizar este servicio Web, solicitan el archivo WSDL para conocer la ubicación del servicio, sintaxis de llamada de funciones, y cómo acceder al servicio Web. Luego utilizan la información en el archivo WSDL para construir una petición SOAP (*Simple Object Access Protocol*) y enviarla hacia el proveedor de servicio.

Una de las ideas centrales detrás de los servicios Web es que las aplicaciones futuras estarán conformadas de una colección de servicios habilitados en la red. Mientras haya dos servicios equivalentes que se publiciten a la red de una forma estándar y neutra, en teoría una aplicación podría seleccionar uno de ellos en base a criterios establecidos de antemano como precio o rendimiento. Además, algunos servicios podrían permitir que fueran copiados entre máquinas, permitiendo así que una aplicación que corra en una máquina (o cluster de máquinas) mejore en rendimiento al copiar servicios útiles a unidades de disco locales.

El Lenguaje de Descripción de Servicios Web (WSDL) es el equivalente de un resumen en XML -- describiendo los servicios Web, donde se ubican, y cómo se pueden invocar.

5.3.2.1.- Anatomía de un documento WSDL

Si se examina cada parte de un documento WSDL se encontrará: **<definitions>.** El elemento <definitions> contiene la definición de uno o más servicios. En la mayoría de los casos, un archivo WSDL define un servicio únicamente. Seguido de la etiqueta de definición se encontrarán declaraciones de algunos atributos.

Dentro de la etiqueta <definitions> se encuentran tres secciones conceptuales:

- **<message>** y **<portType>**, describe *qué* operaciones provee el servicio.
- **<binding>**, describe *cómo* se invocan las operaciones.
- **<service>**, describe *dónde* se ubica el servicio.
- **<documentation>**, cualquier elemento WSDL puede contener información del servicio para el usuario.

5.3.2.2.- Generando la descripción de servicio WSDL

La mayoría de las herramientas incluyen una forma de generar WSDL a partir de una componente, incluyendo el WSTK de IBM y el .NET Studio de Microsoft.

5.3.3.- UDDI (Universal Description, Discovery, and Integration)

A medida que el número de proveedores de servicios Web aumente, será necesario disponer de un sistema de referencia que permita localizar estos servicios. Este es el propósito del proyecto *UDDI*.

Los servicios Web es un nuevo paradigma en el desarrollo de sistemas distribuidos que proveerá una plataforma para todas las transacciones de comercio-*e* de negocio-a-negocio (B2B - *Business-to-Business*) en el Internet, así como también proporcionar una plataforma universal para la integración B2B.

Siempre ha sido un reto la comunicación entre los negocios a nivel de aplicaciones, dada la vasta existencia de plataformas, herramientas, mecanismos, y procesos que cada quien utiliza. La popularidad reciente del XML (*eXtensible Markup Language*) en estos sitios, promete una solución para el intercambio de datos de una forma transparente. También, la evolución de protocolos como el SOAP (*Simple Object Access Protocol*) proporciona una plataforma para el intercambio de servicios sobre la red.

Bien entonces, si el mecanismo de comunicación entre las plataformas es en XML, y la forma de comunicación es en SOAP, ¿cómo sabrán con quien comunicarse y dónde encontrar otros negocios?. La respuesta es el UDDI.

Las capas en la siguiente tabla ilustra la ubicación del UDDI en el contexto del resto de protocolos en la pila de interoperabilidad de servicios Web.

Pila de Interoperabilidad de Servicio Web	**UDDI (Universal Description, Discovery and Integration)**
	SOAP (Simple Object Access Protocol)
	XML (eXtensible Markup Language)
	Protocolos Comunes de Internet (HTTP, TCP/IP)

Descripción detallada de los elementos de la pila:

- **Protocolos Comunes de Internet** como el HTTP (ó pudiera ser SMTP) establece el marco de referencia básico para los servicios Web, aunque los servicios Web no necesariamente están atados a algún protocolo en particular.
- **XML** se ha vuelto el estándar universal para el intercambio de datos y sus estructuras, y establece la base de los servicios Web.
- **SOAP** proporciona un mecanismo sencillo y ligero para el intercambio de información estructurada y textual con XML en un ambiente descentralizado y distribuido.

- **UDDI** define una forma de publicar y descubrir información acerca de servicios Web. Una nube de registros UDDI, o registro de negocios (registro único y publicación generalizada), proporciona información de cómo acceder los servicios Web.

Cómo su nombre lo indica, el estándar UDDI provee un mecanismo para que los negocios se "describan" a si mismos y los tipos de servicios que proporcionan y luego se pueden registrar y publicarse en un Registro UDDI. Tales negocios publicados pueden ser buscados, consultados o "descubiertos" por otros negocios utilizando mensajes con SOAP.

5.3.3.1.- Secciones Blanca, Amarilla y Verde.

Conceptualmente, la información proporcionada de un negocio en un registro UDDI consta de tres componentes:

- La **Sección Blanca** es muy similar a la información que aparece en el directorio telefónico, que incluye nombre, teléfono, y dirección.
- La **Sección Amarilla** es muy similar a su equivalente telefónico, e incluyen categorías de catalogación industrial tradicionales, ubicación geográfica, etc. Mediante el uso de códigos y claves predeterminadas, los negocios se pueden registrar y así facilitar a otros servicios la búsqueda usando estos índices de clasificación.
- La **Sección Verde** contiene la información técnica acerca de los servicios ofrecidos por los negocios. Se incluyen referencias de especificaciones de servicios Web así como también información complementaria para los mecanismos diversos de búsqueda basados en URL.

5.3.3.2.- Estructura central de UDDI

La información que compone un registro UDDI consiste por el momento de cuatro tipos de estructuras de datos (cinco en la versión UDDI 2.0)

- **businessEntity:** Esta estructura captura la información sobre un negocio o entidad y que es utilizada por el negocio para publicar información descriptiva sobre si misma y los servicios que ofrece.
- **businessService:** Esta estructura representa los servicios o procesos de negocios que provee la estructura **businessEntity**.
- **bindingTemplate:** Esta estructura representa los datos importantes que describen las características técnicas de la implementación del servicio ofrecido.
- **tModel:** El papel principal de esta estructura es la de representar una especificación técnica.

5.3.3.3.- Características de UDDI

El UDDI provee dos categorías de API: El API de publicación y el API de consulta. El API de publicación, provee el mecanismo para que los proveedores de servicios se registren ellos mismos y sus servicios en el Registro UDDI.

El API de consulta permite a los subscriptores de servicios buscar los servicios disponibles. El API de consulta provee dos tipos de llamados, un mecanismo de búsqueda y un mecanismo de obtención, cuando ya se tiene disponible toda la información referente a la disponibilidad de un servicio.

6.- Plataforma Tecnológica: Microsoft

6.1.- ASP .NET

ASP.NET es un marco de trabajo generado en *Common Language Runtime* que puede utilizarse en un servidor para generar aplicaciones Web. Las ventajas que ofrece ASP.NET frente a modelos anteriores de programación Web son:

- **Mejor rendimiento.** ASP.NET es código compilado que se ejecuta en el servidor. ASP.NET utiliza compilación *just-in-time*, servicios de caché, y optimización para la máquina donde se ejecute, por lo que el incremento de rendimiento es superior

- **Compatibilidad con herramientas de primer nivel.** El marco de trabajo de ASP.NET se complementa con un diseñador y una caja de herramientas muy completos en el entorno integrado de programación (*Integrated Development Environment*, IDE) de Visual Studio. La edición WYSIWYG, los controles de servidor de arrastrar y colocar y la implementación automática son sólo algunas de las características que proporciona esta eficaz herramienta.

- **Eficacia y flexibilidad.** Debido a que ASP.NET se basa en *Common Language Runtime*, la eficacia y la flexibilidad de toda esa plataforma se encuentra disponible para los programadores de aplicaciones Web. La biblioteca de clases de .NET Framework, la Mensajería y las soluciones de Acceso a datos se encuentran accesibles desde el Web de manera uniforme. ASP.NET es también independiente del lenguaje, por lo que puede elegir el lenguaje que mejor se adapte a la aplicación o dividir la aplicación en varios lenguajes. Además, la interoperabilidad de *Common Language Runtime* garantiza que la inversión existente en programación basada en COM se conserva al migrar a ASP.NET.

- **Simplicidad.** ASP.NET facilita la realización de tareas comunes, desde el sencillo envío de formularios y la autenticación del cliente hasta la implementación y la configuración de sitios. Por ejemplo, el marco de trabajo de página de ASP.NET permite generar interfaces de usuario, que separan claramente la lógica de aplicación del código de presentación, y controlar eventos en un sencillo modelo de procesamiento de formularios de tipo Visual Basic. Además, *Common Language Runtime* simplifica la programación, con servicios de código administrado como el recuento de referencia automático y el recolector de elementos no utilizados.

- **Facilidad de uso.** ASP.NET emplea un sistema de configuración jerárquico, basado en texto, que simplifica la aplicación de la configuración al entorno de servidor y las aplicaciones Web. Debido a que la información de configuración se almacena como texto sin formato, se puede aplicar la nueva configuración sin la ayuda de herramientas de administración local. Esta filosofía de "administración local cero" se extiende asimismo a la implementación de las aplicaciones ASP.NET Framework. Una aplicación ASP.NET Framework se implementa en un servidor sencillamente mediante la copia de los archivos necesarios al servidor. No se requiere el reinicio del servidor, ni siquiera para implementar o reemplazar el código compilado en ejecución.

- **Escalabilidad y disponibilidad.** ASP.NET se ha diseñado teniendo en cuenta la escalabilidad, con características diseñadas específicamente a medida, con el fin de mejorar el rendimiento en entornos agrupados y de múltiples procesadores. Además, el motor de tiempo de ejecución de ASP.NET controla y administra los procesos de cerca, por lo que si uno no se comporta adecuadamente (filtraciones, bloqueos), se puede crear un proceso nuevo en su lugar, lo que ayuda a mantener la aplicación disponible constantemente para controlar solicitudes.

- **Posibilidad de personalización y extensibilidad.** ASP.NET presenta una arquitectura bien diseñada que permite a los programadores insertar su código en el nivel adecuado. De hecho, es posible extender o reemplazar cualquier subcomponente del motor de tiempo de ejecución de ASP.NET con su propio componente escrito personalizado. La implementación de la autenticación personalizada o de los servicios de estado nunca ha sido más fácil.

- **Seguridad.** Con la autenticación de Windows integrada y la configuración por aplicación, se puede tener la completa seguridad de que las aplicaciones están a salvo.

6.1.1.- Formularios Web

El marco de trabajo de la página de formularios Web de ASP.NET es un modelo de programación escalable de *Common Language Runtime* que puede utilizarse en el servidor para generar páginas Web dinámicamente.

Concebido como una evolución lógica de ASP (ASP.NET proporciona compatibilidad sintáctica con las páginas existentes), el marco de trabajo de formularios Web ASP.NET se ha diseñado específicamente para tratar varias deficiencias clave del modelo anterior. En particular, proporciona:

- Capacidad para crear y utilizar controles de la interfaz de usuario reutilizables que puedan encapsular funcionalidades comunes y, así, reducir la cantidad de código que tiene que escribir el programador de una página.
- Capacidad para que los programadores puedan estructurar limpiamente la lógica de la página de forma ordenada (no revuelta).
- Capacidad para que las herramientas de desarrollo proporcionen un fuerte soporte de diseño WYSIWYG (Lo que ve es lo que se imprime) a las páginas (el código ASP existente es opaco para las herramientas).

6.1.1.1.- Escribir la página de formularios Web

Las páginas de formularios Web de ASP.NET consisten en archivos de texto con una extensión de nombre de archivo .aspx. Pueden implementarse por todo un árbol de directorio raíz virtual IIS. Cuando un explorador cliente solicita recursos .aspx, el motor en tiempo de ejecución de ASP.NET analiza y compila el archivo de destino en una clase de .NET Framework. Esta clase puede utilizarse, a continuación, para procesar de forma dinámica las solicitudes entrantes. (Debe observarse que el archivo .aspx sólo se compila la primera que se tiene acceso al mismo; la instancia de tipo compilada se vuelve a utilizar en múltiples solicitudes).

6.1.1.2.- Utilizar bloques de representación ASP <%%>

ASP.NET proporciona compatibilidad sintáctica con páginas ASP existentes. Esto incluye compatibilidad para bloques de representación de código <% %> que pueden entremezclarse con contenido HTML dentro de un archivo .aspx. Estos bloques de código se ejecutan de arriba a abajo en tiempo de representación de página.

6.1.1.3.- Introducción a controles de servidor ASP.NET

Además de (o en vez de) utilizar bloques de código <% %> para programar contenido dinámico, los programadores de páginas ASP.NET pueden utilizar controles de servidor ASP.NET para programar páginas Web. Los controles de servidor se declaran dentro de un archivo .aspx mediante etiquetas personalizadas o etiquetas HTML intrínsecas que contienen un valor de atributo **runat="server"**. Las etiquetas HTML intrínsecas las controla uno de los controles del espacio de nombres **System.Web.UI.HtmlControls**. A cualquier etiqueta que no esté explícitamente asignada a uno de los controles se le asigna el tipo de **System.Web.UI.HtmlControls.HtmlGenericControl**.

Debe tenerse en cuenta que estos controles de servidor mantienen automáticamente cualquier valor introducido por el cliente entre acciones de ida y vuelta al servidor. El estado del control no se almacena en el servidor, sino en un campo del formulario **<input type="hidden">** que recibe acciones de ida y vuelta entre solicitudes. Hay que tener en cuenta que no se necesita ninguna secuencia de comando en el cliente.

6.1.1.4.- Controlar eventos de controles de servidor

Cada control de servidor ASP.NET puede exponer un modelo de objeto con propiedades, métodos y eventos. Los programadores de ASP.NET pueden utilizar este modelo de objeto para modificar e interactuar limpiamente con la página.

6.1.1.5.- Utilizar controles de servidor personalizados

ASP.NET incluye 45 controles de servidor integrados que se pueden utilizar fuera del cuadro. Además de utilizar los controles integrados de ASP.NET, los programadores también pueden utilizar controles desarrollados por otros fabricantes.

6.1.1.6.- Listas, datos y enlace de datos

ASP.NET incluye un conjunto integrado de controles de lista y cuadrícula de datos. Se pueden utilizar para proporcionar una interfaz de usuario personalizada basada en consultas a una base de datos o a otro origen de datos. El control **DataGrid <asp:datagrid runat=server>** proporciona una forma sencilla de visualizar rápidamente resultados de datos mediante una interfaz de usuario tradicional de control de cuadrícula. Como alternativa, los programadores de ASP.NET pueden utilizar el control **DataList <asp:DataList runat=server>** y una plantilla **ItemTemplate** personalizada para personalizar información de datos.

6.1.1.7.- Controles de validación de formulario

El marco de trabajo de la página de formularios Web de ASP.NET proporciona un conjunto de controles de servidor de validación que proporcionan a su vez un modo sencillo a la vez que potente de comprobar errores en los formularios de entrada y, en caso necesario, mostrar mensajes al usuario.

Los controles de validación se agregan a una página ASP.NET con otros controles de servidor. Existen controles para tipos concretos de validación, como la comprobación de intervalos o la coincidencia de modelos, además de **RequiredFieldValidator**, que se asegura de que un usuario omita un campo de entrada.

6.1.2.- Servicios Web de ASP.NET

Common Language Runtime proporciona soporte integrado para crear y exponer servicios Web, utilizando una abstracción de programación coherente con programadores de Web Forms ASP.NET y con usuarios existentes de Visual Basic que resulta familiar para ambos. El modelo resultante es escalable y ampliable y comprende estándares abiertos de Internet (HTTP, XML, SOAP, WSDL) de forma que cualquier cliente o dispositivo que cuente con servicios de Internet puede obtener acceso al modelo y lo puede consumir.

ASP.NET proporciona soporte para servicios Web con el archivo .asmx. Un archivo .asmx es un archivo de texto similar a un archivo .aspx. Estos archivos pueden formar parte de una aplicación ASP.NET que incluya archivos .aspx. De esta forma, se puede asignar una dirección URI a los archivos .asmx, como se hace con los archivos .aspx.

En el ejemplo siguiente se muestra un archivo .asmx muy sencillo.

```
<%@ WebService Language="C#" Class="HolaMundo" %>

using System;
using System.Web.Services;

public class HolaMundo : WebService {

    [WebMethod]
    public String HolaMundo()
   {
        return "Hola Mundo";
    }

}
```

Este archivo comienza con la directiva **WebService** ASP.NET y establece el lenguaje en C#, Visual Basic o JScript. A continuación, importa el espacio de nombres **System.Web.Services**. Debe incluir este espacio de nombres. Seguidamente, se declara la clase HolaMundo. Esta clase se deriva de la clase base **WebService**; tenga en cuenta que la derivación de la clase base **WebService** es opcional. Por último, cualquier método que sea accesible como parte del servicio tiene el atributo **[WebMethod]** en C#, **<WebMethod()>** en Visual Basic o **WebMethodAttribute** en JScript, delante de su firma.

Para que este servicio esté disponible, podemos asignar al archivo el nombre **HolaMundo.asmx** y colocarlo en un servidor denominado **Dominio.com** dentro de un directorio virtual cuyo nombre sea **miCarpeta**. Mediante un explorador Web, se podría insertar la dirección URL **http://Dominio.com/MiCarpeta/HolaMundo.asmx** y la página resultante mostraría los métodos públicos para este servicio Web (aquéllos marcados con el atributo **WebMethod**), así como los protocolos (SOAP o HTTP GET) que se pueden utilizar para invocar dichos métodos. Al insertar la dirección **http://Dominio.com/MiCarpeta/HolaMundo.asmx?WSDL** en el explorador, se devuelve un documento de Lenguaje de descripción del servicio Web (WSDL). Este documento WSDL es muy importante y lo utilizarán los clientes que obtengan acceso al servicio.

6.1.2.1.- Obtener acceso a servicios Web

Además de la tecnología de servidor ASP.NET que permite a los programadores crear servicios Web, .NET Framework proporciona conjuntos de herramientas y código sofisticados para consumir servicios Web. Como los servicios Web se basan en protocolos abiertos como SOAP (*Simple Object Access Protocol*), esta tecnología para cliente también se puede utilizar para consumir servicios Web que no estén basados en ASP.NET.

El SDK incluye una herramienta denominada WSDL.exe (*Web Services Description Language*). Esta herramienta basada en la línea de comandos se utiliza para crear clases *proxy* a partir de WSDL. Por ejemplo, podría escribir:

WSDL http://Dominio.com/MiCarpeta/HolaMundo.asmx?WSDL

para crear una clase proxy denominada HelloWorld.cs.

Esta clase se parecerá mucho a la clase creada en la sección anterior. Contendrá un método denominado *HolaMundo* que devuelve una cadena. Si compila una clase proxy en una aplicación y después llama a su método, la clase proxy empaqueta una solicitud SOAP a través de HTTP y recibe la respuesta codificada por SOAP, que se resolverá como una cadena.

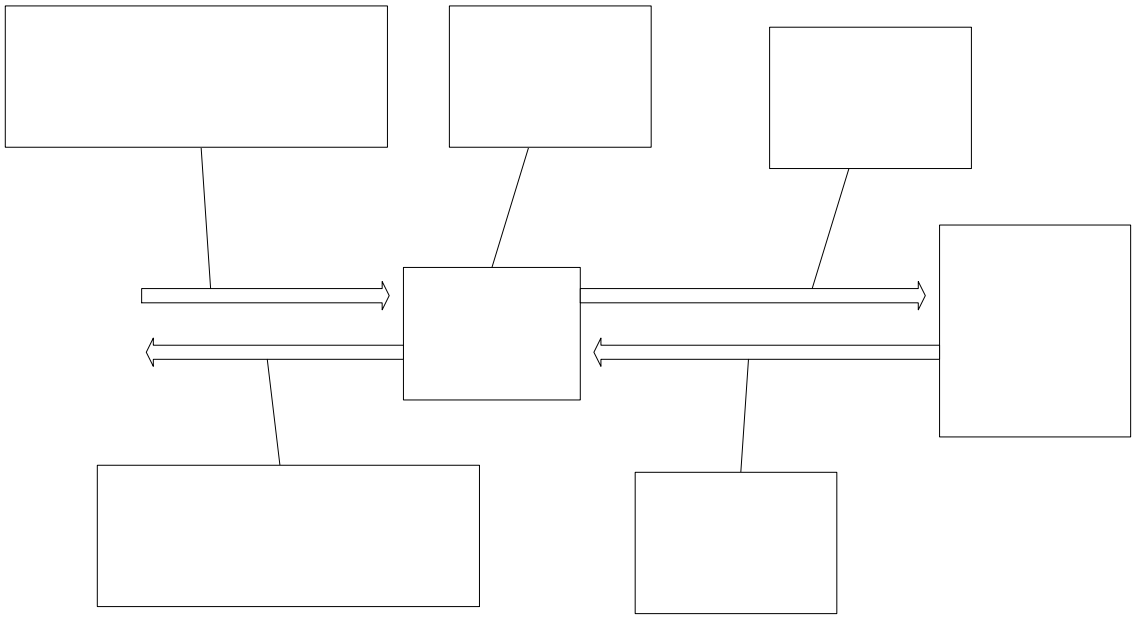
Desde la perspectiva del cliente, el código es sencillo, tal y como se muestra en el siguiente ejemplo.

```
HolaMundo miHolaMundo = new HolaMundo ();
String sReturn = miHolaMundo.HolaMundo ();
```

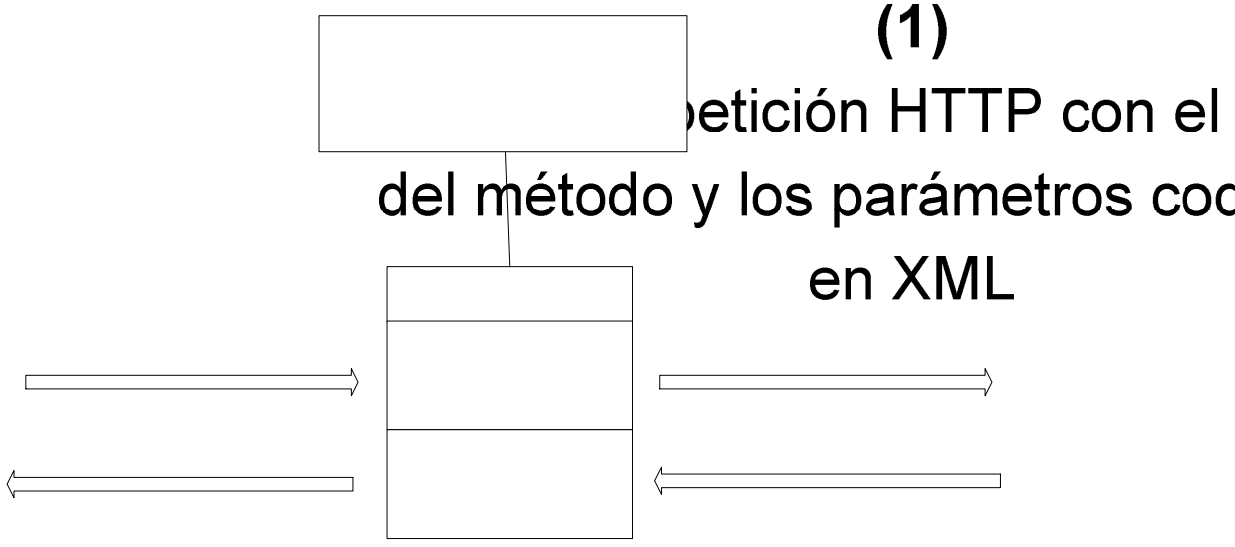
El resultado devuelto sería "Hola Mundo".

6.1.2.2.- Esquema de funcionamiento

6.1.2.2.1.- Servidor



6.1.2.2.2.- Cliente



6.1.2.3.- Cálculo de referencias de servicios Web de XML

En esta sección se muestra que se pueden pasar varios tipos de datos a métodos de **servicios Web** y devolverlos desde los mismos. Dado que la implementación de servicios Web de XML se genera en la parte superior de la arquitectura de serialización de XML, admite un importante número de tipos de datos. En la siguiente tabla se enumeran los tipos de datos admitidos para métodos de **servicios Web** cuando se utiliza el protocolo SOAP (por ejemplo, al utilizar el *proxy* generado por la herramienta del Lenguaje de descripción de servicios Web, WSDL.exe).

Tipo	**Descripción**
Tipos primitivos	Tipos primitivos estándar. En la lista completa de tipos primitivos admitidos se encuentran String, Char, Byte, Boolean, Int16, Int32, Int64, UInt16, UInt32, UInt64, Single, Double, Guid, Decimal, DateTime (como timeInstant de XML), DateTime (como fecha de XML), DateTime (como hora de XML) y XmlQualifiedName (como QName de XML).
Tipos Enum	Tipos de enumeración, como por ejemplo, "public enum color { red=1, blue=2 }"
Matrices de primitivos, Enums	Matrices de los tipos primitivos anteriores, como string[] e int[]
Clases y estructuras	Tipos de clases y estructuras con campos o propiedades públicas. Se pueden serializar los campos y propiedades públicos.
Matrices de clases (estructuras)	Matrices de los tipos anteriores.
DataSet	Tipos de DataSet de ADO.NET (vea la siguiente sección para obtener un ejemplo). También pueden aparecer DataSet como campos en estructuras y clases. Nota: Microsoft Visual Studio.NET y la utilidad del SDK XSD.EXE son compatibles para "establecer tipos inflexiblemente" en DataSet. Estas herramientas generan una clase que se hereda de DataSet para producir DataSet1 al agregar varios métodos, propiedades, etc. que son específicos para un esquema XML concreto. Si se pasa DataSet, los servicios Web de XML siempre transmiten el esquema junto con los datos (de forma que sabe qué tablas y columnas se están pasando) y los tipos correspondientes (por ejemplo, int, string). Si se pasa una subclase de DataSet (por ejemplo, DataSet1), los servicios Web de XML

	suponen que se agregan tablas o columnas al constructor, así como que esas tablas o columnas representan el esquema.
Matrices de DataSet	Matrices del tipo anterior.
XmlNode	XmlNode consiste en una representación en memoria de un fragmento de XML (como un modelo ligero de objeto de documento de XML). Por ejemplo, "<comment>This ispretty neat</comment>" podría almacenarse en un tipo XmlNode. Se pueden pasar tipos XmlNode como parámetros y se agregan al resto de XML que se está pasando al servicio Web de XML (los otros parámetros) de forma compatible con SOAP. Lo mismo sirve para los valores devueltos. Esto permite pasar o devolver XML cuya estructura cambia de llamada en llamada o donde no se pueden conocer todos los tipos que se están pasando. También pueden aparecer tipos XmlNode como campos en estructuras y clases.
Matrices de XmlNode	Matrices del tipo anterior.

Valores devueltos:

Tanto si se llama a un servicio Web de XML mediante SOAP o HTTP GET/POST, todos los tipos anteriores son compatibles con los valores devueltos. Parámetros: Tanto los parámetros por valor, como por referencia (in/out) son compatibles si se utiliza el protocolo SOAP. Los parámetros por referencia pueden enviar el valor en dos direcciones: hasta el servidor y de vuelta al cliente. Cuando se analizan parámetros de entrada a un servicio Web de XML mediante HTTP GET/POST, sólo se admite un conjunto limitado de tipos de datos que deben ser parámetros por valor. A continuación se enumeran los tipos compatibles con los parámetros GET/POST de HTTP:

Tipo	**Descripción**
Tipos primitivos (limitados)	La mayoría de los tipos primitivos estándar. La lista completa de tipos primitivos admitidos comprende los tipos Int32, String, Int16, Int64, Boolean, Single, Double, Decimal, DateTime, UInt16, UInt32, UInt64 y Currency. Desde el punto de vista del cliente, todos estos tipos se transforman en cadenas.
Tipos Enum	Tipos de enumeración, como por ejemplo, "public enum color { red=1, blue=2 }". Desde el punto de vista del cliente, los tipos enum se convierten en clases con una cadena const estática para cada valor.
Matrices de primitivos, Enums	Matrices de los tipos primitivos anteriores, como string[] e int[]

6.1.3.- Información general de una aplicación

ASP.NET define una aplicación como el conjunto de todos los archivos, páginas, controladores, módulos y código ejecutable que se pueden invocar o ejecutar dentro del ámbito de un determinado directorio virtual (y sus subdirectorios) en un servidor de aplicaciones Web. Por ejemplo, una aplicación de "pedido" podría publicarse dentro del directorio virtual "/pedido" de un servidor Web. Para IIS, el directorio virtual se puede configurar en el Administrador de servicios de Internet y contiene todos los subdirectorios, a menos que los propios subdirectorios sean directorios virtuales.

Cada aplicación ASP.NET Framework de un servidor Web se ejecuta dentro de un dominio único de aplicaciones ejecutables de .NET Framework, lo que garantiza el aislamiento de clases (no se producen conflictos de nombres o versiones), el uso seguro de recursos (se impide el acceso a determinados equipos o recursos de red) y el aislamiento de variables estáticas.

ASP.NET mantiene una agrupación de instancias **HttpApplication** durante el período de duración de una aplicación Web. ASP.NET asigna automáticamente una de estas instancias para procesar cada solicitud HTTP entrante recibida por la aplicación. La instancia **HttpApplication** asignada en particular es responsable del proceso de la solicitud a lo largo de todo su período de duración y sólo se puede volver a utilizar después de que la solicitud se haya completado. Esto significa que el código de usuario incluido en la instancia **HttpApplication** no necesita ser reentrante.

6.1.3.1.- Crear una aplicación

Para crear una aplicación ASP.NET Framework, se puede utilizar un directorio virtual existente o crear uno nuevo. Por ejemplo, si Windows 2000 Server se instaló con IIS, probablemente existirá un directorio C:\\InetPub\\WWWRoot. IIS se puede configurar mediante el Administrador de servicios de Internet, que se encuentra en Inicio -> Programas -> Herramientas administrativas. Haga clic con el botón secundario del *mouse* (ratón) en un directorio existente y elija Nuevo (para crear un nuevo directorio virtual) o Propiedades (para convertir un directorio normal existente).

Cuando se coloca una simple página .aspx, como la siguiente, en el directorio virtual, y se obtiene acceso a ella con el explorador, se desencadena el proceso de creación de la aplicación ASP.NET.

```
<%@Page Language="C#"%>
<html>
<body>
<h1> Hola Mundo, <% Response.Write(DateTime.Now.ToString()); %></h1>
</body>
</html>
```

6.1.3.2.- Duración de una aplicación

Una aplicación ASP.NET Framework se crea la primera vez que se realiza una solicitud al servidor; antes de ello, no se ejecuta ningún código ASP.NET. Cuando se realiza la primera solicitud, se crea una agrupación de instancias **HttpApplication** y se provoca el evento **Application_Start**. Las instancias **HttpApplication** procesan esta solicitud y las siguientes hasta que la última instancia termina y se provoca el evento **Application_End**.
Observe que los métodos **Init** y **Dispose** de **HttpApplication** se invocan por cada instancia y, de este modo, pueden utilizarse varias veces entre **Application_Start** y **Application_End**. Sólo se comparten estos eventos entre todas las instancias de **HttpApplication** en una aplicación ASP.NET.

6.1.3.3.- Administrar el estado de las aplicaciones

Ya que distintos subprocesos pueden tener acceso a una aplicación y a todos los objetos que contiene dicha aplicación de forma simultánea, es mejor almacenar con ámbito de aplicación únicamente los datos que se modifican con poca frecuencia. Idealmente, los objetos se inicializan en el evento **Application_Start** y los accesos posteriores son de sólo lectura.

En el siguiente ejemplo se lee un archivo en **Application_Start** (definido en el archivo Global.asax) y el contenido se almacena en un objeto **DataView** en el estado de la aplicación.

```
void Application_Start()
{
   DataSet ds = new DataSet();

   FileStream fs = new
   FileStream(Server.MapPath("schemadata.xml"),FileMode.Open,FileAccess.Read);
   StreamReader reader = new StreamReader(fs);
   ds.ReadXml(reader);
   fs.Close();

   DataView view = new DataView(ds.Tables[0]);
   Application["Fuente"] = view;
}
```

En el método **Page_Load** se recupera el objeto **DataView** y después se utiliza para llenar un objeto **DataGrid**:

```
void Page_Load(Object sender, EventArgs e) {
   DataView Source = (DataView)(Application["Fuente"]);
   ...
   MyDataGrid.DataSource = Source;
   ...
}
```

6.1.3.4.- Utilizar el estado de las sesiones

Es posible almacenar los datos con ámbito de sesión a fin de proporcionar datos individuales a un usuario durante una sesión. En el ejemplo siguiente se inicializan los valores de preferencias de usuario en el evento **Session_Start** del archivo Global.asax.

```
void Session_Start() {
   Session["Color"] = "beige";
   ...
}
```

En la siguiente página de personalización, se modifican los valores de las preferencias de usuario en el controlador de eventos **Submit_Click** con la información aportada por el usuario.

```
protected void Submit_Click(Object sender, EventArgs e) {
   Session["Color"] = Color.Value;
   ...
   Response.Redirect(State["Referer"].ToString());
}
```

Es posible configurar las características del estado de una sesión en la sección **<sessionState>** de un archivo Web.config. Para doblar el tiempo de espera predeterminado de 20 minutos, se puede agregar el código siguiente al archivo Web.config de una aplicación:

```
<sessionState timeout="40"/>
```

De forma predeterminada, ASP.NET almacenará el estado de la sesión en el mismo proceso que atiende la solicitud, de la misma manera que ASP. Si no hay cookies disponibles, se puede hacer un seguimiento de una sesión agregando un identificador de sesión a la dirección URL. Se puede habilitar de la manera siguiente:

```
<sessionState cookieless="true"/>
```

De forma predeterminada, ASP.NET almacenará el estado de la sesión en el mismo proceso que atiende la solicitud, de la misma manera que ASP. Además, ASP.NET puede almacenar datos de sesión en un proceso externo, que podría residir en otro equipo. Para habilitar esta función:

- Inicie el servicio de estado de ASP.NET mediante el complemento Servicios o ejecutando la instrucción "net start aspnet_state" desde la línea de comandos. De manera predeterminada, el servicio de estado escuchará en el puerto 42424. Para cambiar el puerto, modifique la clave del Registro para el servicio: *HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\aspnet_state\Parameters\Port*
- Establezca el atributo **mode** de la sección **<sessionState>** en "StateServer".
- Configure el atributo **stateConnectionString** con los valores del equipo en que se inició aspnet_state.

En el siguiente ejemplo se da por hecho que el servicio de estado se ejecuta en el mismo equipo que el servidor Web ("localhost") y utiliza el puerto predeterminado (42424):

```
<sessionState  mode="StateServer"
  stateConnectionString="tcpip=localhost:42424" />

Sub Page_Load(sender As Object, e As EventArgs)
    Dim Source As New DataView  = CType(Application("Source"), DataView)
    ...
    MyDataGrid.DataSource = Source
    ...
End Sub

function Page_Load(sender:Object, e:EventArgs) : void {
    var Source:DataView = DataView((Application("Source")));
    ...
    MyDataGrid.DataSource = Source;
    ...
}
```

6.1.3.5.- Diseño del sistema de archivos de las aplicaciones ASP.NET

Se puede utilizar ASP.NET para albergar múltiples aplicaciones Web, cada una identificada mediante un prefijo URL único dentro de un sitio Web (un sitio Web se representa en un servidor Web como una combinación **NombreDeHost**/**Puerto**). Por ejemplo, un único servidor Web de Microsoft Internet Information Services (IIS) con dos direcciones IP (una con alias "www.fdi.ucm.es" y otra "intranet") y tres sitios lógicos (http://intranet, http:// www.fdi.ucm.es, http www.fdi.ucm.es port 81) podría exponer las siguientes seis aplicaciones ASP.NET.

URL de la aplicación	**Descripción**
http://intranet	Aplicación "Root" del sitio de la intranet.
http:// www.fdi.ucm.es	Aplicación "Root" en el sitio www.fdi.ucm.es.
http:// www.fdi.ucm.es:81	Aplicación "Root" en el sitio www.fdi.ucm.es, puerto 81.
http://intranet/cursos	Aplicación "cursos" del sitio de la intranet.
http://intranet/dacya	Aplicación "dacya" del sitio de la intranet.
http://intranet/dacya/resolutor/	Aplicación "resolutor" del sitio de la intranet.

Nota: la URL de la aplicación "resolutor" mencionada en la tabla está arraigada dentro del espacio de nombres de la URL de la aplicación DACYA. No obstante, esta notación jerárquica de URL no implica que la aplicación "resolutor" se encuentre incluida o anidada dentro de la aplicación DACYA.

Cada aplicación ASP.NET Framework expuesta en un espacio de nombres de URL está respaldada por un directorio del sistema de archivos ubicado en un recurso compartido local o remoto. Los directorios de aplicaciones no necesitan estar ubicados centralmente en una parte contigua del sistema de archivos; pueden estar esparcidos por todo un disco. Por ejemplo, las aplicaciones ASP.NET mencionadas anteriormente podrían estar ubicadas en los diferentes directorios que aparecen en la tabla siguiente.

URL de la aplicación	**Ruta de acceso física**
http://intranet	c:\inetpub\wwwroot
http:// www.fdi.ucm.es	c:\inetpub\fdiroot
http:// www.fdi.ucm.es:81	d:\fdiroot81
http://intranet/cursos	d:\serverapps\cursosapp
http://intranet/dacya	\\DACYAweb\
http://intranet/dacya/resolutor/	c:\inetpub\wwwroot\resolutor

6.1.3.6.- Resolver referencias de clases en ensamblados

Los ensamblados constituyen la unidad de implementación de clases en *Common Language Runtime*. Los programadores que escriben clases de .NET Framework utilizando la versión 7.0 de Visual Studio .NET producen un nuevo ensamblado con cada proyecto de Visual Studio que compilan. Aunque es posible hacer que un ensamblado ocupe varios archivos ejecutables portables (PE) (varias DLL de módulos), Visual Studio .NET compilará, de forma predeterminada, todo el código del ensamblado en una única DLL (1 proyecto de Visual Studio .NET = 1 ensamblado de .NET Framework = 1 DLL física).

Para utilizar un ensamblado en un equipo, se debe implementar en una caché de ensamblado. La caché de ensamblado puede ser global para un equipo o local para una aplicación determinada. En la caché de ensamblado global del sistema, sólo debería colocarse el código que se va a compartir entre varias aplicaciones. El código específico para una aplicación particular, como la lógica de la mayoría de las aplicaciones Web, se debería implementar en la caché de ensamblado local de la aplicación. Una de las ventajas de distribuir un ensamblado en la caché de ensamblado local de una aplicación es que sólo el código interno de esa aplicación puede tener acceso a él. (Se trata de una característica muy apreciada para escenarios en los que intervienen proveedores de servicios de Internet). También facilita la creación conjunta de versiones de la misma aplicación, ya que las clases son privadas para cada instancia de versión de la aplicación.

Un ensamblado se puede distribuir en la caché de ensamblado local de una aplicación simplemente copiando (o utilizando XCOPY o FTP) los archivos apropiados en un directorio marcado como "ubicación para la caché de ensamblado" de esa determinada aplicación. No es necesario ejecutar ninguna herramienta de registro adicional una vez copiados los archivos apropiados y tampoco es necesario reiniciar el equipo. Esto elimina algunas de las dificultades actualmente asociadas a la distribución de componentes COM dentro de aplicaciones ASP (actualmente, un administrador debe iniciar sesión localmente en el servidor Web y ejecutar Regsvr32.exe).

De forma predeterminada, una aplicación ASP.NET Framework se configura automáticamente para utilizar el subdirectorio \\bin, ubicado inmediatamente bajo la raíz de la aplicación, como su caché de ensamblado local. El directorio \\bin también está configurado para denegar cualquier acceso de un explorador Web, de modo que un cliente remoto no pueda descargar el código y apropiarse del mismo. El siguiente ejemplo muestra una posible distribución de directorios para una aplicación de ASP.NET, donde el directorio \bin se encuentra justo debajo de la raíz de la aplicación.

C:\inetpub\wwwroot
Default.aspx

\bin <= Application assembly cache directory
MyPages.dll

\order
SubmitOrder.aspx

\img
HappyFace.gif

6.1.3.7.- Inicio de aplicaciones ASP.NET Framework y resolución de clases

Las aplicaciones ASP.NET Framework se construyen lentamente la primera vez que un cliente solicita un recurso URL de ellas. Cada aplicación ASP.NET Framework se lanza dentro de un dominio de aplicación exclusivo (**AppDomain**), que consiste en una nueva construcción de *Common Language Runtime* que permite a los *hosts* de procesos ofrecer código extensivo, seguridad y aislamiento de configuraciones durante la ejecución.

ASP.NET se encarga de crear manualmente un dominio de aplicaciones cuando se inicia una nueva aplicación. Como parte de este proceso, ASP.NET proporciona valores de configuración para que *Common Language Runtime* los utilice. Entre estos valores de configuración, se incluyen:

- Las rutas de acceso a directorios que componen la caché de ensamblado local. (Nota: es la arquitectura de aislamiento de dominios para aplicaciones de .NET Framework la que permite a cada aplicación mantener su propia caché de ensamblado local).
- Las restricciones de seguridad de la aplicación (donde puede tener acceso la aplicación en el sistema).

Como ASP.NET no tiene conocimiento, en tiempo de compilación, de las aplicaciones que se escriben por encima de él, no puede utilizar referencias estáticas para resolver y hacer referencia al código de las aplicaciones. En vez de ello, ASP.NET debe utilizar un enfoque de resolución dinámico de clases y ensamblados para realizar la transición del módulo de ejecución de ASP.NET al código de aplicación.

Los archivos de activación de página y configuración de ASP.NET permiten hacer referencia dinámicamente a una clase de .NET Framework compilada para un objetivo especificando una combinación de nombre de clase y ensamblado. El formato de cadena para esta unión sigue el modelo *classname, assemblyname.*
Llegados a este punto, *Common Language Runtime* puede utilizar esta referencia simple de cadena para resolver y cargar la clase apropiada.

6.1.3.8.- Sustitución de código

Los ensamblados de .NET Framework normalmente se compilan y distribuyen en un formato PE basado en DLL de Windows. Cuando el cargador de *Common Language Runtime* resuelve una clase implementada con este tipo de ensamblado, llama a la rutina *LoadLibrary* de Windows en el archivo (la cual bloquea su acceso en disco) y, a continuación, asigna los datos de código en memoria apropiados para la ejecución. Una vez cargado, el archivo DLL permanece bloqueado en disco hasta que el dominio de aplicación que hace referencia a él se destruye o se recicla manualmente.

Aunque ASP.NET no puede impedir que *Common Language Runtime* bloquee una DLL de ensamblado en disco, sí puede garantizar que el módulo de ejecución no cargue nunca realmente las DLL físicas en la caché de ensamblado privada de una aplicación Web. En vez de ello, se realizan copias en servidores de copia de seguridad de las DLL de ensamblados inmediatamente antes de su uso. El módulo de ejecución bloquea y carga entonces estos ensamblados del servidor de copia de seguridad (no los archivos originales).

Como los archivos de ensamblado originales siempre permanecen desbloqueados, existe la libertad de eliminarlos, reemplazarlos o cambiar su nombre sin activar el servidor Web o tener que usar una utilidad de registro. FTP y los métodos similares funcionan sin problemas. ASP.NET mantiene una lista activa de todos los ensamblados cargados dentro del dominio de una aplicación particular y utiliza código de supervisión de cambio de archivos para detectar cualquier actualización en los archivos originales.

6.1.4.- Información general sobre seguridad

Una parte importante de muchas aplicaciones Web radica en la capacidad de identificar usuarios y controlar el acceso a los recursos. Se conoce como autenticación al acto de determinar la identidad de la entidad solicitante. Por lo general, el usuario deberá presentar sus credenciales, como el par nombre de usuario y contraseña, para ser autenticado. Cuando se encuentre disponible una identidad autenticada, deberá determinarse si esa identidad puede tener acceso a un recurso específico. Este proceso se conoce como autorización. ASP.NET e IIS colaboran para proporcionar servicios de autenticación y autorización a las aplicaciones.

Una característica importante de los objetos COM es la capacidad de controlar la identidad con la que se ejecuta el código de objeto COM. Se conoce como representación al hecho de que un objeto COM ejecute código con la identidad de la entidad solicitante. Opcionalmente, las aplicaciones ASP.NET Framework pueden representar solicitudes.

Es posible que algunas aplicaciones también personalicen dinámicamente el contenido en función de la identidad del solicitante o de un conjunto de funciones al que pertenece la identidad del solicitante. Las aplicaciones ASP.NET Framework pueden comprobar dinámicamente si la entidad solicitante actual participa en una función concreta. Por ejemplo, es posible que una aplicación intente comprobar si el usuario actual pertenece a la función del administrador a fin de generar condicionalmente contenido para los administradores

6.1.4.1.- Autenticación y autorización

ASP.NET funciona, junto con IIS, para permitir la autenticación de tipo básica, implícita y Windows. ASP.NET es compatible con el servicio de autenticación de pasaporte de Microsoft, el cual proporciona servicios simples de firma y servicios de perfiles de usuario. ASP.NET también proporciona un robusto servicio para aplicaciones que necesitan utilizar autenticación basada en formularios. La autenticación basada en formularios utiliza "cookies" para autenticar a los usuarios, y permite a la aplicación realizar su propia verificación de credenciales.
Es importante darse cuenta de que los servicios de autenticación de ASP.NET dependen de los servicios de autenticación suministrados por IIS. Por ejemplo, para poder utilizar una autenticación básica en una aplicación IIS, se debe configurar el uso de la autenticación básica para la aplicación mediante el Administrador de servicios Internet.

ASP.NET proporciona dos tipos de servicios de autorización:
Comprobaciones de ACLs o permisos sobre un recurso para determinar si la cuenta de usuario autenticada puede obtener acceso a los recursos
Autorización de URL, la cual autoriza una identidad para partes del espacio Web
Para ilustrar la diferencia, considere un escenario en el que una aplicación está configurada para permitir acceso anónimo mediante la cuenta *IUSR_MIMAQUINA*. Cuando una solicitud de una página ASP.NET (como *"/default.aspx*") recibe autorización, se realiza una comprobación de los ACL de ese archivo (por ejemplo, "*c:\\inetpub\\wwwroot\\default.aspx*") para ver si la cuenta *IUSR_MIMAQUINA* tiene permiso para leer el archivo. Si lo tiene, entonces se autoriza el acceso. La autorización de archivo se realiza automáticamente.

Para autorización de URL, se comprueba el usuario anónimo con los datos de configuración obtenidos para la aplicación ASP.NET. Si se permite el acceso a la dirección URL solicitada, la solicitud se autoriza. En este caso, ASP.NET comprueba si el usuario anónimo dispone de acceso a */Default.aspx* (es decir, la comprobación se realiza sobre la propia URL, no sobre el archivo en el que se resuelve la URL en última instancia).

Ésta podría parecer una diferencia sutil, pero permite a las aplicaciones utilizar esquemas de autenticación como los basados en formularios o la autenticación de Pasaporte, en los cuales los usuarios no se corresponden con un equipo o una cuenta de dominio. También permite la autorización de recursos virtuales, para los cuales no existe un archivo físico subyacente al recurso. Por ejemplo, una aplicación podría optar por asignar todas las solicitudes de archivos que terminan en .stk a un controlador que proporciona cotizaciones de valores según ciertas variables presentes en la cadena de consulta. En ese caso, no existe un archivo .stk con el que hacer las comprobaciones ACL; por lo tanto, se utiliza autorización de URL para controlar el acceso al recurso virtual.

La autorización de archivo se realiza siempre con la cuenta autenticada suministrada por IIS. Si se permite el acceso anónimo, esta cuenta es la cuenta anónima configurada. En cualquier otro caso, se utiliza una cuenta NT. Esto funciona exactamente de la misma forma que ASP.
Las listas ACL de archivos se configuran para un determinado archivo o directorio mediante la ficha **Seguridad** de la página de propiedades del Explorador. La autorización de URL se configura como parte de una aplicación ASP.NET.
Para activar un servicio de autenticación de ASP.NET, se debe configurar el elemento **<authentication>** en el archivo de configuración de la aplicación. Este elemento puede tener uno de los valores de la tabla siguiente.

Valor	Descripción
Ninguno	No hay ningún servicio de autenticación de ASP.NET activo. Observe que los servicios de autenticación de IIS pueden estar aún presentes.
Windows	Los servicios de autenticación de ASP.NET asocian un **WindowsPrincipal** (**System.Security.Principal.WindowsPrincipal**) a la solicitud actual para permitir la autorización de usuarios o grupos de NT.
Formularios	Los servicios de autenticación de ASP.NET administran las "cookies" y desvían a los usuarios no autenticados a una página de inicio de sesión. Este método se suele utilizar en conjunción con la opción IIS que permite el acceso anónimo a una aplicación.
Passport	Los servicios de autenticación de ASP.NET proporcionan un envoltorio apropiado para los servicios suministrados por el kit de desarrollo de software para pasaporte (Passport SDK), el cual debe instalarse en el equipo.

Por ejemplo, el siguiente archivo de configuración permite la autenticación basada en formularios (cookie) para una aplicación:

```
<configuration>
  <system.web>
    <authentication mode="Forms"/>
  </system.web>
</configuration>
```

6.1.5.- Introducción sobre el rendimiento

Aquellas aplicaciones Web que presenten muchas características no serán de utilidad si no funcionan correctamente. Las exigencias de las aplicaciones Web son de tal naturaleza que, ahora más que nunca, se espera del código que haga más en menos tiempo. En esta sección se describen algunos principios clave del rendimiento de aplicaciones Web, sugerencias para escribir código que se ejecute correctamente y herramientas para medir el rendimiento.

ASP.NET proporciona varias mejoras de rendimiento integradas. Por ejemplo, las páginas sólo se compilan una vez y se almacenan en caché para posibles solicitudes posteriores. Como estas páginas compiladas se guardan en disco, seguirán siendo válidas incluso después de reiniciar completamente el servidor. ASP.NET también almacena en caché objetos internos, como variables de servidor, para acelerar el acceso del código del usuario. Además, ASP.NET aprovecha todas las mejoras de rendimiento de la biblioteca *Common Language Runtime*: compilación incremental (Just-in-time), una biblioteca *Common Language Runtime* optimizada para equipos de un solo procesador o multiprocesador, etc.

Sin embargo, todas estas mejoras no sirven de nada si el código está mal escrito. Por último, debe asegurarse de que la aplicación satisface las necesidades de los usuarios. En la próxima sección se describen algunas de las formas comunes de evitar cuellos de botella en lo que se refiere al rendimiento. Pero primero es necesario comprender las métricas siguientes:

- **Rendimiento**: número de solicitudes que una aplicación Web puede atender por unidad de tiempo, generalmente medido en solicitudes/segundo. El rendimiento puede variar en función de la carga (número de subprocesos cliente) del servidor. Normalmente, se suele considerar que ésta es la métrica de rendimiento más importante que hay que optimizar.
- **Tiempo de respuesta**: tiempo transcurrido entre la emisión de una solicitud y el primer byte devuelto al cliente desde el servidor. Éste suele ser el aspecto del rendimiento que mejor puede percibir el usuario cliente. Si una aplicación tarda mucho en responder, el usuario podría impacientarse y pasar a otro sitio. El tiempo de respuesta puede variar independientemente de la tasa de rendimiento (incluso inversamente).
- **Tiempo de ejecución**: tiempo que tarda en procesarse una solicitud, generalmente entre el primer y el último byte devuelto al cliente desde el servidor. El tiempo de ejecución afecta directamente al cálculo del rendimiento.
- **Escalabilidad**: medida de la capacidad de una aplicación de ofrecer mejor rendimiento a medida que se le asignen más recursos (memoria, procesos o equipos). Generalmente se trata de una medida de la tasa de cambio del rendimiento con respecto al número de procesadores.

La base de la programación de aplicaciones con buen rendimiento es lograr un equilibrio entre estas métricas. Ninguna medida individual puede caracterizar el comportamiento de la aplicación en distintas circunstancias; varias medidas realizadas a la vez pueden ofrecer una imagen aproximada del rendimiento de una aplicación.

7.- Lenguaje LRPR para especificación de problemas de restricciones

La declaración de restricciones se realiza mediante el lenguaje **LRPR (Lenguaje de Resolución por Propagación de Restricciones)**, cuya notación EBNF se describe a continuación, y que ha sido creado para el proyecto.

Programa ::= Declaraciones Restricciones FuncionObjetivo?

Declaraciones ::= 'd' 'e' 'c' 'l' 'a' 'r' 'a' 't' 'i' 'o' 'n' 's' ';' Variable+

Variable ::= 'v' 'a' 'r' <espacio>+ NombreVariable <espacio>+ LimiteInf '.' '.' LimiteSuperior';'

NombreVariable ::= <letra>+

LimiteInferior ::= '-'? <digito>+

LimiteSuperior ::= '-'? <digito>+

Restricciones ::= '{' ';' Restriccion* '}' ';'

Restriccion ::= PrimerSumando <espacio>+ SiguienteSumando* <espacio>+ TerminoIndependiente? <espacio>+ Operador '0' ';'

PrimerSumando ::= '-'? Coeficiente? Variable

SiguienteSumando ::= ('+' | '-') Coeficiente? Variable

TerminoIndependiente ::= ('+' | '-')<digito>+

Operador ::= '=' | '<">' | '<' | '<"=' | '>' | '>"='

FuncionObjetivo ::= Objetivo Funcion

Objetivo ::= 'o' 'b' 'j' 'e' 't' 'i' 'v' 'e' <espacio>+ Tipo ';'

Tipo ::= 'm' 'a' 'x' | 'm' 'i' 'n'

Funcion ::= PrimerSumando <espacio>+ SiguienteSumando* <espacio>+ TerminoIndependiente? ';'

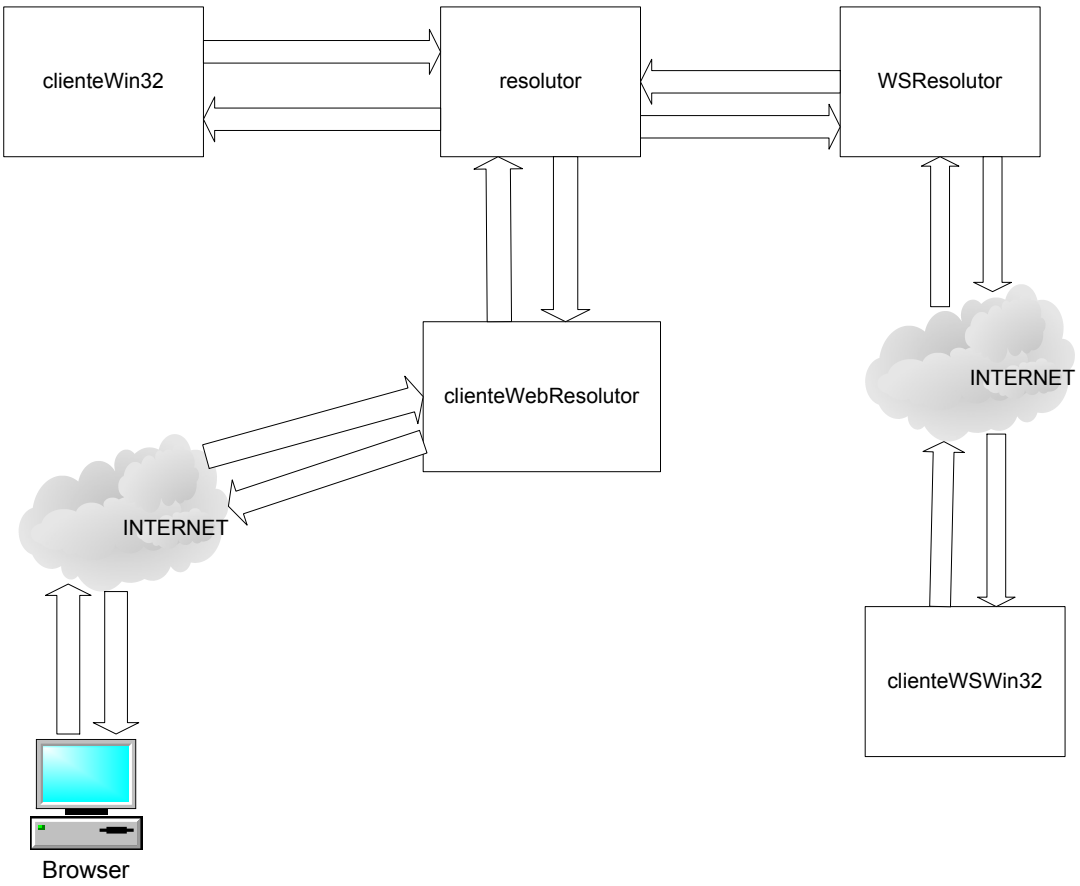
8.- **Implementación de los métodos de optimización**

En este apartado describiremos cada uno de los métodos de optimización integrados en el sistema: Dominios Finitos, algoritmos Genéticos y Ramificación y Acotación.

8.1.- Optimización con Restricciones sobre Dominios Finitos

Está compuesta de cinco componentes, a saber, el componente resolutor (*Resolutor.dll*) que realiza la operativa del algoritmo y que está implementado como *Component Library* de .NET(.dll), un servicio Web (*WSResolutor*) que se encarga de invocar al componente resolutor y que está implementado con la tecnología de Servicios Web de Microsoft .NET, un cliente Windows (*clienteWin32.exe*) que invoca al componente *Resolutor.dll,* un cliente Windows (*clienteWSWin32.exe*) que invoca al servicio Web *WSResolutor,* y una aplicación Web (*clienteWebResolutor*) realizado con la tecnología ASP.NET, y que se encarga de invocar al componente *Resolutor.dll* para la resolución del problema.

En la siguiente figura, se especifica el diagrama de contexto de todos los módulos de los que consta el proyecto.



A continuación se explica cada uno de los componentes de los que consta la implementación de resolución mediante propagación de restricciones.

8.1.1.- Resolutor

El componente resolutor, es un *Component Library(.dll)* de Microsoft .NET, que implementa el algoritmo de de resolución de problemas de variable de dominio finito mediante el método de propagación de restricciones.

Tiene dos métodos de entrada, uno para la resolución del problema en un solo paso, y otro para la resolución paso a paso. El primer método (resolver), recibe los datos del cliente, en forma de arrays, que son las variables (string[]), los límites inferiores (int[]), los límites superiores (int[]), las restricciones (int[][]), y los operadores (string[]), y realiza tanto la propagación inicial como el etiquetado, es decir, todo el algoritmo, y devuelve el resultado en un objeto ArrayList (objeto del framework Microsoft .NET). Este método, es llamado desde la aplicación Windows (clienteWin32), el servicio Web (WSResolutor) y desde la aplicación Web (clienteWebResolutor).

Si se ha especificado función de optimización para el problema, tanto de maximización como de minimización, el resultado devuelto, consiste en la tupla de valores de las variables que optimizan la función objetivo, además del valor de la misma. Si por el contrario no se ha especificado función de optimización, el resultado es una lista de todas las tuplas de valores de las variables que cumplen la restricciones.

Para resolver paso a paso el problema no se puede tener un solo método de entrada, ya que los pasos de propagación se realizan de forma asíncrona desde los clientes, a petición del usuario. Por tanto para este tipo de resolución, hay varios pasos, que se aplican sobre el componente resolutor, en el primer paso, que es la llamada a *resolverPasoAPaso,* se crean los objetos y las reglas. Lo siguiente, es cada paso de propagación que se realiza mediante las llamadas sucesivas a *aplicarReglasPasoAPaso,* que aplica las todas la reglas de la restricción actual, y por último es la búsqueda de resultados, sobre los dominios restringidos en los anteriores pasos.

8.1.1.1.- Diagrama de clases

Las clases de las que se compone el resolutor son las siguientes:

- **CResolutor**, que realiza la lógica del algoritmo, y que trata con los arrays de variables, restricciones y con la función objetivo. Es la clase que hay que instanciar desde los clientes, para poder hacer las llamadas a *resolver, resolverPasoAPaso, aplicarReglasPasoAPaso y búsqueda.*

- **CVariable**, que representa una variable, y que posee un dominio (límite inferior y superior), nombre y flag para indicar el etiquetado. Tiene métodos para acotar los límites y etiquetarla, para que el resolutor pueda trabajar con ella.

- **CRestriccion**, representa una restricción que deben cumplir las soluciones del problema. Viene definida por una lista de sumandos (por ejemplo 2a+3b-4c), el operador (Igual, Menor, MenorIgual, Mayor, MayorIgual y Distinto), y el término independiente (por ejemplo -5). A partir de la restricción, se obtendrán todas las reglas que se deben aplicar a los dominios de las variables de sus sumandos. Cada regla, será un propagador que hay que aplicar al dominio de la variable, por ejemplo, para la restricción *a-b<0* las reglas obtenidas de la restricción son *a<b y b>a,* que recortan el límite superior de *a,*y el límite inferior de *b* respectivamente, en función de los valores mínimo y máximo de los dominios de *b y a* respectivamente.

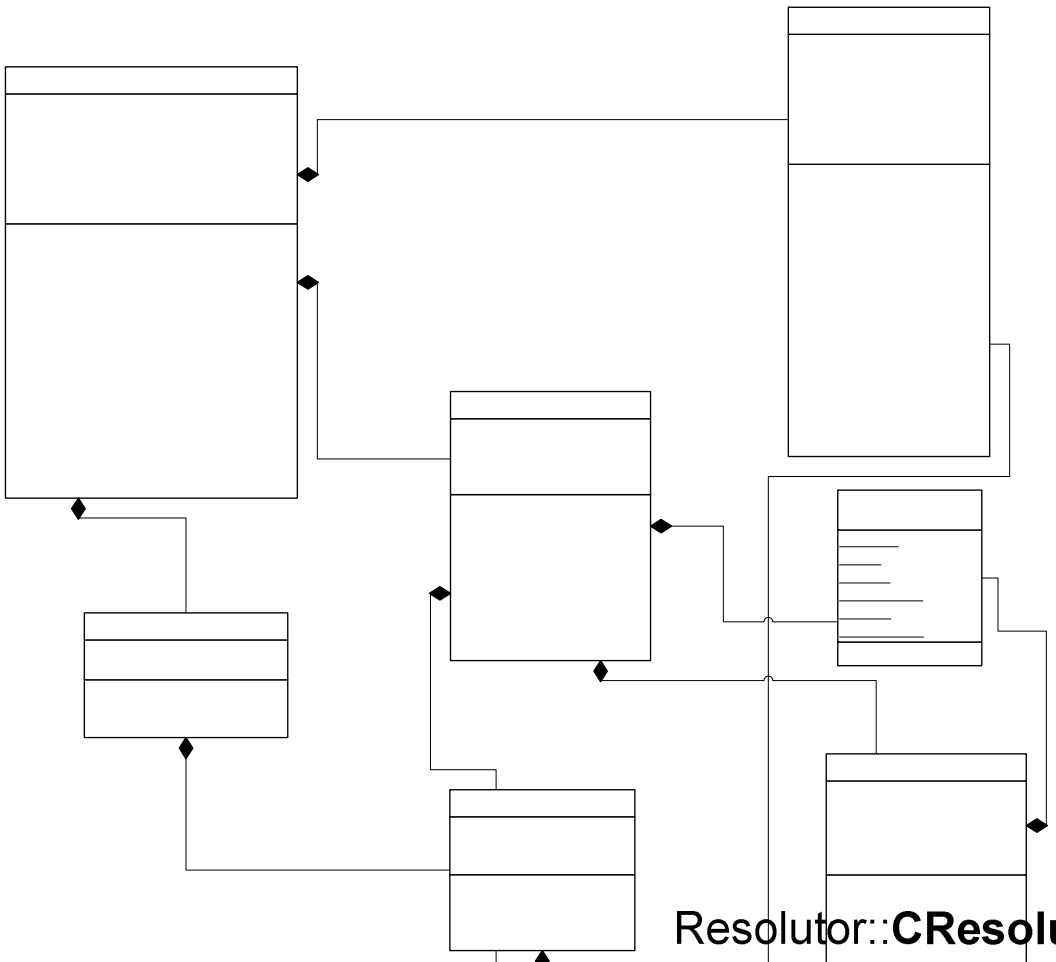
- **CRegla**, clase que representa una regla o propagador y que consta de la variable despejada, cuyo dominio será el que se acote, la lista de sumandos a la derecha del operador de la regla, el término independiente y el denominador, que será el coeficiente de la variable despejada.

- **CSumando**, es la clase que forma las restricciones y la función objetivo, por si sola no tiene sentido, porque son las componentes de las restricciones, por ejemplo en la restricción *-2a+3b-c<>0*, la lista de sumandos es *-2a,3b y –c*. Cada sumando viene definido por una variable y un coeficiente.

- **CObjetivo**, es la función que hay que calcular para cada solución que se obtiene, está representada en esta clase mediante la lista de sumandos que la componen y el término independiente, un ejemplo es *3a-4b+d+3.*

- **Coperador**, simplemente es un tipo enumerado que representa los operadores disponibles.

A continuación se muestra el diagrama de clases UML del componente resolutor:

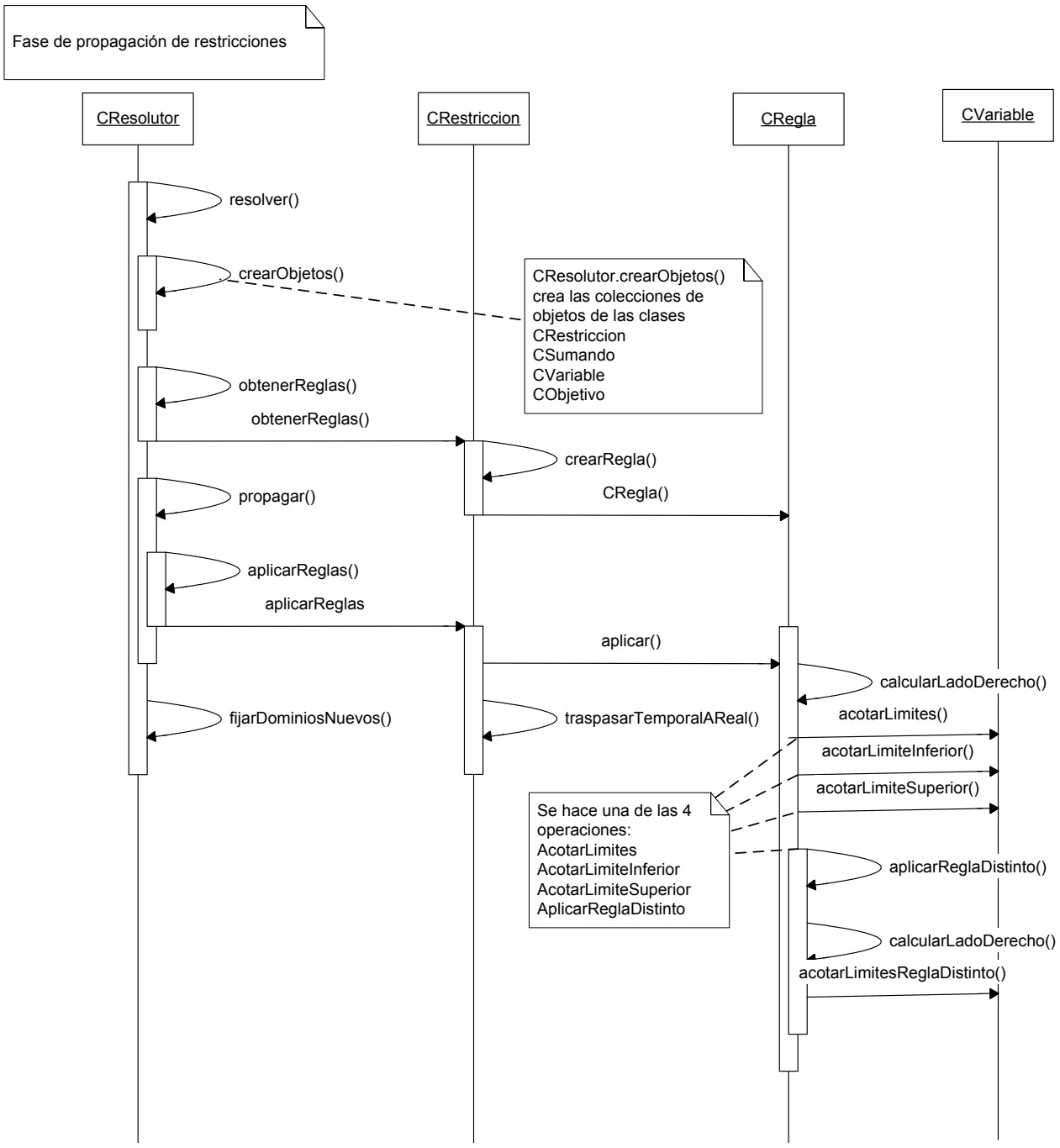


Resolutor::**CResolutor**

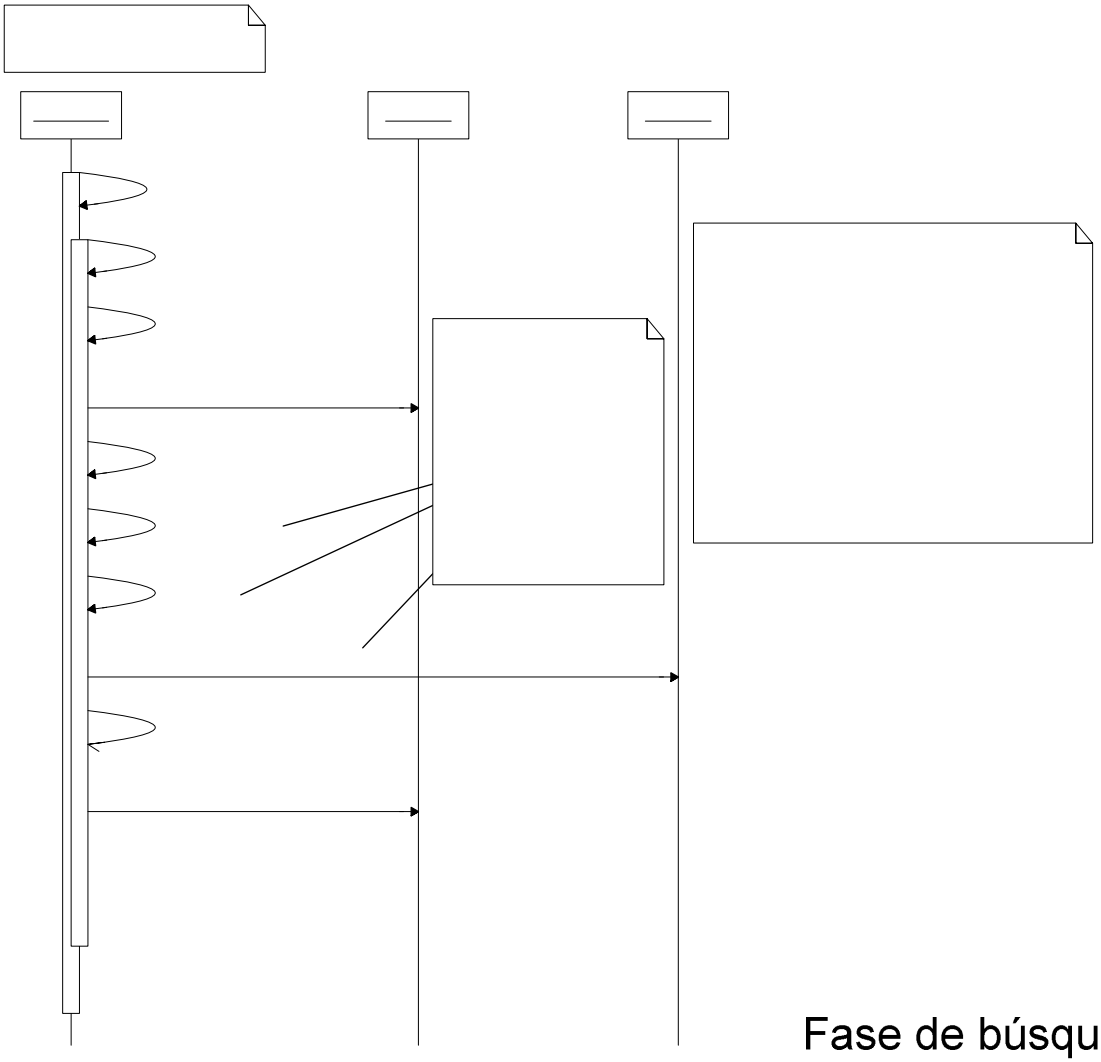
-m_variables : ArrayList
-m_restricciones : ArrayList
-m_enumeratorRestricciones : IEnumerator
-m_numeroSoluciones : int
-m_FuncionObjetivo : CObjetivo
-m_Maximizacion : bool
-m_ValorActualOptimizacion : int
+numeroSoluciones()
+numeroRestricciones()
+resolver()
-crearObjetos()
-propagar()
+busqueda()
-obtenerReglas()
-aplicarReglas()
-variablesVacias()
+unitarias()

8.1.1.2.- Diagramas de secuencia

El siguiente diagrama de secuencia UML muestra la fase de propagación de restricciones que se realiza tanto inicialmente, como en cada búsqueda de soluciones.



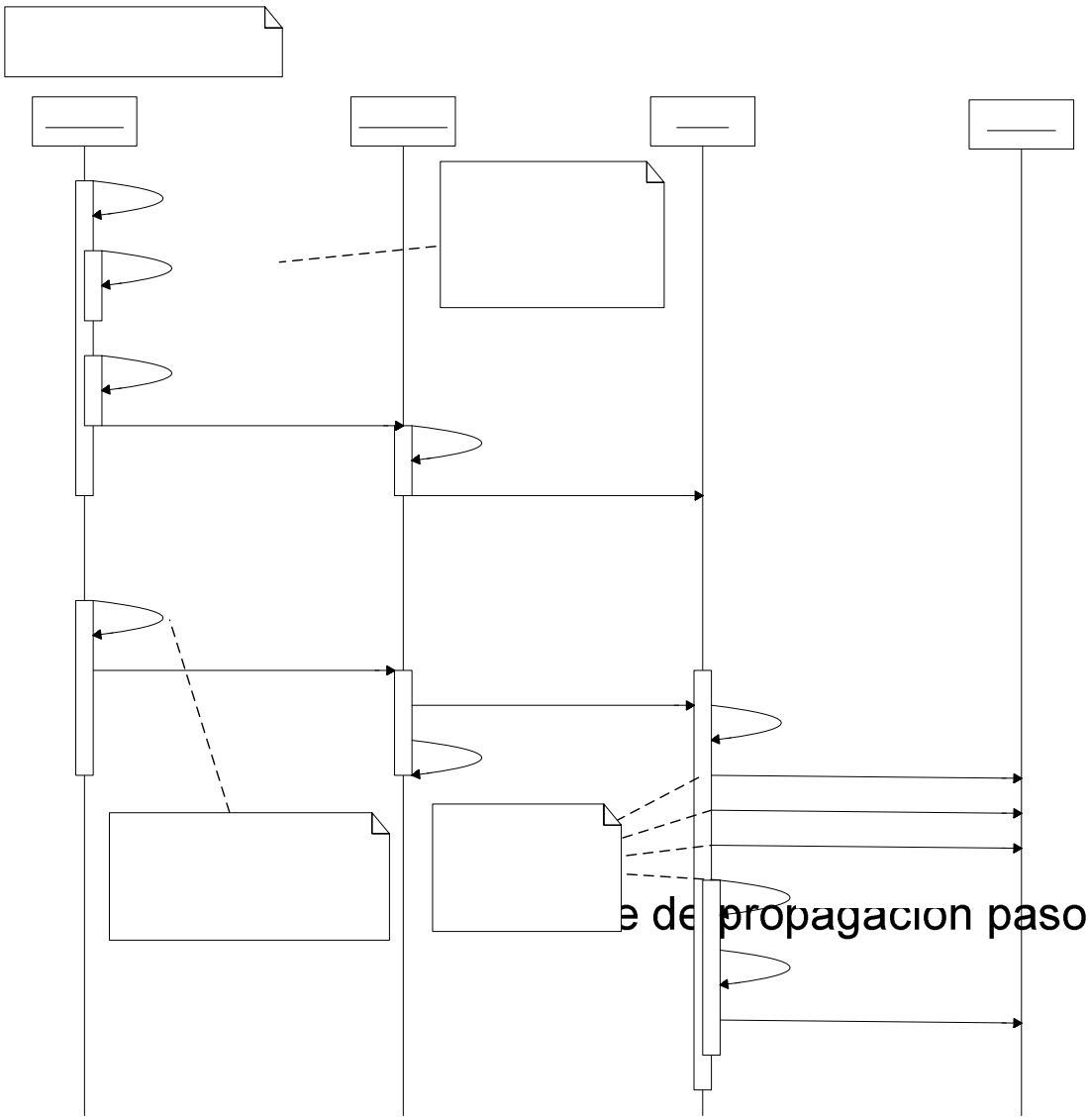
El siguiente diagrama de secuencia UML muestra la fase de búsqueda de soluciones del resolutor, que se realiza después de la propagación inicial.



Fase de búsqueda d

CResolutor

Este diagrama muestra la propagación inicial paso a paso.



8.1.2.- WSResolutor

Este componente es un servicio Web, que estará instalado en Internet Information Server, para que el cliente pueda invocar los métodos Web, deberá agregar una referencia Web a **http://servidor/WSResolutor.asmx?WSDL**, con lo que recibirá el fichero XML de descripción de servicios Web, que Visual Studio utiliza para poder extraer la información de los métodos y parámetros de éstos.

El cliente entonces podrá invocar al único método Web disponible en el servicio (*resolver*), que simplemente invoca al método *resolver* del componente resolutor que estará instalado en el mismo servidor Internet Information Server para que pueda estar accesible. Con esto conseguimos que el cliente no tenga que tener instalado el resolutor en su máquina local, con las ventajas que esto supone para mantener siempre la última versión sin hacer nada, simplemente ejecutarlo.

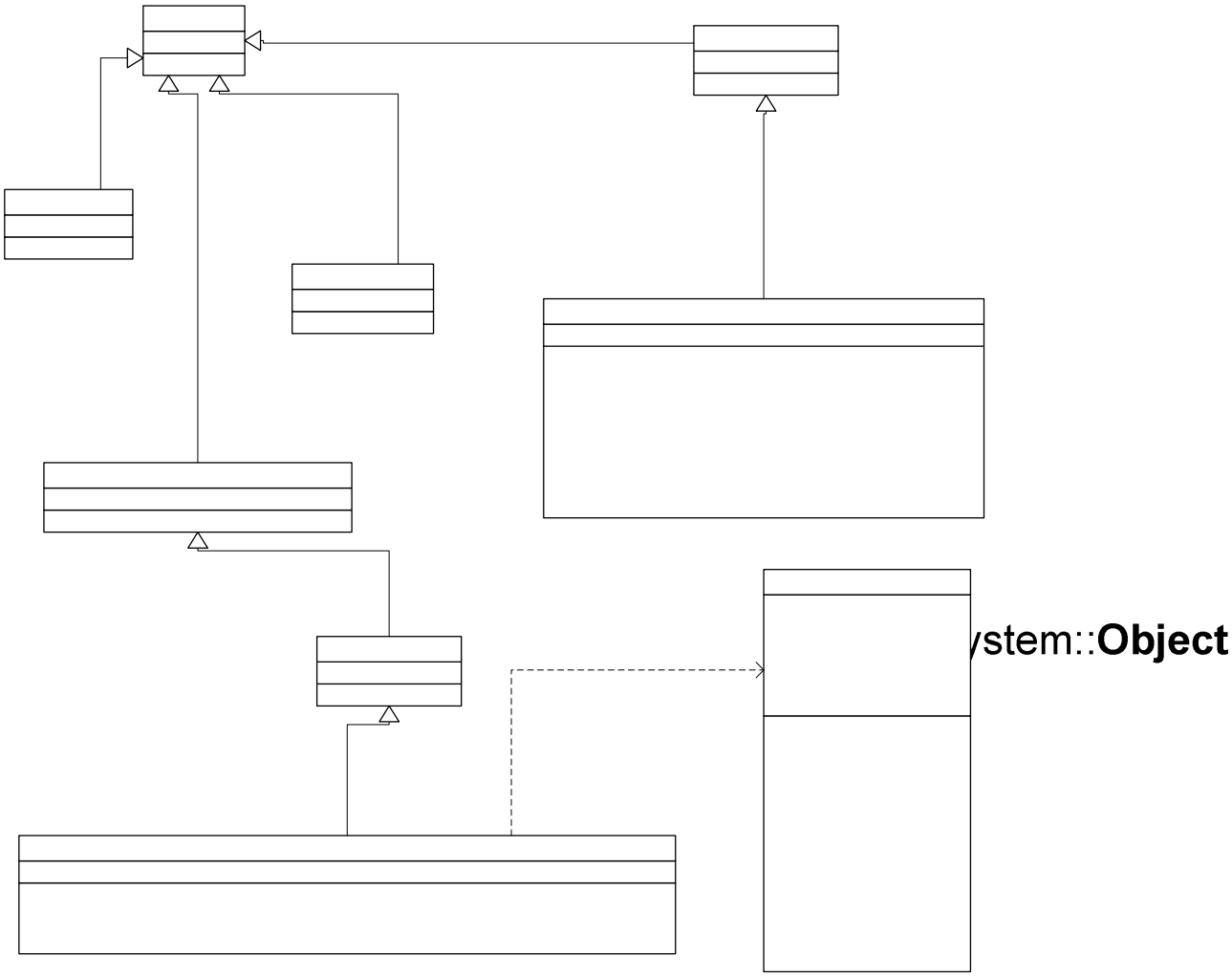
El método (resolver), recibe los datos del cliente, en forma de arrays, que son las variables (string[]), los límites inferiores (int[]), los límites superiores (int[]), las restricciones (int[][]), y los operadores (string[]), y realiza tanto la propagación inicial como el etiquetado, es decir, todo el algoritmo, y devuelve el resultado en un objeto ArrayList (objeto del framework Microsoft .NET). Este método, es llamado desde la aplicación Windows (clienteWSWin32) y desde la aplicación Web (clienteWebResolutor).

8.1.2.1.-Diagrama de clases

Las clases de las que se compone el servicio Web son las siguientes:

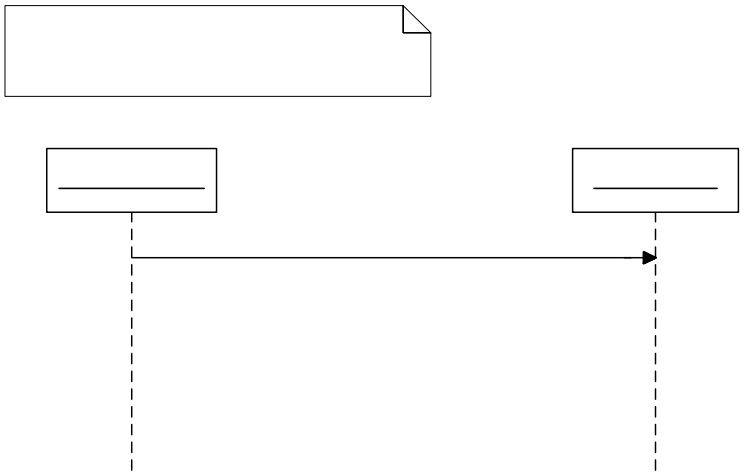
- **Global**, clase que representa la aplicación ASP.NET que alberga el servicio Web, y que dispone de los manejadores de eventos para el inicio de aplicación inicio de sesión etc…

- **WSResolutor**, es el servicio propiamente dicho, que implementa el método resolver, que recoge los parámetros enviados por el cliente, y hace una llamada al componente resolutor con esos mismos parámetros. La salida del componente resolutor es recogida por el servicio y devuelta al cliente.


ystem::Object
System::EventArgs

8.1.2.2.-Diagramas de secuencia

El siguiente diagrama de secuencia UML muestra el comportamiento del servicio.



Funcionamiento del se

8.1.3.- ClienteWin32

Aplicación realizada con Windows Forms de .NET, compuesta de un interfaz gráfico, que permite describir el problema, mediante la declaración de variables y sus restricciones (lenguaje *LRPR*), en una ventana, y que permite la resolución completa del problema, o bien verificar paso a paso la propagación inicial del las restricciones sobre los dominios de las variables.

8.1.3.1.- Diagrama de clases

Las clases de las que se compone este componente son las siguientes:

- **CodeEditor**, formulario donde se escribe el código en lenguaje *LRPR* para describir el problema, y que dispone de un menú para interactuar con ficheros (*.rpr*) y los botones para verificar el código fuente, para resolver el problema y para resolverlo paso a paso.

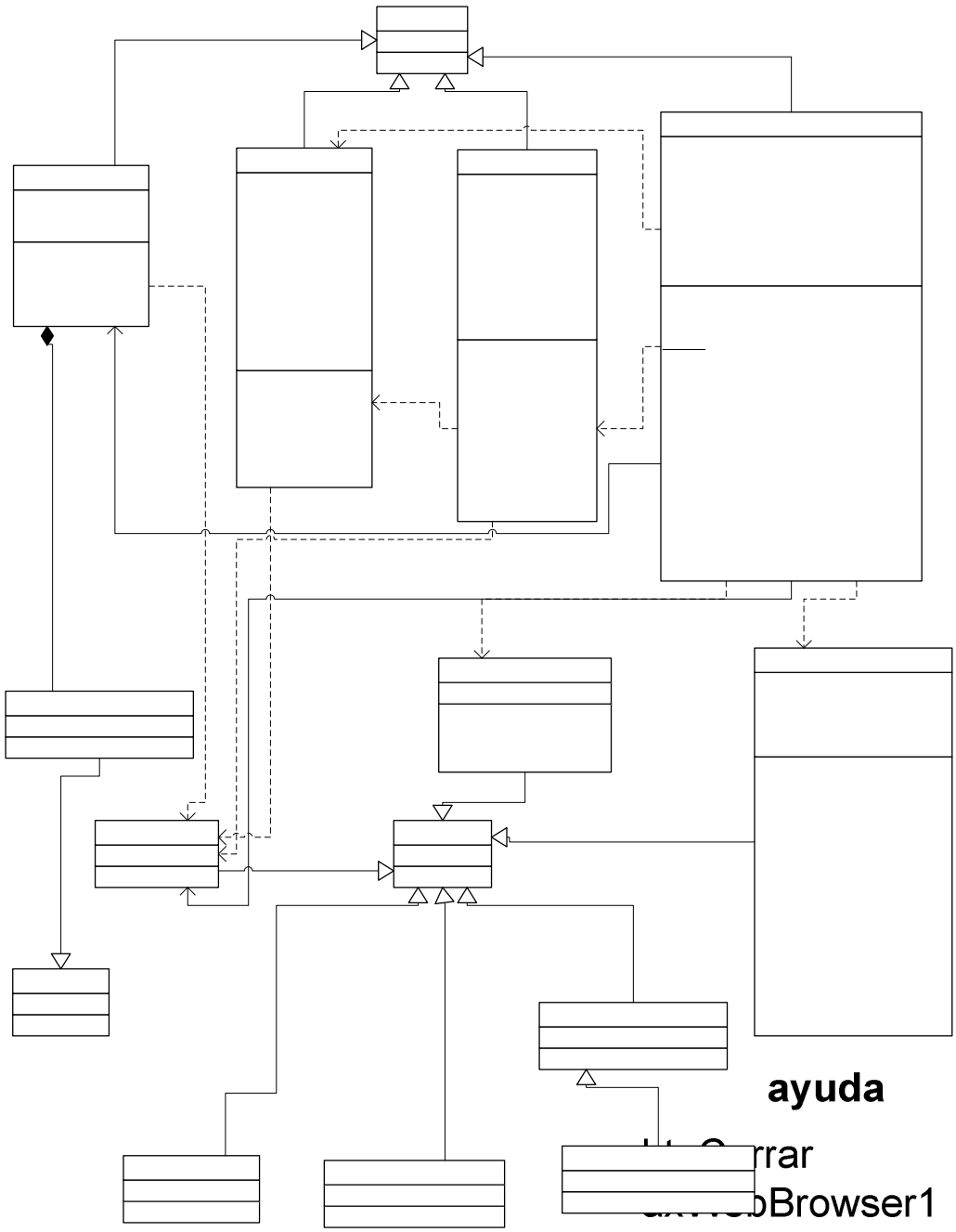
- **propagacion**, formulario al que se pasa desde el *CodeEditor* cuando se quiere resolver el problema paso a paso. Dispone de un *DataGrid* (control de Microsoft .NET para mostrar datos tabulados), donde se muestran los dominios de las variables, un botón para resolver el problema, y un botón para regresar al *CodeEditor.*

- **resultado**, formulario al que se accede desde el *CodeEditor* o bien desde *propagación,* cuando se opta por resolver el problema. Muestra las soluciones así como el tiempo tardado en resolverlo.

- **ayuda**, formulario que muestra una página de ayuda de sintaxis del lenguaje *LRPR*.

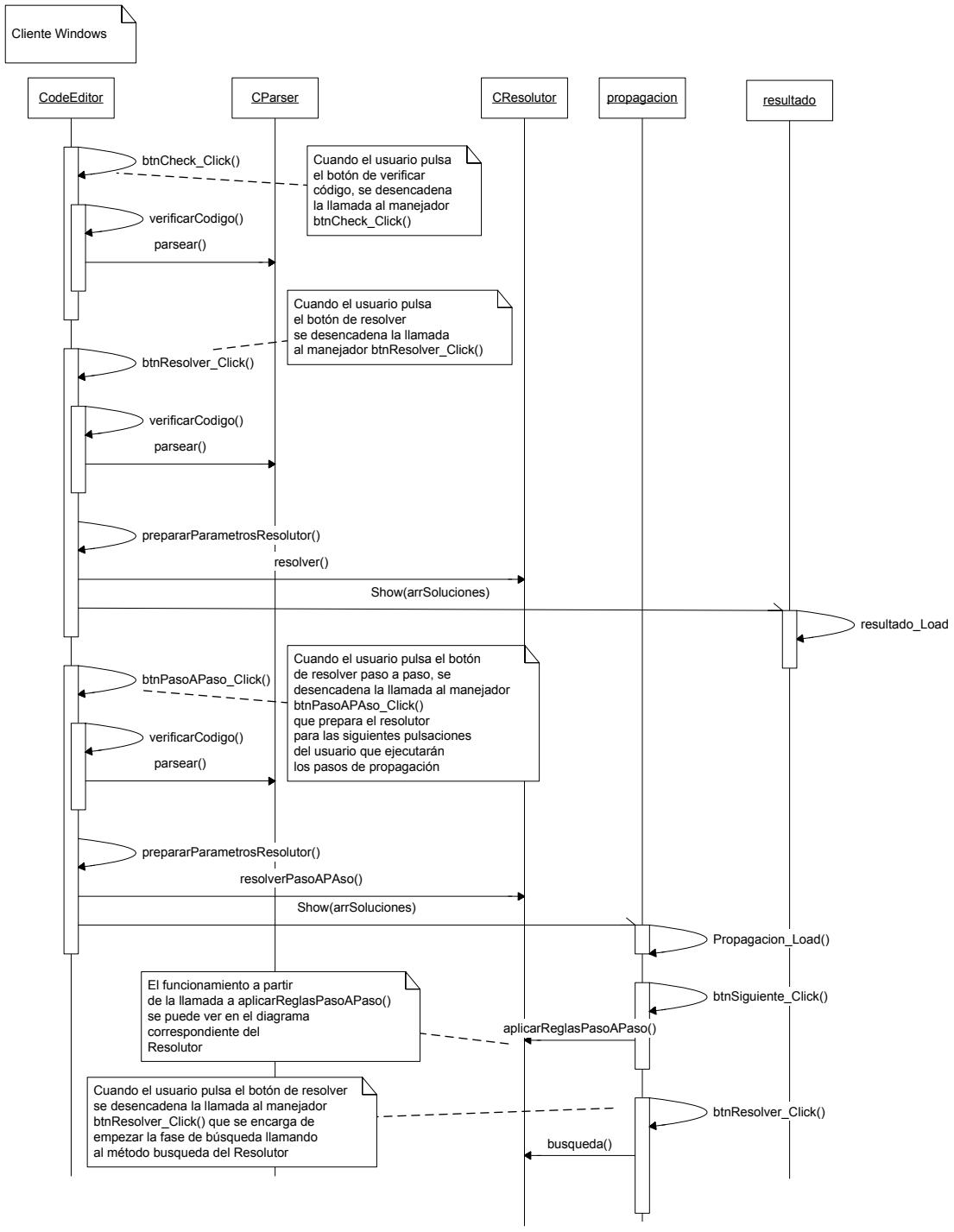
- **CParser**, clase que realiza el parseo del código fuente *LRPR* y que tiene como método principal *resolver* al que se le pasa el código a parsear y genera las colecciones de objetos necesarios para el problema.

- **Global**, clase que representa la aplicación ASP.NET, y que dispone de los manejadores de eventos para el inicio de aplicación inicio de sesión etc.…


ayuda
rrar
bBrowser1
-components = null
+ayuda()
#Dispose()
-InitializeComponent()
-ayuda_Load()
-btnCerrar_Click()
1 -axWebBrowser1

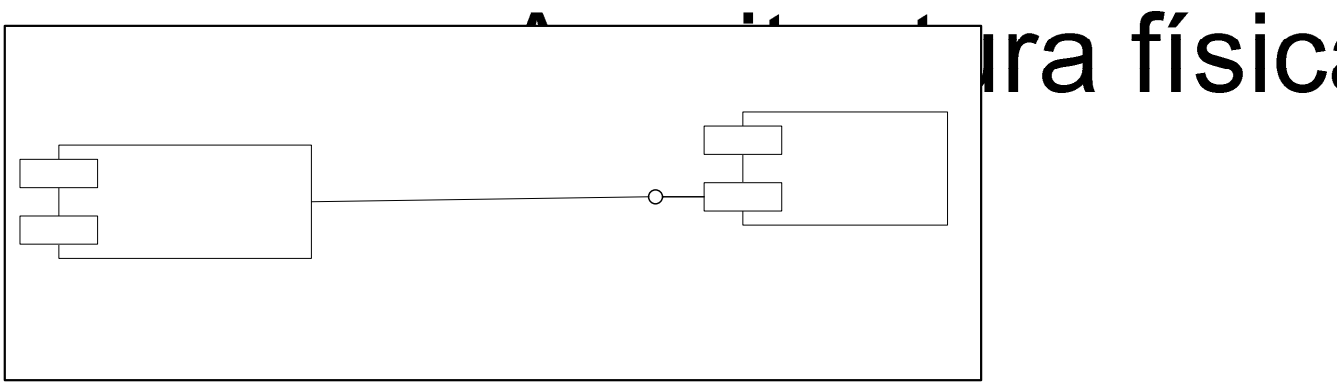
8.1.3.2.- Diagramas de secuencia

El siguiente diagrama de secuencia UML muestra el comportamiento de clienteWin32.



8.1.3.3.- Arquitectura

Toda la lógica reside en una sola máquina, que tiene instalado el Framework de .NET, necesario para ejecutar la aplicación.



Los componentes lógicos de la aplicación, son la propia aplicación Win32, que establece el interfaz de usuario, y el componente (dll), que realiza la resolución del problema mediante la implementación del algoritmo de propagación de restricciones.

8.1.4.- ClienteWebResolutor

Aplicación realizada con Web Forms de .NET (ASP .NET), que proporciona un interfaz de usuario para describir el problema a resolver (lenguaje LRPR), en una ventana, mediante la declaración de variables y restricciones, y que permite la resolución completa, o bien, la verificación paso a paso de la propagación inicial de restricciones, para la resolución posterior del problema.

8.1.4.1.- Diagrama de clases

Las clases del clienteWebResolutor de describen a continuación:

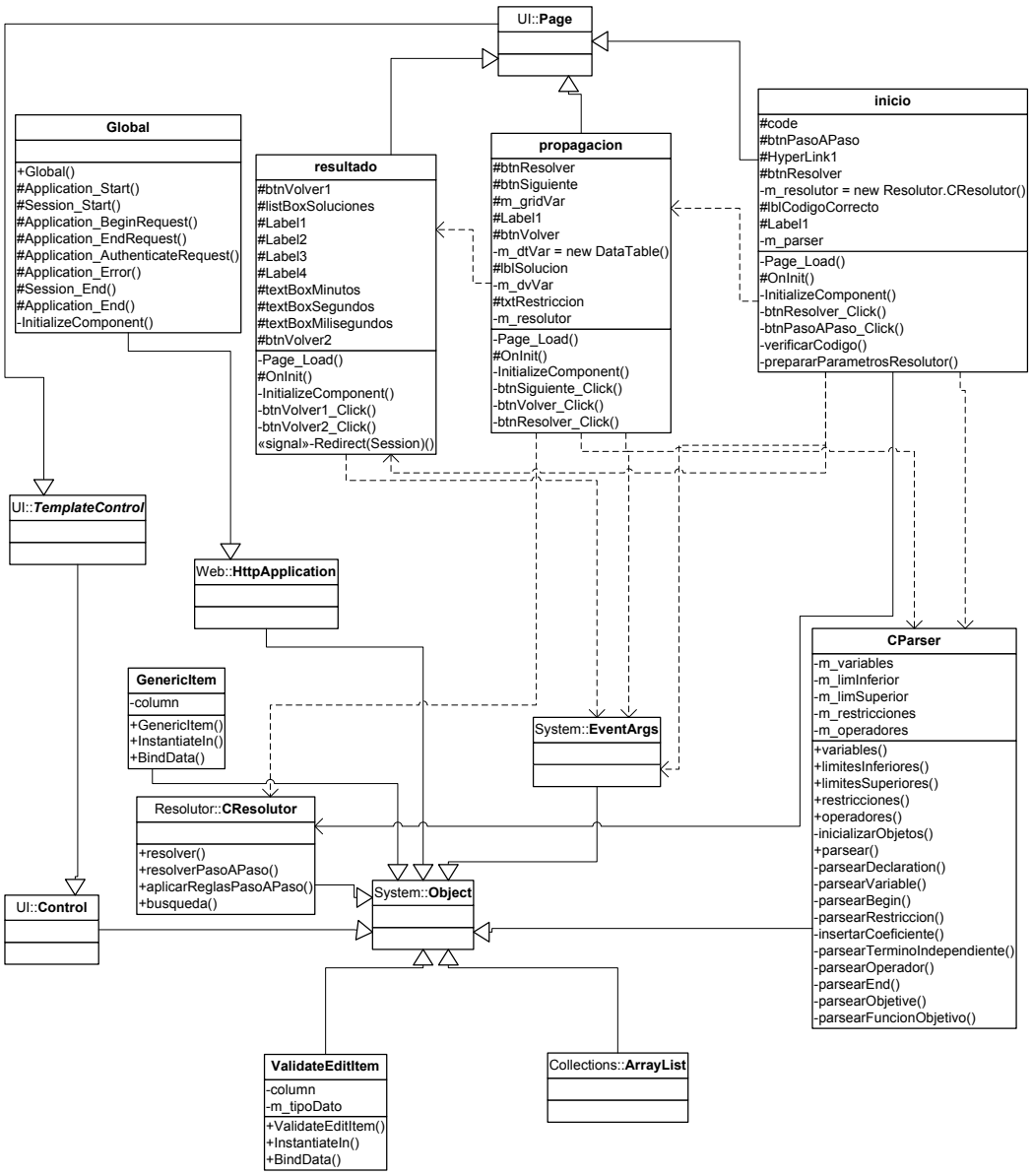
- **inicio**, página donde se escribe el código en lenguaje *LRPR* para describir el problema, y que dispone de los botones para resolver el problema y para resolverlo paso a paso.

- **propagacion**, página a la que se pasa desde *inicio* cuando se quiere resolver el problema paso a paso. Dispone de un *DataGrid* (control de Microsoft .NET para mostrar datos tabulados), donde se muestran los dominios de las variables, un botón para resolver el problema, y un botón para regresar a *inicio.*

- **resultado**, página a la que se accede desde *inicio* o bien desde *propagación,* cuando se opta por resolver el problema. Muestra las soluciones así como el tiempo tardado en resolverlo.

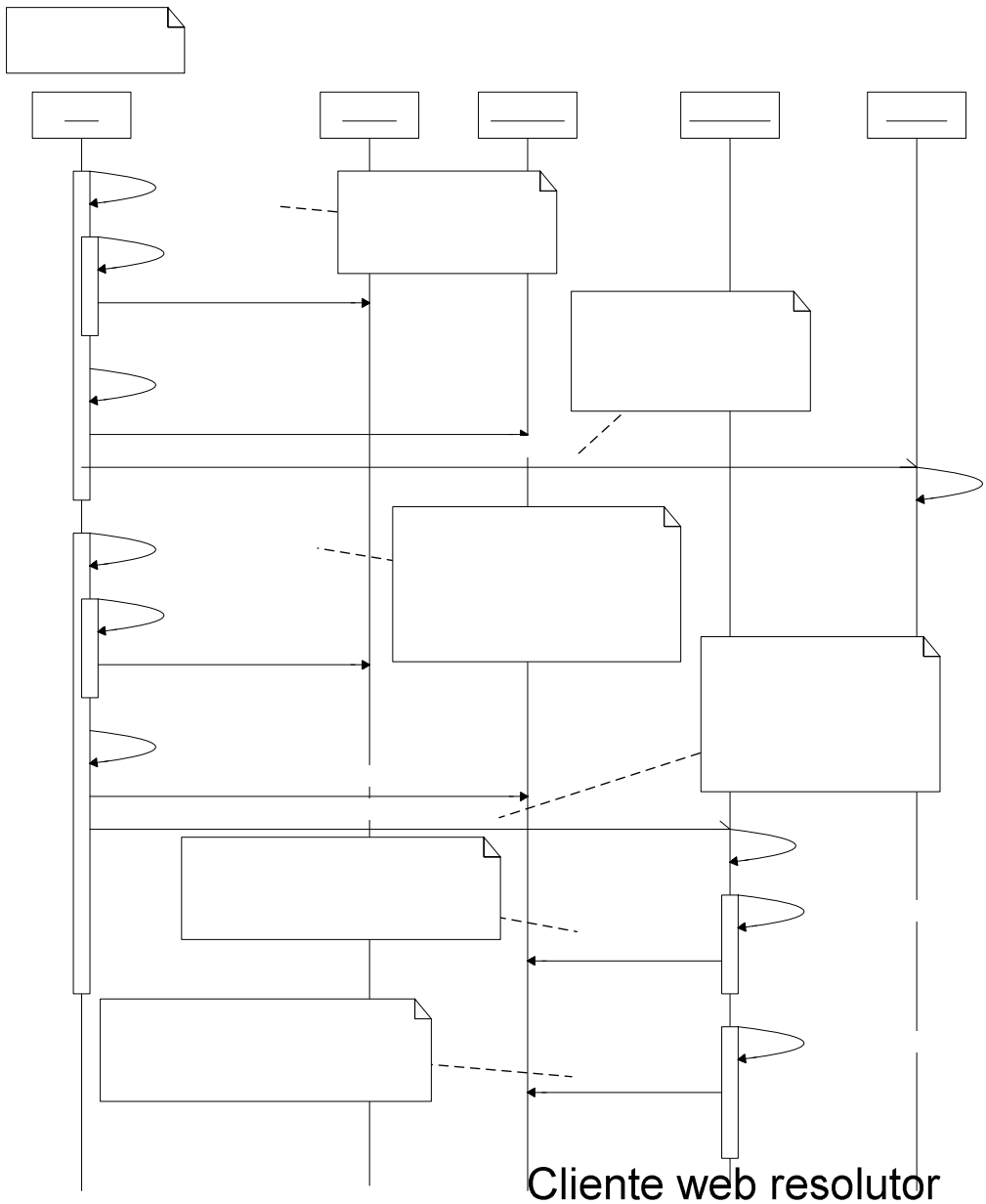
- **CParser**, clase que realiza el parseo del código fuente *LRPR* y que tiene como método principal *resolver* al que se le pasa el código a parsear y genera las colecciones de objetos necesarios para el problema.

- **Global**, clase que representa la aplicación ASP.NET, y que dispone de los manejadores de eventos para el inicio de aplicación inicio de sesión etc.…


UI::Page
Global
+Global()
#Application_Start()
#Session_Start()
#Application_BeginRequest()
#Application_EndRequest()
#Application_AuthenticateRequest()
#Application_Error()
#Session_End()
#Application_End()
-InitializeComponent()
resultado
#btnVolver1
#listBoxSoluciones
#Label1
#Label2
#Label3
#Label4
#textBoxMinutos
#textBoxSegundos
#textBoxMilisegundos
#btnVolver2
-Page_Load()
#OnInit()
-InitializeComponent()
-btnVolver1_Click()
-btnVolver2_Click()
«signal»-Redirect(Session)()
propagacion
#btnResolver
#btnSiguiente
#m_gridVar
#Label1
#btnVolver
-m_dtVar = new DataTable()
#lblSolucion
-m_dvVar
#txtRestriccion
-m_resolutor
-Page_Load()
#OnInit()
-InitializeComponent()
-btnSiguiente_Click()
-btnVolver_Click()
-btnResolver_Click()
inicio
#code
#btnPasoAPaso
#HyperLink1
#btnResolver
-m_resolutor = new Resolutor.CResolutor()
#lblCodigoCorrecto
#Label1
-m_parser
-Page_Load()
#OnInit()
-InitializeComponent()
-btnResolver_Click()
-btnPasoAPaso_Click()
-verificarCodigo()
-prepararParametrosResolutor()
UI::TemplateControl
Web::HttpApplication
GenericItem
-column
+GenericItem()
+InstantiateIn()
+BindData()
System::EventArgs
CParser
-m_variables
-m_limInferior
-m_limSuperior
-m_restricciones
-m_operadores
+variables()
+limitesInferiores()
+limitesSuperiores()
+restricciones()
+operadores()
-inicializarObjetos()
+parsear()
-parsearDeclaration()
-parsearVariable()
-parsearBegin()
-parsearRestriccion()
-insertarCoeficiente()
-parsearTerminoIndependiente()
-parsearOperador()
-parsearEnd()
-parsearObjetive()
-parsearFuncionObjetivo()
Resolutor::CResolutor
+resolver()
+resolverPasoAPaso()
+aplicarReglasPasoAPaso()
+busqueda()
UI::Control
System::Object
ValidateEditItem
-column
-m_tipoDato
+ValidateEditItem()
+InstantiateIn()
+BindData()
Collections::ArrayList

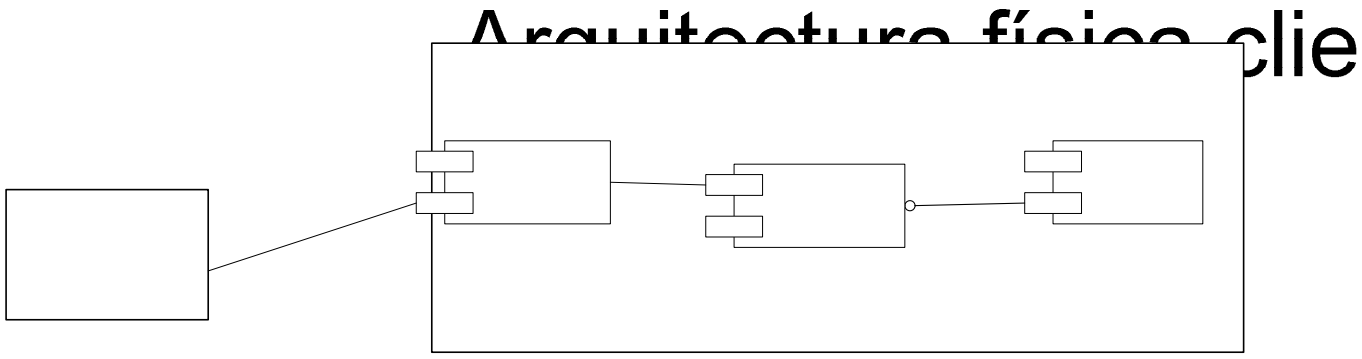
8.1.4.2.- Diagramas de secuencia

El siguiente diagrama de secuencia UML muestra el comportamiento del clienteWebResolutor.



8.1.4.3.- Arquitectura

El cliente es un navegador de internet, que se conecta a una URL, el servidor descarga sobre el navegador la página inicial, en la cual, se introduce el problema a resolver. Cuando se opta por resolver, el navegador envía la información al servidor http, que resuelve el problema, y devuelve la página de resultados al cliente.



Los componentes lógicos son, el navegador, que se conecta al servidor, donde reside el servidor Web (http), que recoge la petición Web, y que la reenvía a la aplicación ASP .NET, que la procesará, ésta, después de preparar la información, hace una petición al componente Resolutor (dll), que resuelve el problema, y devuelve los resultados a la aplicación Web, que a su vez los devuelve al servidor http, que forma la página de resultados que se envía al cliente.

Cliente
Browser

HTTP

8.1.5.- ClienteWSWin32

Aplicación realizada con Windows Forms de .NET, compuesta de un interfaz gráfico, que permite describir el problema, mediante la declaración de variables y sus restricciones (lenguaje LRPR), en una ventana, y que permite la resolución completa del problema.

8.1.5.1.- Diagrama de clases

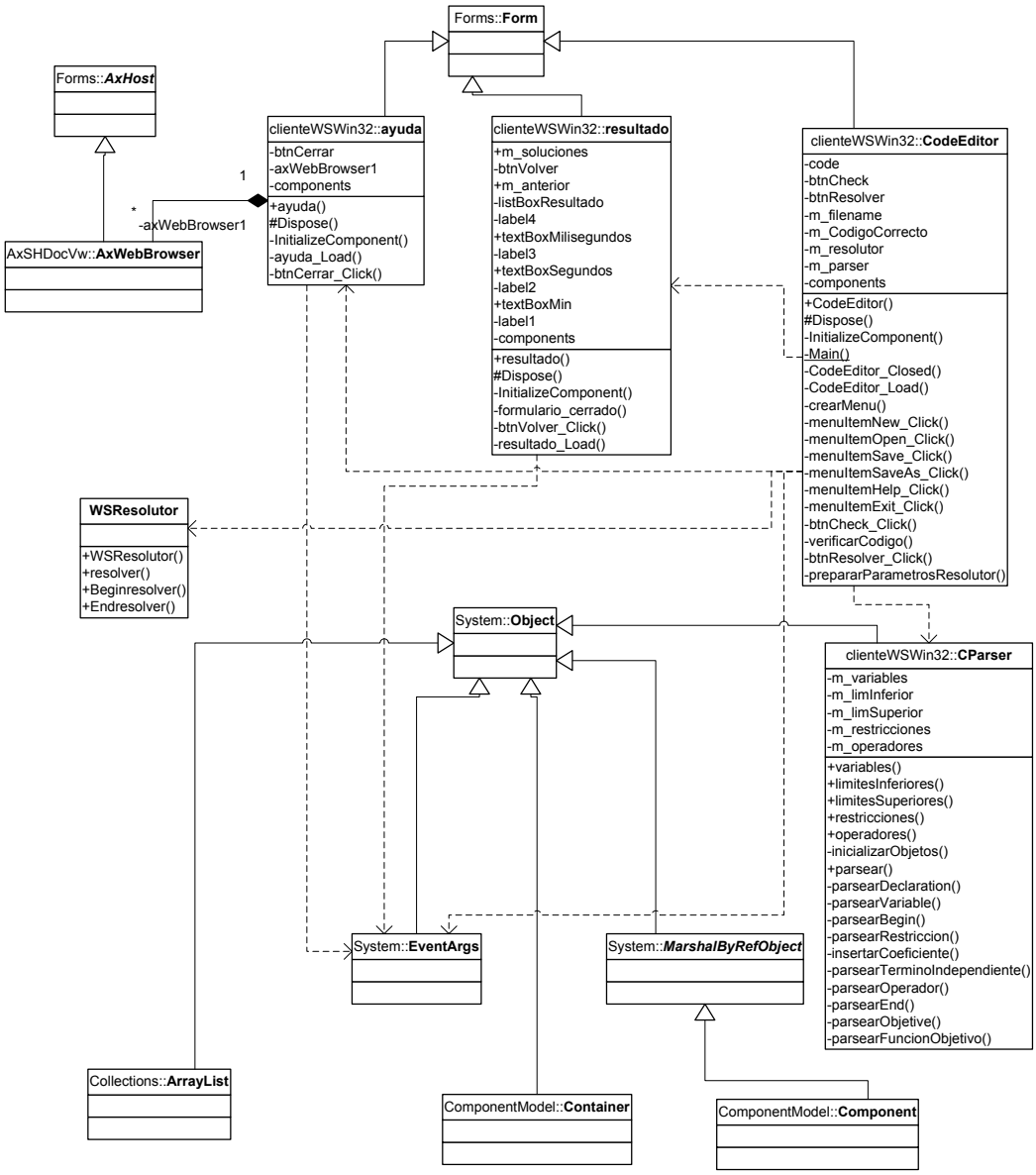
Estas son las clases del clienteWSWin32:

- **CodeEditor**, formulario donde se escribe el código en lenguaje *LRPR* para describir el problema, y que dispone de un menú para interactuar con ficheros (*.rpr*) y los botones para verificar el código fuente, para resolver el problema y para resolverlo paso a paso.

- **resultado**, formulario al que se accede desde el *CodeEditor* o bien desde *propagación,* cuando se opta por resolver el problema. Muestra las soluciones así como el tiempo tardado en resolverlo.

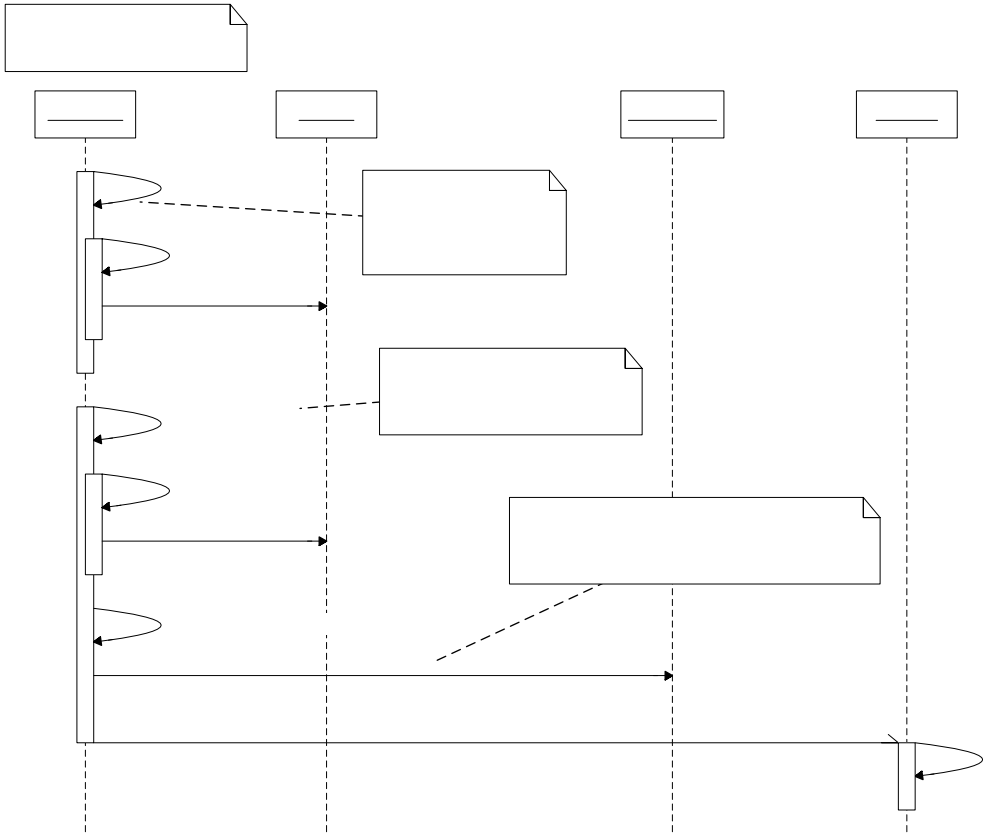
- **ayuda**, formulario que muestra una página de ayuda de sintaxis del lenguaje *LRPR*.

- **CParser**, clase que realiza el parseo del código fuente *LRPR* y que tiene como método principal *resolver* al que se le pasa el código a parsear y genera las colecciones de objetos necesarios para el problema.


Forms::Form
Forms::AxHost
clienteWSWin32::ayuda
-btnCerrar
-axWebBrowser1
-components
+ayuda()
#Dispose()
-InitializeComponent()
-ayuda_Load()
-btnCerrar_Click()
clienteWSWin32::resultado
+m_soluciones
-btnVolver
+m_anterior
-listBoxResultado
-label4
+textBoxMilisegundos
-label3
+textBoxSegundos
-label2
+textBoxMin
-label1
-components
+resultado()
#Dispose()
-InitializeComponent()
-formulario_cerrado()
-btnVolver_Click()
-resultado_Load()
clienteWSWin32::CodeEditor
-code
-btnCheck
-btnResolver
-m_filename
-m_CodigoCorrecto
-m_resolutor
-m_parser
-components
+CodeEditor()
#Dispose()
-InitializeComponent()
-Main()
-CodeEditor_Closed()
-CodeEditor_Load()
-crearMenu()
-menuItemNew_Click()
-menuItemOpen_Click()
-menuItemSave_Click()
-menuItemSaveAs_Click()
-menuItemHelp_Click()
-menuItemExit_Click()
-btnCheck_Click()
-verificarCodigo()
-btnResolver_Click()
-prepararParametrosResolutor()
1
*
-axWebBrowser1
AxSHDocVw::AxWebBrowser
WSResolutor
+WSResolutor()
+resolver()
+Beginresolver()
+Endresolver()
System::Object
clienteWSWin32::CParser
-m_variables
-m_limInferior
-m_limSuperior
-m_restricciones
-m_operadores
+variables()
+limitesInferiores()
+limitesSuperiores()
+restricciones()
+operadores()
-inicializarObjetos()
+parsear()
-parsearDeclaration()
-parsearVariable()
-parsearBegin()
-parsearRestriccion()
-insertarCoeficiente()
-parsearTerminoIndependiente()
-parsearOperador()
-parsearEnd()
-parsearObjetive()
-parsearFuncionObjetivo()
System::EventArgs
System::MarshalByRefObject
Collections::ArrayList
ComponentModel::Container
ComponentModel::Component

8.1.5.2.- Diagramas de secuencia

El siguiente diagrama de secuencia UML muestra el comportamiento del clienteWSWin32.



Cliente Windows Web Service

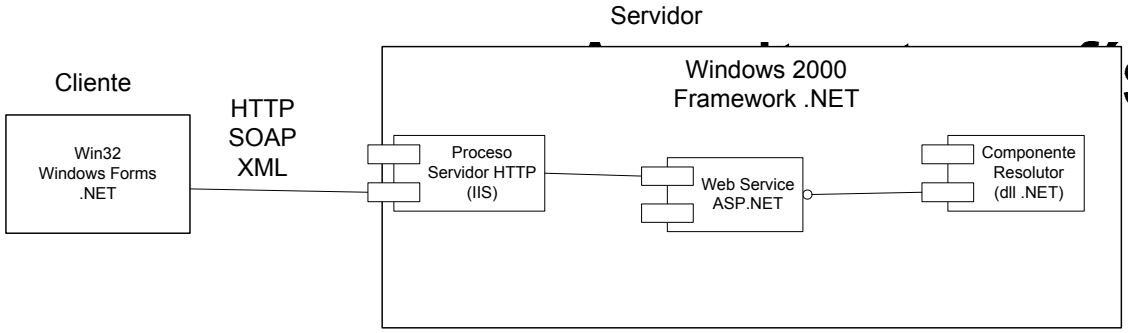
CodeEditor

btnCheck_Click()

8.1.5.3.- Arquitectura

El cliente es una aplicación Windows Forms de .NET, que establece el interfaz de usuario, para la descripción del problema, y que cuando se decide resolver, se envía la petición a un servicio Web, a través de internet (http,xml, soap), a un servidor http, que una vez que lo resuelve, devuelve la solución mediante xml al cliente.

Arquitectura lógica clienteWSWin32



Los componentes lógicos son, la aplicación Windows Forms, en la parte cliente, que se conecta vía internet, por protocolo http, al servidor Web, éste recoge la petición, y la reenvía al servicio Web, que a su vez, invoca al componente Resolutor, que se encuentra en el servidor, y que resuelve el problema, una vez resuelto, devuelve el resultado al servicio Web, que se lo devuelve al servidor http, que lo envía mediante XML al cliente.

8.2.- Ramificación y poda con Backtracking

Está compuesta de cinco componentes, que son:

1.- *ResolutorRPR.dll*: el componente resolutor que realiza la operativa del algoritmo y que está implementado como *Component Library* de .NET(.dll)

2.- C*liente.exe:* un cliente Windows que invoca al componente *ResolutorRPR.dll*.

3.- *clienteWebRPR.exe:* una aplicación Web realizada con la tecnología ASP.NET que se encarga de invocar al componente *ResolutorRPR.dll* para la resolución del problema.

4.- *WSResolutorRPR*: un servicio web que se encarga de invocar al componente resolutor y que está implementado con la tecnología de Servicios Web de Microsoft .NET.

5.- *ClienteWS.exe:* un cliente Windows que invoca al servicio web *WSResolutorRPR*.

A continuación se explica cada uno de los componentes de los que consta la implementación de la resolución mediante el empleo del algoritmo de ramificación y poda con backtracking.

8.2.1.- ResolutorRPR

El componente ResolutorRPR, es un *Component Library(.dll)* de Microsoft .NET, que implementa el algoritmo de resolución de problemas de variable de dominio finito mediante el método de ramificación y poda con backtracking.

El método principal de este componente es el método *resolverRPR*, que se encarga de la resolución del problema a partir de los parámetros que recibe como entrada. Estos datos que recibe el método del cliente en forma de arrays son los siguientes:

- Nombre de las variables del problema (string[])
- Límites inferiores (int[])
- Límites superiores (int[])
- Restricciones a aplicar (int[][])
- Operadores a aplicar para cada restricción
- Función objetivo
- Objetivo de optimización (maximizar o minimizar la función objetivo)

El método resolverRPR devuelve como resultado un objeto ArrayList (objeto del framework Microsoft .NET) con los pares *variable = valor* resultado de la resolución o bien, en caso de no encontrar solución, la cadena *NO HAY SOLUCIÓN*.

Este método será llamado tanto desde la aplicación Windows (Cliente) como desde el servicio Web (WSResolutorRPR).

8.2.1.1.- Diagrama de clases

Las clases desarrolladas para la implementación de este componente ha sido:

- **CResolutorRPR**, que realiza la lógica del algoritmo y que trata con los arrays de variables, límites inferiores y superiores, las restricciones, la función objetivo y el objetivo de la optimización. Esta es la clase que hay que instanciar desde los clientes para poder hacer la llamada al método encargado de la resolución de cada problema.

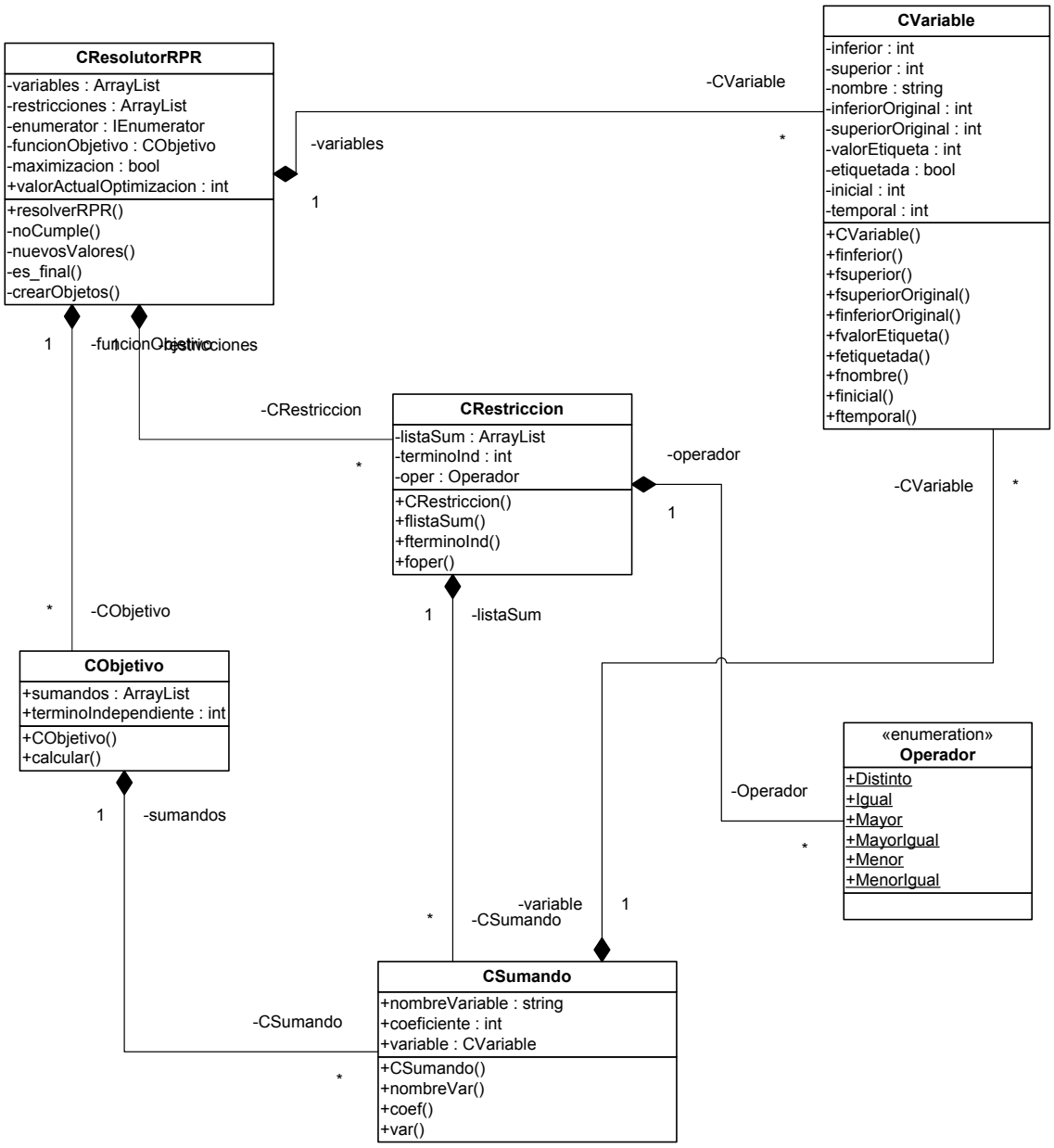
- **CRestriccion,** cada instancia de esta clase representará una restricción que debe cumplir la solución al problema. Cada restricción vendrá definida por la lista de los sumandos, el operador y el término independiente.

- **CSumando,** es la clase con la que se formarán los sumandos de cada restricción y de la función objetivo. Cada sumando viene definido por una variable y un coeficiente.

- **CObjetivo,** es la función que hay que calcular para la solución que se obtenga. Está representada mediante la lista de sumandos que la componen y el término independiente.

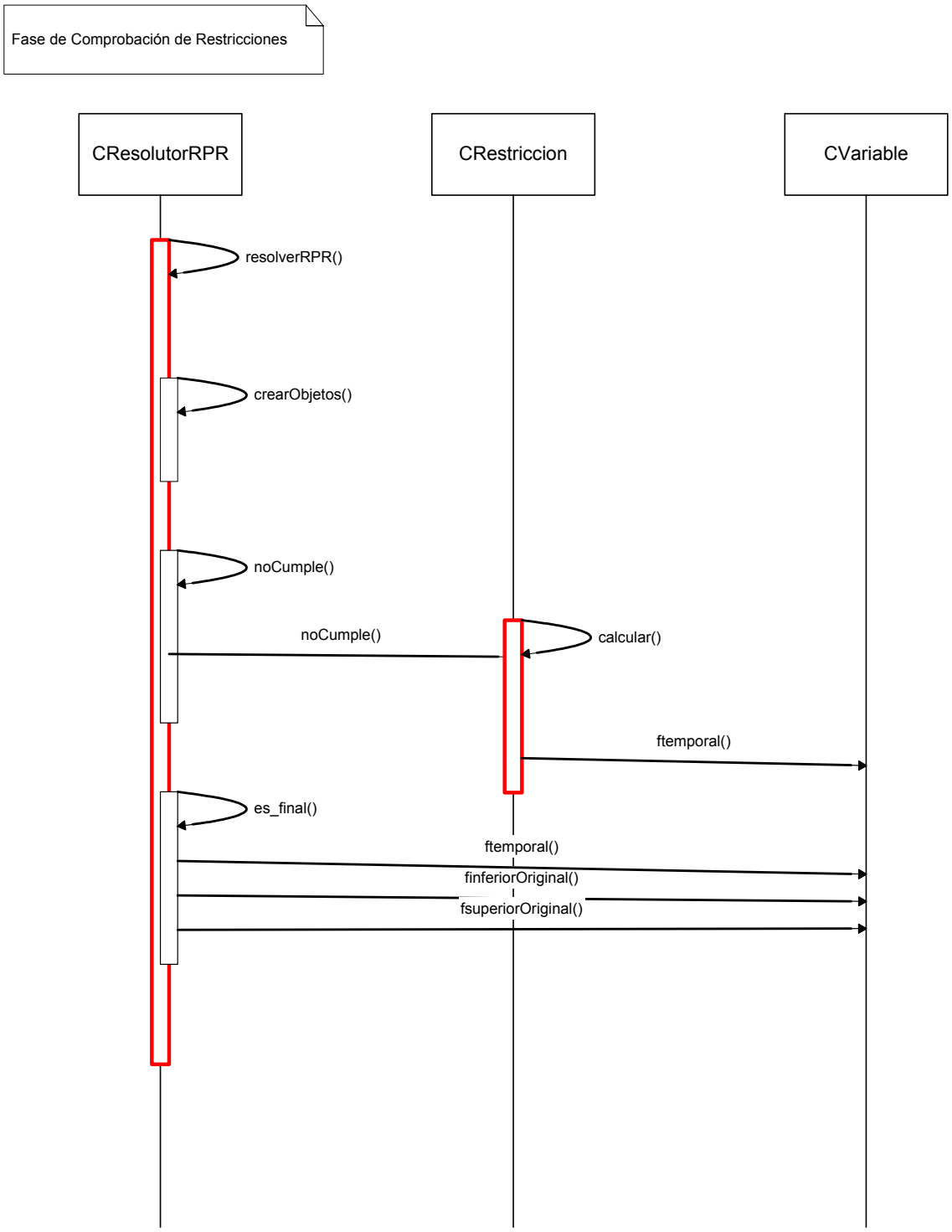
- **Operador,** es un tipo enumerado que representa los operadores disponibles, que serán: Igual, Menor, MenorIgual, Mayor, MayorIgual o Distinto.

A continuación se muestra el diagrama de clases UML del componente ResolutorRPR:

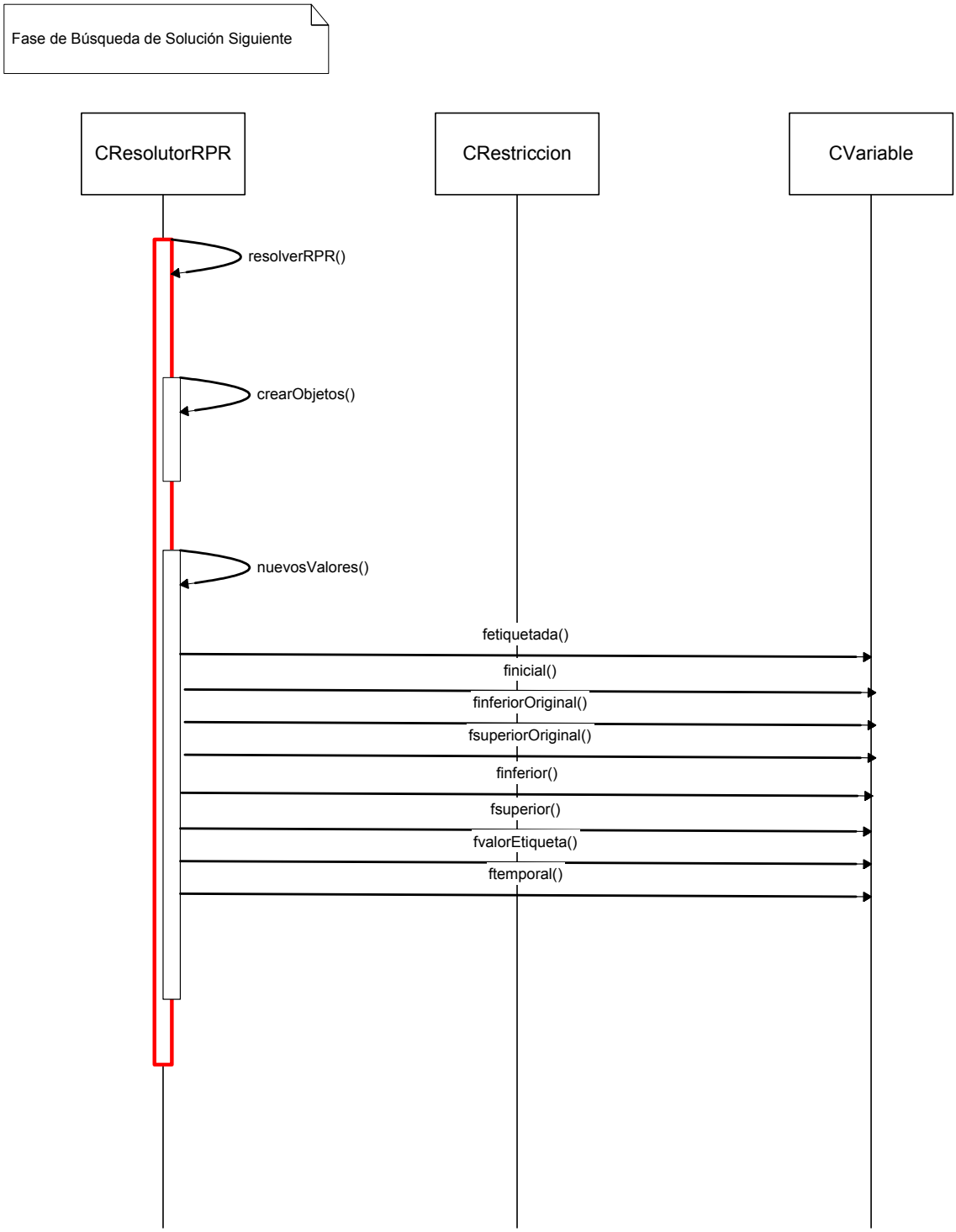

CResolutorRPR
-variables : ArrayList
-restricciones : ArrayList
-enumerator : IEnumerator
-funcionObjetivo : CObjetivo
-maximizacion : bool
+valorActualOptimizacion : int
+resolverRPR()
-noCumple()
-nuevosValores()
-es_final()
-crearObjetos()
CVariable
-inferior : int
-superior : int
-nombre : string
-inferiorOriginal : int
-superiorOriginal : int
-valorEtiqueta : int
-etiquetada : bool
-inicial : int
-temporal : int
+CVariable()
+finferior()
+fsuperior()
+fsuperiorOriginal()
+finferiorOriginal()
+fvalorEtiqueta()
+fetiquetada()
+fnombre()
+finicial()
+ftemporal()
-CVariable
-variables
1
*
1
1
-funcionObjetivo
-restricciones
-CRestriccion
*
CRestriccion
-listaSum : ArrayList
-terminoInd : int
-oper : Operador
+CRestriccion()
+flistaSum()
+fterminoInd()
+foper()
-operador
1
-CVariable
*
*
-CObjetivo
1
-listaSum
CObjetivo
+sumandos : ArrayList
+terminoIndependiente : int
+CObjetivo()
+calcular()
«enumeration»
Operador
+Distinto
+Igual
+Mayor
+MayorIgual
+Menor
+MenorIgual
-Operador
*
1
-sumandos
*
-variable
-CSumando
1
CSumando
+nombreVariable : string
+coeficiente : int
+variable : CVariable
+CSumando()
+nombreVar()
+coef()
+var()
-CSumando
*

8.2.1.2.- Diagramas de secuencia

En el siguiente diagrama de secuencia UML se muestra la fase de comprobación de restricciones de la resolución:



En el siguiente diagrama de secuencia UML se muestra la fase de búsqueda de la siguiente solución posible:



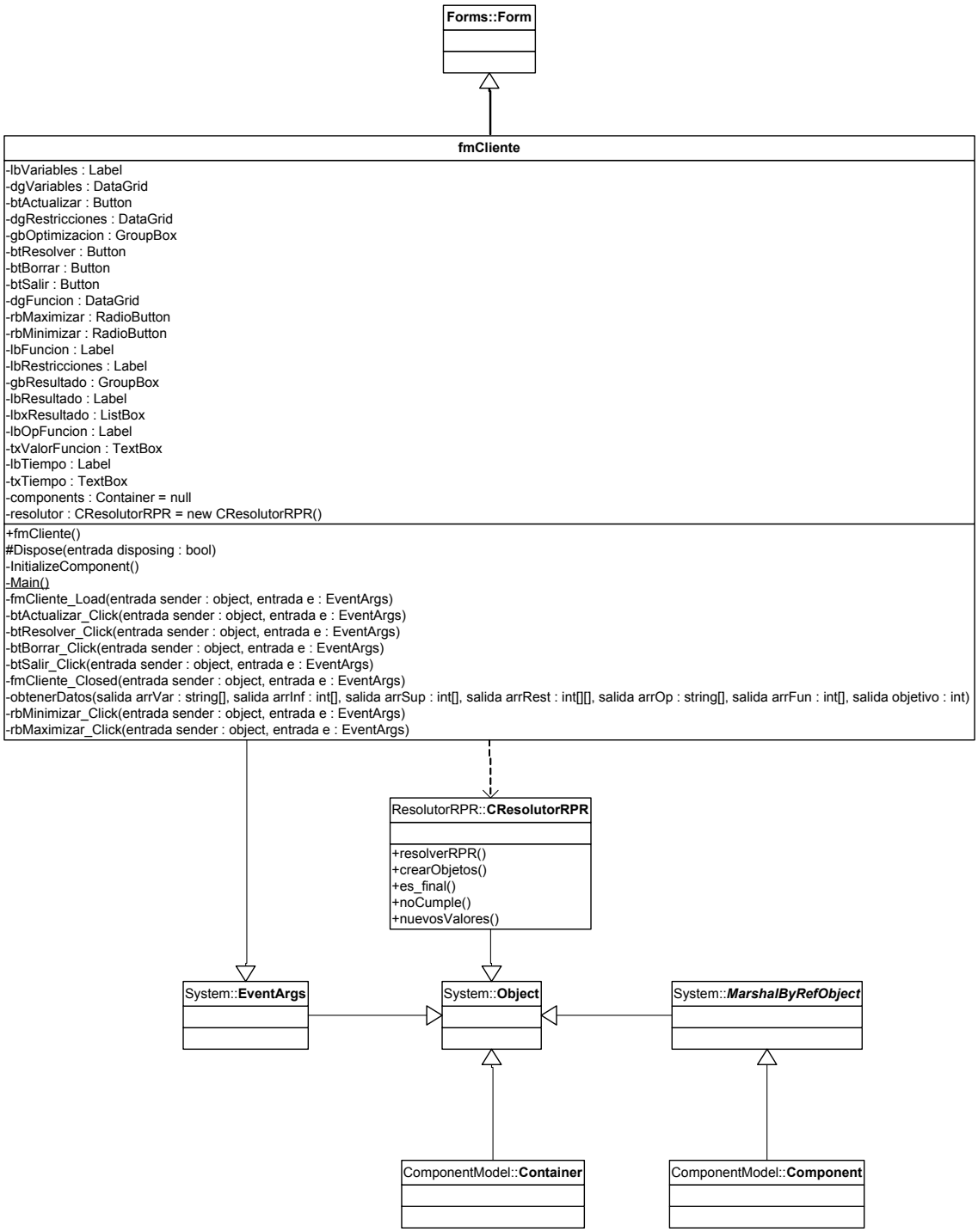
8.2.2.- Cliente

Aplicación realizada con Windows Forms de .NET compuesta de un interfaz gráfico que permite describir el problema sobre una misma ventana mediante la declaración de las variables y sus límites, de las restricciones, de la función a optimizar y del objetivo de la optimización de modo que se pueda llevar a cabo la resolución completa del problema.

8.2.2.1.- Diagrama de clases

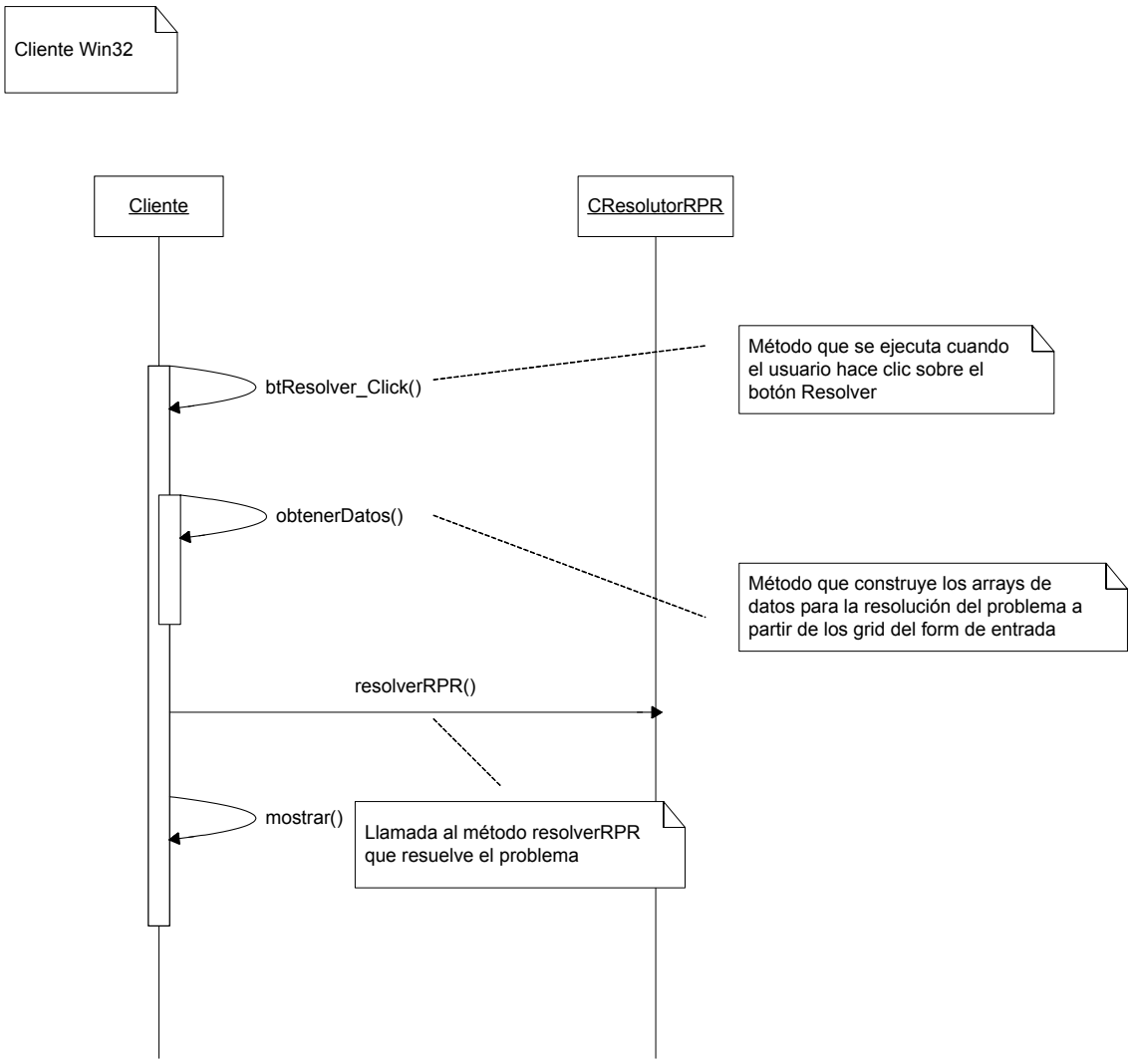
La clase desarrollada para la implementación de este componente ha sido:

- **Cliente**, formulario en el que el usuario deberá introducir las variables a tratar en la resolución, sus dominios, las restricciones a cumplir, la función de optimización y el objetivo.


Forms::Form
fmCliente
-lbVariables : Label
-dgVariables : DataGrid
-btActualizar : Button
-dgRestricciones : DataGrid
-gbOptimizacion : GroupBox
-btResolver : Button
-btBorrar : Button
-btSalir : Button
-dgFuncion : DataGrid
-rbMaximizar : RadioButton
-rbMinimizar : RadioButton
-lbFuncion : Label
-lbRestricciones : Label
-gbResultado : GroupBox
-lbResultado : Label
-lbxResultado : ListBox
-lbOpFuncion : Label
-txValorFuncion : TextBox
-lbTiempo : Label
-txTiempo : TextBox
-components : Container = null
-resolutor : CResolutorRPR = new CResolutorRPR()
+fmCliente()
#Dispose(entrada disposing : bool)
-InitializeComponent()
-Main()
-fmCliente_Load(entrada sender : object, entrada e : EventArgs)
-btActualizar_Click(entrada sender : object, entrada e : EventArgs)
-btResolver_Click(entrada sender : object, entrada e : EventArgs)
-btBorrar_Click(entrada sender : object, entrada e : EventArgs)
-btSalir_Click(entrada sender : object, entrada e : EventArgs)
-fmCliente_Closed(entrada sender : object, entrada e : EventArgs)
-obtenerDatos(salida arrVar : string[], salida arrInf : int[], salida arrSup : int[], salida arrRest : int[][], salida arrOp : string[], salida arrFun : int[], salida objetivo : int)
-rbMinimizar_Click(entrada sender : object, entrada e : EventArgs)
-rbMaximizar_Click(entrada sender : object, entrada e : EventArgs)
ResolutorRPR::CResolutorRPR
+resolverRPR()
+crearObjetos()
+es_final()
+noCumple()
+nuevosValores()
System::EventArgs
System::Object
System::MarshalByRefObject
ComponentModel::Container
ComponentModel::Component

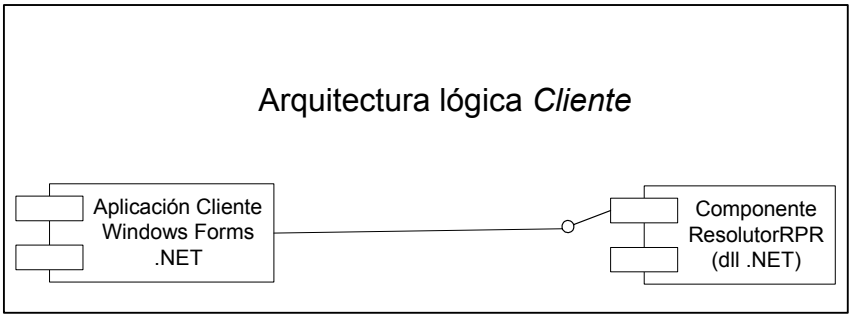
8.2.2.2.-Diagrama de secuencia

El siguiente diagrama de secuencia UML muestra el comportamiento del componente Cliente.



8.2.2.3.- Arquitectura

Toda la lógica reside en una sola máquina, que tiene instalado el Framework de .NET necesario para ejecutar la aplicación.



Los componentes lógicos de la aplicación son:

- La aplicación Win32 que establece el interfaz de usuario
- El componente .dll que realiza la resolución del problema mediante la implementación del algoritmo de ramificación y poda con backtracking.

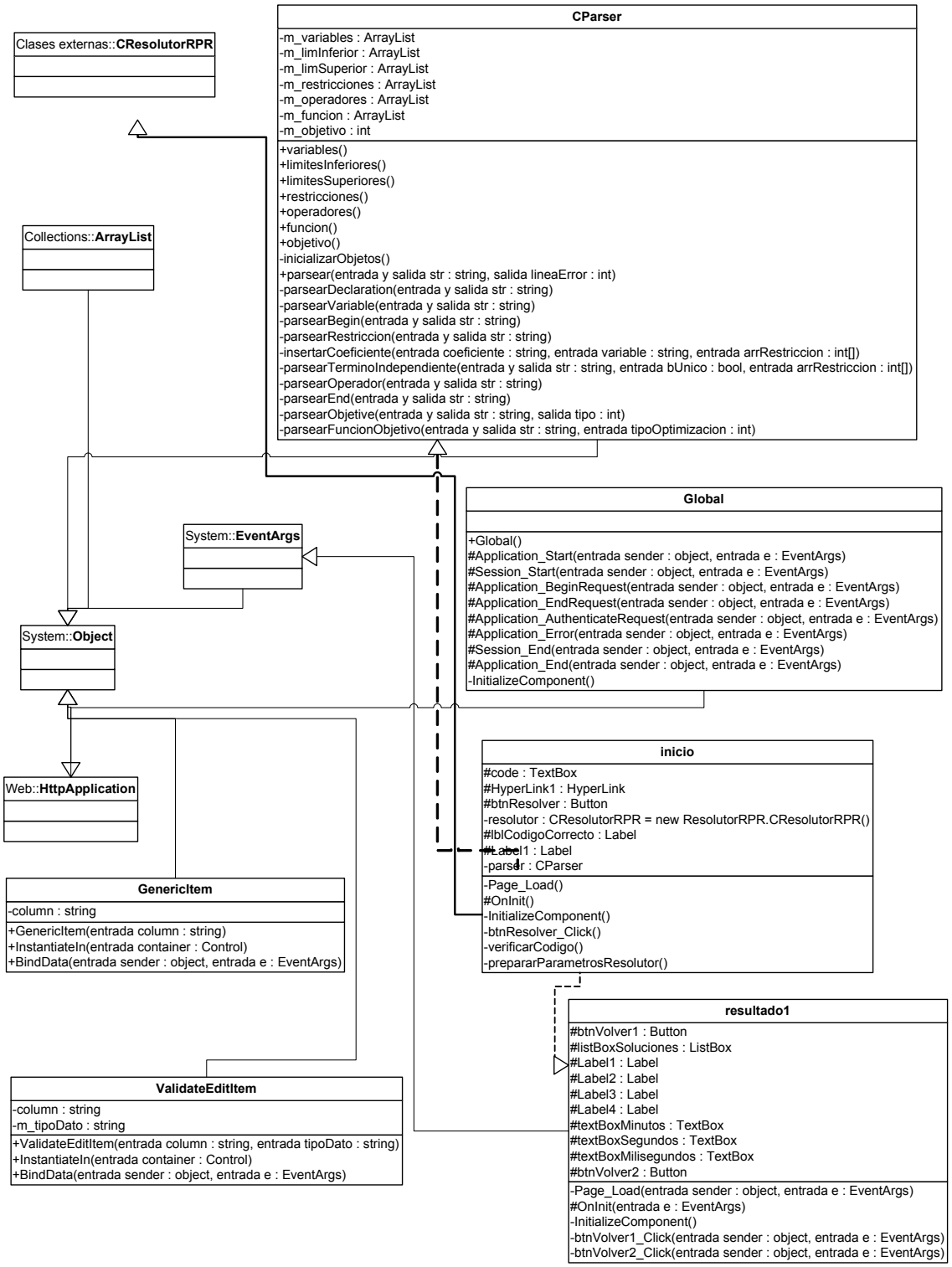
8.2.3.- ClienteWebRPR

Aplicación realizada con Web Forms de .NET (ASP .NET), que proporciona un interfaz de usuario para describir el problema a resolver (lenguaje LRPR), en una ventana, mediante la declaración de variables y restricciones, y que permite la resolución completa del problema.

8.2.3.1.- Diagrama de clases

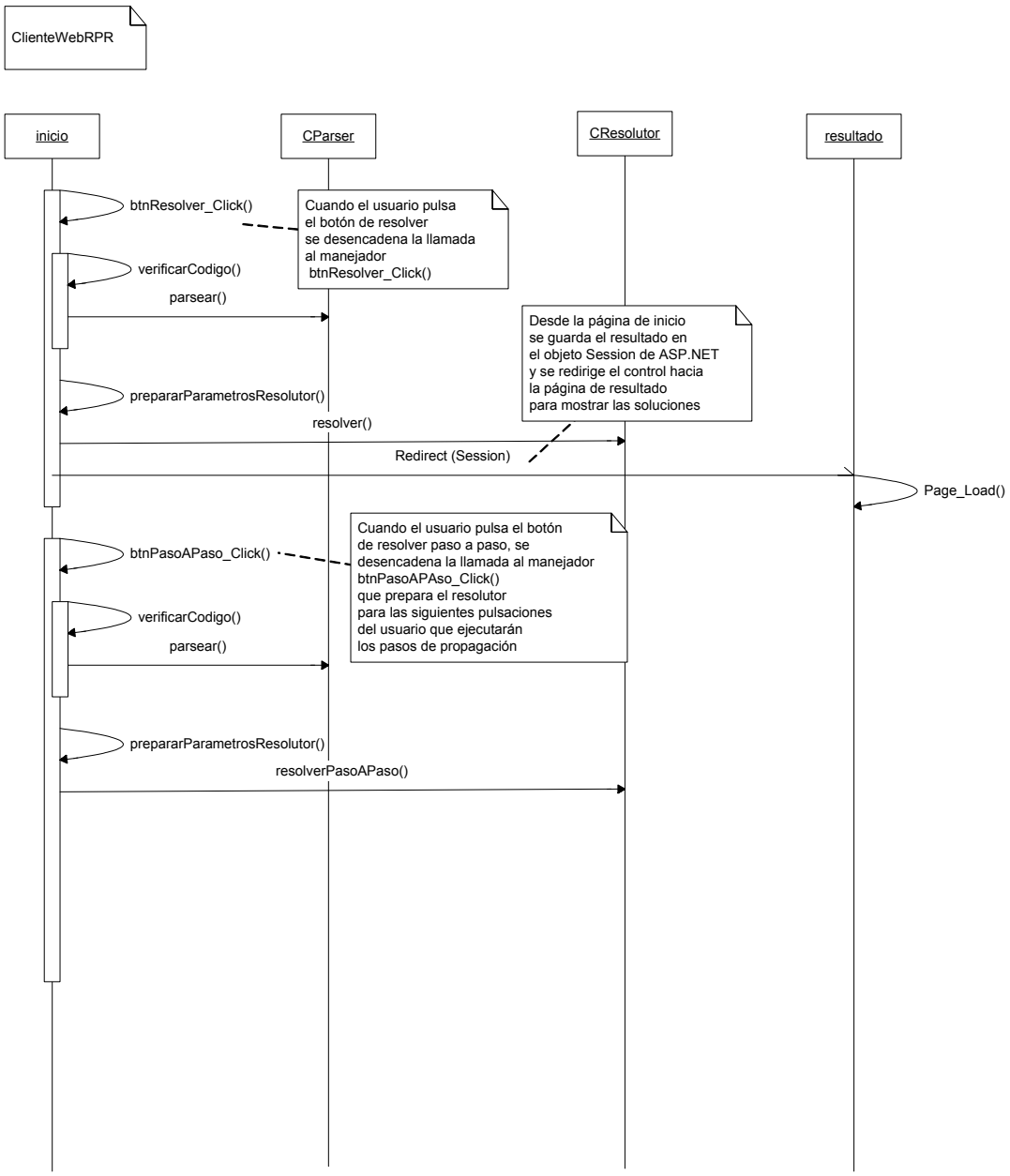
Las clases empleadas en la implementación de la aplicación clienteWebRPR, son las siguientes:

- **Inicio:** página donde se escribe el código en lenguaje *LRPR* para describir el problema, y que dispone del botón necesario para llevar a cabo la resolución del problema.
- **Resultado:** página a la que se accede desde *inicio* cuando se opta por resolver el problema. Muestra el valor de cada variable para la solución obtenida junto con el valor de la función optimización y el tiempo tardado en resolverlo.
- **Cparser:** clase que realiza el parseo del código fuente LRPR y que tiene como método principal resolver, al que se le pasa el código a parsear y genera las colecciones de objetos necesarios para la resolución del problema.
- **Global:** clase que representa la aplicación ASP.NET y que dispone de los manejadores de eventos para el inicio de aplicación, inicio de sesión, etc...


Clases externas::CResolutorRPR
CParser
-m_variables : ArrayList
-m_limInferior : ArrayList
-m_limSuperior : ArrayList
-m_restricciones : ArrayList
-m_operadores : ArrayList
-m_funcion : ArrayList
-m_objetivo : int
+variables()
+limitesInferiores()
+limitesSuperiores()
+restricciones()
+operadores()
+funcion()
+objetivo()
-inicializarObjetos()
+parsear(entrada y salida str : string, salida lineaError : int)
-parsearDeclaration(entrada y salida str : string)
-parsearVariable(entrada y salida str : string)
-parsearBegin(entrada y salida str : string)
-parsearRestriccion(entrada y salida str : string)
-insertarCoeficiente(entrada coeficiente : string, entrada variable : string, entrada arrRestriccion : int[])
-parsearTerminoIndependiente(entrada y salida str : string, entrada bUnico : bool, entrada arrRestriccion : int[])
-parsearOperador(entrada y salida str : string)
-parsearEnd(entrada y salida str : string)
-parsearObjetive(entrada y salida str : string, salida tipo : int)
-parsearFuncionObjetivo(entrada y salida str : string, entrada tipoOptimizacion : int)
Collections::ArrayList
Global
+Global()
#Application_Start(entrada sender : object, entrada e : EventArgs)
#Session_Start(entrada sender : object, entrada e : EventArgs)
#Application_BeginRequest(entrada sender : object, entrada e : EventArgs)
#Application_EndRequest(entrada sender : object, entrada e : EventArgs)
#Application_AuthenticateRequest(entrada sender : object, entrada e : EventArgs)
#Application_Error(entrada sender : object, entrada e : EventArgs)
#Session_End(entrada sender : object, entrada e : EventArgs)
#Application_End(entrada sender : object, entrada e : EventArgs)
-InitializeComponent()
System::EventArgs
System::Object
Web::HttpApplication
inicio
#code : TextBox
#HyperLink1 : HyperLink
#btnResolver : Button
-resolutor : CResolutorRPR = new ResolutorRPR.CResolutorRPR()
#lblCodigoCorrecto : Label
#Label1 : Label
-parser : CParser
-Page_Load()
#OnInit()
-InitializeComponent()
-btnResolver_Click()
-verificarCodigo()
-prepararParametrosResolutor()
GenericItem
-column : string
+GenericItem(entrada column : string)
+InstantiateIn(entrada container : Control)
+BindData(entrada sender : object, entrada e : EventArgs)
resultado1
#btnVolver1 : Button
#listBoxSoluciones : ListBox
#Label1 : Label
#Label2 : Label
#Label3 : Label
#Label4 : Label
#textBoxMinutos : TextBox
#textBoxSegundos : TextBox
#textBoxMilisegundos : TextBox
#btnVolver2 : Button
-Page_Load(entrada sender : object, entrada e : EventArgs)
#OnInit(entrada e : EventArgs)
-InitializeComponent()
-btnVolver1_Click(entrada sender : object, entrada e : EventArgs)
-btnVolver2_Click(entrada sender : object, entrada e : EventArgs)
ValidateEditItem
-column : string
-m_tipoDato : string
+ValidateEditItem(entrada column : string, entrada tipoDato : string)
+InstantiateIn(entrada container : Control)
+BindData(entrada sender : object, entrada e : EventArgs)

8.2.3.2.- Diagrama de secuencia

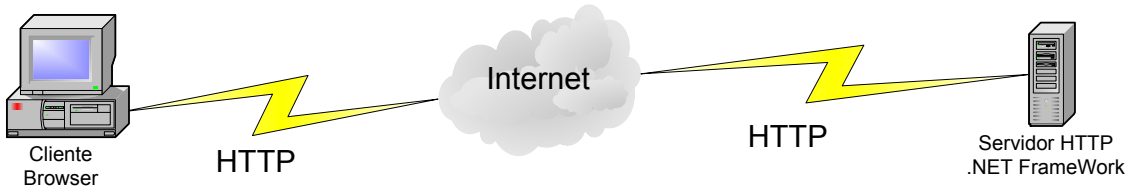
El siguiente diagrama de secuencia UML muestra el comportamiento de la aplicación clienteWebRPR:



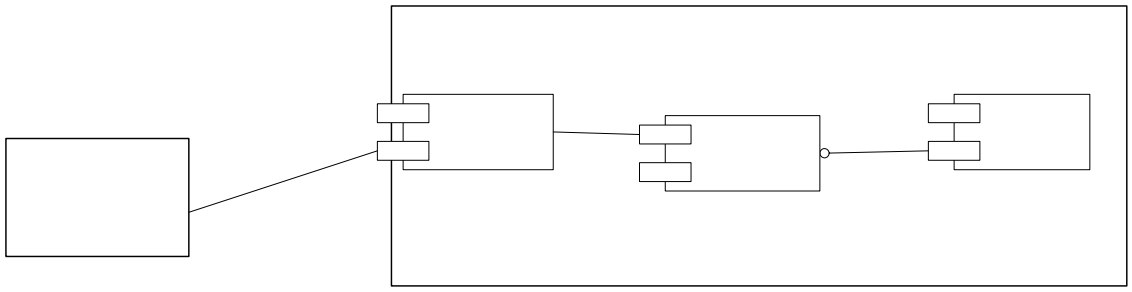
8.2.3.3.- Arquitectura

Aplicación realizada con Web Forms de .NET (ASP .NET), que proporciona un interfaz de usuario para describir el problema a resolver (lenguaje LRPR), en una ventana, mediante la declaración de variables y restricciones.

Arquitectura física clienteWebRPR



El cliente es un navegador de Internet que se conecta a una URL de modo que el servidor descarga sobre el navegador la página inicial, en la cual, se introduce el problema a resolver. Cuando se opta por resolver, el navegador envía la información al servidor http, que resuelve el problema, y devuelve la página de resultados al cliente.



El navegador, se conecta al servidor donde reside el servidor Web (http) que recoge la petición Web y que la reenvía a la aplicación ASP .NET, que la procesará. Después de preparar la información hace una petición al componente ResolutorRPR (dll) que resuelve el problema y devuelve los resultados a la aplicación Web que, a su vez, los devuelve al servidor http que construye la página de resultados que se envía al cliente.

8.2.4.- WSResolutorRPR

Este componente es un servicio Web que estará instalado en el Internet Information Server (IIS).

Para que el cliente pueda invocar los métodos Web deberá agregar una referencia Web a **http://<*servidor*>/WSResolutorRPR.asmx?WSDL**, con lo que recibirá el fichero XML de descripción de servicios Web que Visual Studio utiliza para poder extraer la información de los métodos y parámetros que estos servicios necesitan.

El cliente podrá invocar al único método Web disponible en el servicio (*resolverRPR*), que simplemente realizará una invocación al método *resolverRPR* del componente CResolutorRPR que debe estar instalado en el mismo servidor IIS para que pueda ser accedido. Con esto conseguimos que el cliente no tenga que tener instalado el resolutor en su máquina local, con las ventajas que esto supone para mantener siempre la última versión de forma transparente para los distintos clientes.

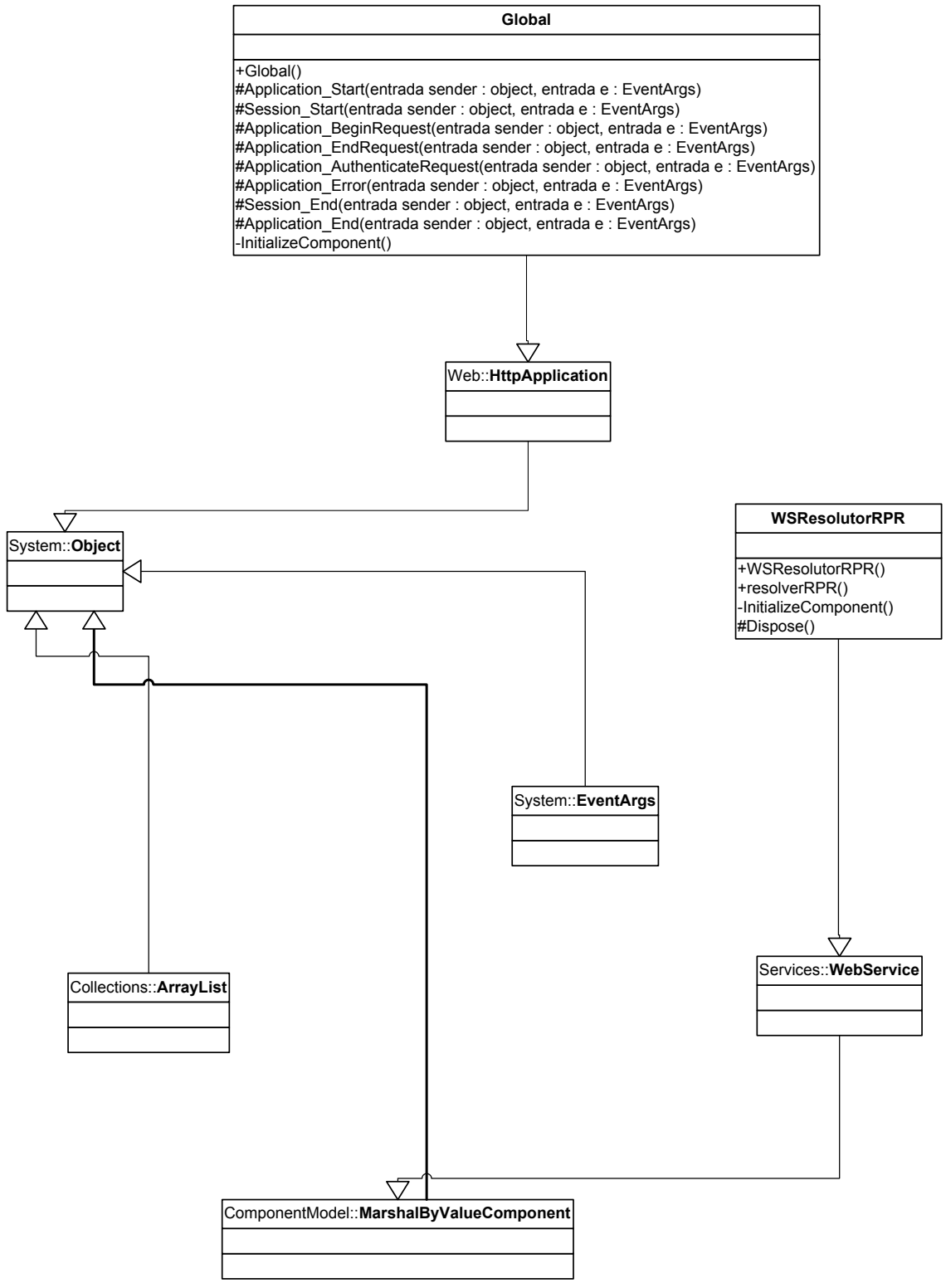
El método de resolución del problema (resolverRPR) que, como ya se ha explicado, recibe los datos del cliente en forma de arrays, resuelve completamente el problema planteado y devuelve el resultado en un objeto ArrayList (objeto del framework Microsoft .NET). Este método, es llamado desde la aplicación Windows (Cliente) y desde la aplicación Web (ClienteWS).

8.2.4.1.- Diagrama de clases

La clase desarrollada para la implementación de este componente ha sido:

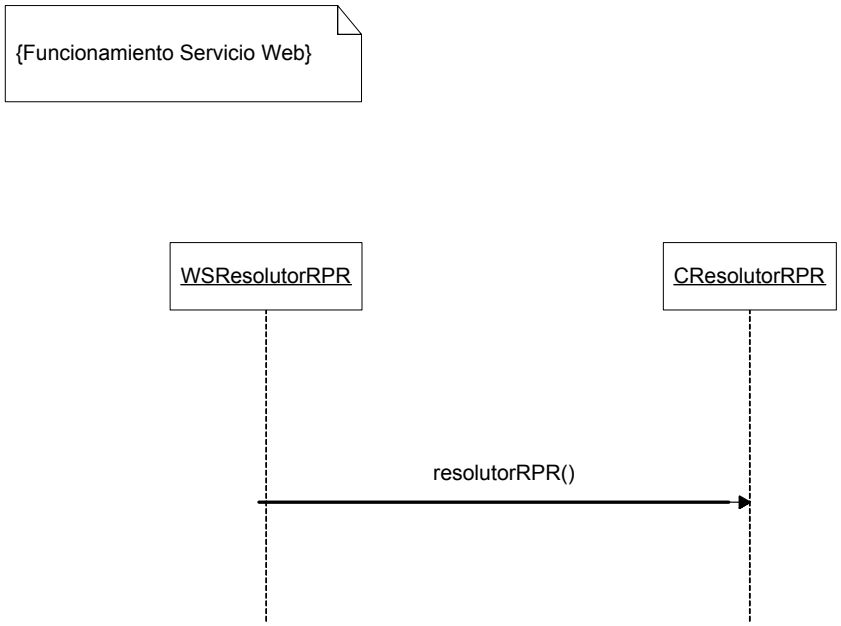
- **WSResolutorRPR,** contiene el método resolverRPR, es decir, recoge los parámetros enviados por el cliente y hace una llamada al componente CResolutorRPR con esos mismos parámetros. La salida del componente es recogida por el servicio y luego devuelta al cliente.

- **Global**, clase que representa la aplicación ASP.NET que alberga el servicio Web, y que dispone de los manejadores de eventos para el inicio de aplicación, el inicio de sesión etc,...


Global
+Global()
#Application_Start(entrada sender : object, entrada e : EventArgs)
#Session_Start(entrada sender : object, entrada e : EventArgs)
#Application_BeginRequest(entrada sender : object, entrada e : EventArgs)
#Application_EndRequest(entrada sender : object, entrada e : EventArgs)
#Application_AuthenticateRequest(entrada sender : object, entrada e : EventArgs)
#Application_Error(entrada sender : object, entrada e : EventArgs)
#Session_End(entrada sender : object, entrada e : EventArgs)
#Application_End(entrada sender : object, entrada e : EventArgs)
-InitializeComponent()
Web::HttpApplication
System::Object
WSResolutorRPR
+WSResolutorRPR()
+resolverRPR()
-InitializeComponent()
#Dispose()
System::EventArgs
Collections::ArrayList
Services::WebService
ComponentModel::MarshalByValueComponent

8.2.4.2.- Diagrama de secuencia

El siguiente diagrama de secuencia UML muestra el comportamiento del servicio:



8.2.5.- ClienteWS

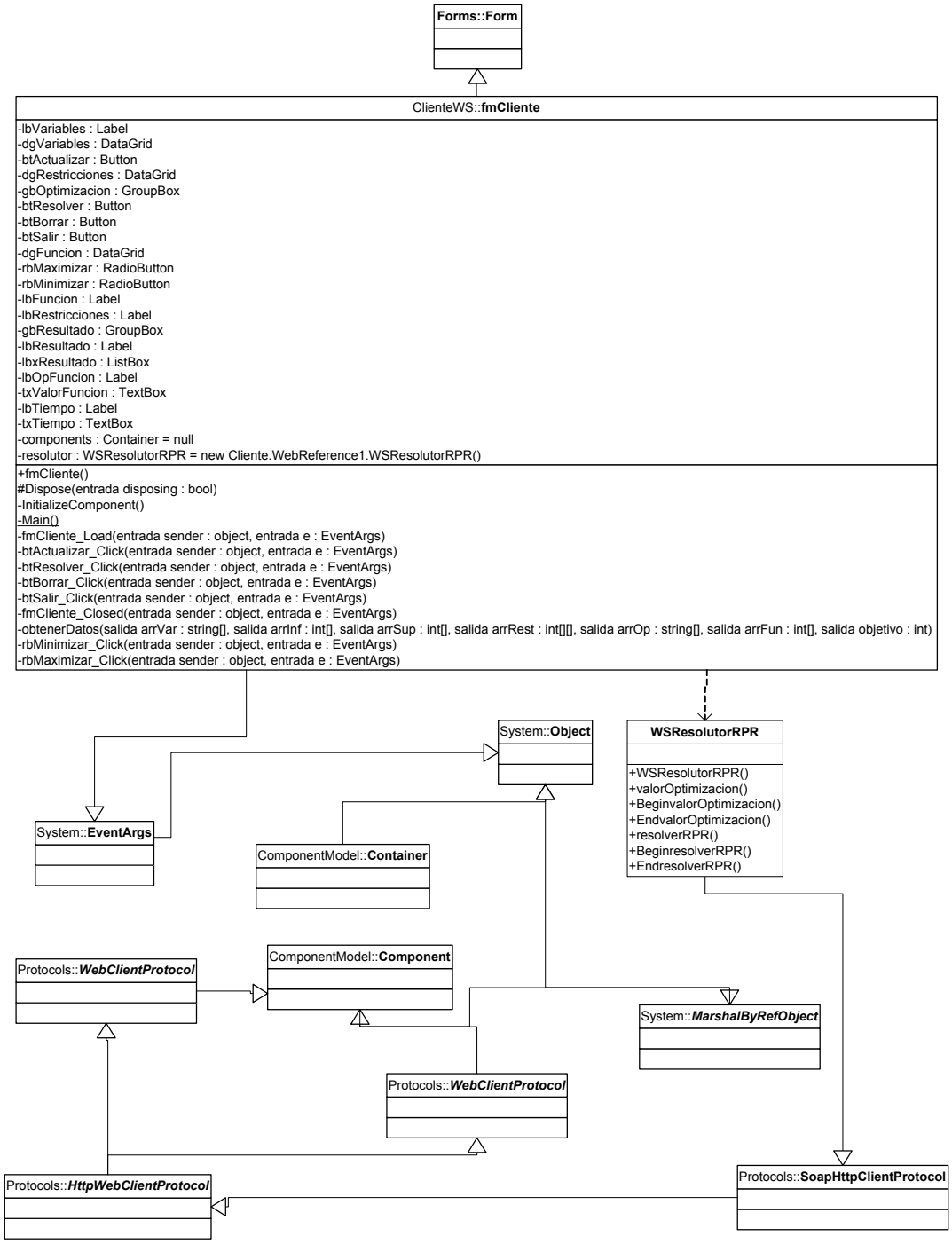
Aplicación realizada con Windows Forms de .NET compuesta de un interfaz gráfico que permite describir el problema sobre una misma ventana mediante la declaración de las variables y sus límites, de las restricciones, de la función a optimizar y del objetivo de la optimización de modo que se pueda llevar a cabo la resolución completa del problema.

8.2.5.1.- Diagrama de clases

La clase desarrollada para la implementación de este componente ha sido:

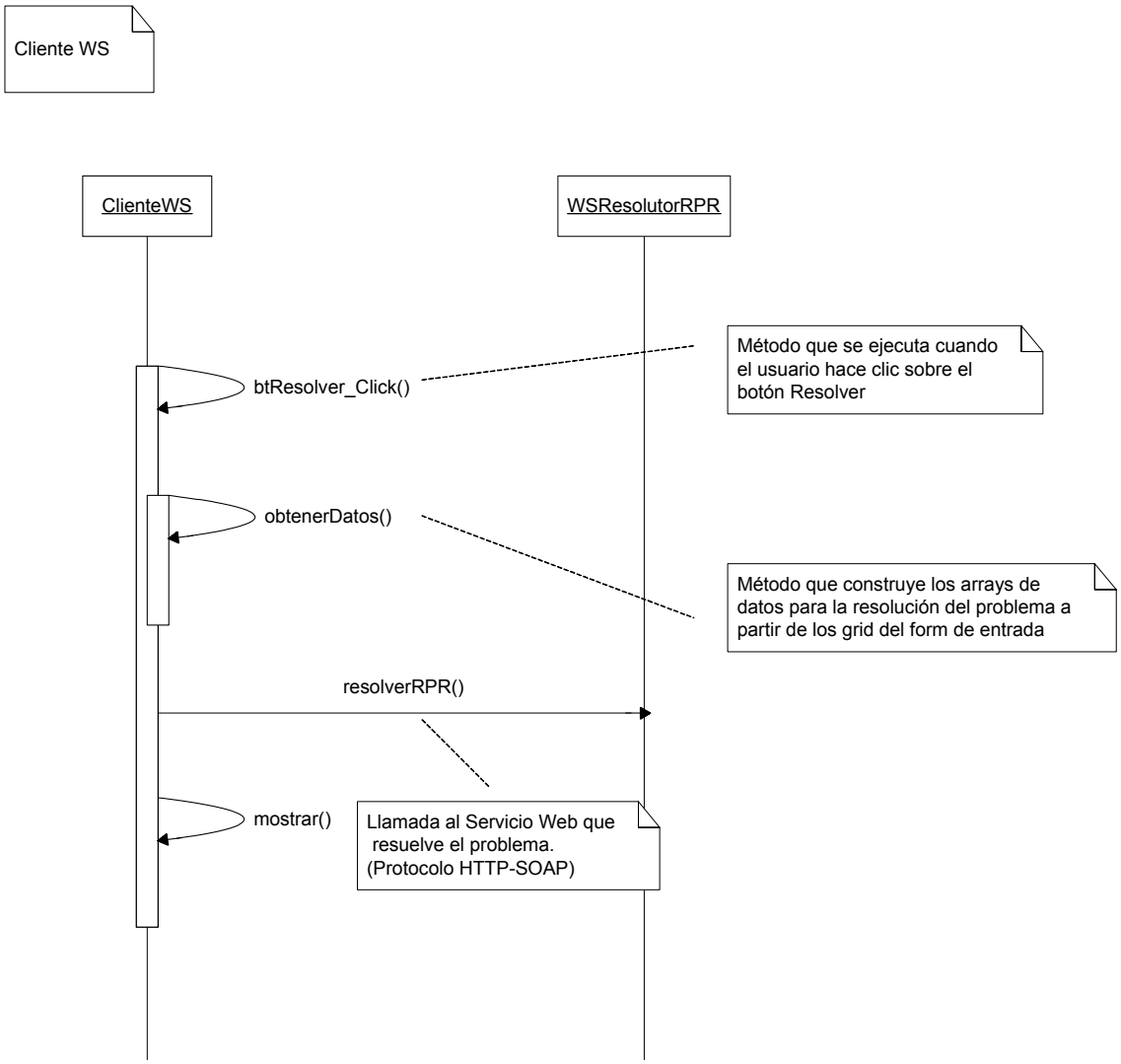
- **Cliente,** formulario en el que el usuario deberá introducir las variables a tratar en la resolución, sus dominios, las restricciones a cumplir, la función de optimización y el objetivo.

- **Global**, clase que representa la aplicación ASP.NET que alberga el servicio Web, y que dispone de los manejadores de eventos para el inicio de aplicación, el inicio de sesión etc,...


Forms::Form
ClienteWS::fmCliente
-lbVariables : Label
-dgVariables : DataGrid
-btActualizar : Button
-dgRestricciones : DataGrid
-gbOptimizacion : GroupBox
-btResolver : Button
-btBorrar : Button
-btSalir : Button
-dgFuncion : DataGrid
-rbMaximizar : RadioButton
-rbMinimizar : RadioButton
-lbFuncion : Label
-lbRestricciones : Label
-gbResultado : GroupBox
-lbResultado : Label
-lbxResultado : ListBox
-lbOpFuncion : Label
-txValorFuncion : TextBox
-lbTiempo : Label
-txTiempo : TextBox
-components : Container = null
-resolutor : WSResolutorRPR = new Cliente.WebReference1.WSResolutorRPR()
+fmCliente()
#Dispose(entrada disposing : bool)
-InitializeComponent()
-Main()
-fmCliente_Load(entrada sender : object, entrada e : EventArgs)
-btActualizar_Click(entrada sender : object, entrada e : EventArgs)
-btResolver_Click(entrada sender : object, entrada e : EventArgs)
-btBorrar_Click(entrada sender : object, entrada e : EventArgs)
-btSalir_Click(entrada sender : object, entrada e : EventArgs)
-fmCliente_Closed(entrada sender : object, entrada e : EventArgs)
-obtenerDatos(salida arrVar : string[], salida arrInf : int[], salida arrSup : int[], salida arrRest : int[][], salida arrOp : string[], salida arrFun : int[], salida objetivo : int)
-rbMinimizar_Click(entrada sender : object, entrada e : EventArgs)
-rbMaximizar_Click(entrada sender : object, entrada e : EventArgs)
System::Object
WSResolutorRPR
+WSResolutorRPR()
+valorOptimizacion()
+BeginvalorOptimizacion()
+EndvalorOptimizacion()
+resolverRPR()
+BeginresolverRPR()
+EndresolverRPR()
System::EventArgs
ComponentModel::Container
Protocols::WebClientProtocol
ComponentModel::Component
System::MarshalByRefObject
Protocols::WebClientProtocol
Protocols::HttpWebClientProtocol
Protocols::SoapHttpClientProtocol

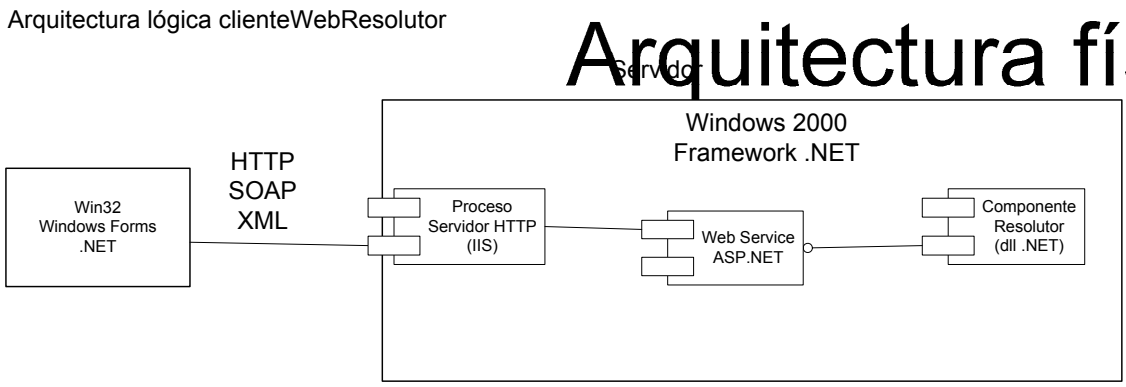
8.2.5.2.- Diagrama de secuencia

El siguiente diagrama de secuencia UML muestra el comportamiento del componente ClienteWS.



8.2.5.3.- Arquitectura

El cliente es una aplicación Windows Forms de .NET que establece el interfaz de usuario para la descripción del problema y que, cuando se decide resolver, realiza una llamada al método resolver de un servicio Web a través de Internet y mediante un servidor http (http, xml, soap) que, una vez que lo resuelve, devuelve la solución mediante xml al cliente.



Los componentes lógicos en este caso son:

- La aplicación Windows Forms (en la parte cliente) que se conecta vía Internet mediante protocolo http al servidor Web
- El servidor Web que recoge la petición y la reenvía al servicio Web
- El servicio Web que, a su vez, invoca al componente ResolutorRPR (dll) que se encuentra en el servidor y que será el que finalmente resuelva el problema. Una vez resuelto, devuelve el resultado al servicio Web, que se lo devuelve al servidor http, que lo envía mediante XML al cliente.

8.3.- Algoritmo genético

Se han diseñado e implementado seis componentes, que se combinan para dar lugar a tres aplicaciones distintas. Las tres son sistemas de optimización de restricciones basadas en un algoritmo genético, pero cada una correspondiente a un método de acceso distinto. Se presentan brevemente los seis componentes y sus interacciones:

- **Parser**: Component Library de .NET (Parser.dll)
  Biblioteca de clases que implementa un analizador del lenguaje LRPR. Este componente es reutilizado en las tres aplicaciones.

- **AG**: Component Library de .NET (AG.dll)
  Biblioteca de clases que implementa el algoritmo genético. Este componente es reutilizado en las tres aplicaciones.

- **InterfazWindowsAG**: aplicación para Windows (InterfazWindowsAG.exe)
  Es una aplicación autónoma tradicional con una interfaz rica construida con clases formulario de Windows Form de .NET. El componente InterfazWindowsAG tiene una referencia al componente AG y otra referencia al componente Parser. Esta aplicación se ejecuta en servidor con .NET Framework, desde un terminal del propio servidor.

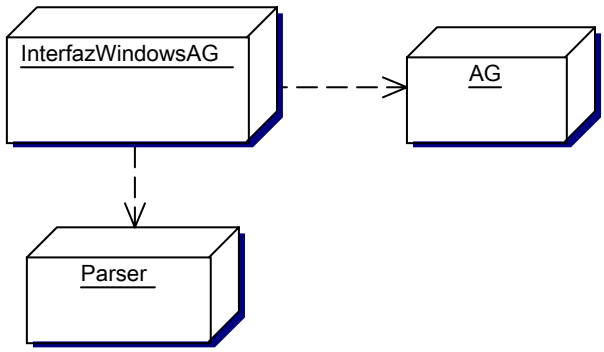


Figura 1.- Diagrama de contexto de la aplicación Windows

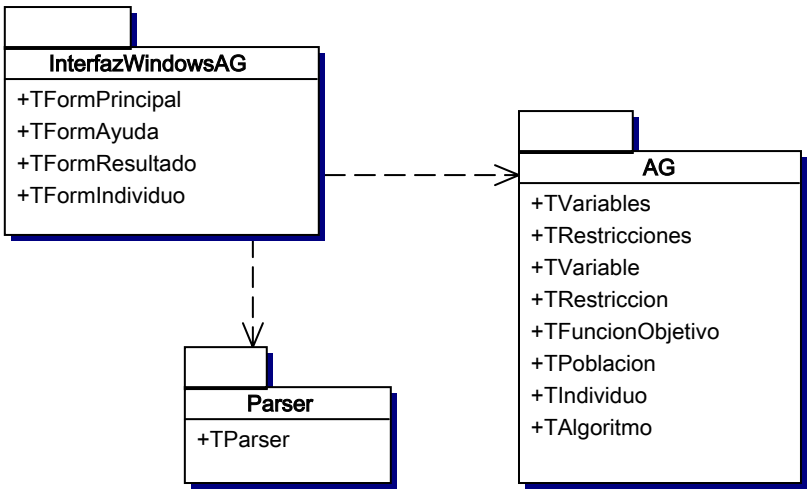


Figura 2.- Diagrama de paquetes de la aplicación Windows

- **InterfazWebAG**: aplicación Web de ASP .NET (interfazWebAG.exe)
  Es una aplicación web tradicional que sirve páginas HTML, realizadas con Web Forms de .NET. El componente InterfazWebAG tiene una referencia al componente AG y otra referencia al componente Parser. Esta aplicación se ejecuta en un servidor Web configurado con Microsoft Internet Information Services (IIS) y con .NET Framework, accediendo a ella a través de un browser ubicado en una máquina remota con conexión a Internet.

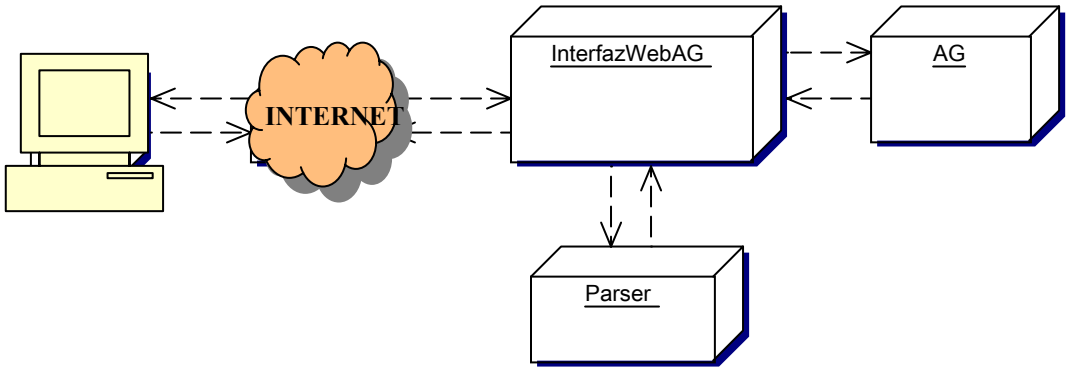


Figura 3.- Diagrama de contexto de la aplicación Web

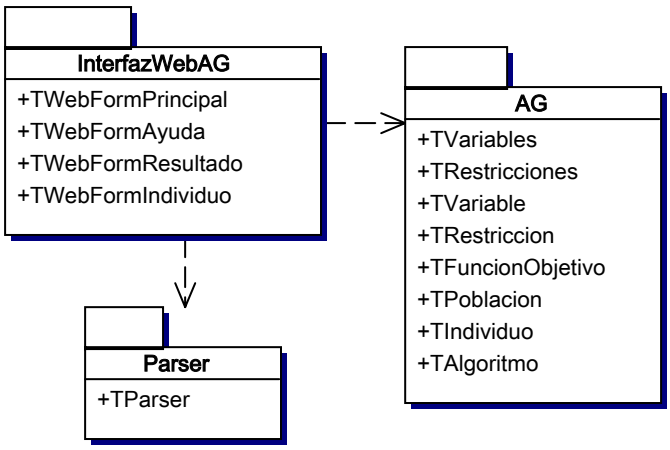


Figura 4.- Diagrama de paquetes de la aplicación Web

- **ServicioWebAG**: Servicio Web de ASP .NET (ServicioWebAG.dll)
  Es un Servicio Web XML que recibe la llamada externa del componente InterfazServicioWebAG. El componente ServicioWebAG tiene una referencia al componente AG.

- **InterfazServicioWebAG**: aplicación para Windows que invoca a un Servicio Web de ASP .NET (interfazServicioWebAG.exe)

  Es una aplicación distribuida con una interfaz rica construida con clases formulario de Windows Form de .NET. InterfazServicioWebAG tiene una referencia web al componente ServicioWebAG y otra referencia al componente Parser. Esta aplicación se ejecuta en una máquina remota con conexión a Internet, que se conecta a un servidor Web configurado con Microsoft Internet Information Services (IIS) y con .NET Framework mediante un Servicio Web.

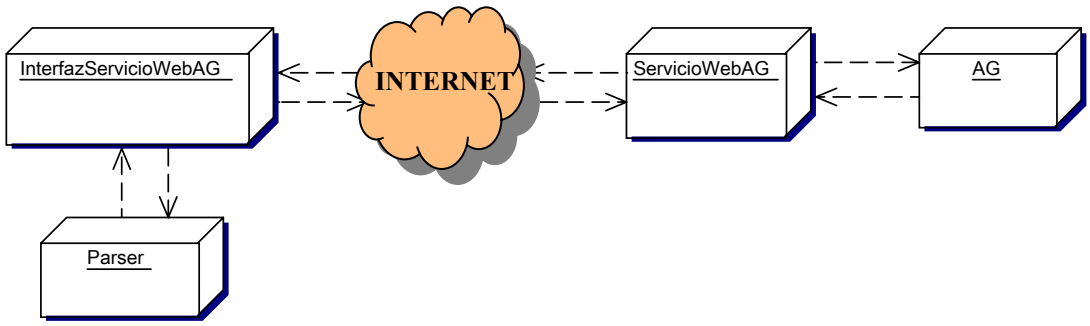


Figura 5.- Diagrama de contexto de la aplicación con Servicio Web

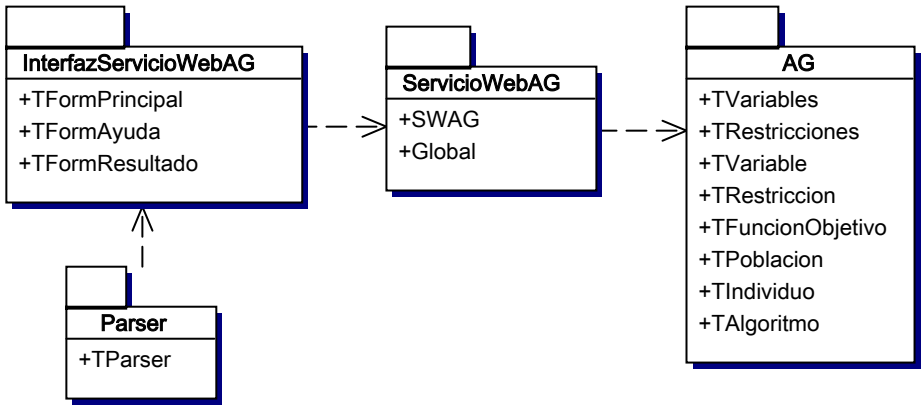


Figura 6.- Diagrama de paquetes de la aplicación con Servicio Web

8.3.1.- Parser

Biblioteca de clases que implementa un analizador del lenguaje LRPR. Este componente es reutilizado en las tres aplicaciones, ya que a él mantienen una referencia el componente InterfazWindowsAG, el componente InterfazWebAG y el componente InterfazServicioWebAG. Por lo tanto, el componente Parser deberá estar presente en el cliente donde se ejecuten estas aplicaciones.

8.3.1.1.- Diagrama de clases

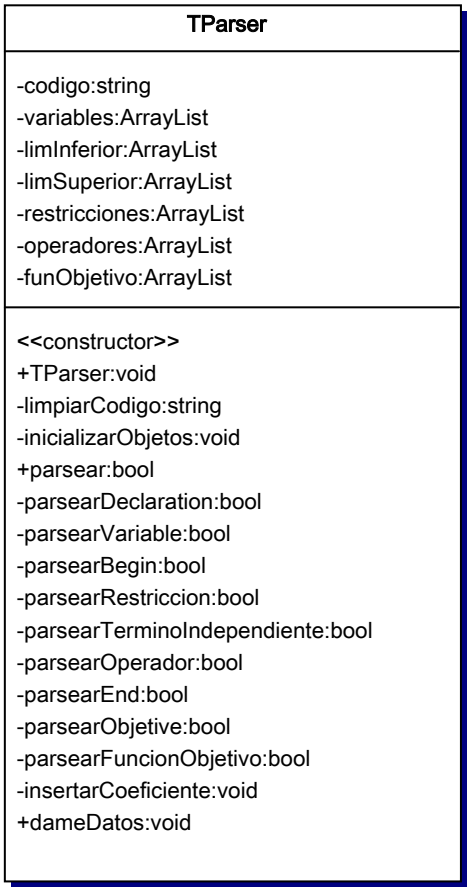
El componente **Parser** esta formado por las siguientes clases implementadas en el lenguaje C# de Microsoft .NET y desarrolladas en Visual Studio .NET:

- **TParser**: Clase que implementa un analizador del lenguaje *LRPR*. Dado un programa escrito en este lenguaje, que representa un problema de optimización de restricciones, comprueba que cumple la sintaxis del lenguaje *LRPR*. A la par que analiza el código, extrae toda la información que comprende el problema de optimización:
  - *variables*, obtiene un array de strings con los nombres de las variables declaradas, y dos arrays de enteros con los límites superiores e inferiores de las mismas.
  - *restricciones*, obtiene un array de arrays de enteros, donde cada array de enteros representa a cada restricción definida y almacena los coeficientes de las variables de la restricción (0 si no aparece la variable) en el orden en el que han sido declaradas éstas y el término independiente en la última posición; también obtiene un array de strings con los operadores de cada restricción.
  - función objetivo, obtiene un array de enteros con los coeficientes de las variables en la función objetivo (0 si no aparece la variable), donde la penúltima posición de este array representa el término independiente de la función objetivo y la última posición representa el tipo de optimización, -1 si es mínimo y 1 si es máximo. La función objetivo puede estar presente o no. Si no hay función objetivo declarada estos arrays tendrán como valor null.

Toda esta información la almacena en sus atributos privados (*variables, limInferior, limSuperior,...*), que va rellenando su método *parsear()*, el cual invoca a otros métodos que analizan cada elemento sintáctico de un programa escrito el *LRPR* (métodos *parsearDeclaration(), parsearVariables(), parsearBegin(),...*). Éste método finaliza cuando encuentra un error de sintaxis en el código devolviendo el valor false. En caso contrario devuelve el valor true y a continuación se puede invocar su método *dameDatos()* que devuelve todos los array citados. Antes de parsear el código de un programa se hace un tratamiento previo con su método *limpiarCódigo()* que eliminar los tabuladores, espacios en blanco y saltos de línea. Este código tratado se almacena en el atributo privado *código*.

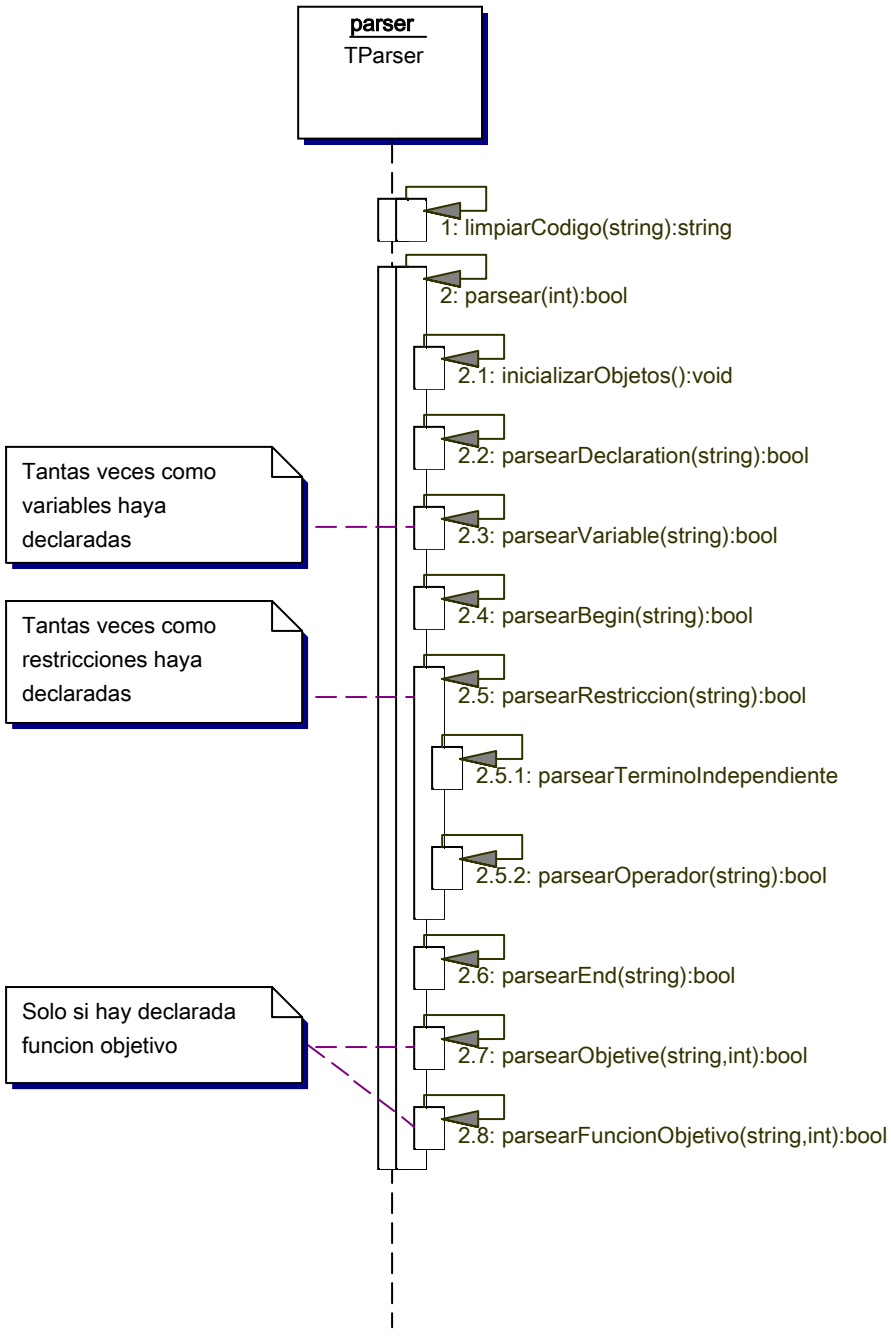
Esta clase es el único elemento del proyecto de C# *Parser*, que es una biblioteca de clases cuyo archivo de resultado es *Parser.dll*.

Este es el diagrama de clases UML correspondiente al componente **Parser**:



8.3.1.2.- Diagrama de secuencia

Este es el diagrama de secuencia UML que detalla el comportamiento del componente **Parser**:



8.3.2.- AG

Biblioteca de clases que implementa el algoritmo genético. Este componente es reutilizado en las tres aplicaciones, ya que a él mantienen una referencia el componente InterfazWindowsAG, el componente InterfazWebAG y el componente ServicioWebAG que a su vez es referenciado por el componente InterfazServicioWeb.

El componente AG tiene un conjunto de clases para representar de forma abstracta los elementos de un problema de restricciones. Estas clases son *TVariables, TVariable, TRestricciones, TRestriccion* y *TFuncionObjetivo*. El otro conjunto de clases que posee el componente AG tienen que ver con el algoritmo genético propio: *TAlgoritmo, TPoblacion* y *TIndividuo*. La clase *TPoblacion* y *TIndividuo* representan los elementos sobre los que trabaja un algoritmo genético, una población que a su vez es un conjunto de individuos. La clase *TAlgoritmo* implementa el esquema general de un algoritmo genético, pero de dos formas diferentes: con un método *algoritmoGenetico()* que proporciona una ejecución directa del algoritmo genético y dos métodos, *iniciarAlgoritmoGenetico()* *algoritmoGeneticoPaso()* que proporcionan una ejecución paso a paso del algoritmo genético.

Los parámetros del algoritmo genético son el número de generaciones, el tamaño de la población, la probabilidad de la aplicación del operador de cruce y la aplicación de la aplicación del operador de mutación. Estos datos se almacenarán en atributos privados de la clase *TAlgoritmo* y el componente AG los recibirá del componente que lo invoque.

Los datos del algoritmo genético son un array de strings con los nombres de las variables declaradas, dos arrays de enteros con los límites inferiores y superiores de las variables (con estos datos se creará un objeto de tipo *TVariables*), un array de arrays de enteros con las restricciones definidas, un array de strings con los operadores de las restricciones (con estos datos se creará un objeto de tipo *TRestricciones*) y un array de enteros con la función objetivo (con estos datos se creará un objeto de tipo *TFuncionObjetivo*). Estos datos se almacenarán en atributos privados de la clase *TAlgoritmo* y el componente AG los recibirá del componente que lo invoque.

8.3.2.1.- Diagrama de clases

El componente **AG** esta formado por las siguientes clases implementadas en el lenguaje C# de Microsoft .NET y desarrolladas en Visual Studio .NET:

- **TAlgoritmo**: Clase que representa el algoritmo genético. Los atributos son los parámetros y los datos del algoritmo genético, cuyo constructor se encarga de rellenarlos. Tiene los métodos necesarios para ejecutar el algoritmo genético en sus dos modalidades: ejecución directa, método *algoritmoGénetico()*, y ejecución pasos a paso, método *inciarAlgoritmoGenético()* para el primer paso y *algoritmoGenéticoPaso()* para los pasos sucesivos.

- **TPoblacion**: Clase que representa una población sobre la que se aplica el algoritmo genético. Su atributo población representa una colección de objetos de la clase *TIndividuo*. Posee métodos para crear una población, generar la población inicial y aplicar los operadores de selección, cruce y mutación sobre una población.

- **TIndividuo**: Clase que representa un individuo sobre el que se aplica el algoritmo genético. Sus atributos aptitud, puntuación y puntuación acumulada sirven para evaluar al individuo. Su atributo genes codifica la información. Tiene métodos para crear un individuo, generar aleatoriamente su cadena de genes y aplicar los operadores de mutación y de cruce sobre un individuo.

- **TFuncionObjetivo**: Clase que representa la función objetivo del problema de optimización. Sus atributos son los coeficientes y el término independiente de dicha función objetivo, además del tipo de optimización (máximo o mínimo). Tiene un método para calcular su valor.

- **TRestricciones**: Clase que representa el conjunto de restricciones del problema de optimización.

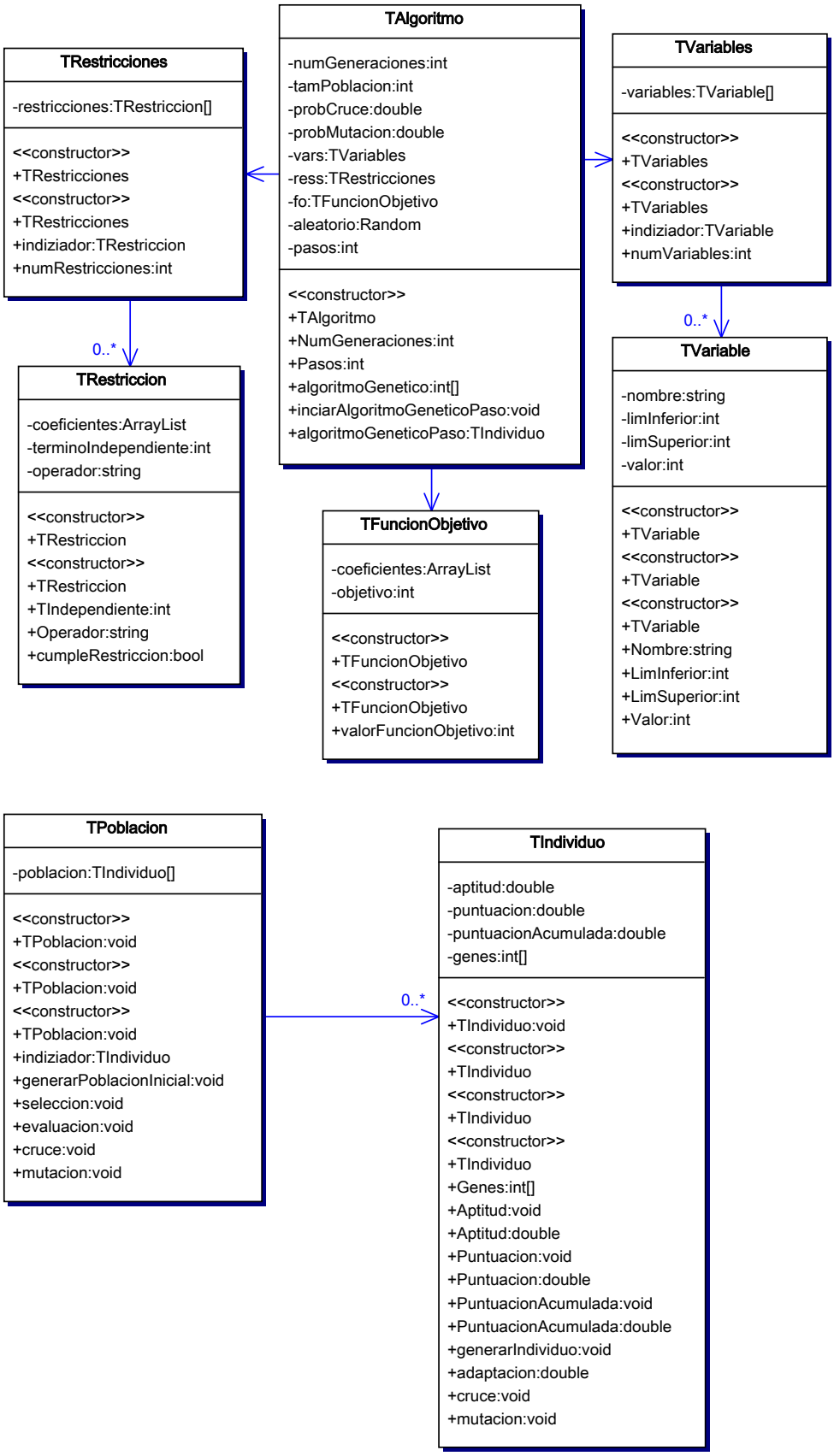
- **TRestriccion**: Clase que representa una restricción del problema de optimización. Sus atributos son los coeficientes, el termino independiente y el operador de dicha restricción. Tiene métodos para calcular si un conjunto de valores cumple la restricción.

- **TVariables**: Clase que representa el conjunto de variables del problema de optimización.

- **TVariable**: Clase que representa una variable del problema de optimización. Sus atributos son el nombre de la variable y su dominio, dado por el límite inferior y el límite superior.

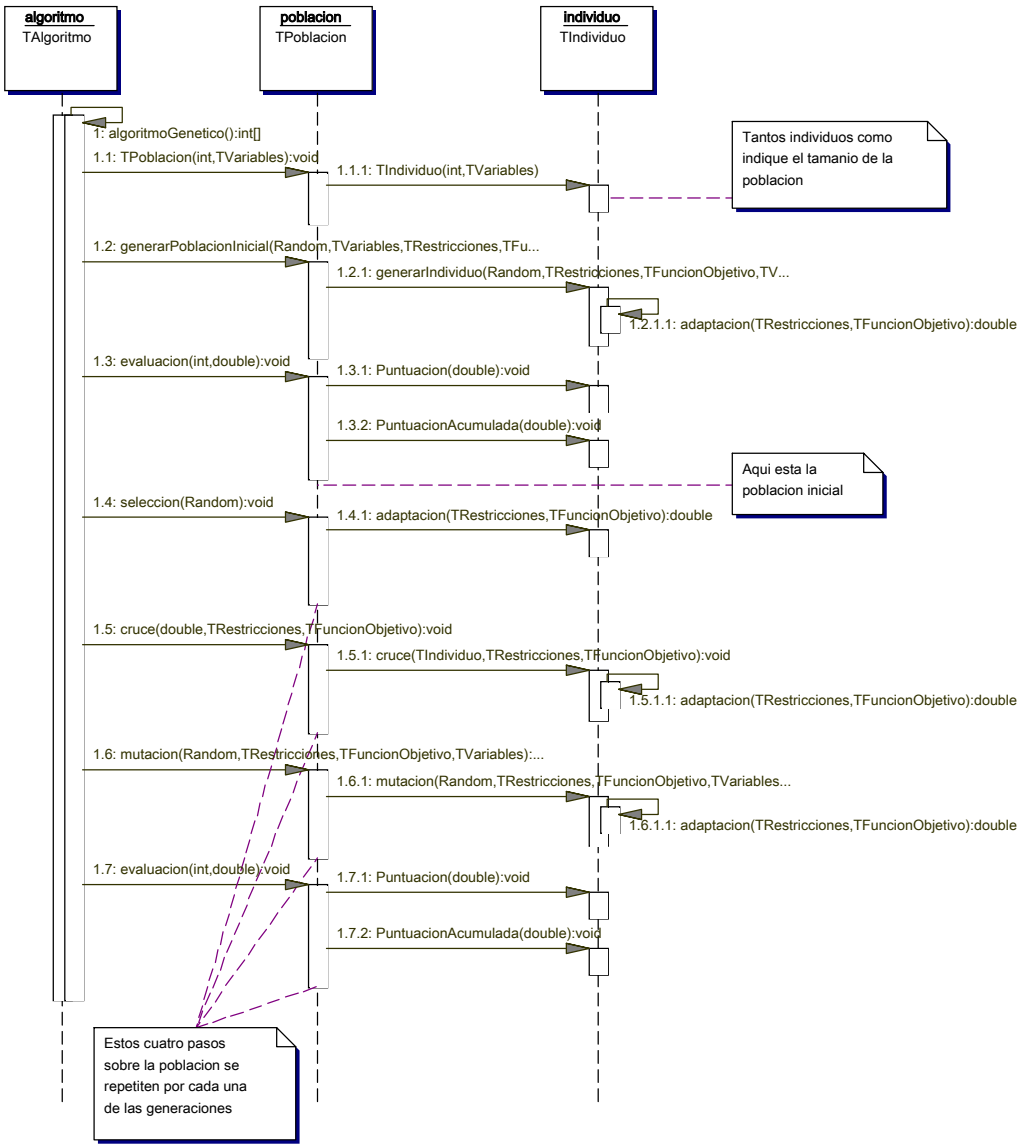
Estas clases son los elementos del proyecto de C# AG, que es una biblioteca de clases cuyo archivo de resultado es AG.dll

Este es el diagrama de clases UML correspondiente al componente **interfazWindowsAG**:

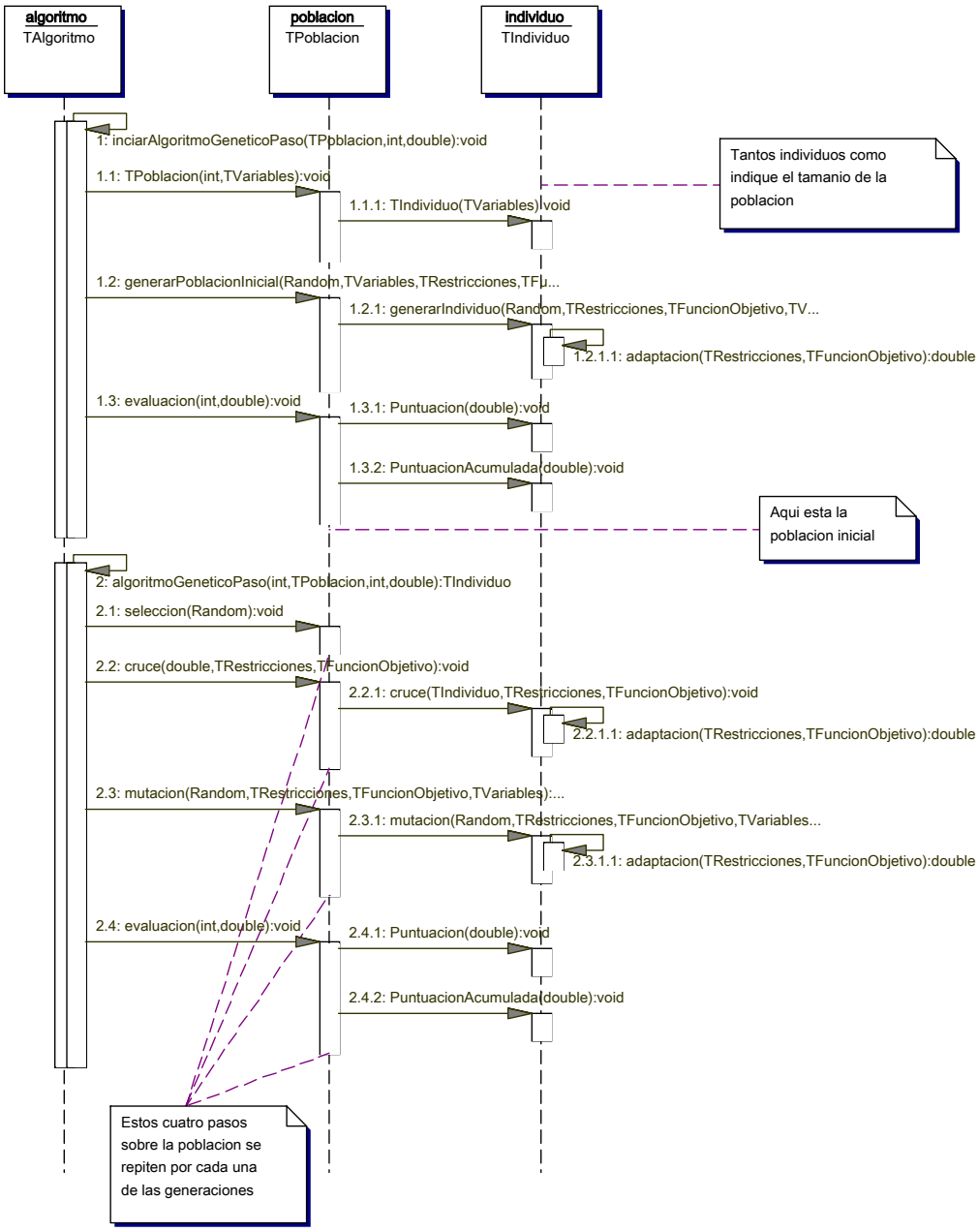


8.3.2.1.- Diagramas de secuencia

Este es el diagrama de secuencia UML que detalla el comportamiento del componente **AG** durante una ejecución directa:



Este es el diagrama de secuencia UML que detalla el comportamiento del componente **AG** durante una ejecución paso a paso:



8.3.3.- InterfazWindowsAG

Es una aplicación para Windows tradicional con una interfaz rica construida con clases formulario de Windows Form de .NET. El componente InterfazWindowsAG tiene una referencia al componente AG y otra referencia al componente Parser.

8.3.3.1.- Diagrama de clases

El componente **interfazWindowsAG** esta formado por las siguientes clases implementadas en el lenguaje C# de Microsoft .NET y desarrolladas en Visual Studio .NET:

- **TFormPrincipal**: formulario windows que se muestra al lanzar la aplicación. En él el usuario puede editar o cargar directamente desde un fichero un programa escrito con el lenguaje *LRPR*, para seguidamente solicitar la verificación del código. Si el programa es correcto, entonces el usuario puede ordenar la resolución del problema mediante el algoritmo evolutivo, permitiendo elegir una ejecución directa o una ejecución paso a paso, donde es configurable el número de pasos a dar. También permite modificar los valores por defecto de los parámetros del algoritmo evolutivo.

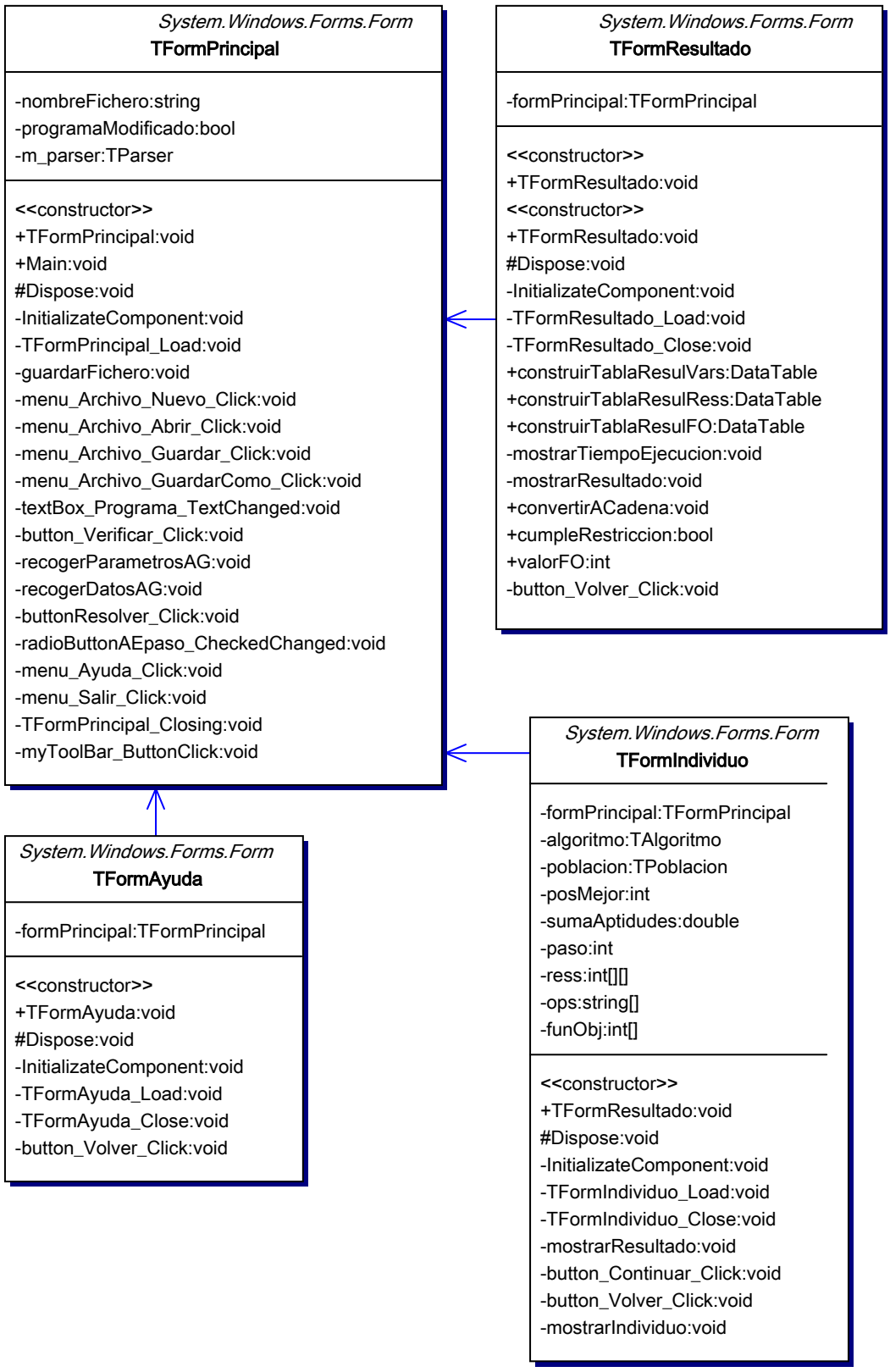
- **TFormResultado**: formulario windows que se muestra tras la solicitud de resolver directamente un problema de optimización representado por un programa verificado. En este formulario se presenta toda la información relativa al problema y a la solución encontrada por el algoritmo genético en forma de tablas. También muestra el tiempo empleado en la resolución del problema.

- **TFormIndividuo**: formulario windows que se muestra tras la solicitud de resolver paso a paso un problema de optimización representado por un programa verificado. En este formulario se presenta toda la información relativa al problema y a la solución parcial hasta el paso actual encontrada por el algoritmo evolutivo en forma de tablas. También muestra las características del mejor individuo en la generación correspondiente al paso actual. Inicialmente se muestra la solución parcial encontrada en el paso 0, es decir, el mejor individuo de la generación inicial. Después el usuario solicita en este formulario dar un nuevo salto, que significa avanzar tantas generaciones como sea el número de pasos. Se mostrarán tantos formularios como número de generaciones entre número de pasos. Cuando se llegue al número de generaciones o se sobrepase, se mostrará la solución final encontrada por el algoritmo genético.

- **TFormAyuda**: formulario Windows que se muestra tras la solicitud de información por parte del usuario. Presenta la sintaxis del lenguaje LRPR utilizada para escribir los programas de restricciones.

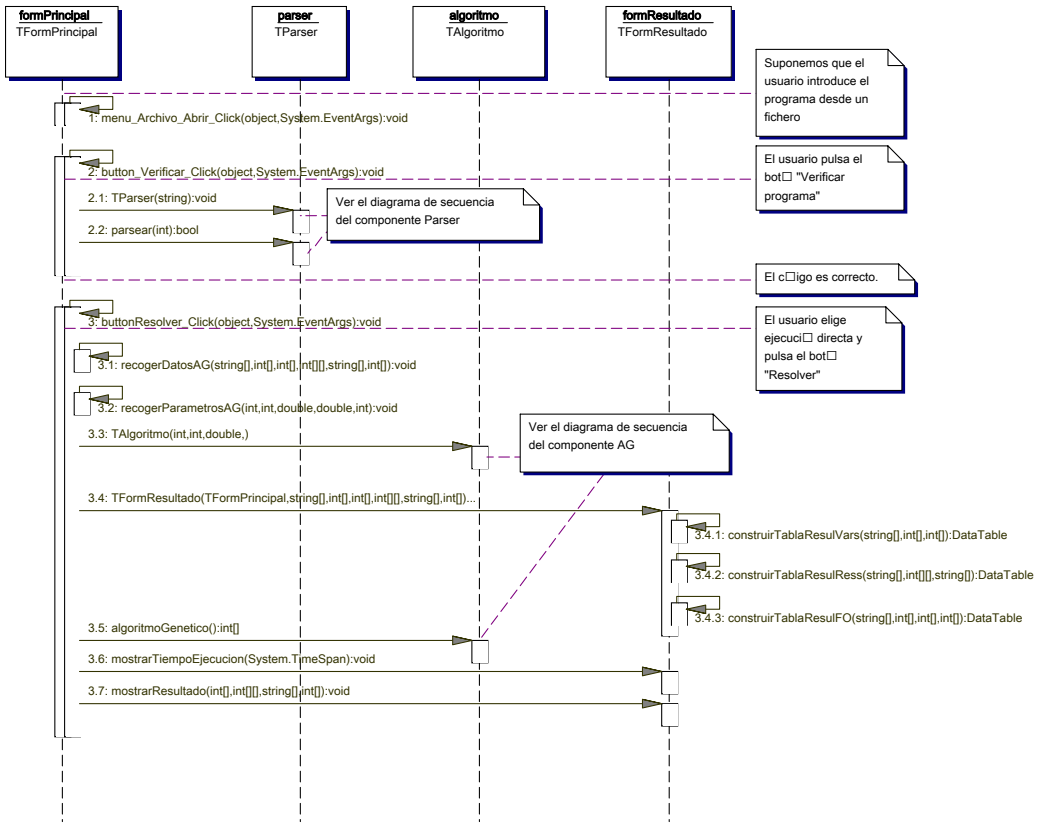
Estas clases son los elementos del proyecto de C# interazWindowsAG, que es una aplicación para Windows cuyo archivo de resultado es interfazWindowsAG.exe

Este es el diagrama de clases UML correspondiente al componente interfazWindowsAG:

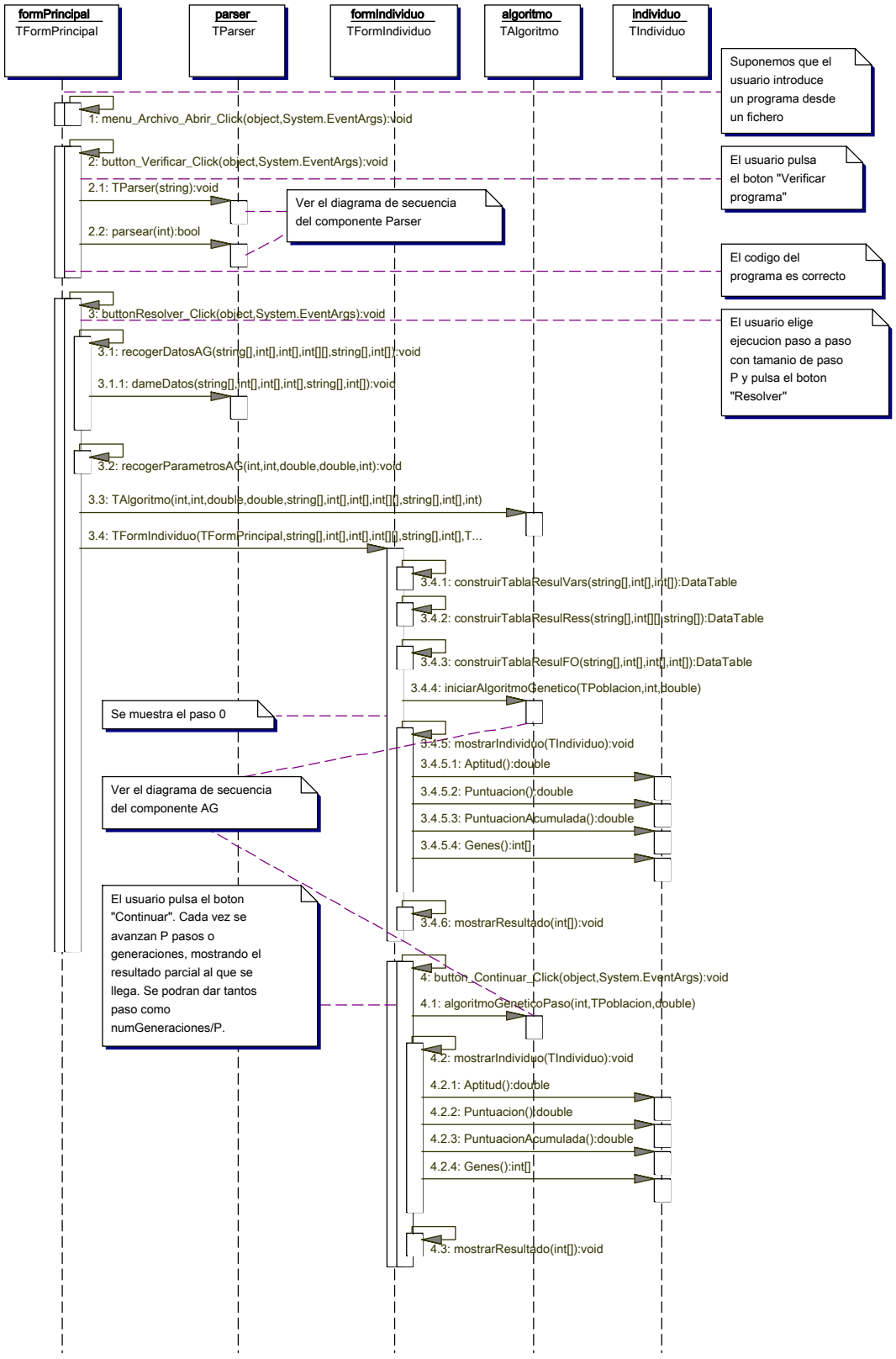


8.3.4.2.- Diagramas de secuencia

Este es el diagrama de secuencia UML que detalla el comportamiento del componente **interfazWindowsAG** durante una ejecución directa del algoritmo genético:



Este es el diagrama de secuencia UML que detalla el comportamiento del componente **interfazWindowsAG** durante una ejecución paso a paso del algoritmo genético:

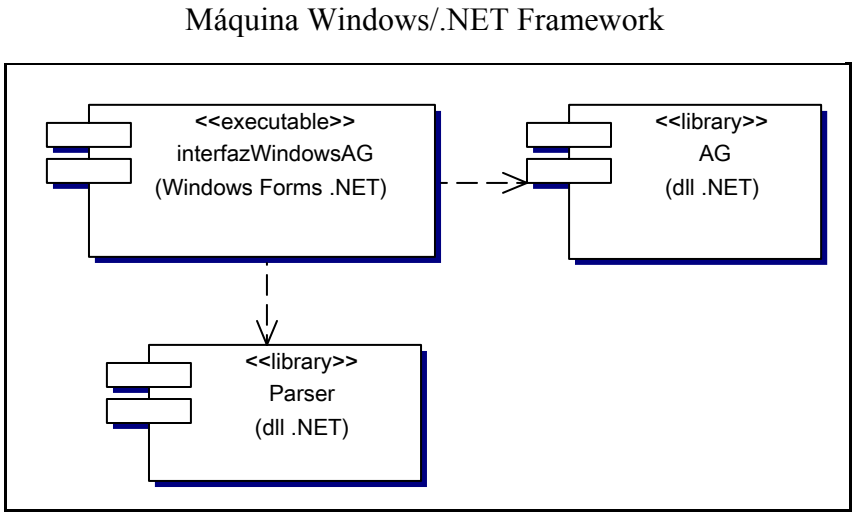


8.3.4.3.- Arquitectura

- Arquitectura física aplicación para Windows

En la aplicación para Windows, todos los componentes residen en una única máquina. Para ejecutar la aplicación dicha máquina debe tener instalado Microsoft Windows y .NET Framework.

- Arquitectura lógica aplicación para Windows



Los componentes lógicos de la aplicación son el componente interfazWindowsAG.exe, que establece la interfaz de usuario, el componente Parser.dll, que analiza los programas LRPR, y el componente AG.dll, que resuelve problemas de restricciones mediante un algoritmo evolutivo.

8.3.4.- InterfazWebAG

Es una aplicación web tradicional que sirve páginas HTML, realizadas con Web Forms de .NET. El componente InterfazWebAG tiene una referencia al componente AG y otra referencia al componente Parser.

8.3.4.1.- Diagrama de clases

El componente **interfazWebAG** esta formado por las siguientes clases implementadas en el lenguaje C# de Microsoft .NET y desarrolladas en Visual Studio .NET:

- **WebFormPrincipal**: formulario web que se muestra al lanzar la aplicación. En él el usuario puede editar un programa escrito con el lenguaje *LRPR*, para seguidamente solicitar la verificación del código. Si el programa es correcto, entonces el usuario puede ordenar la resolución del problema mediante el algoritmo evolutivo, permitiendo elegir una ejecución directa o una ejecución paso a paso, donde es configurable el número de pasos a dar. También permite modificar los valores por defecto de los parámetros del algoritmo evolutivo.

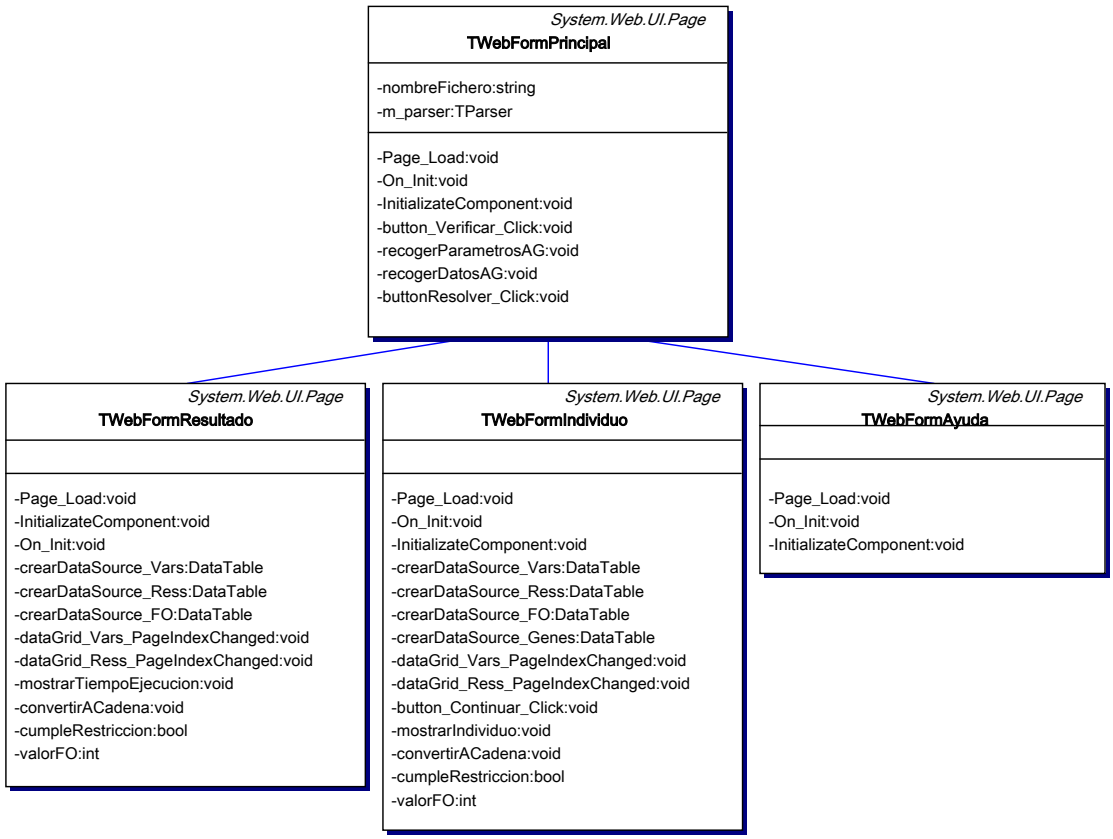
- **WebFormResultado**: formulario web que se muestra tras la solicitud de resolver directamente un problema de optimización representado por un programa verificado. En este formulario se presenta toda la información relativa al problema y a la solución encontrada por el algoritmo genético en forma de tablas. También muestra el tiempo empleado en la resolución del problema.

- **WebFormIndividuo**: formulario web que se muestra tras la solicitud de resolver paso a paso un problema de optimización representado por un programa verificado. En este formulario se presenta toda la información relativa al problema y a la solución parcial hasta el paso actual encontrada por el algoritmo evolutivo en forma de tablas. También muestra las características del mejor individuo en la generación correspondiente al paso actual. Inicialmente se muestra la solución parcial encontrada en el paso 0, es decir, el mejor individuo de la generación inicial. Después el usuario solicita en este formulario dar un nuevo salto, que significa avanzar tantas generaciones como sea el número de pasos. Se mostrarán tantos formularios como número de generaciones entre número de pasos. Cuando se llegue al número de generaciones o se sobrepase, se mostrará la solución final encontrada por el algoritmo genético.

- **WebFormAyuda**: formulario web que se muestra tras la solicitud de información por parte del usuario. Presenta la sintaxis del lenguaje LRPR utilizada para escribir los programas de restricciones.

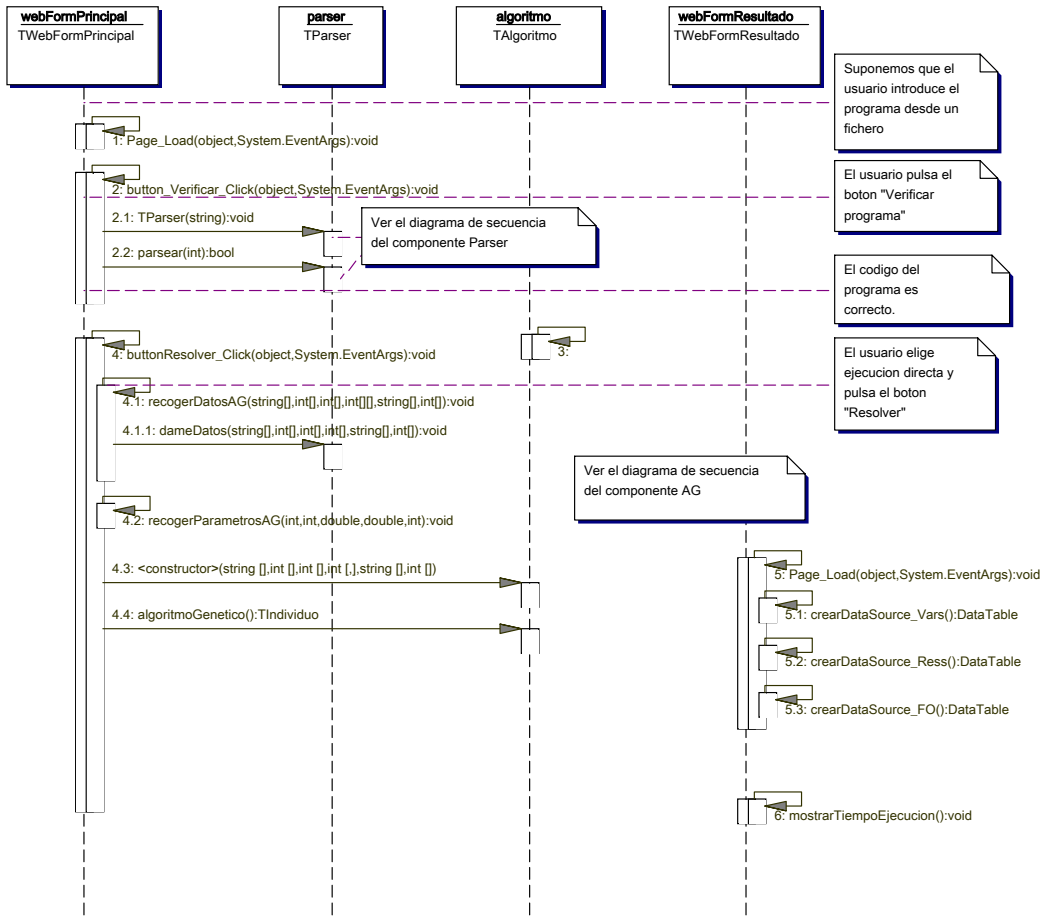
Estas clases son los elementos del proyecto de C# interazWebAG, que es una aplicación Web ASP .NET cuyo archivo de resultado es interfazWebAG.exe

Este es el diagrama de clases UML correspondiente al componente **interfazWebAG**:

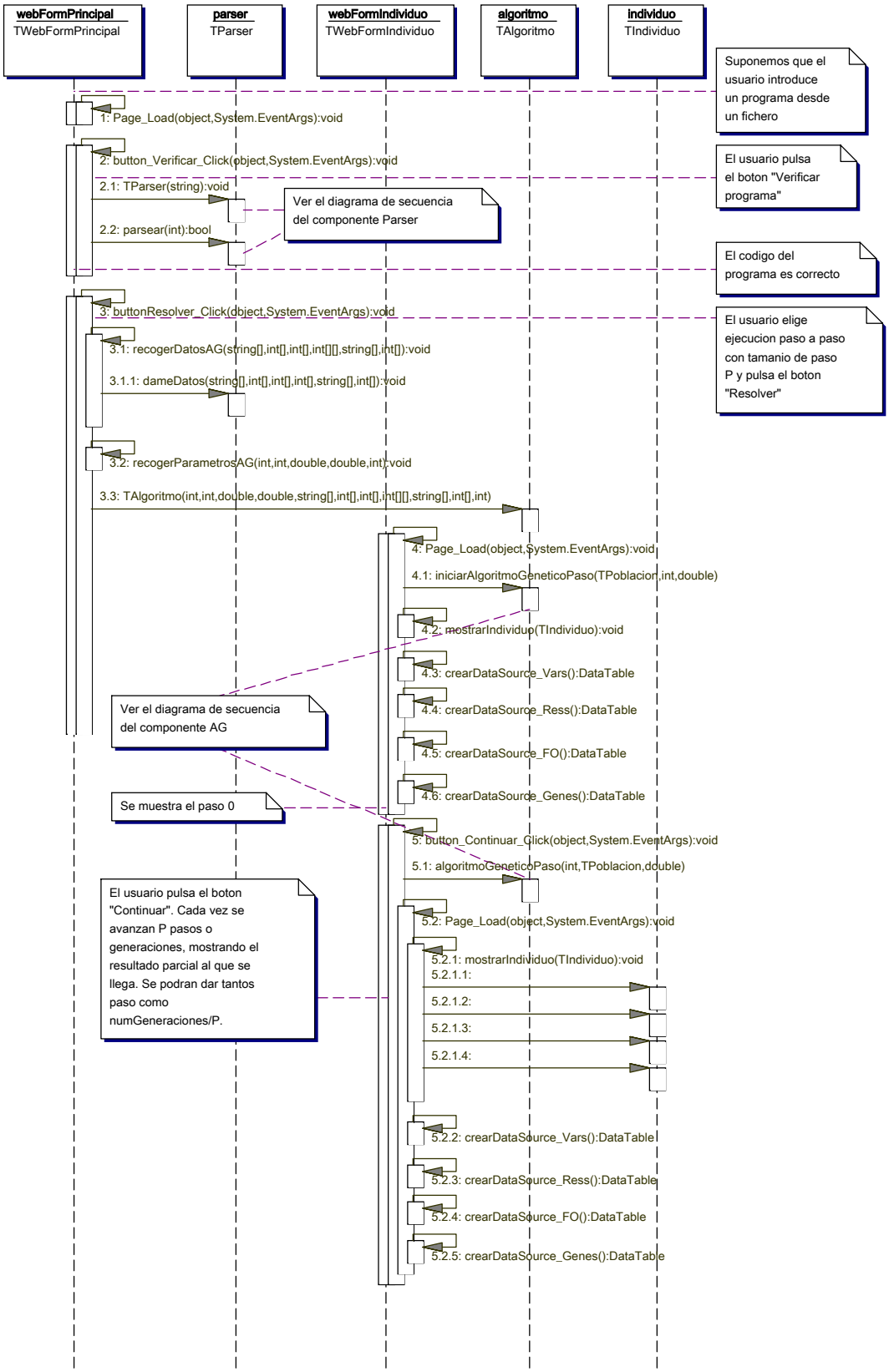


8.3.4.2.- Diagramas de secuencia

Este es el diagrama de secuencia UML que detalla el comportamiento del componente **interfazWebAG** durante una ejecución directa del algoritmo genético:

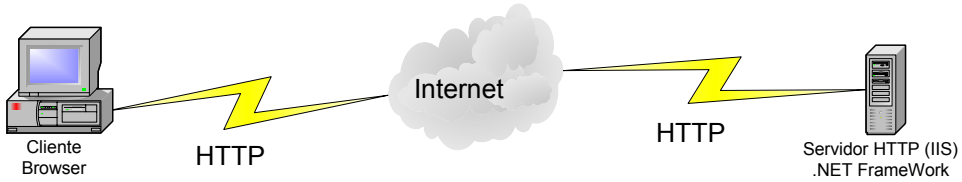


Este es el diagrama de secuencia UML que detalla el comportamiento del componente **interfazWebAG** durante una ejecución paso a paso del algoritmo genético:



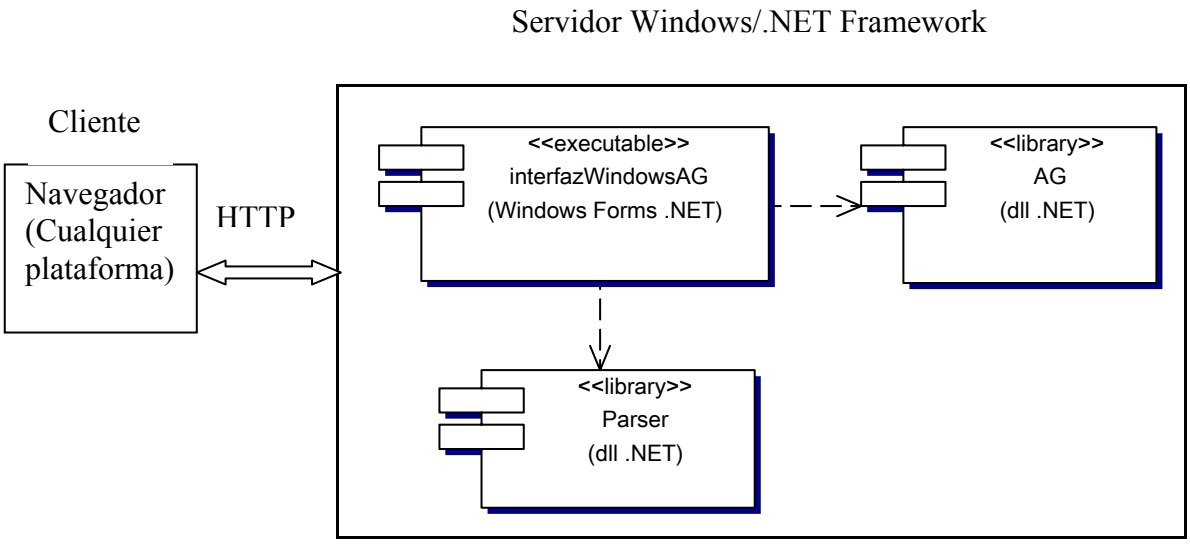
8.3.4.3.- Arquitectura

- Arquitectura física aplicación web



El cliente es un browser ejecutado en cualquier máquina con acceso a Internet. El servidor es un servidor http (Internet Information Server) que tiene instalado .NET Framework y donde residen todos los componentes de esta aplicación (interfazWebAG.exe, AG.dll y Parser.dll). El cliente se conecta a la URL de la página de inicio de la aplicación WebFormPrincipal.aspx. El servidor descarga la página inicial sobre el navegador, en la cual se introduce el problema de restricciones a resolver. Cuando el usuario solicita resolver, el navegador envía la información al servidor http, que resuelve el problema, y devuelve la página de resultados que se muestra en el browser.

- Arquitectura lógica aplicación web



8.3.5.- InterfazServicioWebAG

Es una aplicación distribuida para Windows con una interfaz rica construida con clases formulario de Windows Form de .NET que invoca a un Servicio Web de ASP .NET. InterfazServicioWebAG tiene una referencia web al componente ServicioWebAG y otra referencia al componente Parser

8.3.5.1.- Diagrama de clases

El componente **interfazServicioWebAG** esta formado por las siguientes clases implementadas en el lenguaje C# de Microsoft .NET y desarrolladas en Visual Studio .NET:

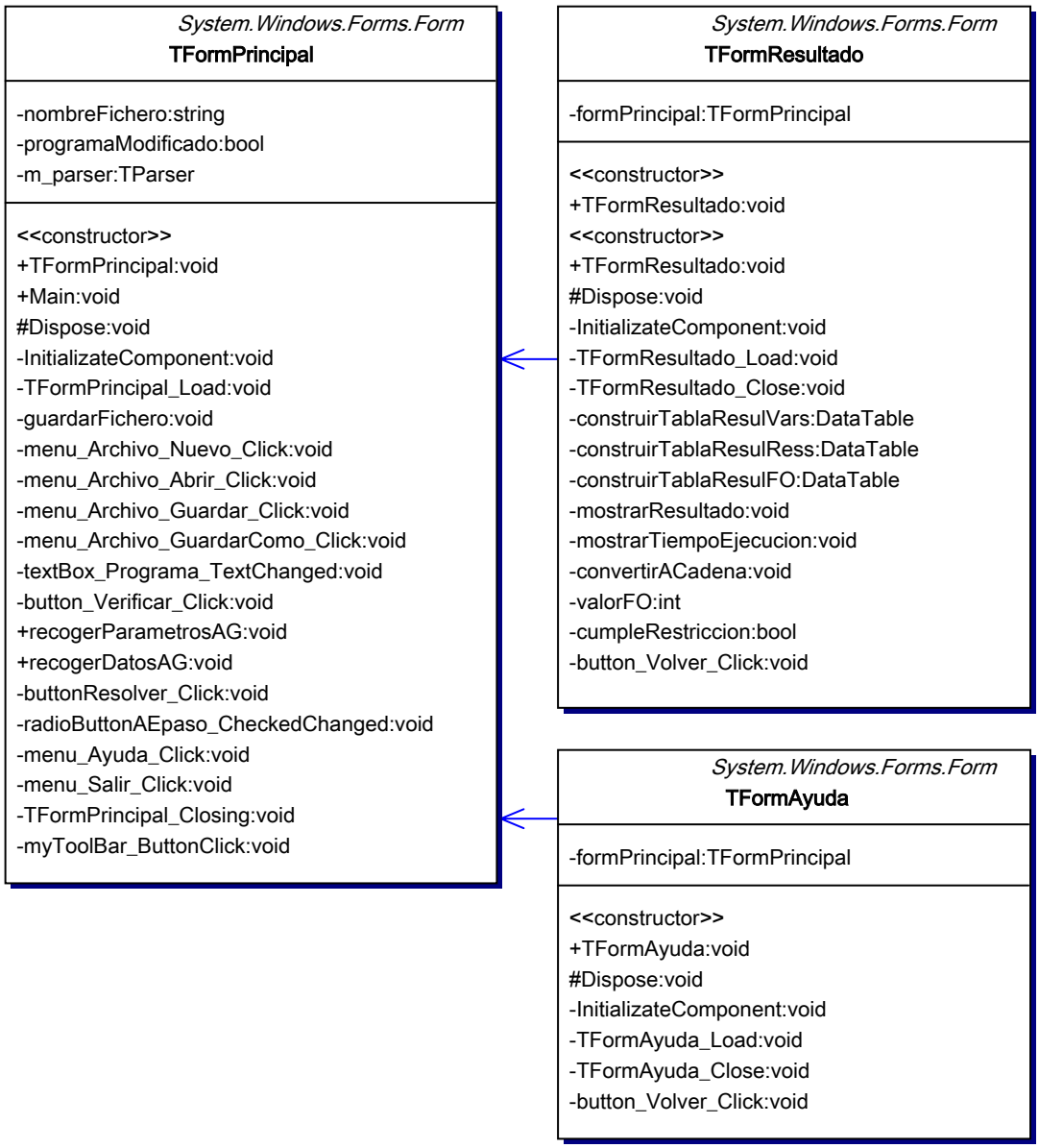
- **TFormPrincipal**: formulario windows que se muestra al lanzar la aplicación. En él el usuario puede editar o cargar directamente desde un fichero un programa escrito con el lenguaje *LRPR*, para seguidamente solicitar la verificación del código. Si el programa es correcto, entonces el usuario puede ordenar la resolución del problema mediante el algoritmo evolutivo, permitiendo únicamente elegir una ejecución directa. También permite modificar los valores por defecto de los parámetros del algoritmo evolutivo.

- **TFormResultado**: formulario windows que se muestra tras la solicitud de resolver directamente un problema de optimización representado por un programa verificado. En este formulario se presenta toda la información relativa al problema y a la solución encontrada por el algoritmo genético en forma de tablas. También muestra el tiempo empleado en la resolución del problema.

- **TFormAyuda**: formulario Windows que se muestra tras la solicitud de información por parte del usuario. Presenta la sintaxis del lenguaje LRPR utilizada para escribir los programas de restricciones.

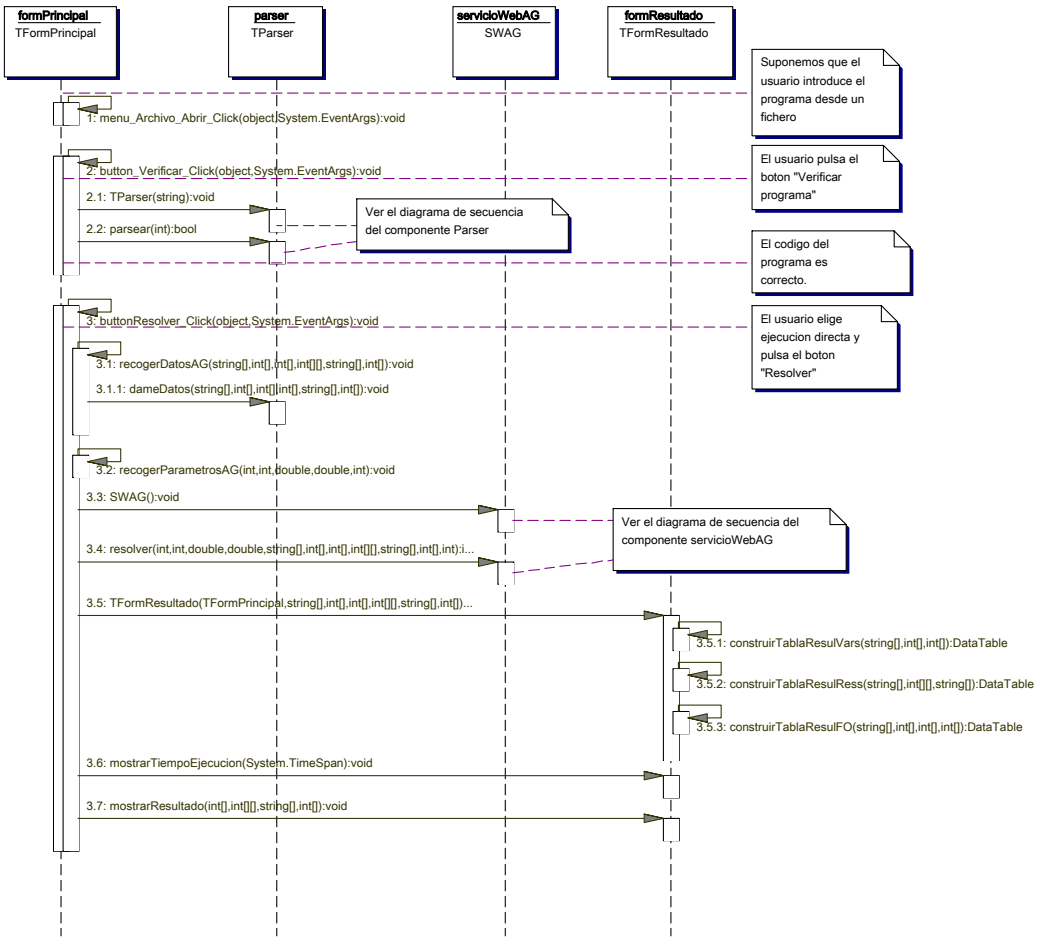
Estas clases son los elementos del proyecto de C# interfazServicioWebAG, que es una aplicación Windows cuyo archivo de resultado es interfazServicioWebAG.exe

Este es el diagrama de clases UML correspondiente al componente **interfazServicioWebAG**:



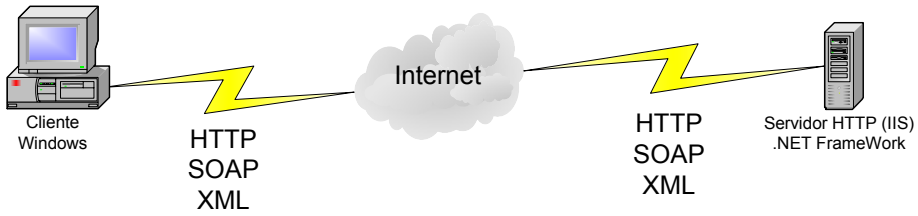
8.3.5.2.- Diagrama de secuencia

Este es el diagrama de secuencia UML que detalla el comportamiento del componente **interfazServicioWebAG** durante una ejecución directa del algoritmo genético:



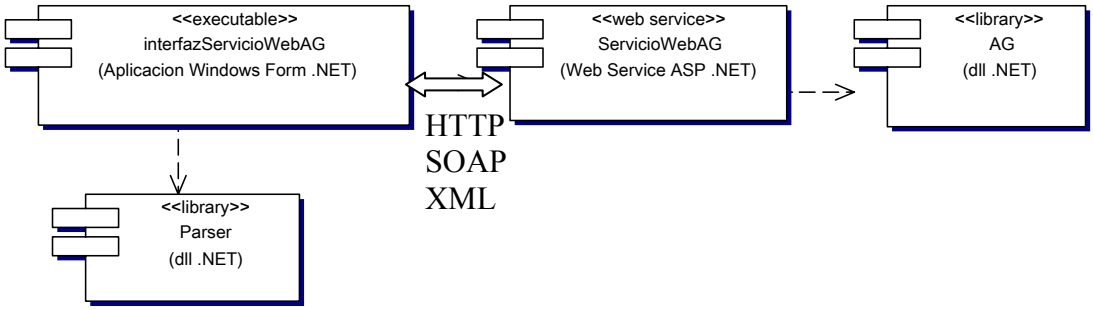
8.3.5.3.- Arquitectura

- Arquitectura física aplicación servicio web



El cliente es una aplicación para Windows, que establece la interfaz de usuario, y se ejecuta en una máquina remota con conexión a Internet. En el cliente se encuentra el componente InterfazServicioWebAG.dll y el componente Parser.dll. El servidor es un servidor http (Internet Information Server) que tiene instalado .NET Framework. El usuario introduce el problema de restricciones en el cliente y solicita resolver, se envía la petición al servicio web a través de Internet (http, soap, xml). El componente ServicioWebAG.dll y AG.dll se encuentran en el servidor. El servicio web referencia al componente AG que resuelve el problema. El servidor devuelve la solución al cliente mediante XML.

- Arquitectura lógica aplicación servicio web



8.3.6.- ServicioWebAG

Es un Servicio Web XML que recibe la llamada externa del componente InterfazServicioWebAG. El componente ServicioWebAG tiene una referencia al componente AG.

8.3.6.1.- Diagrama de clases

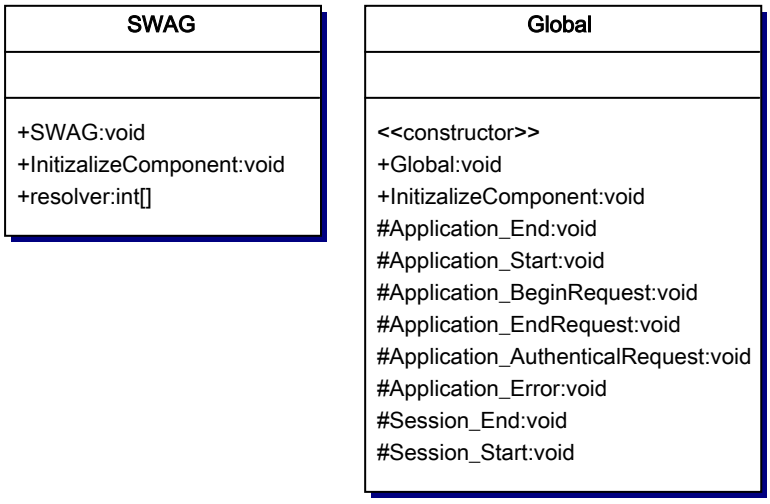
El componente **interfazServicioWebAG** esta formado por las siguientes clases implementadas en el lenguaje C# de Microsoft .NET y desarrolladas en Visual Studio .NET:

- **SWAG**: clase que implementa el servicio web mediante un método *resolver()* con atributo [WebMethod]. Este método recibe los parámetros del algoritmo genético y los datos del problema de restricciones, necesarios para crear una instancia de la clase TAlgoritmo y así poder invocar al método *algoritmoGenetico()* del componente AG. Éste último método devuelve la cadena de genes del mejor individuo tras la ejecución del algoritmo genético, que a su vez el método resolver() devuelve al componente que le haya invocado. En la aplicación servicio web, el componente que invoca al componente ServicioWebAG es intefazServicioWebAG.

- **Global**: clase que contiene código para responder a los eventos de la aplicación provocados por ASP.NET.

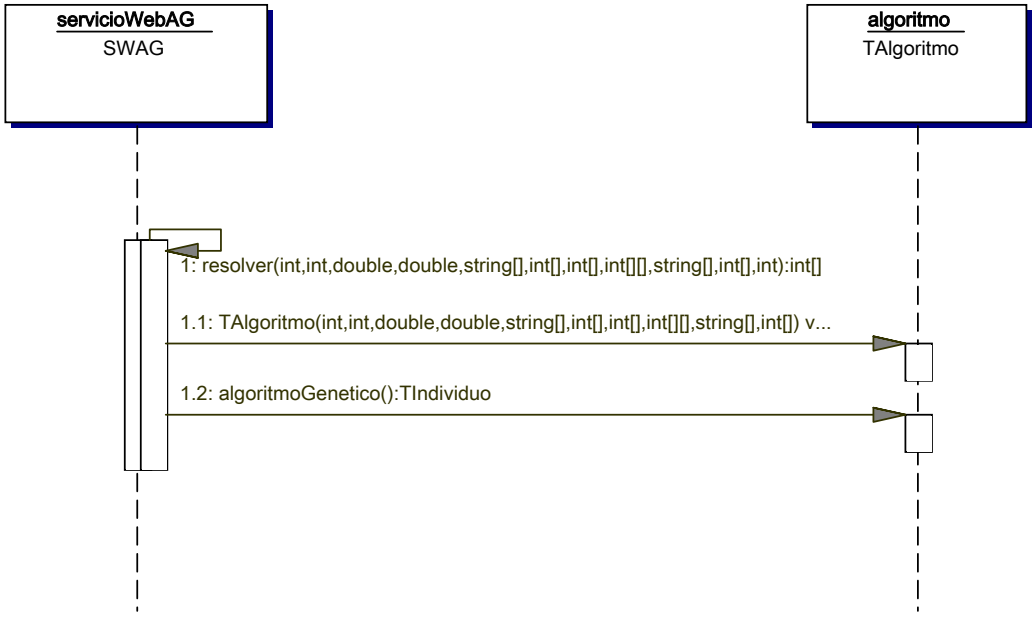
Estas clases son los elementos del proyecto de C# ServicioWebAG, que es un servicio Web cuyo archivo de resultado es ServicioWebAG.dll

Este es el diagrama de clases UML correspondiente al componente **interfazServicioWebAG**:



8.3.6.2.- Diagrama de secuencia

Este es el diagrama de secuencia UML que detalla el comportamiento del componente **ServicioWebAG** durante una ejecución directa del algoritmo genético:



9.- Utilización y ejemplos

9.1.- Optimización con Restricciones sobre Dominios Finitos

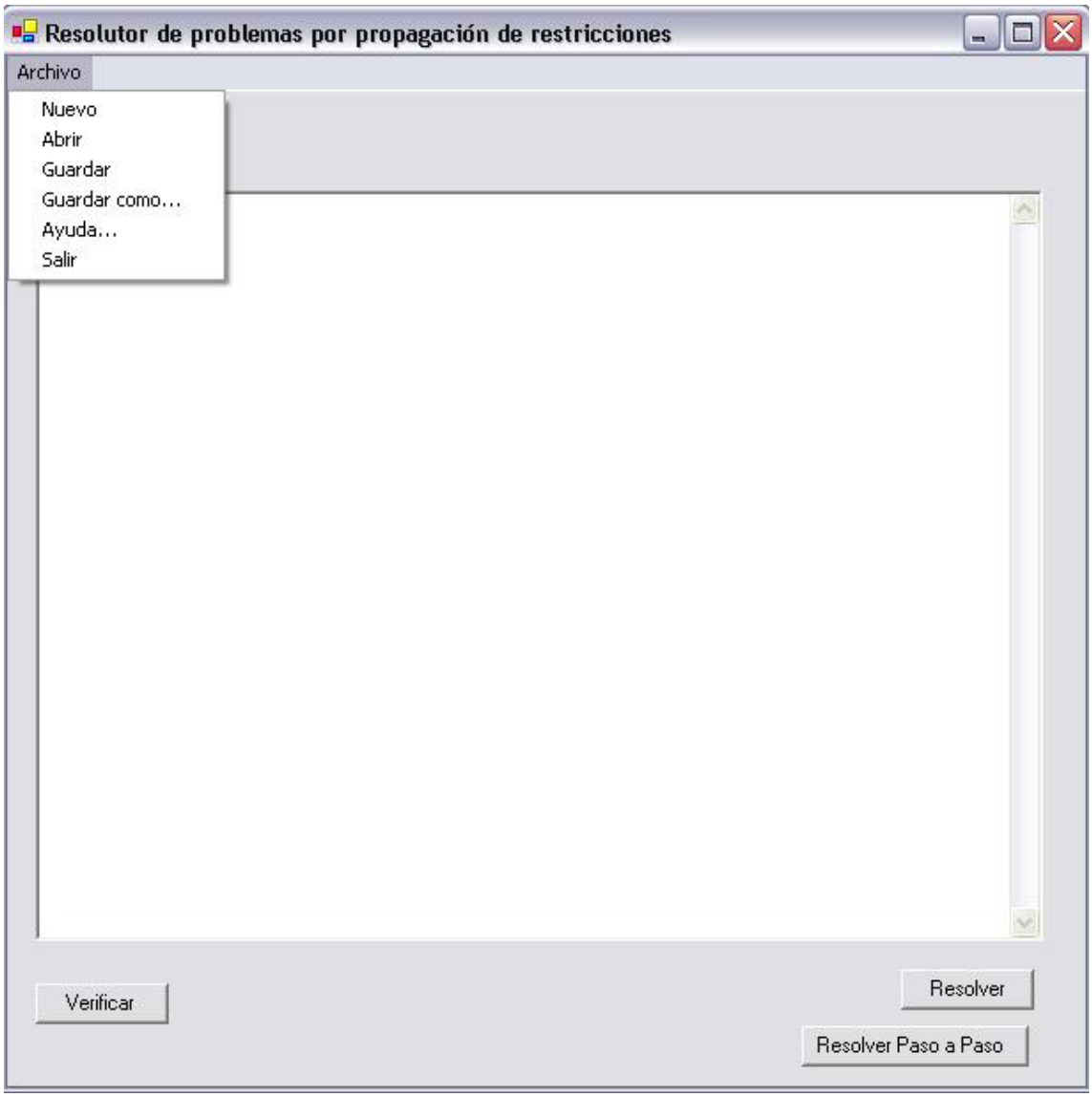
Veremos a continuación la forma de utilizar las herramientas diseñadas en el proyecto, a fin de que cualquier usuario pueda modelar problemas de restricciones sobre dominios finitos. Se dividirá en dos partes, a saber, los clientes nativos, tanto el que se ejecuta sobre el servidor, como el que se ejecuta en un cliente remoto con conexión a través de un Servicio Web (clienteWin32 y clienteWSWin32 respectivamente) y, el cliente que se ejecuta desde una máquina remota a través de un browser de Internet (clienteWebResolutor).

9.1.1.- Clientes nativos (*clienteWin32 y clienteWSWin32*)

Para ejecutar estos clientes, es necesario que la máquina donde se ejecuten tenga instalado el Framework de .NET y, para el caso de clienteWin32, debe existir el componente resolutor (*resolutor.dll*). La aplicación se llama *clienteWin32.exe, y* se ejecuta como cualquier fichero de estas características, haciendo doble clic sobre él. Para el caso de *clienteWSWin32*, además de que en la máquina donde se ejecute debe estar instalado el Framework de .NET, debe haber conexión a través de internet, al servidor donde resida el Servicio Web del resolutor. La aplicación se llama *clienteWSWin32.exe, y* se ejecuta como cualquier fichero de estas características, haciendo doble clic sobre él

El funcionamiento de ambos clientes, es exactamente igual, excepto porque desde el cliente que se conecta al Servicio Web, no se puede realizar la ejecución de la propagación inicial paso a paso, así que pasaremos a describir las partes comunes, y por último se explicará la ejecución de la propagación paso a paso.

La pantalla inicial es la siguiente:

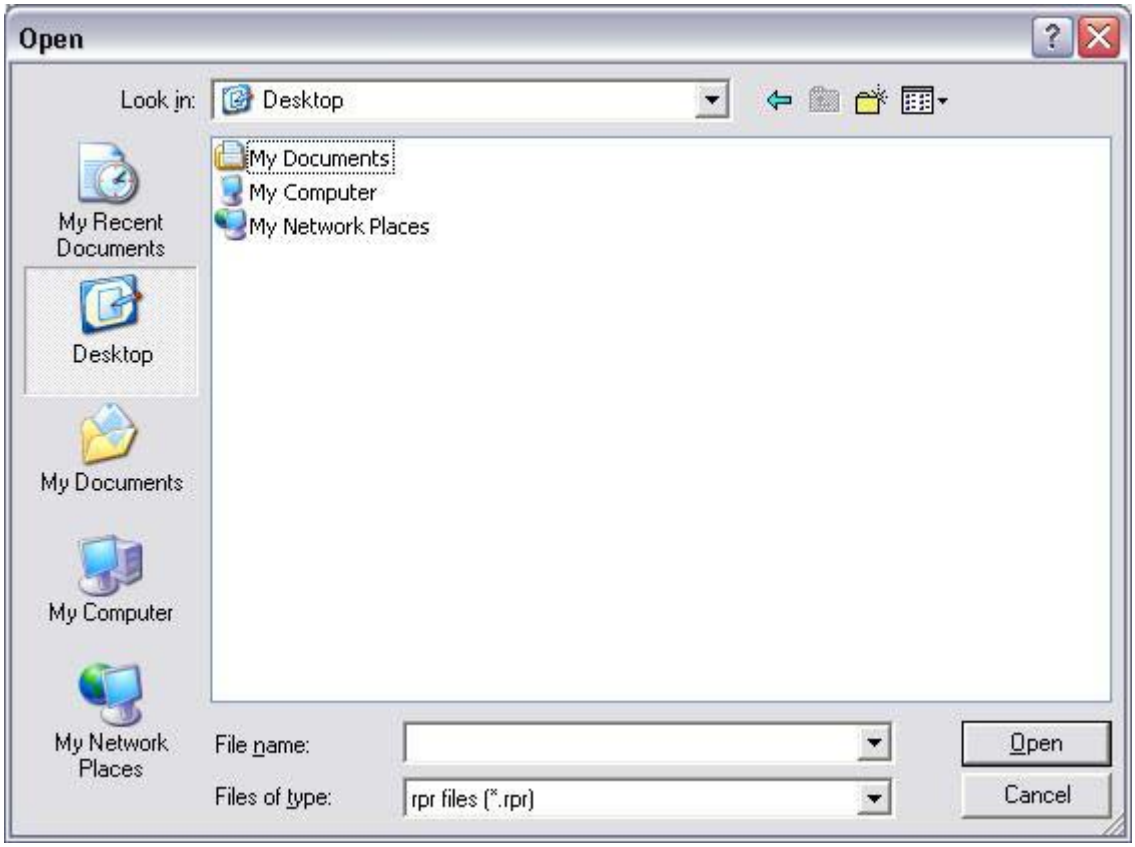


Como vemos, tiene un menú con las opciones de **Nuevo**, **Abrir**, **Guardar**, **Guardar como...** aplicadas a archivos, **Ayuda** y **Salir,** que explicaremos a continuación. Tiene también una zona para escribir el código fuente que describe el problema a resolver, código que se puede escribir a mano, o bien se puede cargar desde un archivo con extensión *.rpr.* Además hay un botón para verificar la sintaxis del código fuente, un botón para resolver el problema sin ninguna interrupción, y un botón sólo disponible en la versión que se ejecuta en el servidor (*clienteWin32*) para la ejecución con un paso intermedio en el que se puede observar el efecto de la propagación inicial de restricciones sobre los dominios de las variables.

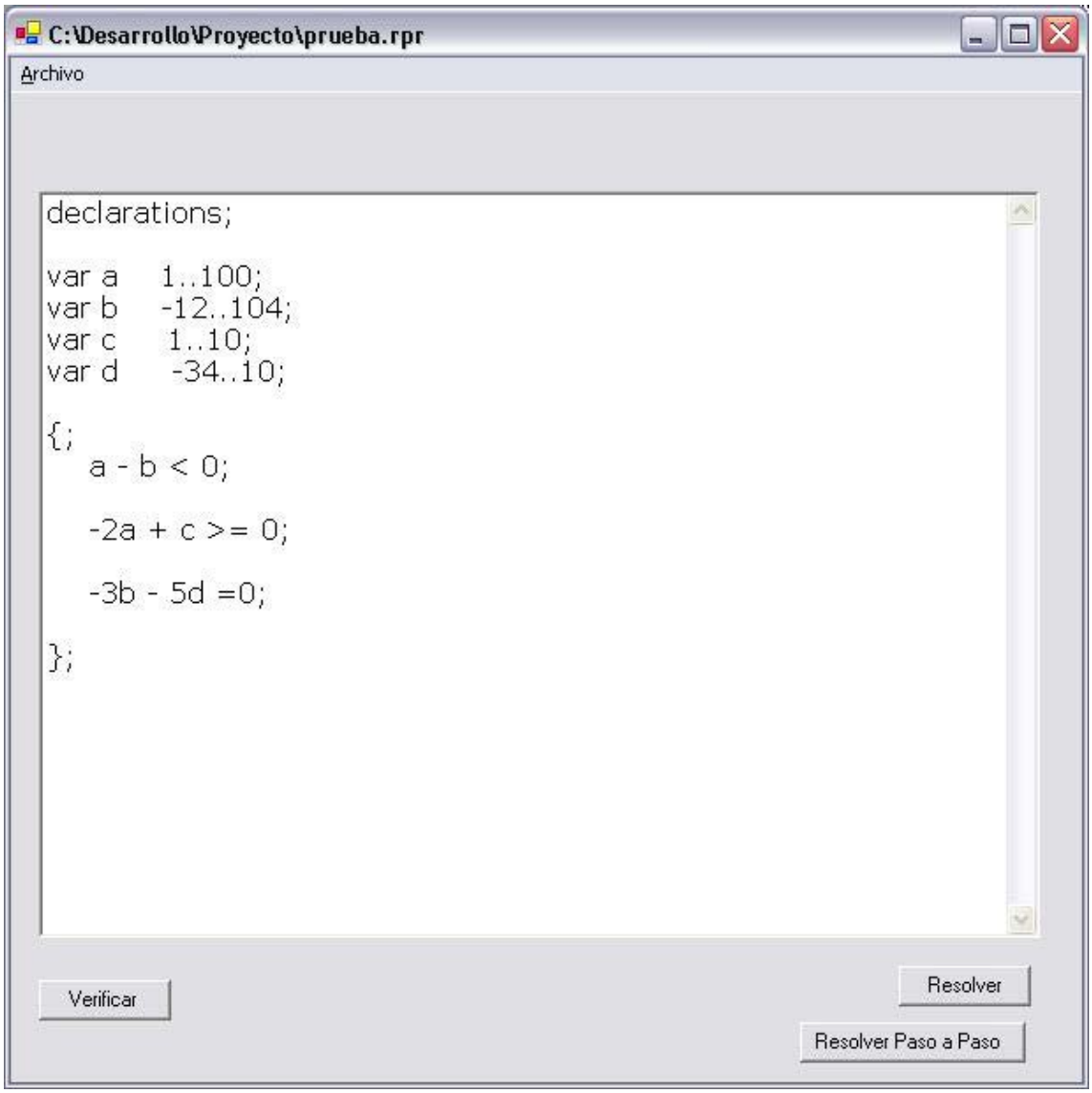
Si seleccionamos la opción **Nuevo** en el menú, la pantalla queda disponible para empezar a insertar código fuente del lenguaje *LRPR*, que describe el problema a solucionar:


Resolutor de problemas por propagación de restricciones
Archivo
Verificar
Resolver
Resolver Paso a Paso

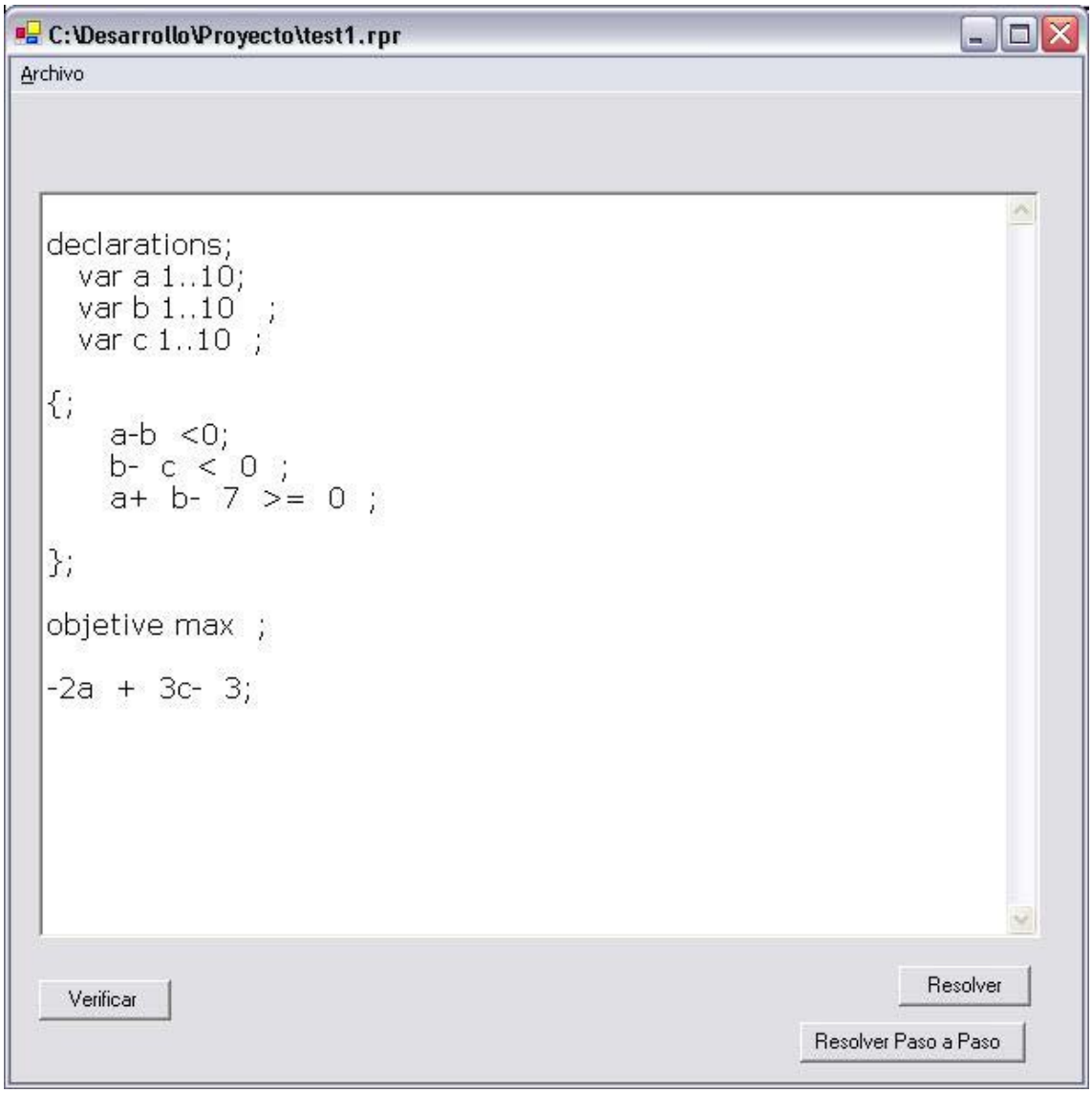
Si sobre el menú pulsamos la opción **Abrir**, aparece el cuadro de diálogo para seleccionar un archivo de extensión *.rpr*, y cargar su código fuente en el área de código de la pantalla. Lo observamos en la siguiente figura:



En la siguiente figura observamos como queda cargado en código fuente en el área de código de la pantalla:

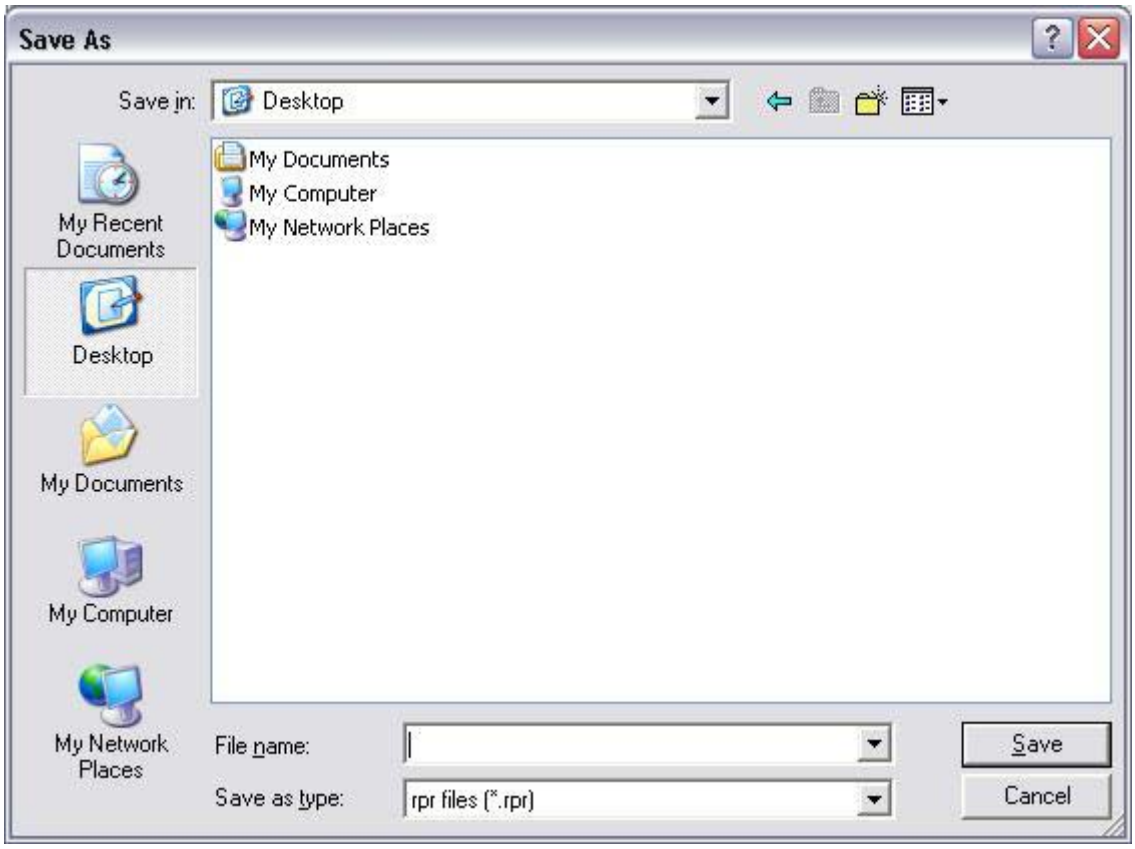


En este caso el problema no describe función objetivo, que es una expresión función de las variables del problema, y que queremos o bien maximizar, o bien minimizar. Así, de todas las soluciones que tiene el problema, sólo una maximiza o minimiza la función indicada en el problema. En la siguiente figura se muestra un problema que define una función objetivo de maximización, lo que significa que se le pide al sistema que devuelva la tupla de valores de las variables del problema, que estando dentro de los dominios iniciales, cumplan las restricciones y el valor de **-2a + 3c- 3** sea el máximo posible.

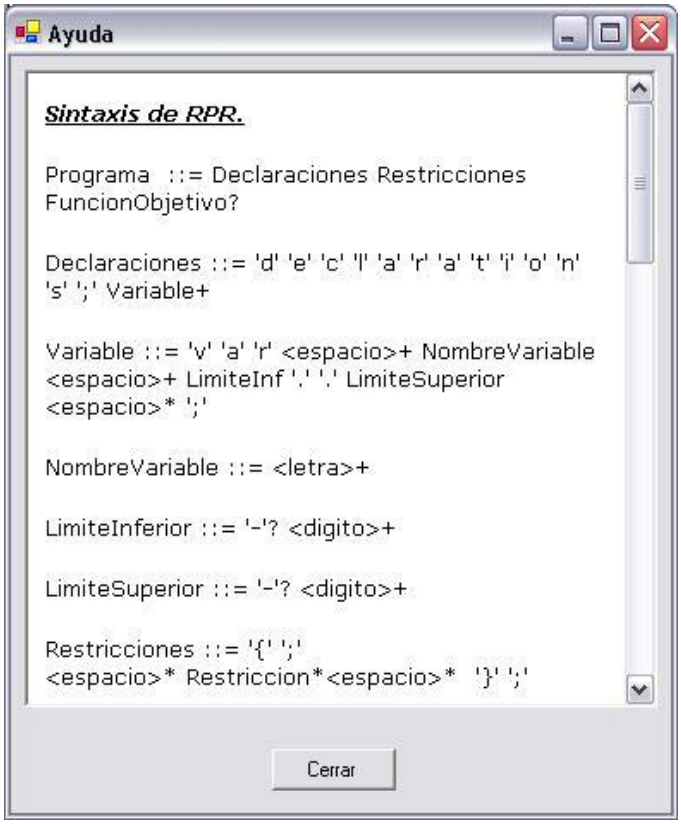


Al seleccionar la opción **Guardar** del menú, el fichero se guarda automáticamente sobrescribiendo el fichero que está actualmente abierto. En caso de que no haya ningún fichero abierto, el funcionamiento será el mismo que el de la opción **Guardar como...** que se explica a continuación.

La pulsación de la opción **Guardar como...**, provoca la apertura de un cuadro de diálogo mediante el cual, se selecciona la ubicación donde se desea almacenar el fichero con extensión *.rpr* que contendrá el código fuente escrito en el área de código en el momento de la pulsación de la opción. El cuadro de diálogo es el siguiente:

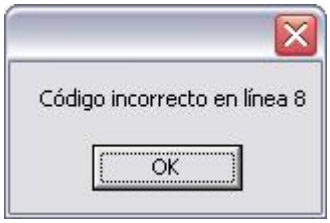


La opción **Ayuda** muestra una ventana con la sintaxis del lenguaje *LRPR* y un ejemplo de problema:

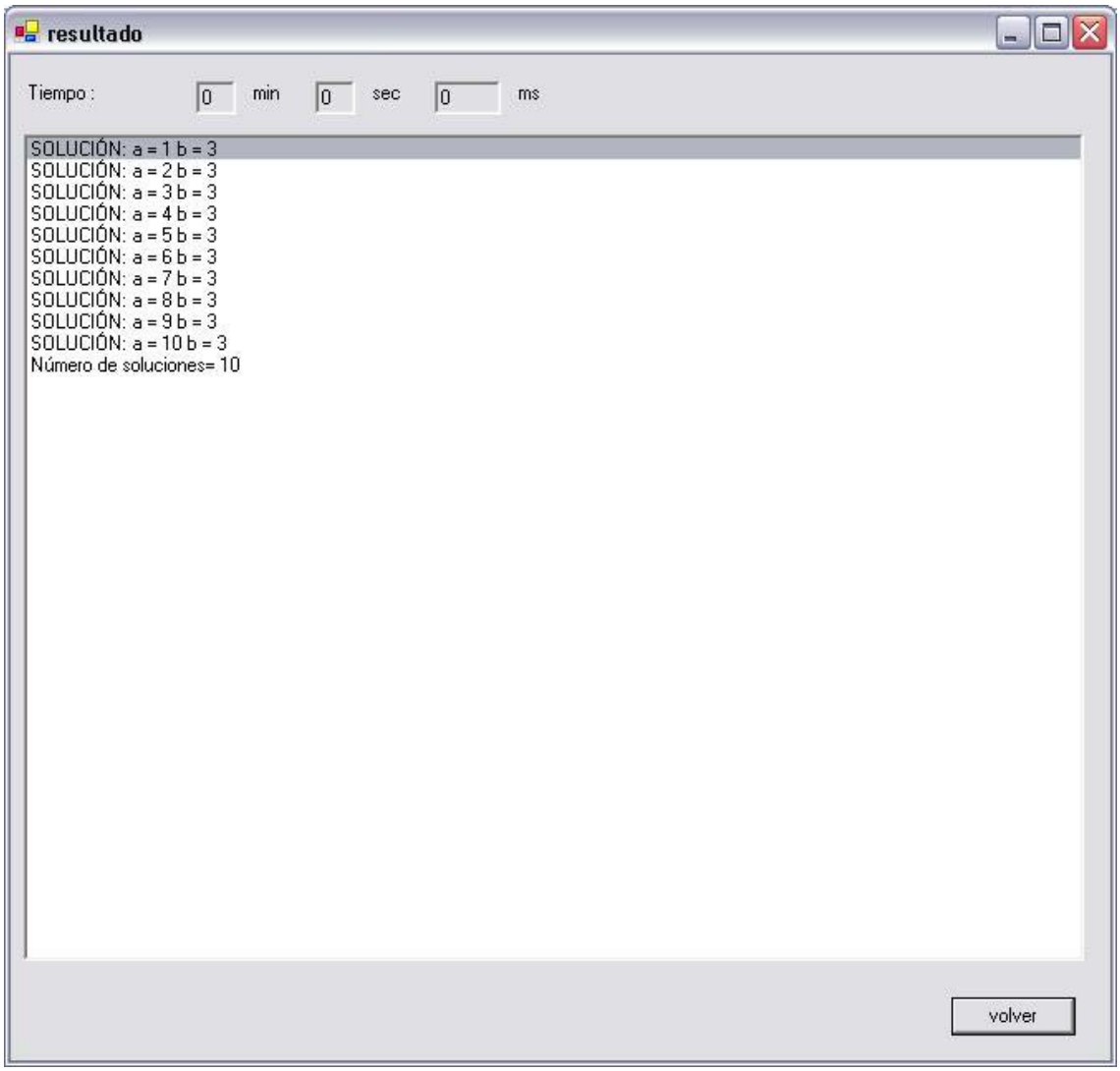


Si una vez que tenemos el código fuente sobre el área de código, pulsamos el botón **Verificar**, hay dos opciones, si es correcto, aparecerá el siguiente aviso:

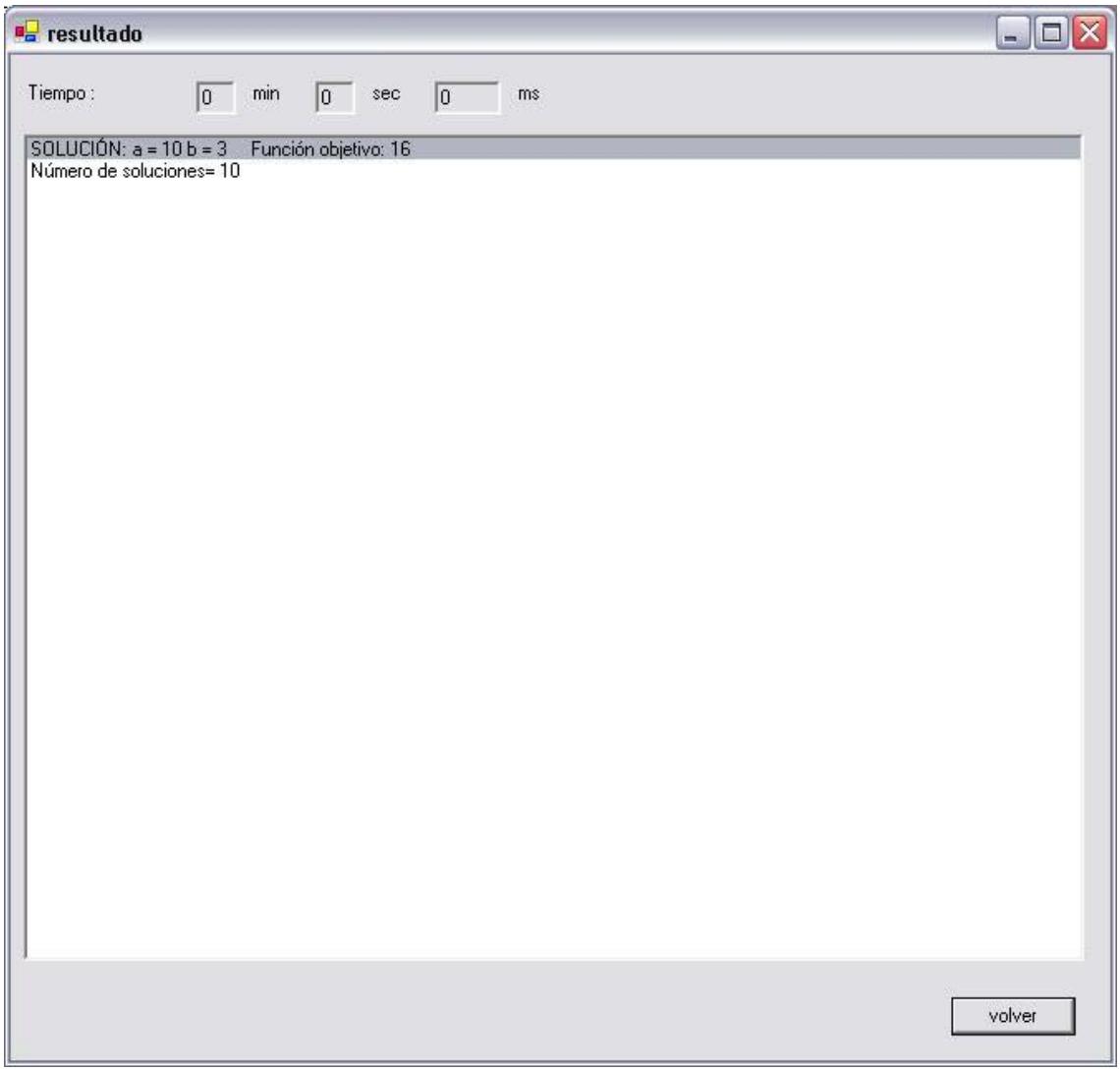
Y si es incorrecto, el sistema nos avisa, indicando el número de línea donde se encuentra el error:



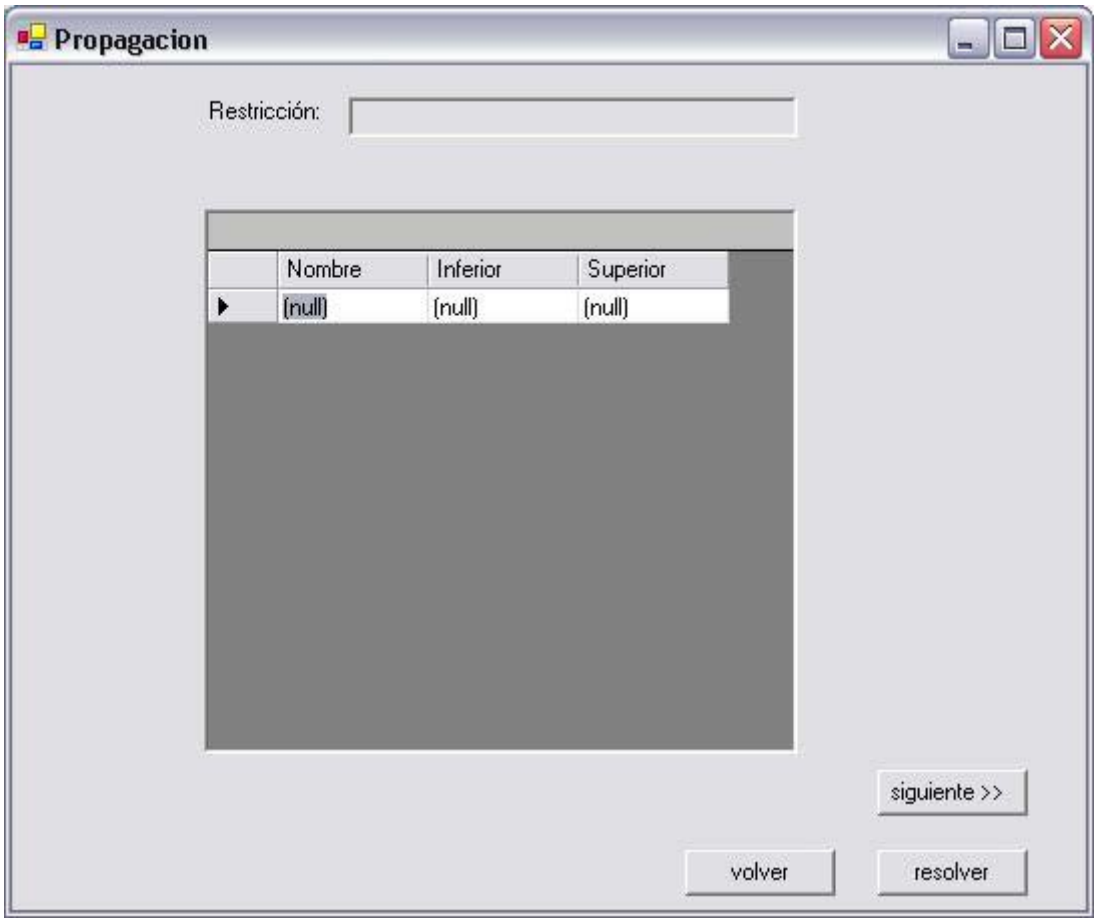
Después de la verificación de la sintaxis del código fuente, pasamos a la resolución del problema. Al pulsar el botón **Resolver**, se ejecutará el algoritmo que resuelve el problema, y si no se ha especificado función objetivo se mostrará una pantalla con la solución, el tiempo de ejecución y un botón para regresar a la pantalla inicial (**"Volver"**), la pantalla es la que sigue:



Si por el contrario se especificó una función objetivo, la solución se muestra en la misma pantalla, indicando sólo la solución que optimiza la función objetivo, y el valor de ésta:

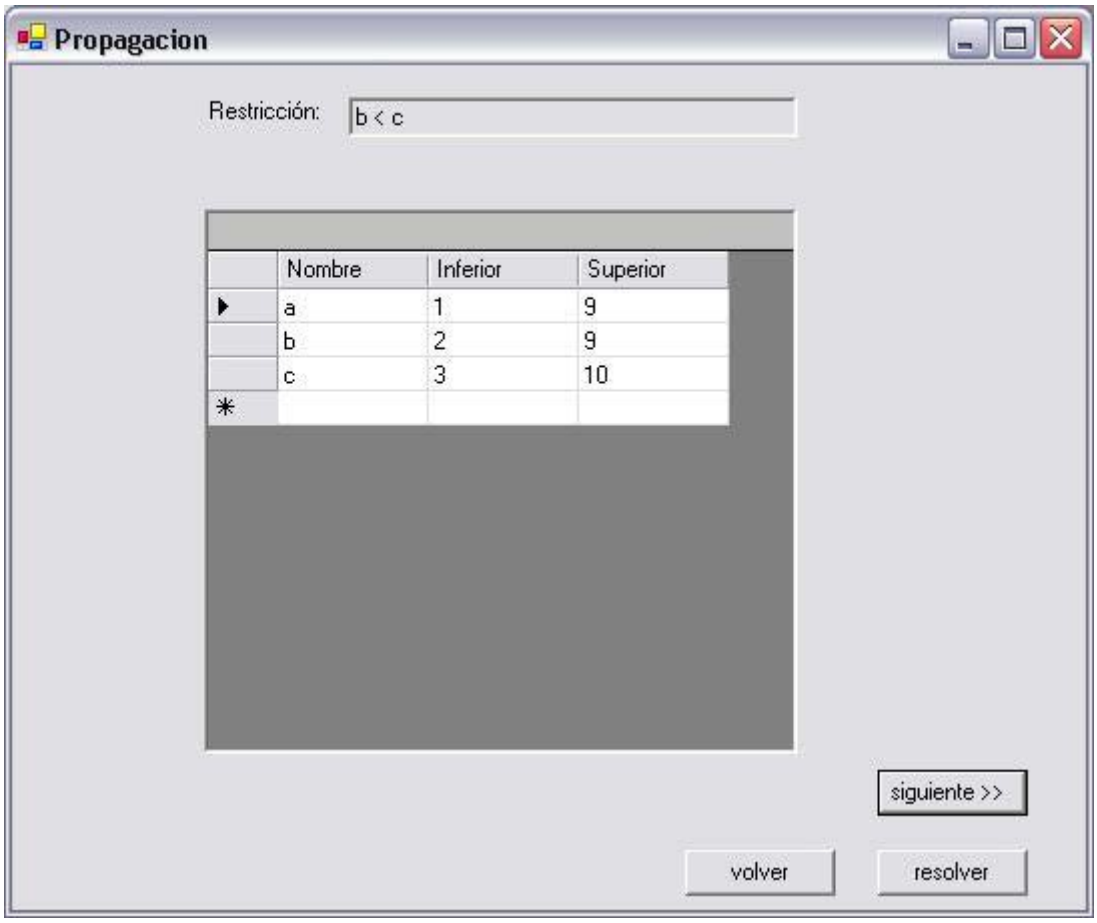


En el cliente que se ejecuta en el servidor (*clienteWin32*), existe la opción de **Resolver Paso a Paso**, que consiste en que antes de la fase de búsqueda de soluciones, se puede hacer un seguimiento de cómo evolucionan los dominios de las variables durante la propagación inicial de restricciones. Al pulsar el botón de **Resolver Paso a Paso**, se muestra la siguiente ventana:



Como podemos ver, esta ventana tiene los siguientes elementos, el campo **Restricción** en el que se muestra la restricción que provoca el cambio de los dominios, el área de dominios de las variables, el botón **Siguiente**, que sirve para ejecutar un paso de propagación (aplicar una restricción a los dominios), el botón **Resolver**, que resuelve el problema mostrando las ventanas vistas anteriormente, y el botón **Volver**, que vuelve a la pantalla inicial.

Si se pulsa el botón **Siguiente**, observamos cómo se muestran tanto la restricción que se propaga, como los valores de los dominios, en la siguiente ventana:



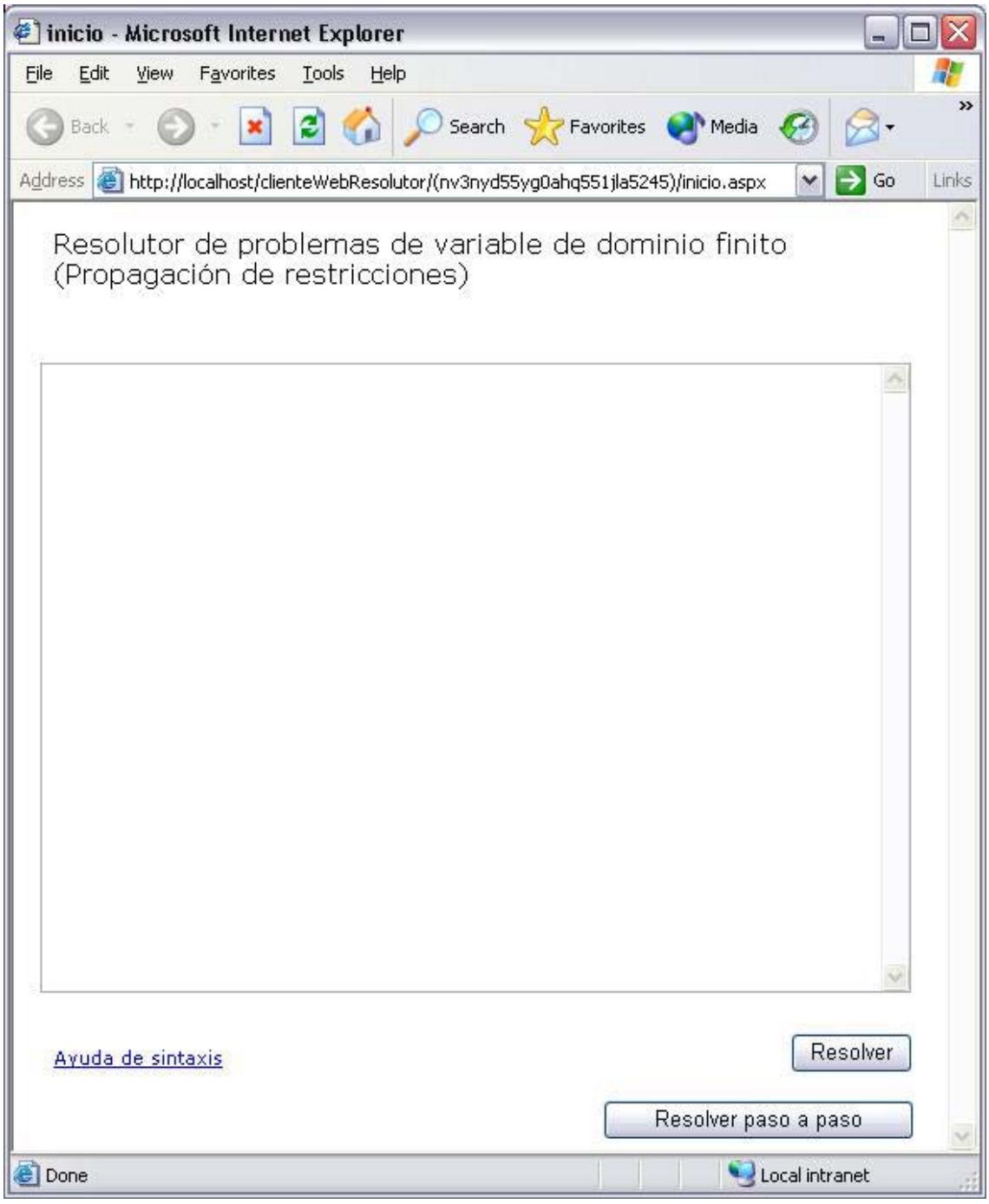
Vemos cómo han quedado los dominios después de la propagación de la restricción **b<c**.

9.1.2.- Cliente Web *(clienteWebResolutor)*

Para poder utilizar este cliente, sólo es necesario tener un navegador en una máquina conectada a Internet, y acceder a la URL *http://servidor/clienteWebResolutor/inicio.aspx*, donde *servidor* será el servidor Web IIS (Internet Information Server) donde resida la aplicación Web *clienteWebResolutor.*

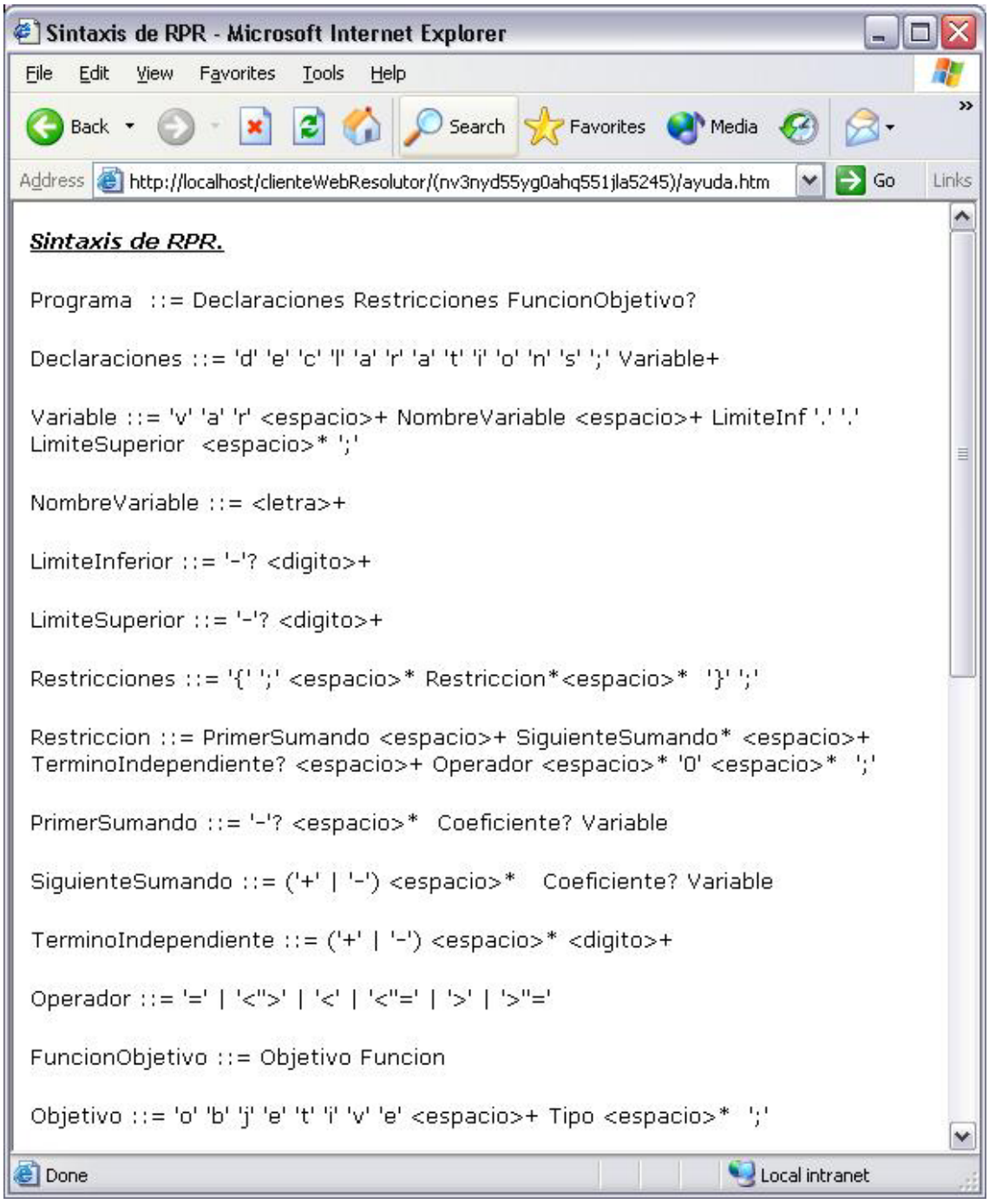
La forma de utilizar este cliente conceptualmente es igual a la de los anteriores, la única diferencia, radica en el interfaz de usuario, que al ser HTML, es más pobre que el de los clientes nativos. Pasamos a explicarlo.

La ventana de inicio es la siguiente:

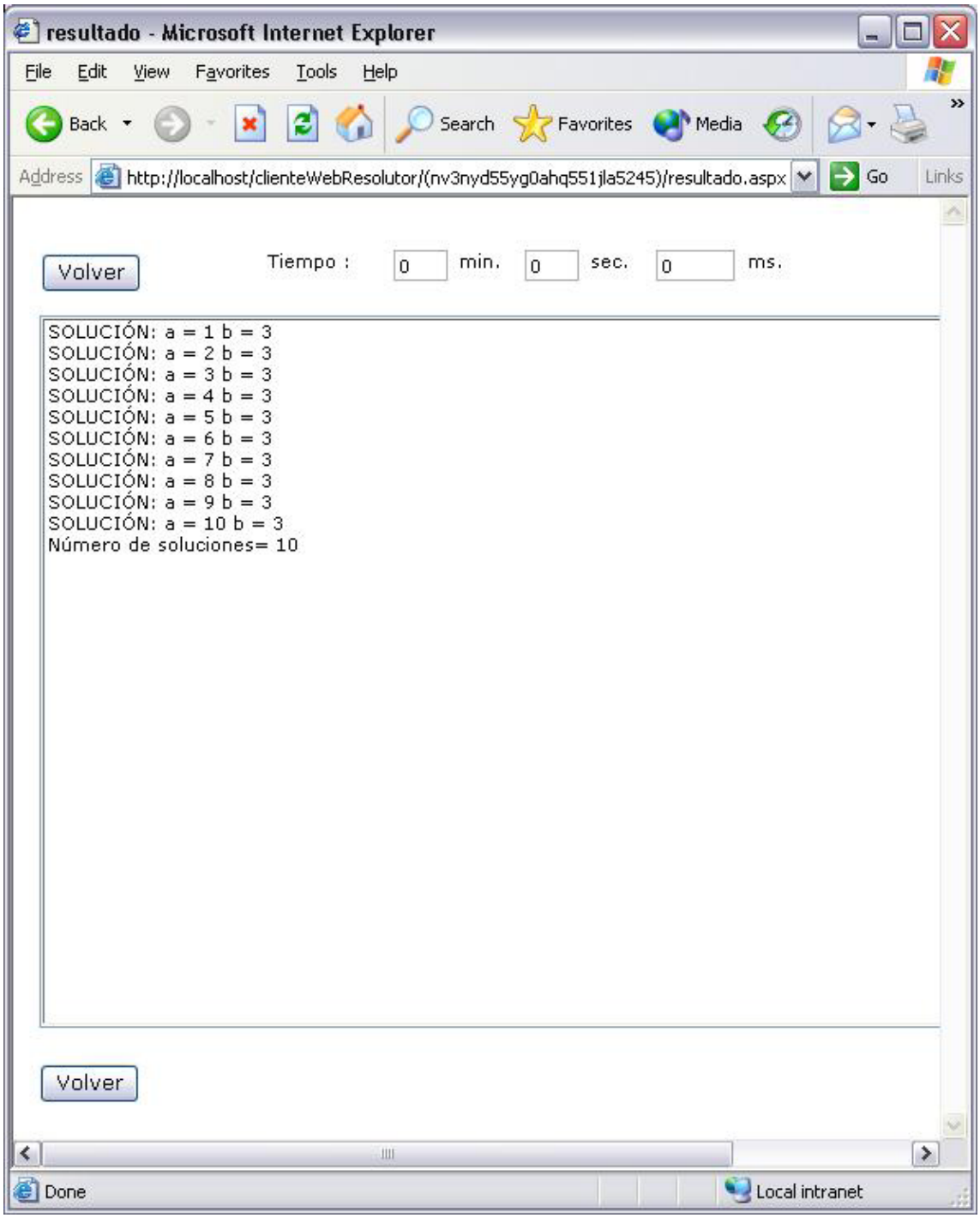


Como podemos observar, no hay menú, y por tanto para la introducción de código fuente en el área de código, es necesario escribirlo manualmente o bien hacer "Copy & Paste" desde un fichero externo en el que previamente habremos guardado el código. Hay un enlace **Ayuda de sintaxis** Además están disponibles los botones **Resolver** y **Resolver Paso A Paso**.

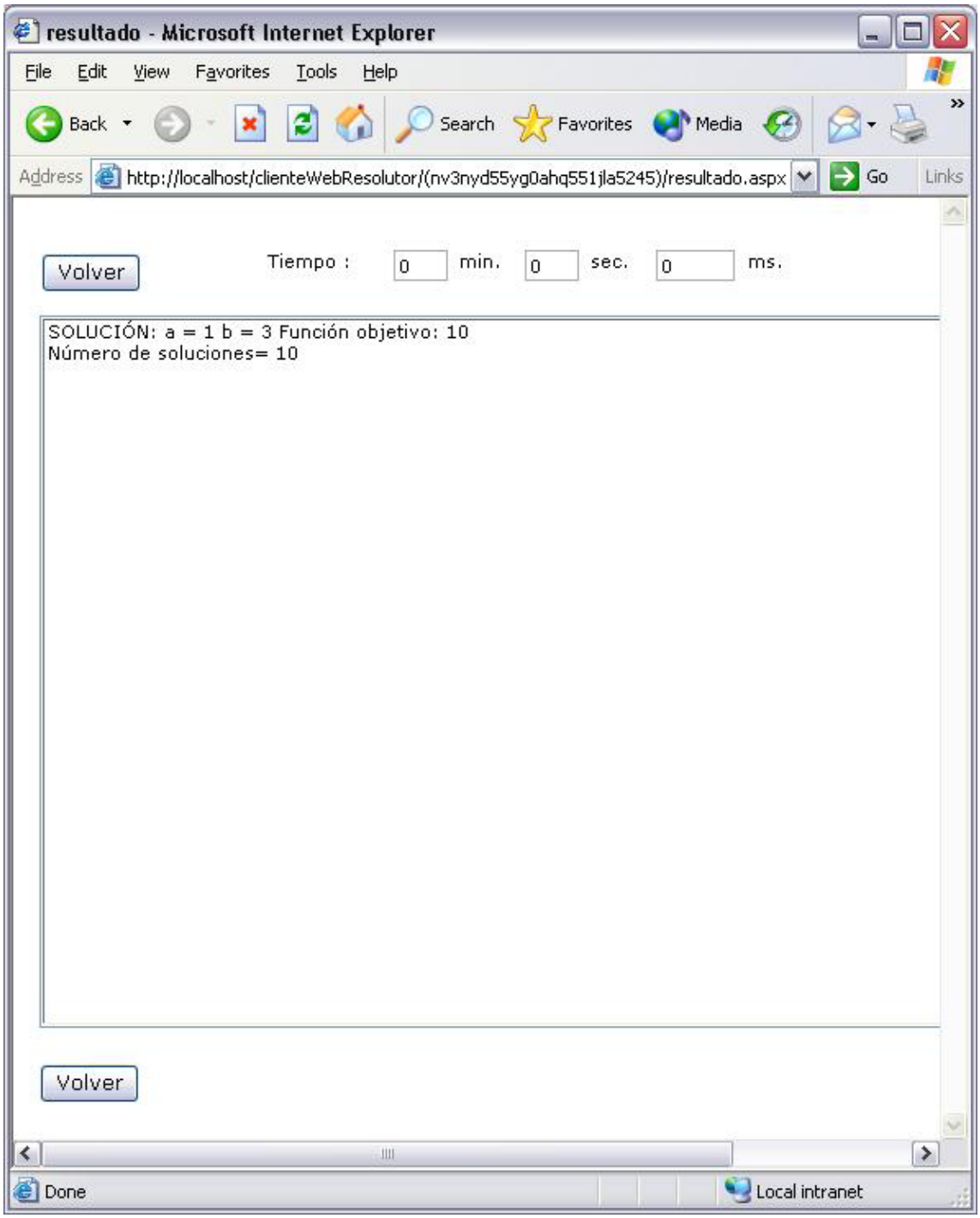
Al pulsar el enlace **Ayuda de sintaxis**, se muestra la ventana siguiente con la notación *EBNF* del lenguaje *LRPR*, un ejemplo de problema, y un enlace para volver a la pantalla inicial, como se muestra a continuación:



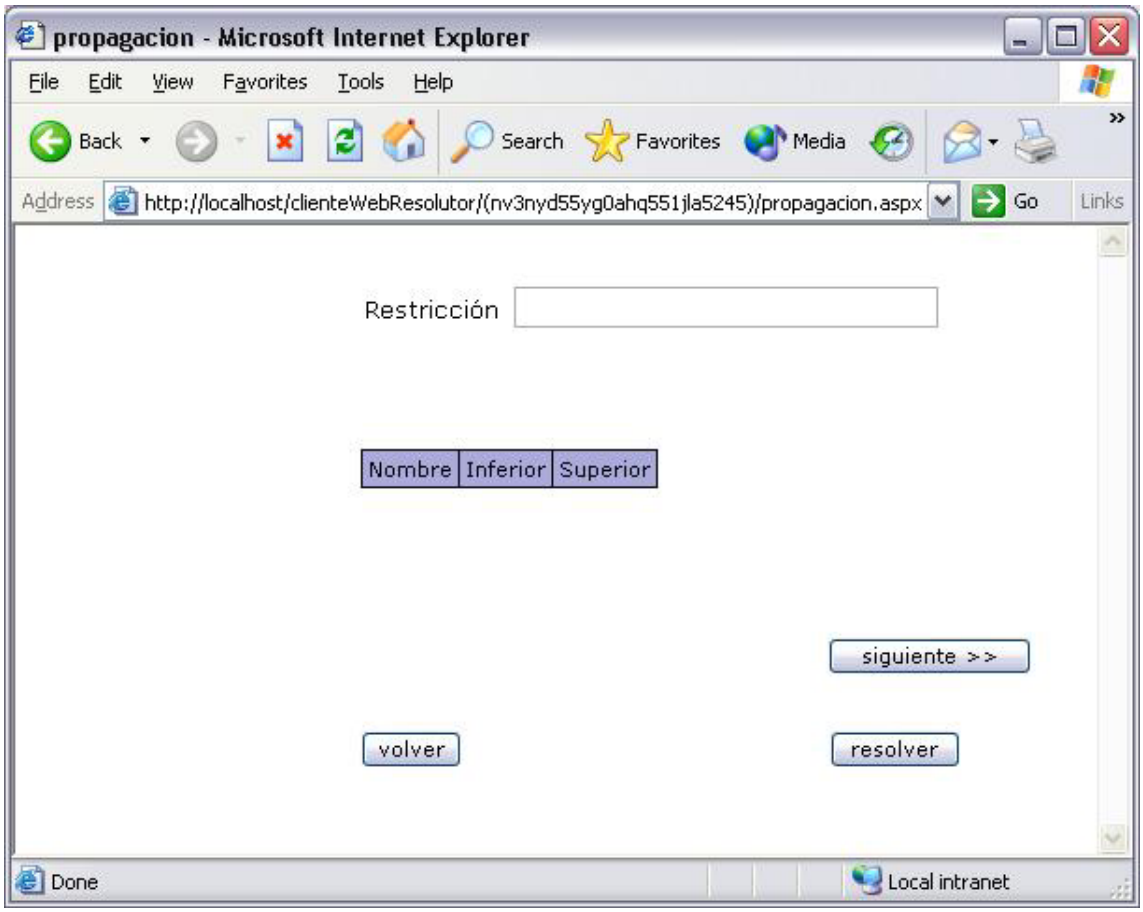
Al pulsar el botón **Resolver**, se ejecutará el algoritmo que resuelve el problema, y si no se ha especificado función objetivo se mostrará una pantalla con la solución, el tiempo de ejecución y un enlace para regresar a la pantalla inicial (**"Volver"**), la pantalla es la que sigue:



Si por el contrario se especificó una función objetivo, la solución se muestra en la misma pantalla, indicando sólo la solución que optimiza la función objetivo, y el valor de ésta:

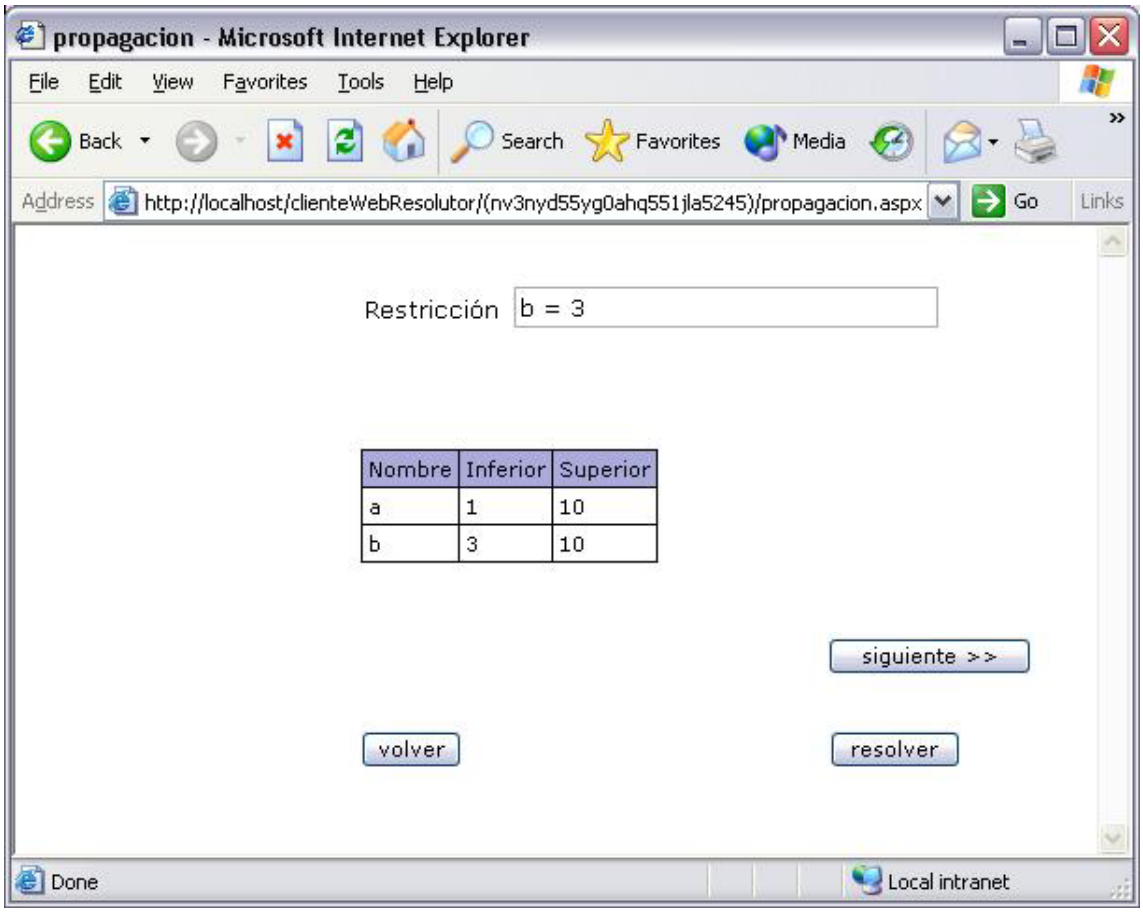


Al pulsar el botón de **Resolver Paso a Paso**, para observar el cambio de los dominios en la propagación inicial, se muestra la siguiente ventana:



Como podemos ver, esta ventana tiene los siguientes elementos, el campo **Restricción** en el que se muestra la restricción que provoca el cambio de los dominios, el área de dominios de las variables, el botón **Siguiente**, que sirve para ejecutar un paso de propagación (aplicar una restricción a los dominios), el botón **Resolver**, que resuelve el problema mostrando las ventanas vistas anteriormente, y el botón **Volver**, que vuelve a la pantalla inicial.

Si se pulsa el botón **Siguiente**, observamos cómo se muestran tanto la restricción que se propaga, como los valores de los dominios, en la siguiente ventana:



Vemos cómo han quedado los dominios después de la propagación de la restricción **b=3**.

Como conclusión decir, que si se dispone del framework de .NET instalado en la máquina, es preferible ejecutar el *clienteWSWin32* ya que se tendrán disponibles las ventajas del interfaz de usuario nativo que es mucho más rico y amigable como hemos podido observar, y tendremos las ventajas de la ejecución del algoritmo en un servidor remoto mediante una petición a un Servicio Web, que descargará la máquina local del trabajo de la resolución del problema.

9.2.- Ramificación y poda con Backtracking

Una vez vista la arquitectura de la aplicación y su división en componentes y clases, vamos a explicar el modo de funcionamiento.

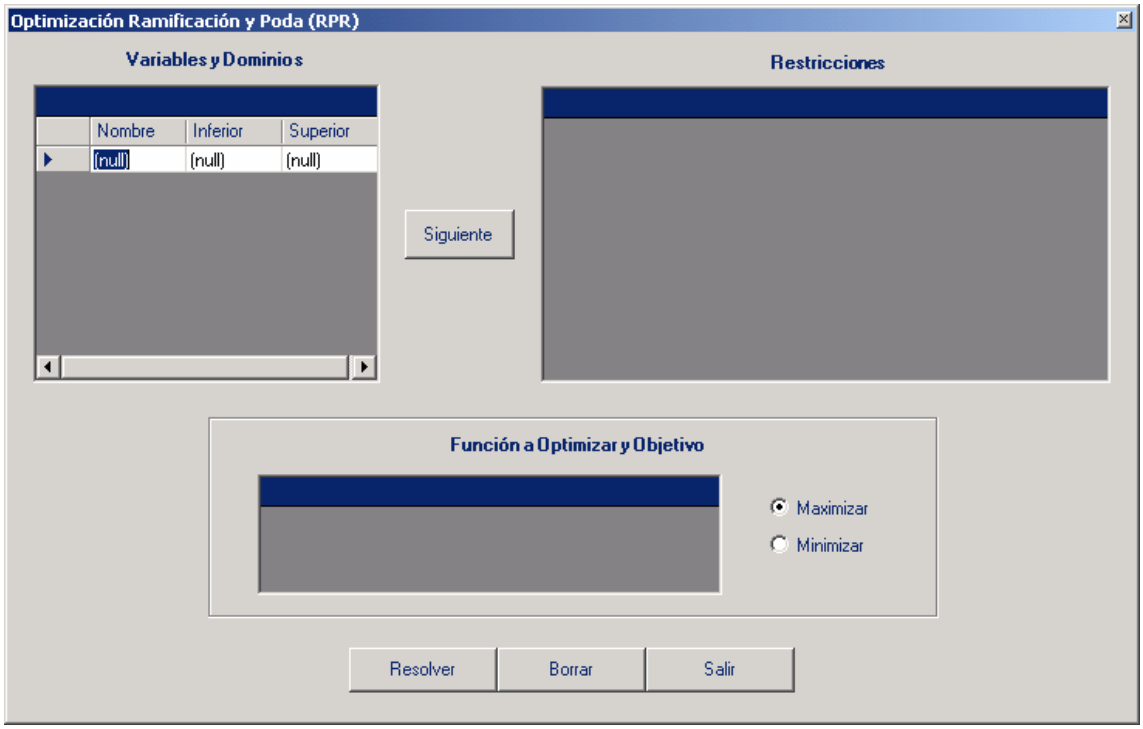
La visualización de la ejecución de la aplicación se llevará a cabo de dos modos diferentes según que parte de la implementación estemos tratando. En primer lugar, se muestra la visualización de la aplicación para el caso del cliente Win32 o el servicio Web y, en segundo lugar, la visualización para el caso de la aplicación ASP (en cuanto a la parte del Algoritmo de Ramificación y Poda con Backtracking se refiere).

Para ejecutar el cliente Win32 o el servicio Web, será necesario que la máquina donde se ejecuten tenga instalado el Framework de .NET y, para el caso de cliente Win32, debe existir el componente Resolutor (*ResolutorRPR.dll*). La aplicación se llama C*liente.exe, y* se ejecuta haciendo doble clic sobre él.
Para el caso del servicio Web, además del Framework de .NET, debe haber conexión a través de Internet, al servidor donde resida el Servicio Web del resolutor.

El funcionamiento de ambos clientes es el siguiente:

9.2.1.- Cliente Win32 / WSResolutorRPR

Una vez ejecutado el archivo *Cliente.exe* nos aparece la siguiente pantalla:



Como se puede ver, se distinguen 4 zonas fundamentalmente que son:

1.- Grid para las variables del problema y sus dominios,
2.- Grid para la introducción de las restricciones del problema,
3.- Panel inferior para los coeficientes de la función a optimizar y para la selección del objetivo de la optimización,
4.- Botonera inferior con las acciones que puede llevar a cabo el usuario

Como iremos viendo a continuación, cada una de estas zonas irá variando según el momento de la ejecución en el que nos encontremos.

- Selección de variables y dominios

En esta primera zona de la pantalla es en la que el usuario de la aplicación deberá introducir el nombre de las variables implicadas en el problema a resolver junto con los límites inferior y superior de su dominio correspondiente.

El nombre de cada variable puede ser cualquier cadena de caracteres (siempre distinto del espacio en blando). Los límites inferior y superior deben pertenecer a los números enteros.

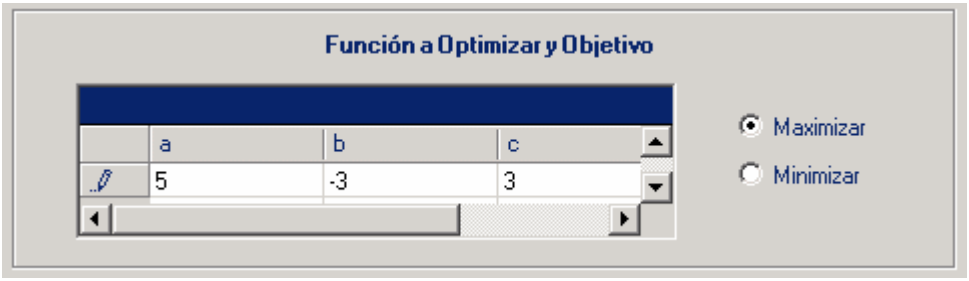
Una vez que el usuario a introducido todas las variables y dominios con los que desea que sea resuelto el problema deberá hacer clic sobre el botón *Siguiente* para que el grid de restricciones y el correspondiente a la función de optimización se completen con el nombre de dichas variables de forma que el usuario sólo tenga que introducir sus coeficientes.

- Introducción de las restricciones

Una vez actualizado el grid de restricciones, el usuario deberá introducir los coeficientes de cada variable para todas las restricciones que la solución debe de cumplir. Además, para cada una de estas restricciones, deberá introducir el término independiente teniendo en cuenta que inicialmente se encuentra situado a la izquierda del operador, es decir, que será cambiado de signo al pasarlo a la derecha del mismo. Y, por último, el usuario deberá introducir el operador para cada una de dichas restricciones.

- Introducción de la función objetivo y del sentido de la optimización

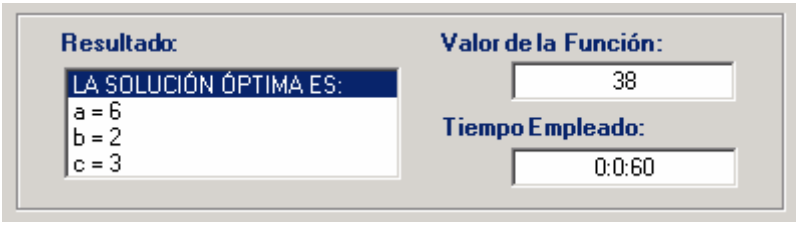
Además de actualizarse el grid de restricciones, tras hacer clic sobre el botón *Siguiente,* se actualiza también el grid correspondiente a la función de optimización en el que el usuario deberá de introducir los coeficientes y el término independiente (también situado a la izquierda) de la función a optimizar por el problema (una única línea de dicho grid).



Una vez introducidos estos coeficientes podrá seleccionar el sentido de la optimización, es decir, si desea que la solución óptima seleccionada finalmente *Maximice* o *Minimice* la función introducida. Por defecto, el sentido de la optimización es el de Maximizar la función.

➢ Obtención del resultado

Finalmente, tras haber introducido todos los datos, el usuario deberá hacer clic sobre el botón *Resolver* (situado en la botonera inferior) para obtener la solución óptima, en el caso de que exista, o un mensaje indicando que no hay solución. Al hacer clic sobre este botón, la pantalla inicial se amplía de modo que aparece una zona inferior en la que se puede ver lo siguiente:

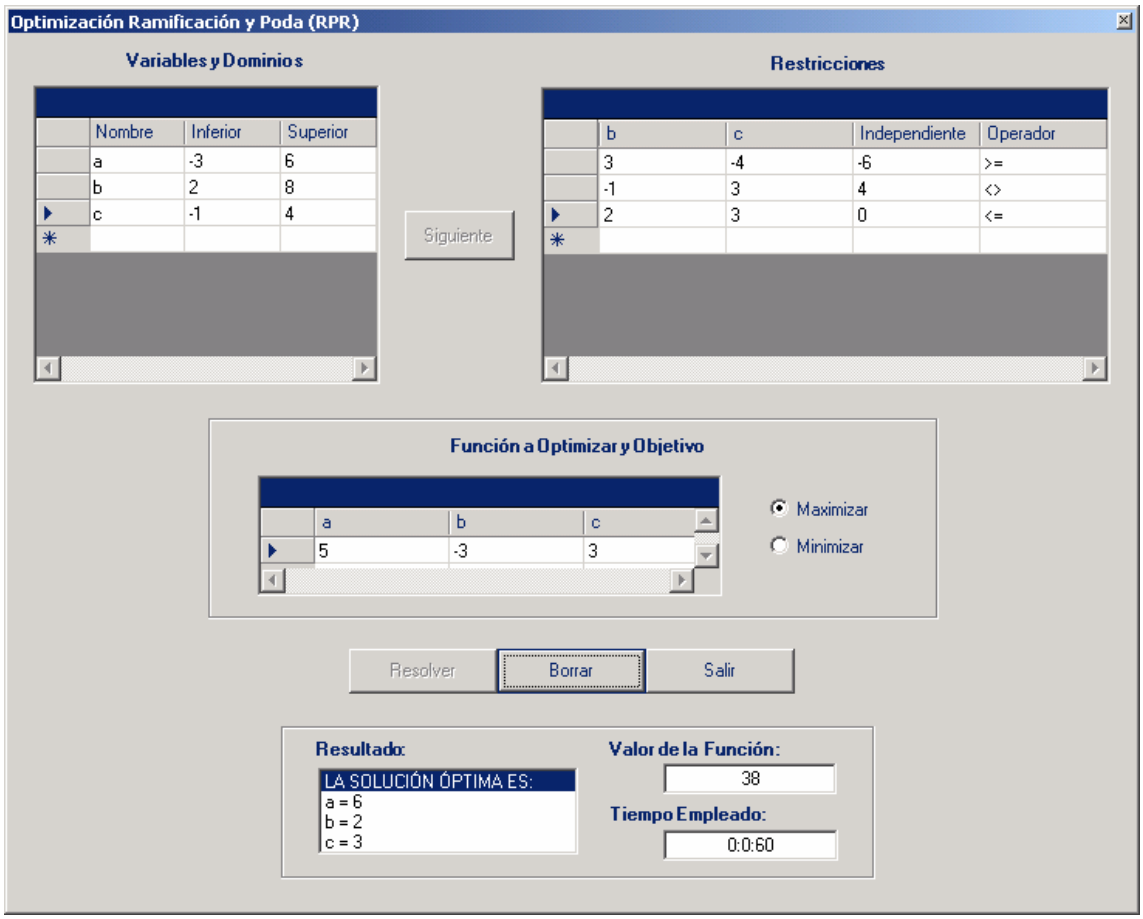


- valor seleccionado por el algoritmo para optimizar la función en el sentido seleccionado,
- valor final de la función optimizada,
- tiempo empleado en la resolución del problema (expresado en minutos, segundos y milisegundos con el formato mm:ss:$m_i m_i$)

El aspecto de la botonera inferior puede cambiar dependiendo del momento de la ejecución del problema en el que nos encontremos, es decir, los botones se verán activados o desactivados según se puedan llevar a cabo o no determinadas acciones.

- Resultado final de la ejecución

De este modo, el resultado final de la ejecución del problema planteado, quedaría visualizado por pantalla de la siguiente forma:



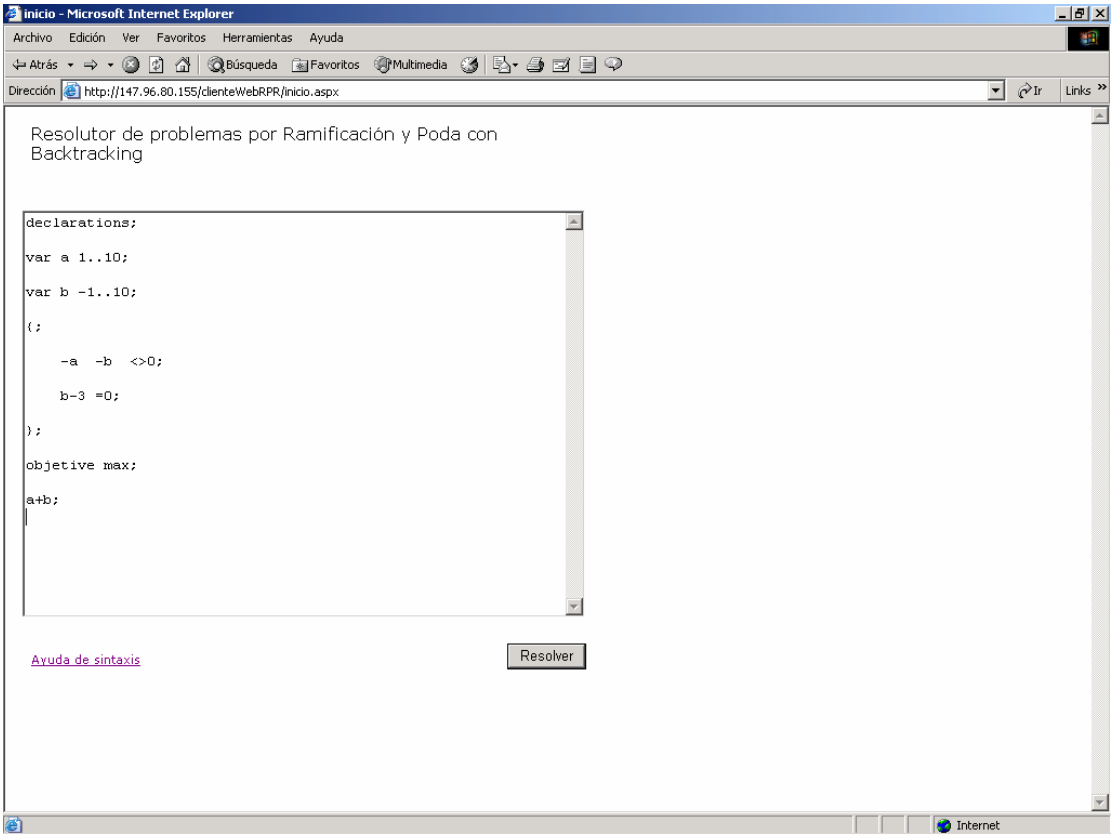
9.2.2.- ClienteWebRPR

Para poder utilizar este cliente (ASP.NET), sólo es necesario tener un navegador en una máquina conectada a Internet, y acceder a la URL *http://servidor/clienteWebResolutor/inicio.aspx,* donde *servidor* será el servidor Web IIS (Internet Information Server) donde resida la aplicación Web*ClienteWebRPR.*

La principal diferencia entre el modo de funcionamiento de este cliente y los vistos anteriormente reside en el interfaz de usuario:

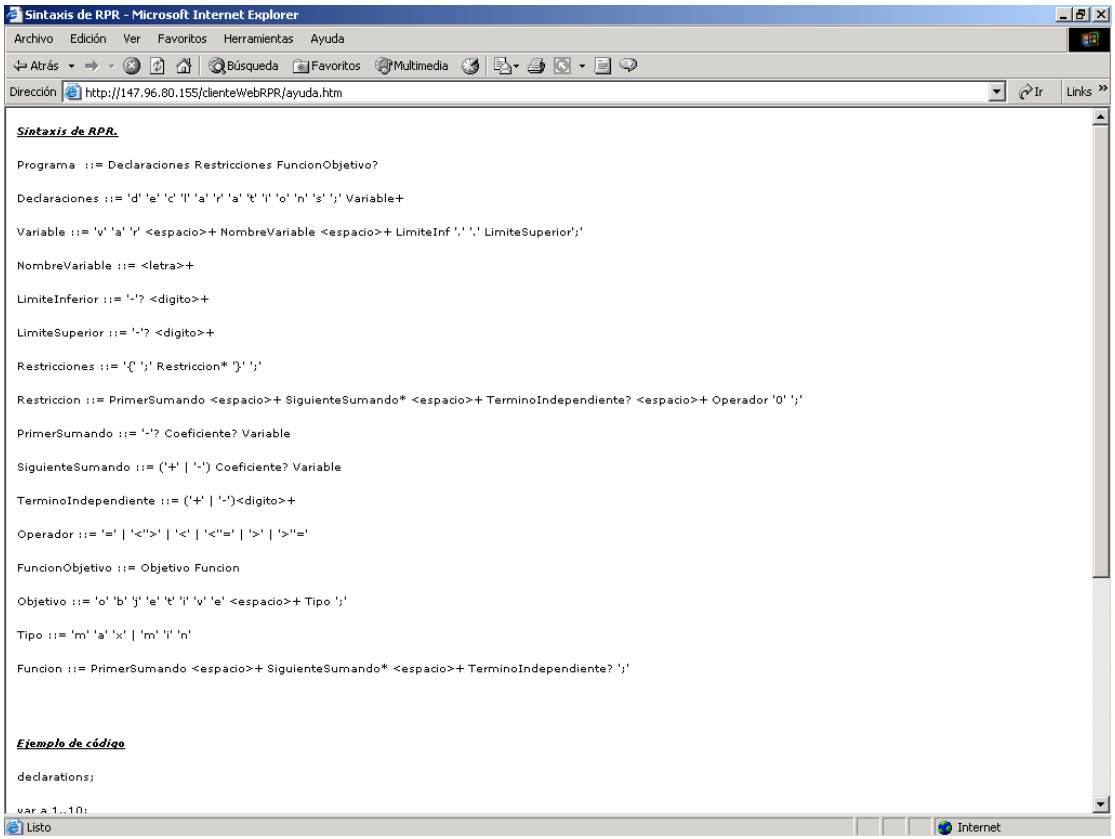
- Introducción de las variables, dominios, restricciones y de la función objetivo

Para el caso del cliente Web hemos utilizado una clase Parser de modo que facilite notablemente la introducción de todos estos valores mediante el empleo de un mini-lenguaje cuyas especificaciones aparecen en la página de ayuda que se muestra al hacer clic sobre el link "Ayuda de sintaxis".



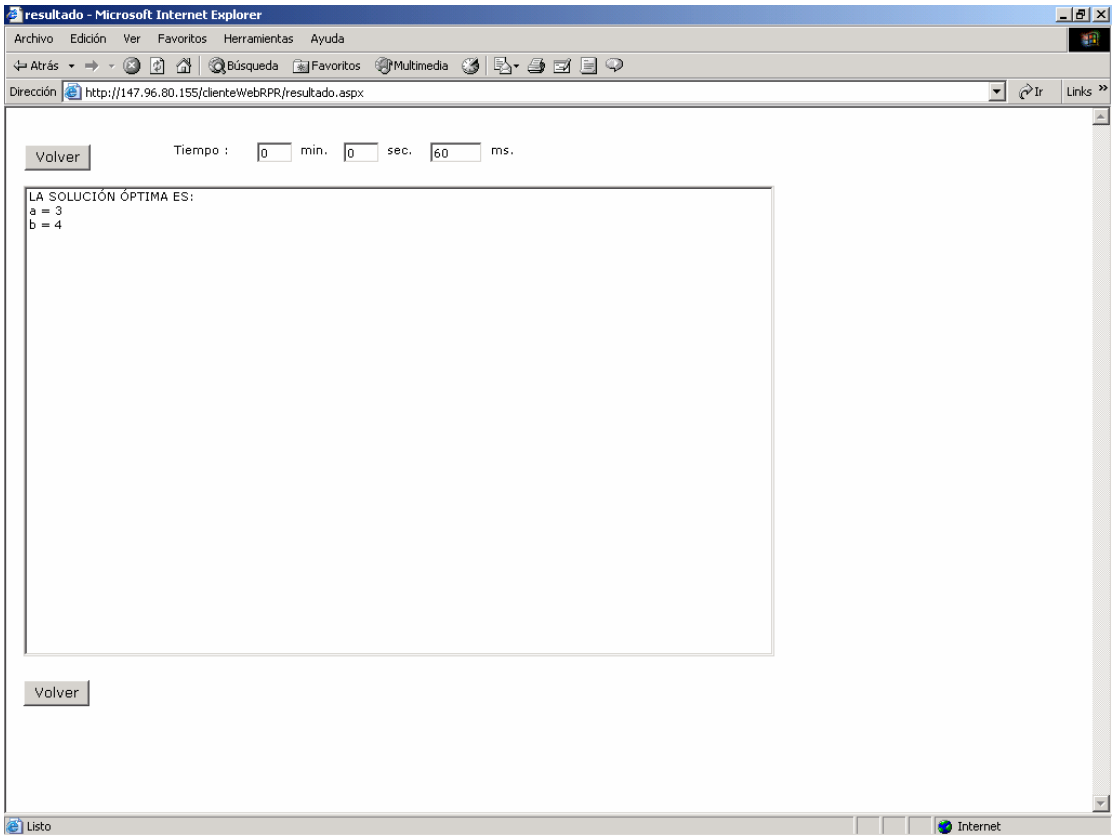
- Ayuda

La página de ayuda para el lenguaje empleado en la especificación de los parámetros a emplear en la resolución del problema es la siguiente:



- Resultado final de la ejecución

De este modo, el resultado final de la ejecución del problema planteado, quedaría visualizado por pantalla de la siguiente forma:



9.3.- Algoritmo genético

En este apartado se presenta la forma de uso de las aplicaciones que utilizan un algoritmo genético como método de optimización.

En primer lugar se comentará el funcionamiento de la aplicación windows y de la aplicación servicio web, ya que las interfaces de usuario de ambas aplicaciones son idénticas, así pues se explicarán juntas. Estas interfaces comprenden un conjunto de formularios Windows.

Después se detallará el funcionamiento de la aplicación web, cuya interfaz de usuario está compuesta por páginas HTML que se visualizan en un browser.

9.3.1.- Aplicación Windows y Aplicación Servicio Web

El archivo ejecutable de la aplicación windows es "interfazWindowsAG.exe". Para lanzar esta aplicación basta hacer doble clic sobre este fichero. En la máquina donde se ejecute la aplicación debe estar instalada la .NET Framework, y además deben estar también el componente "AG.dll" y el componente "Parser.dll" en el mismo directorio donde se quiera ejecutar "interfazWindowsAG.exe".

El archivo ejecutable de la aplicación web es "interfazServicioWebAG.exe". Para lanzar esta aplicación basta hacer doble clic sobre este fichero. En la máquina donde se ejecute la aplicación debe estar instalada la .NET Framework, y además debe tener conexión a través de Internet al servidor donde resida el servicio web del algoritmo genético, "servicioWebAG.dll". También debe estar el componente "Parser.dll" en el mismo directorio donde se quiera ejecutar "interfazServicioWebAG.exe"

Como se ha mencionado anteriormente, ambas aplicaciones tienen la misma GUI, por lo que su funcionamiento es igual, si exceptuamos la diferencia que la aplicación servicio web no dispone de ejecución paso a paso. Esta diferencia se resaltará cuando se explique cómo resolver un problema de restricciones. Comenzamos a detallar el funcionamiento de las aplicaciones siguiendo el flujo normal de las acciones del usuario desde la pantalla inicial hasta que obtiene un resultado del problema.

La figura 1 muestra es la pantalla inicial que aparece al lanzar cualquiera de las dos aplicaciones. Tiene tres zonas diferenciadas: parte superior, parte central y parte inferior. En la parte superior de la pantalla se encuentra el menú principal y la barra de tareas. En la parte central hay una zona de texto para introducir el código de un programa y un botón para solicitar la verificación del código introducido. La parte inferior del formulario se refiere al método de optimización, en este caso un algoritmo genético, y en ella se puede introducir los parámetros del algoritmo, elegir el tipo de ejecución deseada, directa o paso a paso, y ordenar la resolución el problema.

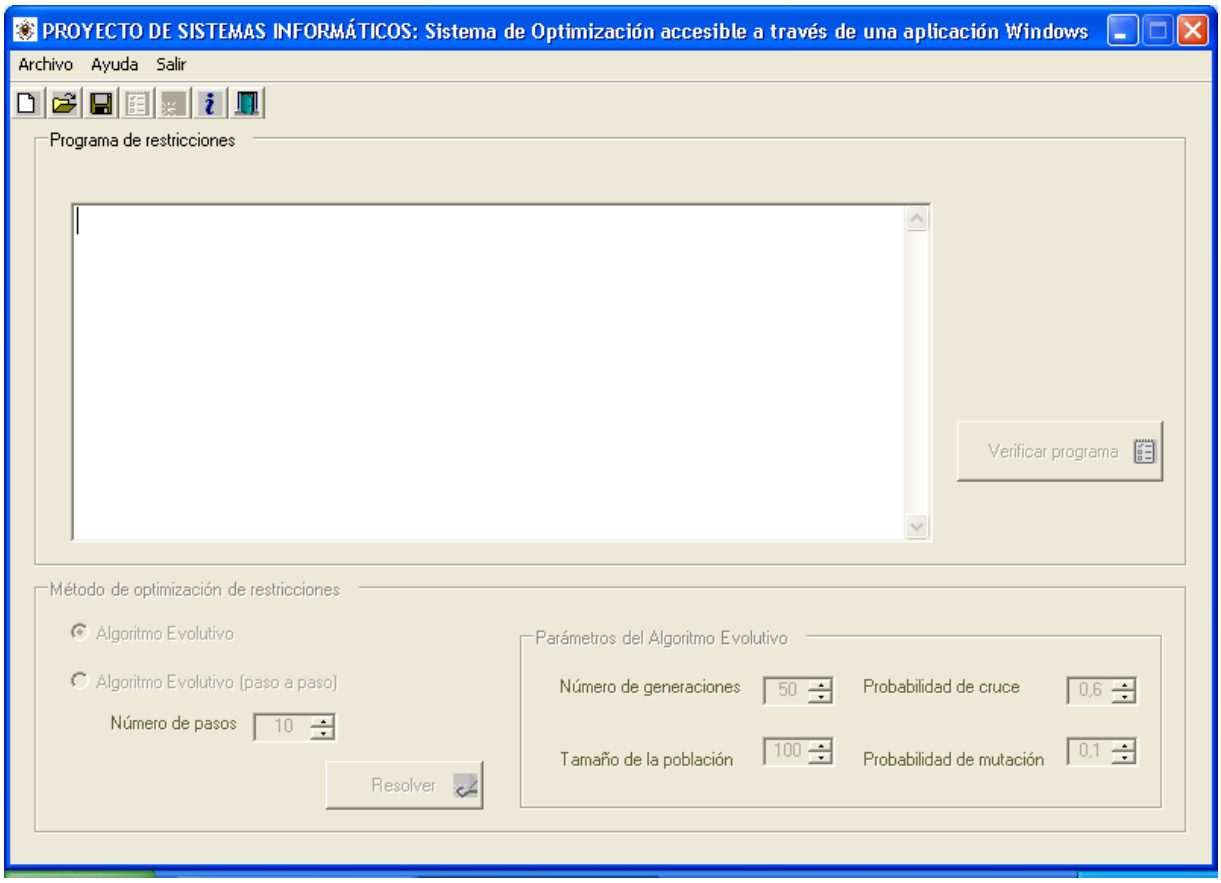


Figura 7.- Pantalla inicial

Las opciones disponibles del menú principal se muestran en la figura 2. Bajo "Archivo" se agrupan una serie de operaciones que se pueden realizar sobre ficheros. Son "Nuevo", "Abrir","Guardar" y "Guardar como". Las otras dos opciones del menú principal son "Ayuda", que solicita información y "Salir", que cierra la aplicación.

La barra de tareas muestra gráficamente las opciones más usuales sobre la aplicación, facilitando el acceso rápido a las mismas. Las imágenes de izquierda a derecha se corresponden con las opciones "Nuevo", "Abrir", "Guardar", "Verificar", "Ejecutar", "Ayuda" y "Salir".

Se puede observar que algunas opciones son accesibles de dos formas distintas, así al pulsar la opción "Ayuda" del menú principal o el penúltimo icono de la barra de tareas la aplicación nos muestra un formulario con información sobre la sintaxis del lenguaje LRPR. Ver figura 3. En esta figura si se pulsa el botón "Volver" se regresa a la figura 1. Siempre se regresará a la pantalla desde donde se halla solicitado la ayuda.

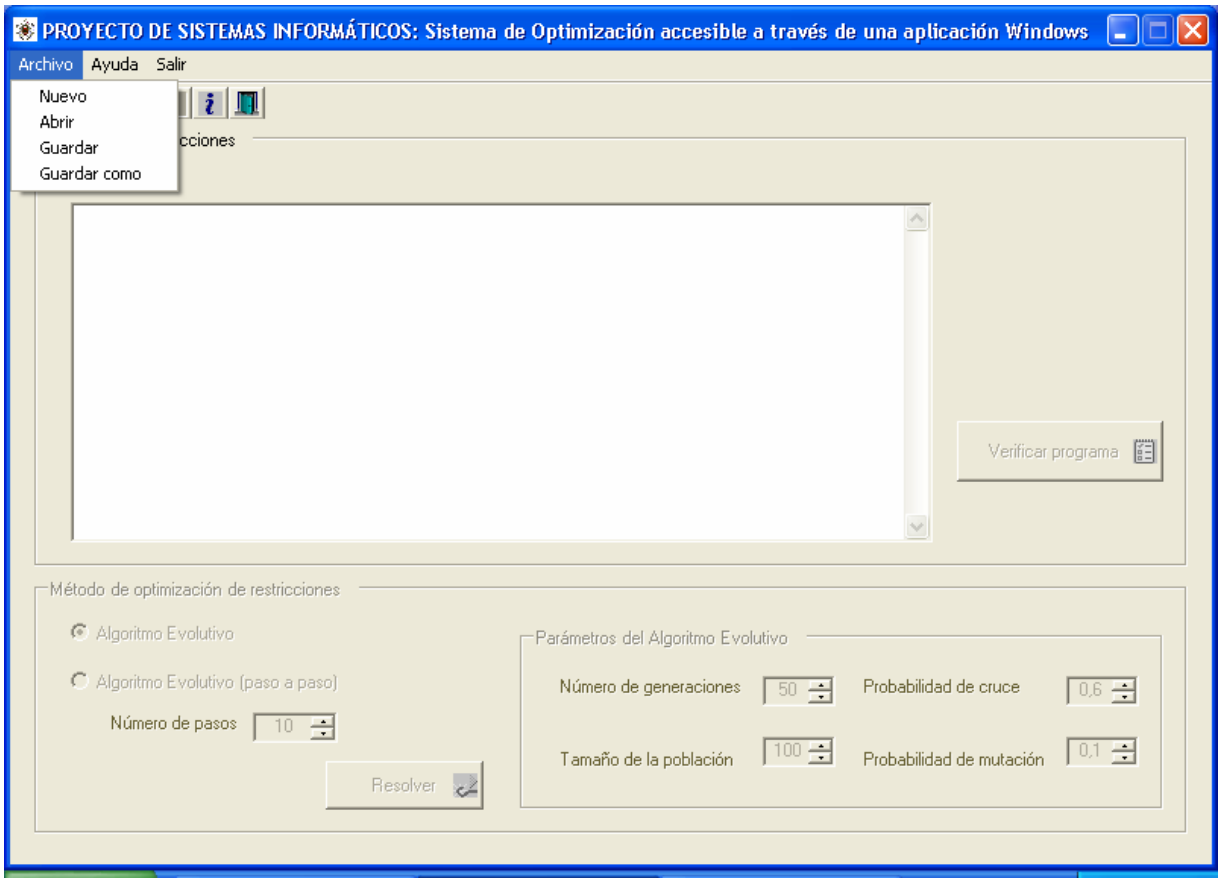


Figura 8.- Opciones disponibles

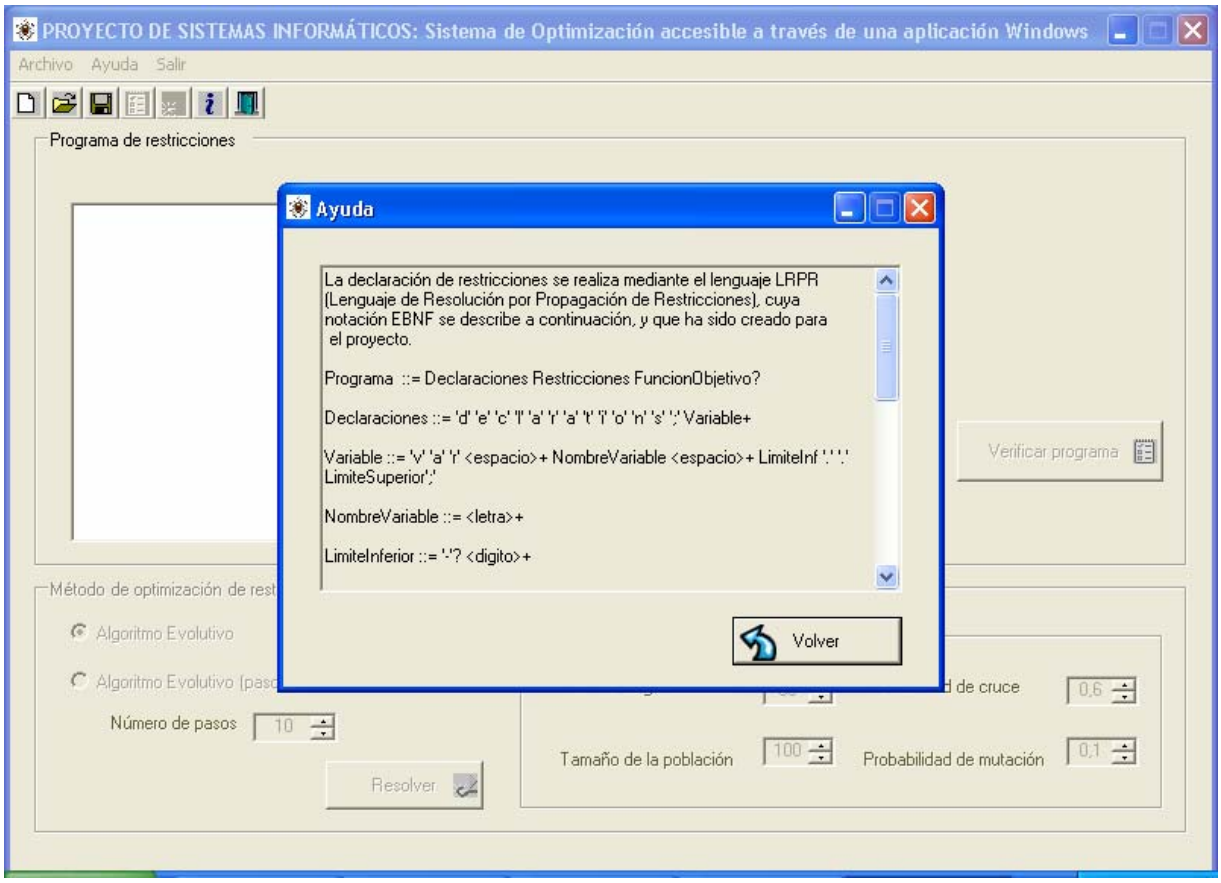


Figura 9.- Ayuda

En este estado inicial las opciones de verificar y de ejecutar no están activas: esta desactivado el cuarto y el quinto icono de la barra de tareas, el botón "Verificar Programa" y todo el panel de la parte inferior de la pantalla. Así se guía al usuario en el primer paso a dar, por que además de cerrar la aplicación y solicitar ayuda, sólo puede introducir el código del programa de restricciones en el cuadro designado para ello.

El programa debe estar escrito con el lenguaje LRPR, un lenguaje ideado para el proyecto y cuya sintaxis muestra la opción de ayuda de la aplicación. Un programa en este lenguaje representa el problema de restricciones a optimizar. El usuario puede introducir el código del programa de dos maneras: editándolo directamente sobre el cuadro de texto o recuperando el programa desde un archivo. Los archivos que maneja la aplicación son ficheros de texto, pero con extensión rpr, para asociarlos con el lenguaje LRPR.

La aplicación esta provista de un control excepcional sobre los ficheros con todas las opciones del menú "Archivo" enumeradas anteriormente, que se explicarán a continuación una por una. Las acciones sobre ficheros están siempre accesibles.

Tras pulsar la opción "Abrir", la aplicación muestra un cuadro de diálogo de apertura de ficheros, donde el usuario puede elegir de forma cómoda el archivo requerido, tal como se aprecia en la figura 4.

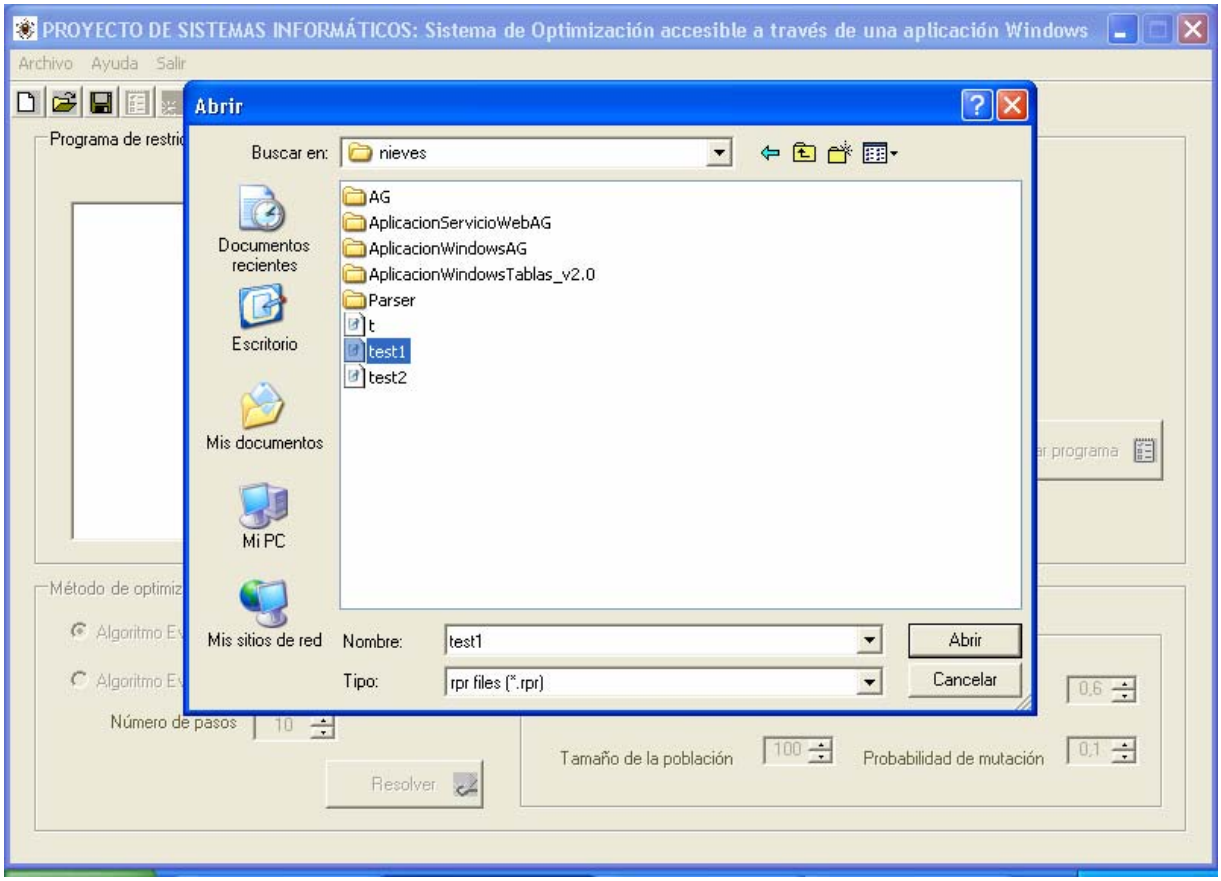


Figura 10.- Abrir

Se selecciona por ejemplo el fichero test1.rpr, y se pulsa el botón "Aceptar" del cuadro de diálogo de la figura 4. Entonces se carga el programa de test1 en el área de texto como se aprecia en la figura 5. Obsérvese que sobre el área de texto aparece la ruta completa del archivo recién abierto.

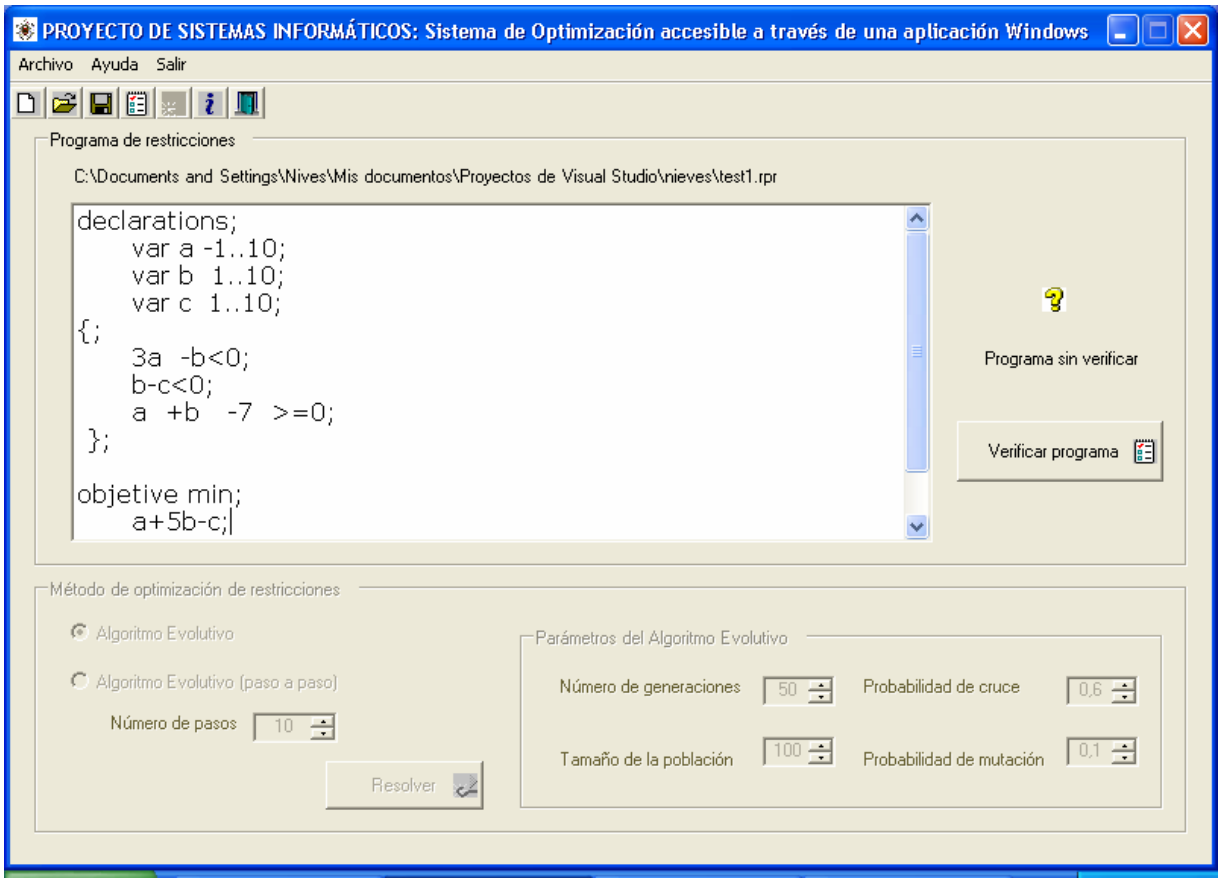


Figura 11.- Programa sin verificar

Cada vez que se pulse la opción "Nuevo", la aplicación volverá al estado representado por la figura 1, limpiado todo el área de texto.

Con la opción "Guardar" el usuario podrá almacenar en archivos .rpr programas que haya recuperado anteriormente y que después haya modificado en el área de texto. Con la opción "Guardar como" el usuario podrá cambiar de nombre a los archivos abiertos. En este caso, la aplicación mostrará un cuadro de diálogo para guardar ficheros, similar al de la figura 4, donde el usuario podrá especificar el nombre del fichero y la ruta de almacenamiento.

Si se pulsa la opción "Guardar" y el código del programa no había sido almacenado previamente, entonces la aplicación cambia la acción por "Guardar como". Si se solicitan alguna de estas dos opciones y el texto del programa es vacío, la aplicación no realiza ninguna acción. Si el programa está modificado, y se solicita la acción "Nuevo", "Abrir" o "Salir", la aplicación preguntará al usuario si quiere guardar el fichero para no perder información, realizando la opción "Guardar" ó "Guardar como", según las condiciones ya explicadas.

Obsérvese que la opción de ejecución aún no esta activa, para guiar una vez más al usuario que intuye de esta manera que el siguiente paso a dar es la verificación del programa.

En cuanto el área de texto contenga algo, ya sea por que se abra un archivo como se acaba de ver, o por que el usuario escriba directamente en el cuadro, la opción de verificación se activará, es decir, pasarán a estar accesibles tanto el cuarto botón de la barra de tareas como el botón "Verificar programa". Además en la parte central derecha aparecerá una imagen (interrogación amarilla) acompañada de un texto para informar al usuario del estado del programa, en este momento, sin verificar. La figura 5 refleja este estado.

Al solicitar la verificación del programa, el parser del lenguaje LRPR analiza el código, y entonces pueden ocurrir dos cosas: que el código sea incorrecto, situación representada en la figura 6, o que el código sea correcto, situación descrita por la figura 7.

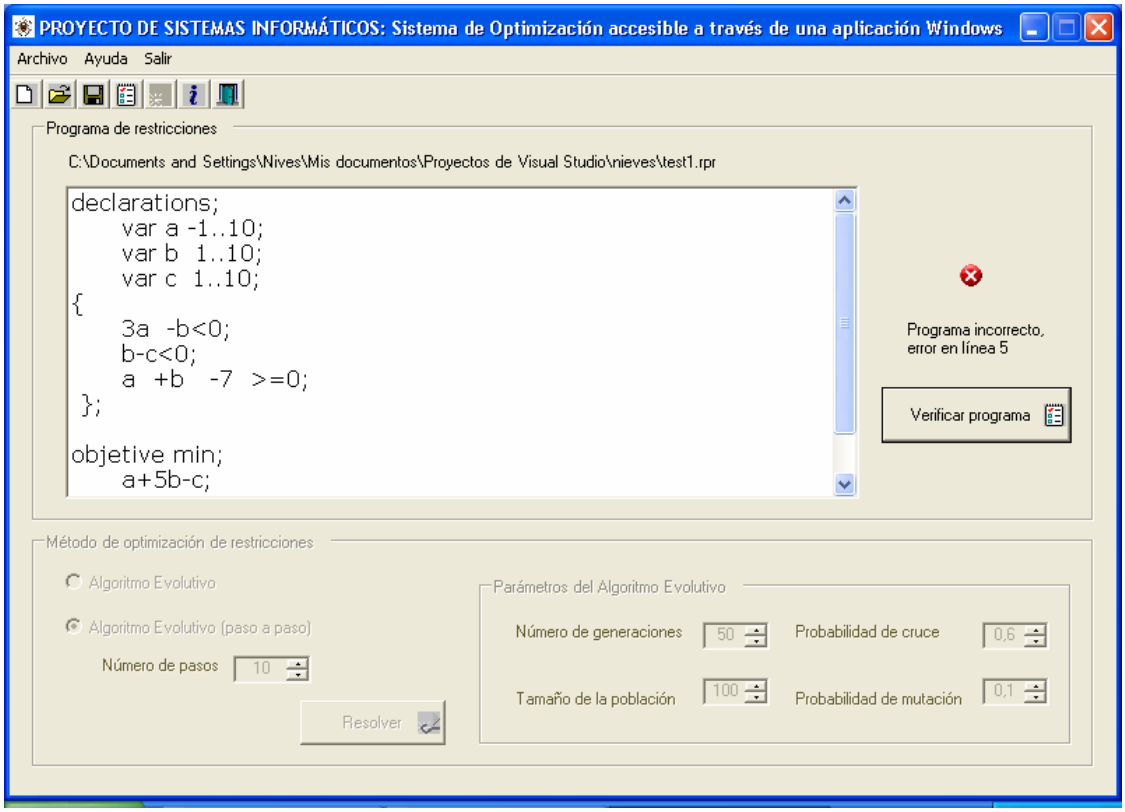


Figura 12.- Programa incorrecto

El parser del lenguaje LRPR detecta en un programa dos tipos distintos de errores: error de sintaxis y error de uso de variables no declaradas en la definición de las restricciones. El error de sintaxis ocurre cuando un programa no cumple la sintaxis del lenguaje LRPR, al no encontrarse con el símbolo esperado. Este es el tipo de error que aparece en la figura 6, pues en la línea 5, después de'{' se esperaba un punto y coma ';'. Cuando ocurre un error como éste, en la parte central derecha aparece una imagen (aspa roja) acompañada de un texto para informar al usuario del estado del programa, en este caso, incorrecto. Obsérvese en la figura 6.

Si en la figura 4 el usuario pulsa el botón "Verificar programa" o el cuarto icono de la barra de tareas, se llegará a la figura 7, pues el código contenido en el área de texto es correcto. En la parte central derecha de esta pantalla hay una imagen (tick verde) acompañada de un texto con la que se informa del estado correcto del programa.

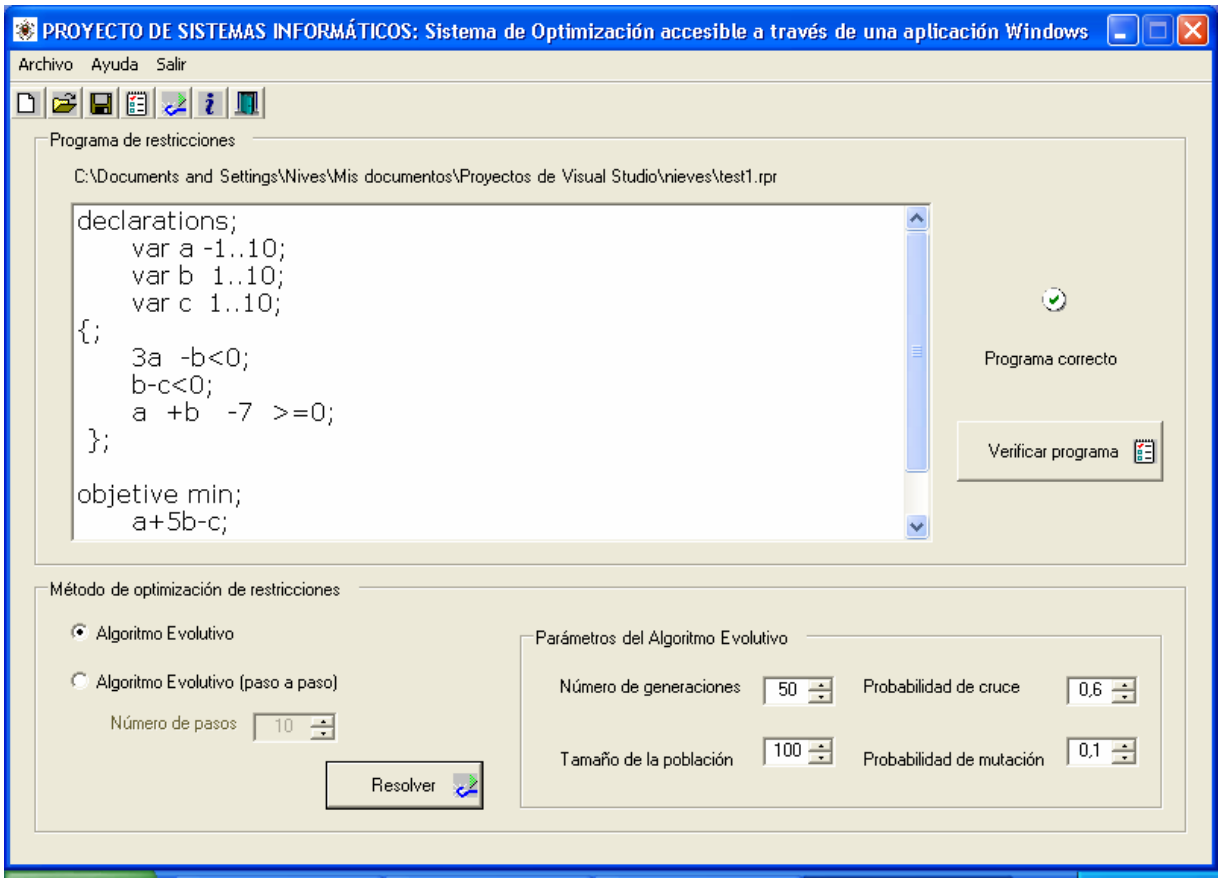


Figura 7.- Programa correcto y ejecución directa

Si en la figura 7, estado correcto del programa, o en la figura 6, estado incorrecto del programa, el usuario modifica el código, añadiendo o borrando algo en el área de texto, la aplicación regresará a la figura 5, estado no verificado del programa.

En la figura 7 se observa, que si tras su verificación un programa es correcto, la opción de ejecución se activa, es decir, pasan a estar disponibles el quinto icono de la barra de tareas y todo el panel de la parte inferior, "Método de optimización de restricciones".

En este panel hay un apartado "Parámetros del Algoritmo Evolutivo", donde el usuario puede modificar los parámetros típicos de un algoritmo genético: el número de generaciones, el tamaño de la población o número de individuos, la probabilidad de aplicación del operador de cruce y la probabilidad de aplicación del operador de mutación. Estos parámetros tienen un valor por defecto con el que el algoritmo genético muestra unos resultados fiables en la mayoría de las ejecuciones. La posibilidad de variarlos es interesante cuando el usuario pretende estudiar el comportamiento del algoritmo genético, pero si no es así, no es conveniente variar los valores predeterminados.

Lo más importante de este panel es que permite elegir al usuario entre dos formas ejecutar el algoritmo genético, mediante unos radiobuttons autoexcluyentes. Así sí se selecciona "Algoritmo Evolutivo" tenemos una ejecución directa, que es la opción por defecto, y sí se cambia a "Algoritmo Evolutivo (paso a paso)" se tiene una ejecución paso a paso.

Una vez elegida la forma de ejecución del algoritmo, para lanzar la resolución del problema de restricciones se debe pulsar el botón "Resolver" o el quinto icono de la barra de tareas.

En la figura 8 aparece el formulario al que se llega tras una ejecución directa del algoritmo evolutivo para resolver el problema de restricciones. En ella se muestra toda la información sobre la solución encontrada por el algoritmo genético, además del tiempo que ha llevado la resolución del problema. La figura 8 corresponde a una solución encontrada a partir de la figura 7.

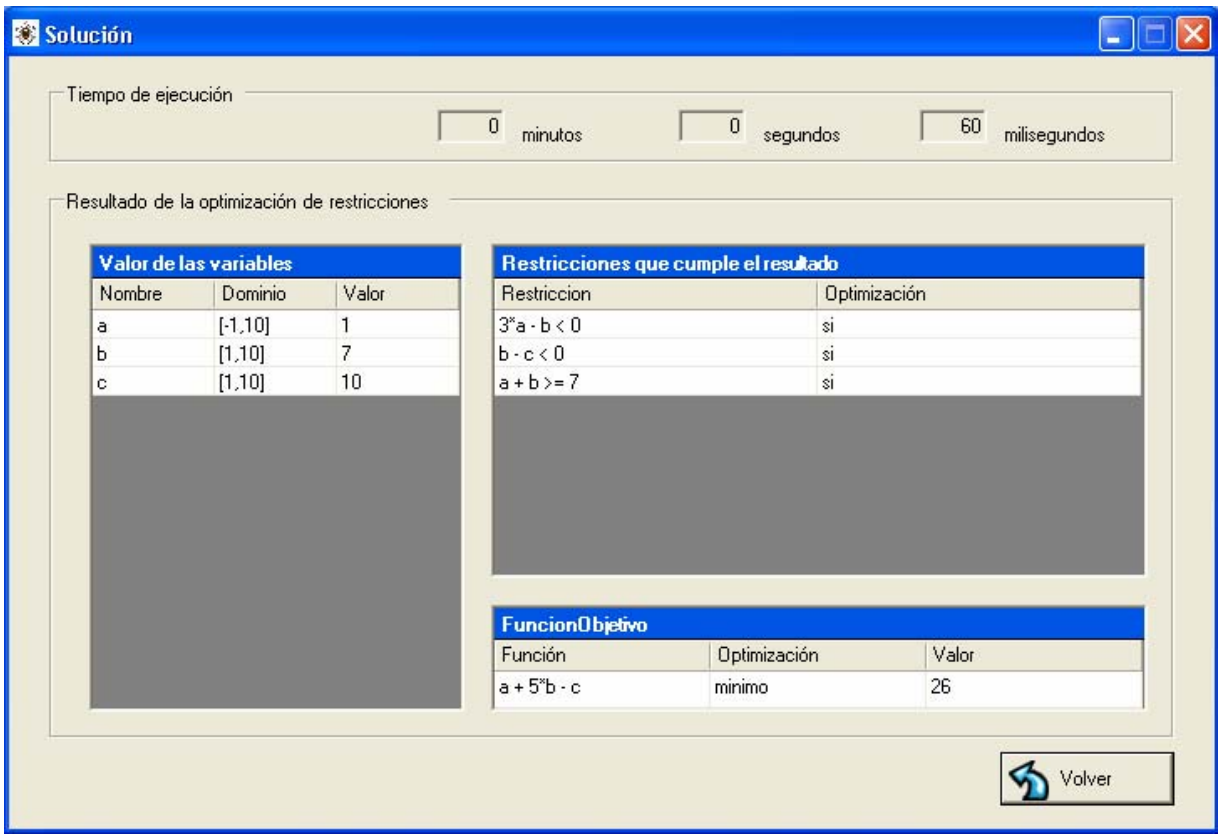


Figura 8.- Resultado ejecución directa

No sólo se muestra información del resultado encontrado, sino que se presenta los valores de entrada del problema de restricciones, es decir, las variables, las restricciones y la función objetivo, si la había. La información se agrupa en tres tablas en el panel "Resultado de optimización de restricciones". La primera tabla de nombre "Valor de las variables" tiene tres columnas, "Nombre", donde aparecen los nombres de las variables declaradas, "Dominio", donde aparecen los límites inferiores y superior de cada variable, y "Valor", donde se escribe el valor encontrado por el algoritmo genético para cada variable. La tabla de nombre "Restricciones que cumplen el resultado", tiene dos columnas, "Restricción", donde aparecen las restricciones definidas en el problema, y "Optimización", donde se escribe si la restricción se cumple para el valor de las variables encontrado. La tabla de nombre "Función Objetivo", tiene tres columnas, "Función", donde aparece la función objetivo definida en el problema, "Optimización", donde aparece el tipo de optimización a realizar sobre la función objetivo, y "Valor", donde se escribe el valor de la función objetivo dado por el valor resultado de las variables. Si no se ha declarado función objetivo en el programa de restricciones esta tabla aparecerá vacía.

Para realizar una ejecución paso a paso, estando en la figura 7 se tiene que seleccionar el segundo radiobutton, "Algoritmo Evolutivo (paso a paso)". La aplicación ofrece la posibilidad de decidir el número de pasos o generaciones que se avanzan cada vez. Por defecto son 10 generaciones.

Una vez elegida la ejecución paso a paso, se pulsa el botón "Resolver", llegando a la figura 9 que se corresponde con el paso 0, donde se muestra información sobre el mejor individuo de la población inicial en la generación inicial.

Esta forma de ejecución tiene un fin didáctico, por lo que se muestra en este y los siguientes formularios que se generen para esta forma de ejecución, la correspondencia entre la población sobre la que se aplica el algoritmo genético y el valor de las variables del resultado encontrado hasta ese momento por el algoritmo genético. Ese resultado se extraerá del mejor individuo de la población de la generación en la que nos encontremos. En el panel "Representación del individuo" se muestran los valores con los que se puntúa cada individuo para seleccionar el mejor de ellos, aptitud, puntuación y puntuación acumulada, y la cadena de genes de un individuo, donde cada gen representa una variable y almacena su valor.

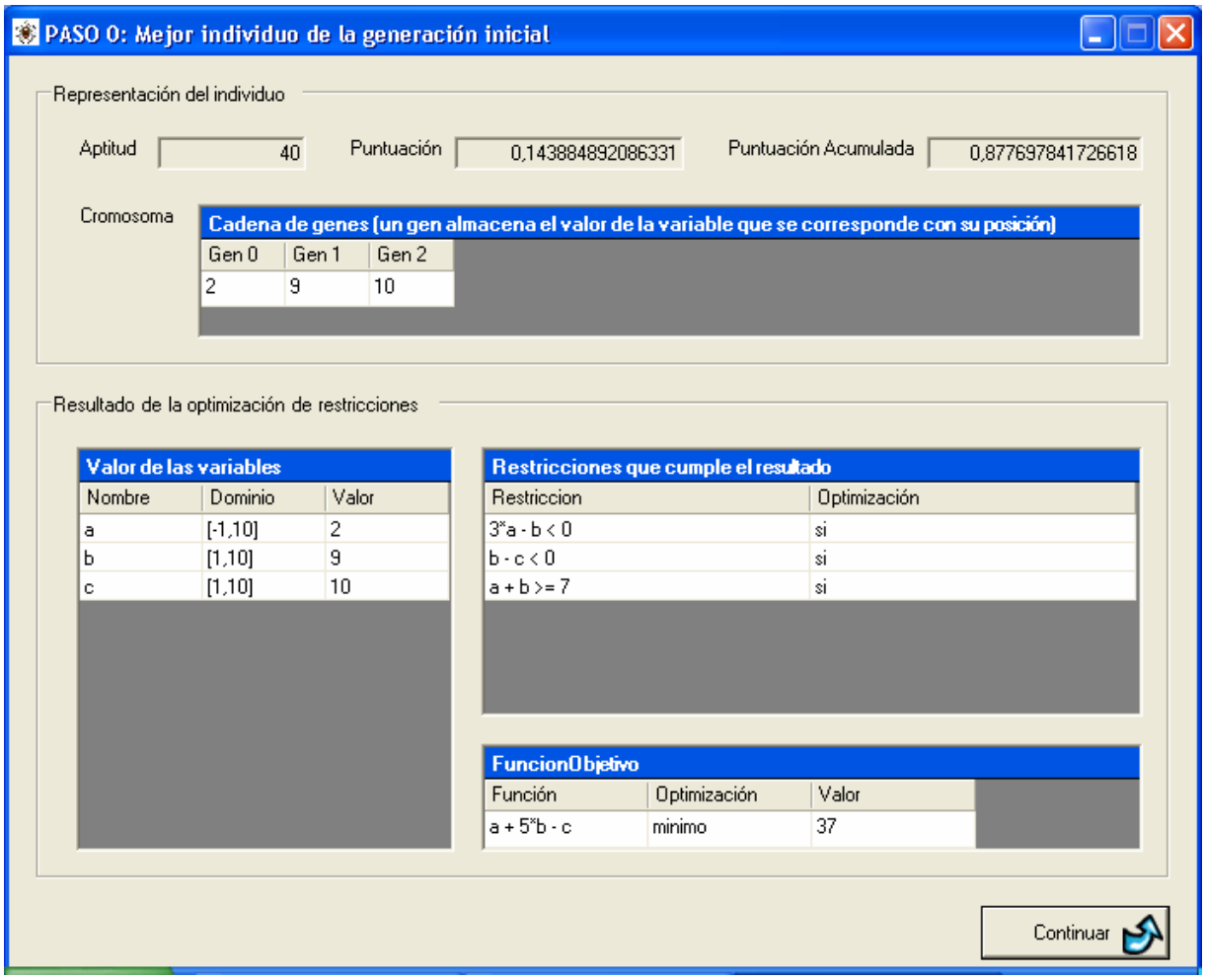


Figura 9.- Primer Paso

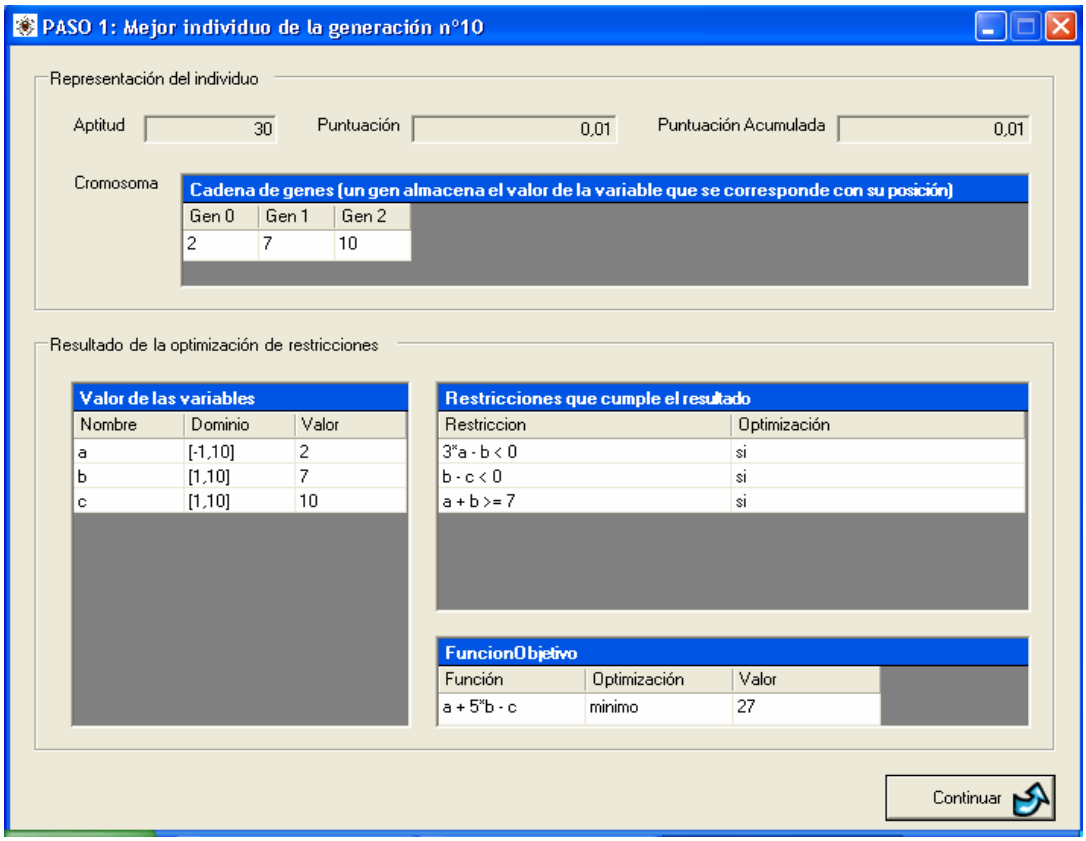


Figura 10.- Siguiente paso

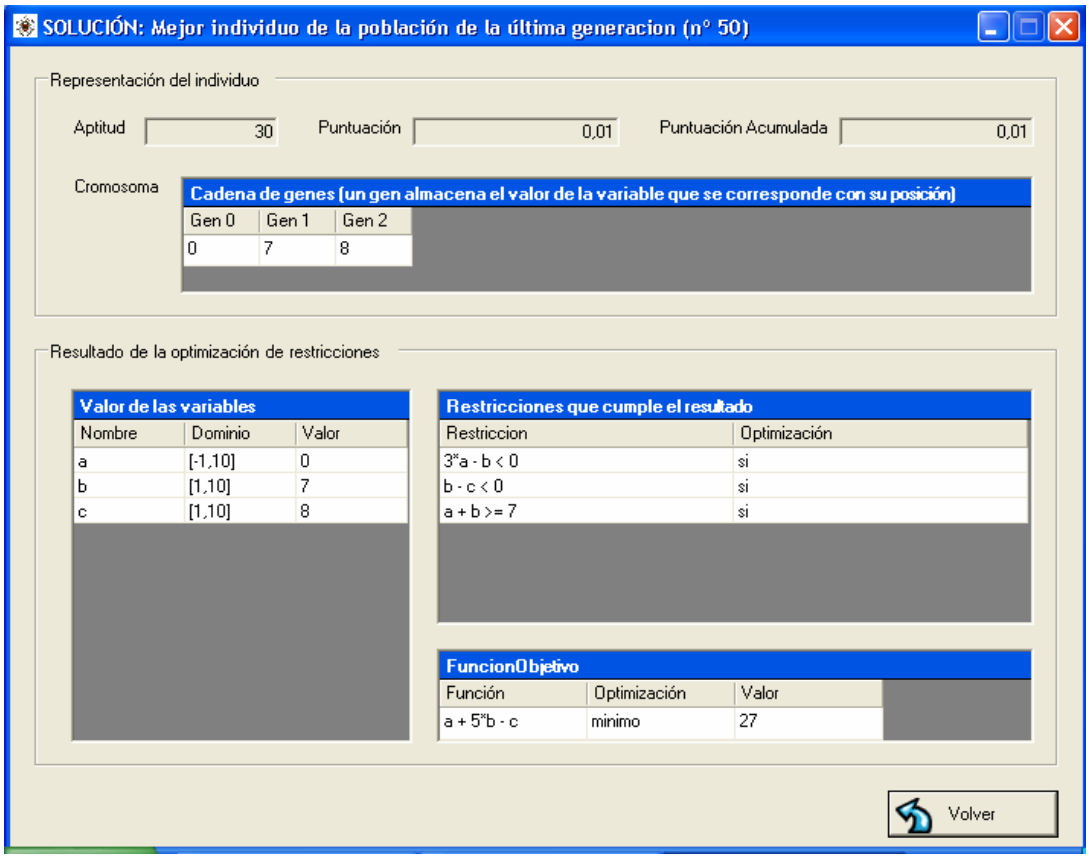


Figura 11.- Último paso

Si en la figura 9 el usuario pulsa el botón continuar se llega a la figura 10, en la que se han avanzado tantos pasos como se hayan especificado en la figura 7 (en el ejemplo 10 pasos, por lo que la figura 10 muestra la información del mejor individuo de la generación 10).

El usuario puede dar tantos saltos como número de generaciones divido entre el número de pasos. Cada vez que el usuario pulse el botón "Continuar" se avanzarán los pasos especificados. En la figura 11 se muestra al resultado final por que se llega o se sobrepasa el número de generaciones totales. En este formulario desaparece el botón "Continuar", por lo que el usuario no puede dar más saltos, y en su lugar se encuentra el botón "Volver".

Si en la figura 8 o en la figura 11 se pulsa el botón "Volver", la aplicación regresa a la figura 7. También se regresa a esta aplicación si se cierra los formularios de las figuras 8 a 11.

La aplicación servicio web no tiene implementado la posibilidad de ejecutar el algoritmo genético paso a paso. Así se le informa al usuario cuando intenta resolver el problema utilizando este método como se ve en la figura 12.

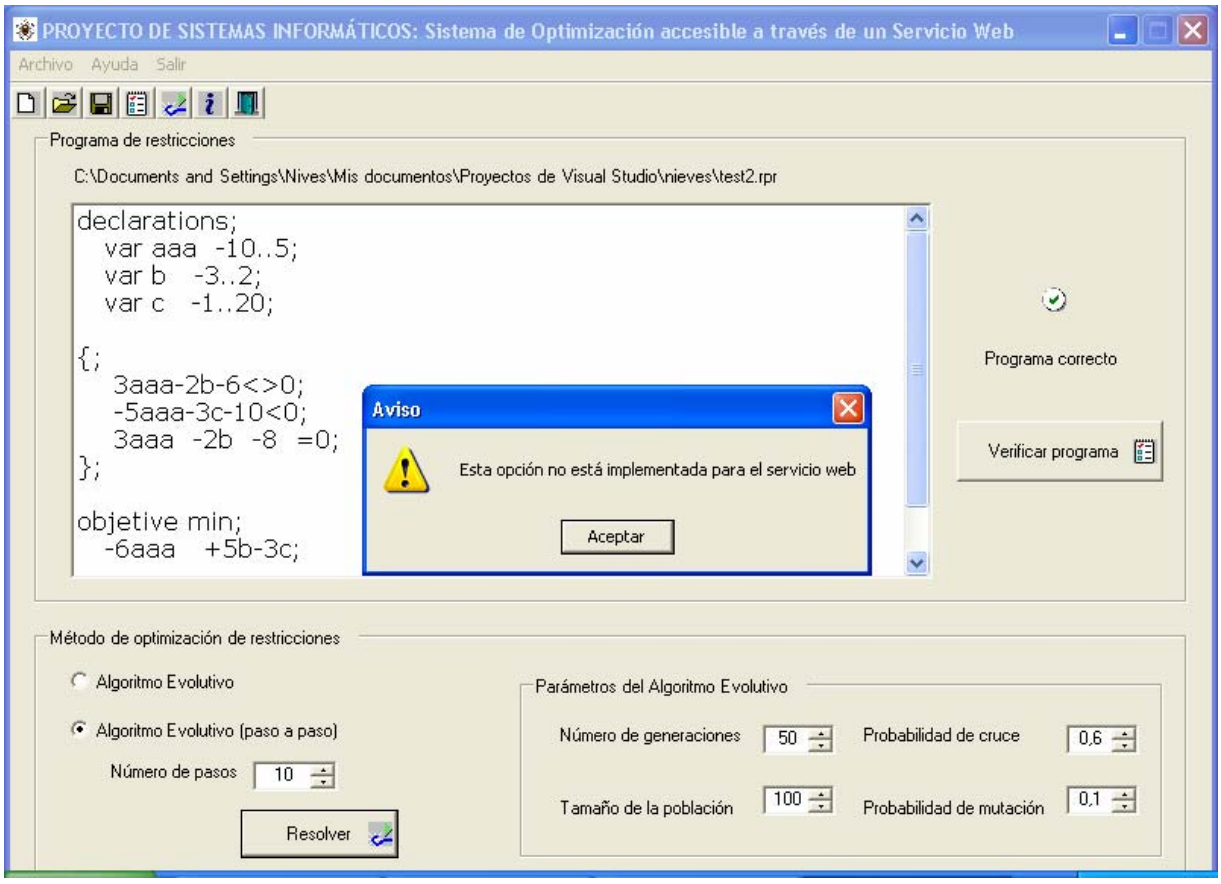


Figura 12.- Ejecución paso a paso en aplicación servicio web

El usuario puede repetir toda esta secuencia de pasos para resolver tantos problemas de restricciones como desee. Cuando desea abandonar la aplicación debe pulsar la opción "Salir" del menú principal lo el último icono de la barra de tareas. Entonces, la aplicación le solicitará confirmación de salida, tal como se muestra en la figura 13. Si el usuario decide salir y el código del área de texto está modificado, se le pregunta para guardar los cambios en un fichero.

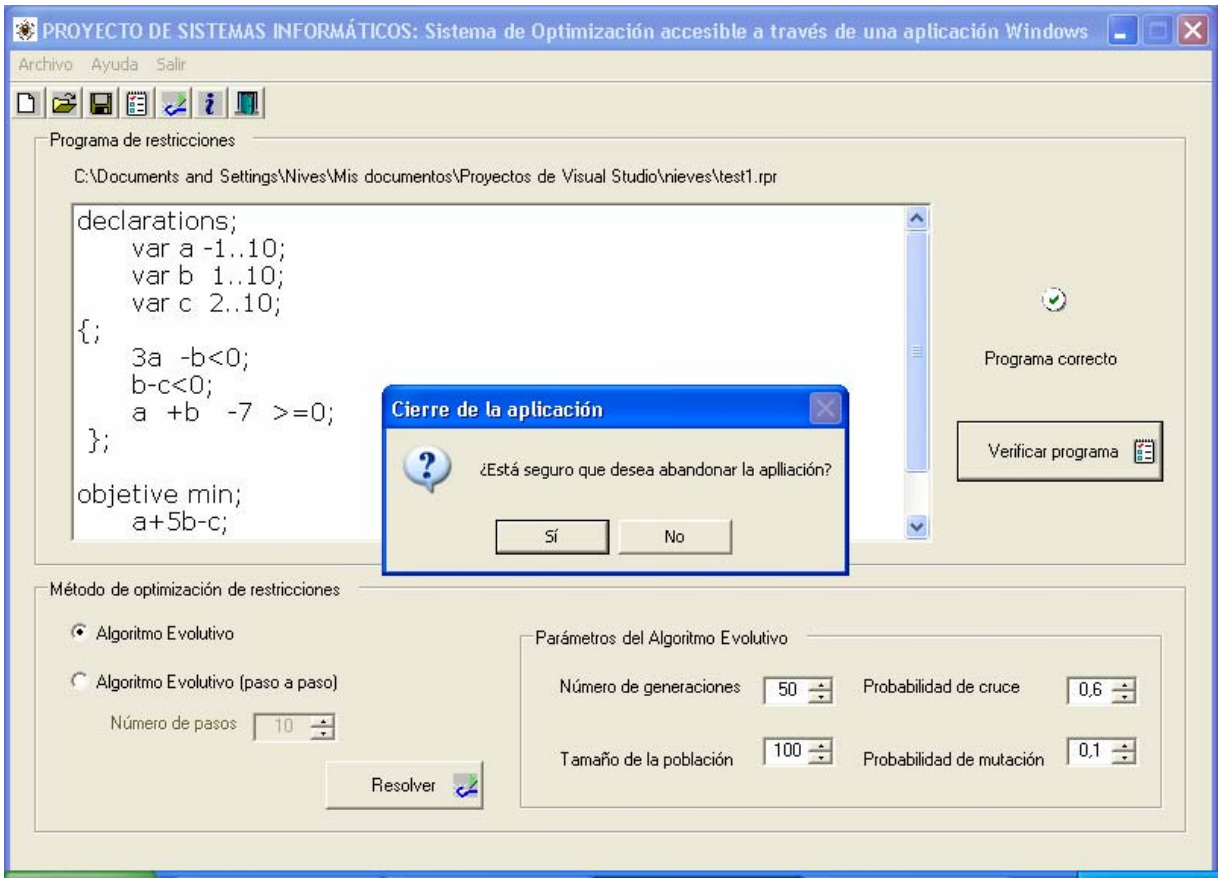


Figura 13.- Cerrar la aplicación

9.3.2.- Aplicación Web

El archivo ejecutable de la aplicación web es "interfazWebAG.exe". Para lanzar esta aplicación basta hacer doble clic sobre este fichero. En la máquina donde se ejecute la aplicación debe estar instalada la .NET Framework, y además debe estar también el componente "AG.dll" y el componente "Parser".dll en el mismo directorio donde se quiera ejecutar "interfazWebAG.exe".

Una forma más que está disponible para acceder a esta aplicación es la URL http://servidor/aplicacionWeb/WebFormPrincipal.aspx desde un navegador en un máquina con conexión a Internet.

La aplicación web tiene la misma filosofía de funcionamiento que las otras dos aplicaciones explicadas en el apartado anterior, la aplicación windows y la aplicación servicio web. Es decir, son necesarios los mismo pasos para llegar de un problema de restricciones a un resultado: escribir el programa en LRPR, verificarlo y ejecutar el algoritmo genético.

En la figura 14 vemos la página inicial de la aplicación web, que posee los mismos elementos que tiene el formulario inicial de la aplicación windows y de la aplicación web, aunque con un aspecto típico de una página web. Faltan la barra de tareas y el menú principal de la parte superior. Y es que la aplicación web no tiene manejo de archivos. La única posibilidad de introducir el programa en el área de texto es que el usuario lo edite directamente o bien lo traiga desde un editor cualquiera de texto mediante selección y copia (disponible en el menú contextual "Copiar" ó Ctrl.+C) y lo pegue en el área de texto (disponible en el menú contextual "Pegar" ó Ctrl.+V). Esto ha sido así porque System.Web.Controls de .NET no posee cuadros de diálogos para abrir y guardar de ficheros, lo que hace a la aplicación web más incómoda para trabajar con ella.

También se observa en la figura 14 que desde el principio está disponible la opción de verificar, está activado el botón "Verificar programa". No tiene sentido verificar un código vacío, aunque si el usuario lo intenta la aplicación le informará del error. Más adelante se explica el motivo de que esto no esté controlado como en las aplicaciones de formularios windows, que guiaban al usuario activando cada vez la siguiente acción a realizar. La opción de ejecución no está disponible, ya que el botón "Resolver" no está activado, aunque el resto del panel inferior "Método de optimización de restricciones" sí lo esté.

La opción de ayuda es un link, más propio de una aplicación web, situado al principio y al final de la página de inicio, como se aprecia en la figura 14. Cuando se pulsa aparece la página de la figura 15, mostrando información sobre la sintaxis del lenguaje LRPR. Pulsando el link "Volver" de la figura 15, se regresa a la página de inicio.

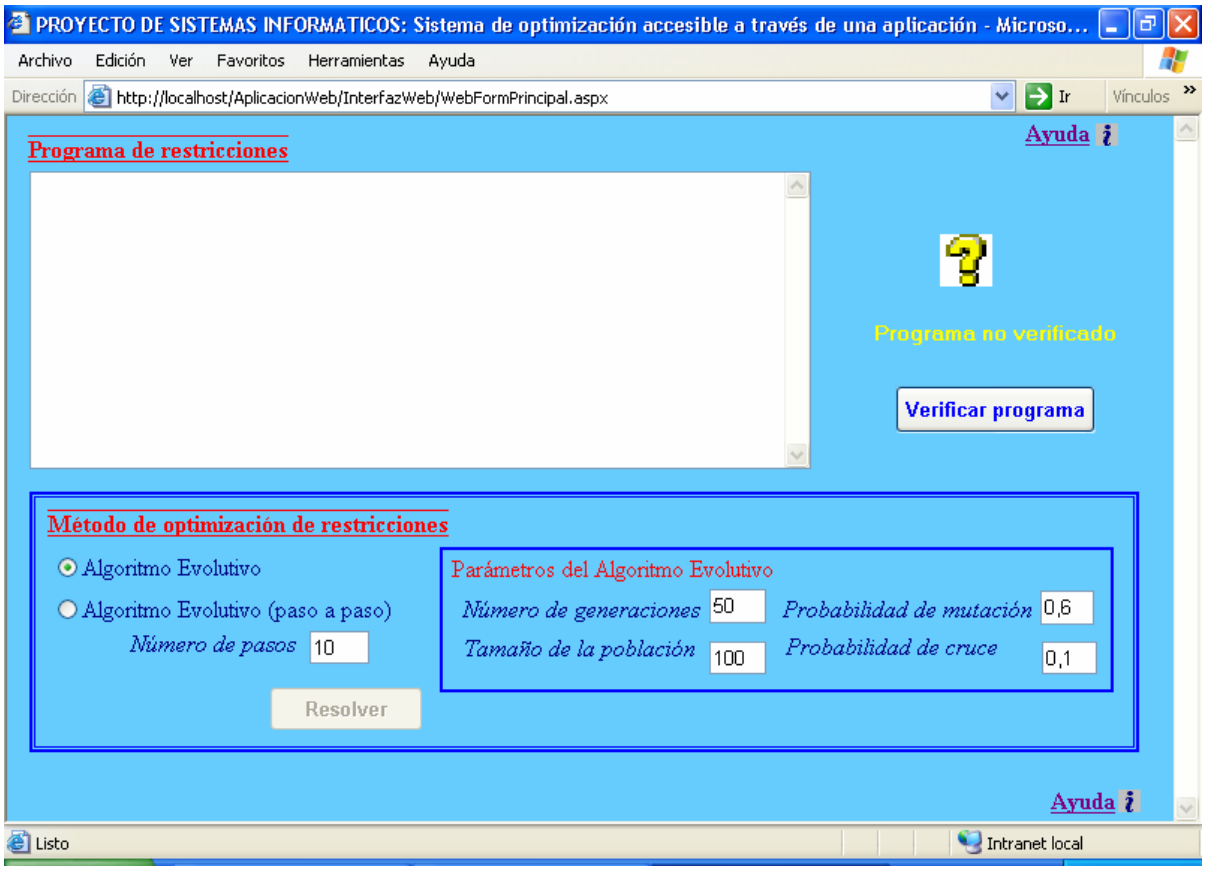


Figura 14.- Pantalla inicial

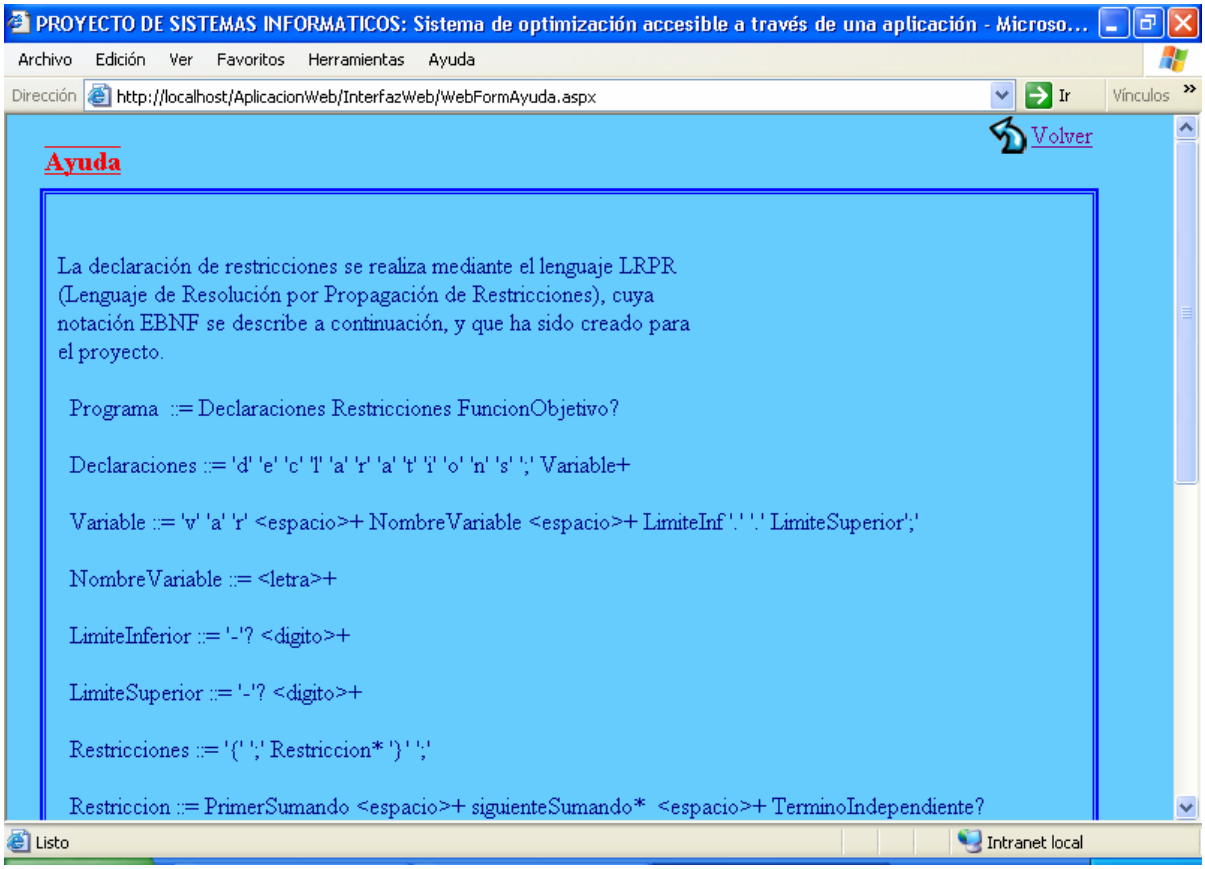


Figura 15.- Ayuda

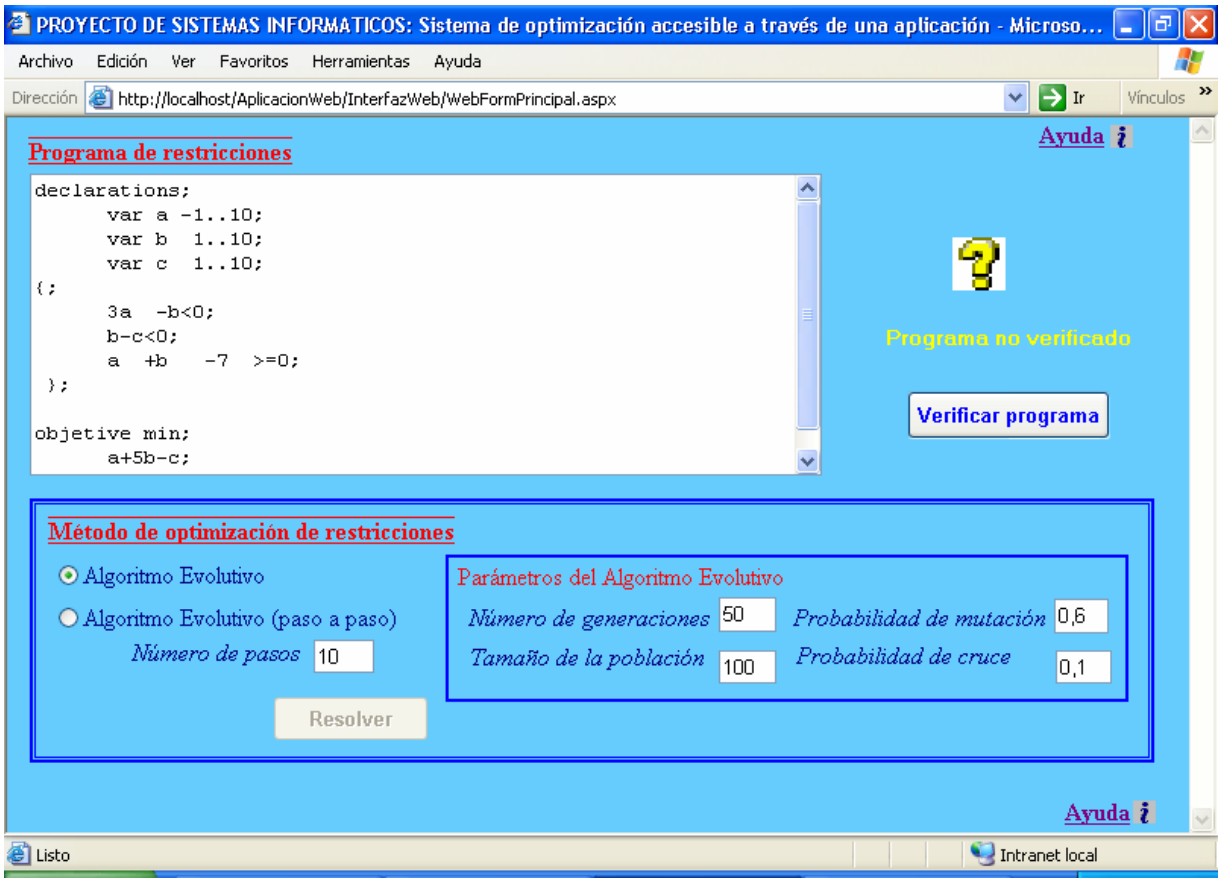


Figura 16.- Programa sin verificar

Si en la figura 14 se introduce el código de un programa, ya sea mediante la secuencia de operaciones selección-copiar-pegar desde un editor de texto o escribiéndolo el usuario directamente, se llega a la figura 16,donde tenemos un programa sin verificar. La acción natural siguiente es verificarlo pulsando el botón "Verificar programa".

Al solicitar la verificación del programa, el parser del lenguaje LRPR analiza el código, y entonces pueden ocurrir dos cosas: que el código sea incorrecto, situación representada en la figura 17, o que el código sea correcto, situación descrita por la figura 18. El estado de un programa se sigue mostrando mediante una imagen acompañada de texto en la parte central derecha.

Los formularios web responden mal a algunos eventos, por ejemplo, no capturan cuando el código de un programa se modifica en el área de texto. Entonces la información del estado de un programa no cambia dinámicamente si un programa se modifica en el área de texto, no se pasa a estado sin verificar, al figura 16. Por lo tanto la información del estado correcto o incorrecto de un programa sólo es válida recién verificado. Esto conlleva a que siempre va a estar disponible la opción de verificación.

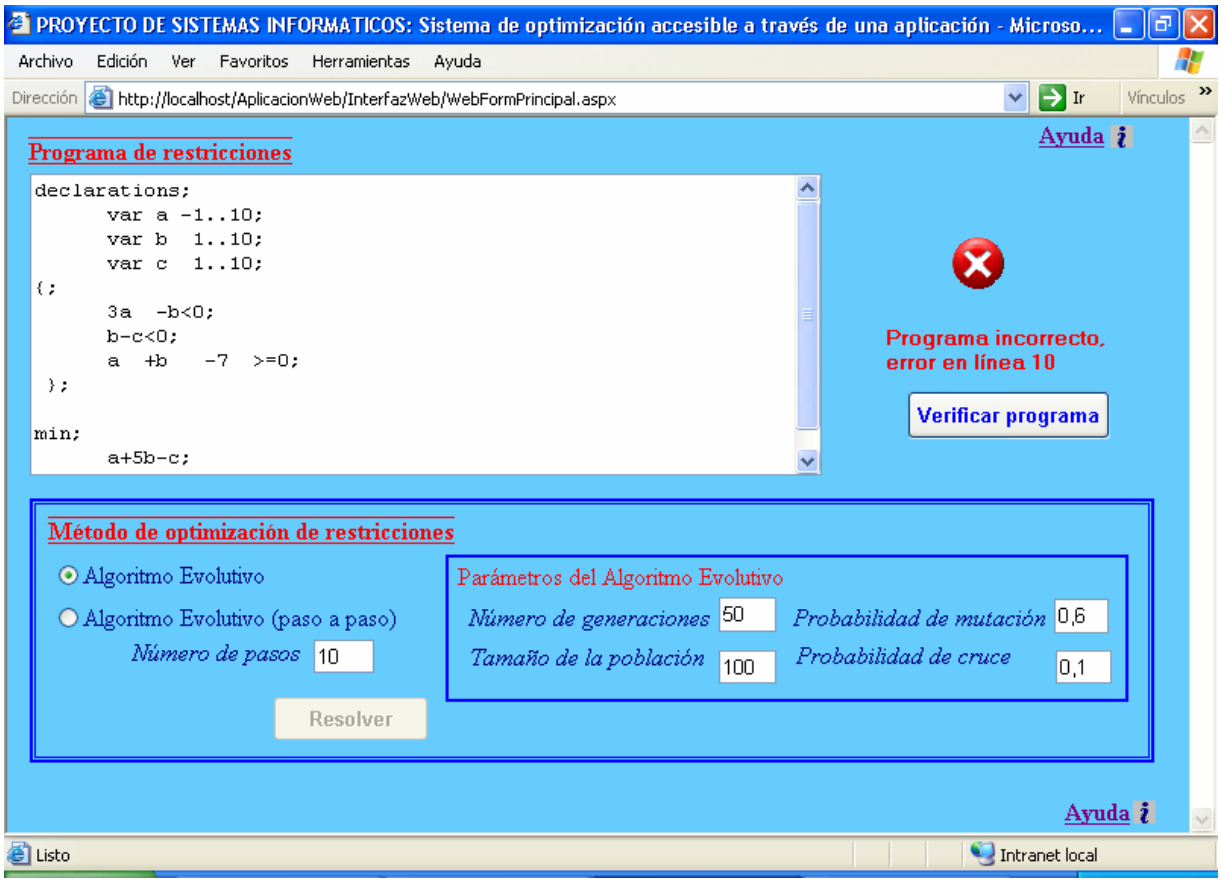


Figura 17.- Programa incorrecto

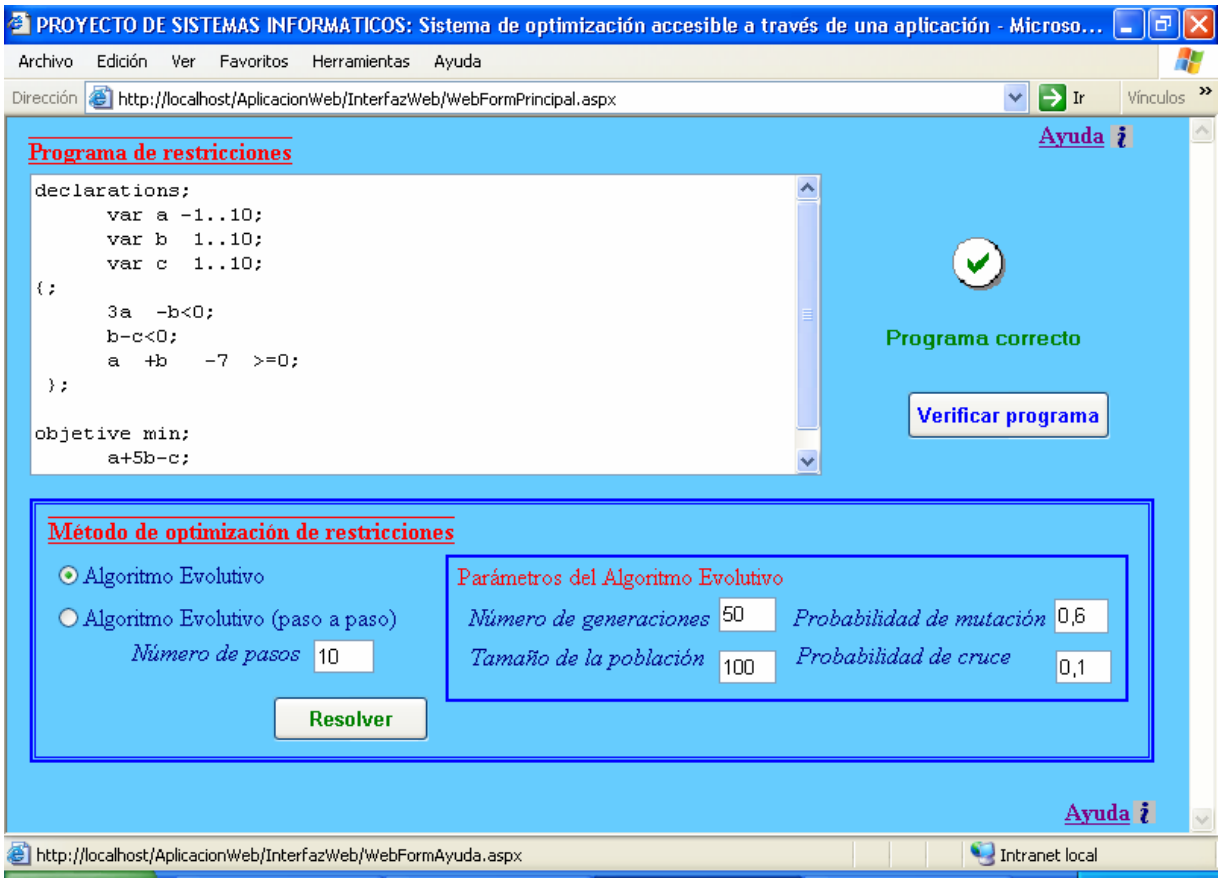


Figura 18.- Programa correcto

Cuando un programa es correcto tras su verificación, se activa la opción de ejecutar, volviéndose disponible el botón "Resolver". Siempre se verifica antes de ejecutar, porque puede ocurrir que un programa resulte correcto, luego se modifique, y como no se puede capturar este evento, la opción de resolver sigue estando disponible. Así que si el usuario pulsa el botón "Resolver" puede volver a la figura 19 o similar, donde el programa es incorrecto.

El panel "Método de optimización de restricciones" es exactamente igual que en la aplicación windows y la aplicación servicio web: permite variar los parámetros del algoritmo genético, permite elegir entre dos opciones de ejecución, directa (opción por defecto) o paso a paso, también se puede elegir el numero de pasos a avanzar en cada salto,...

Si en la figura 18 se pulsa el botón "Resolver", la aplicación muestra la figura 19, donde se muestra el resultado de una ejecución directa del algoritmo genético.

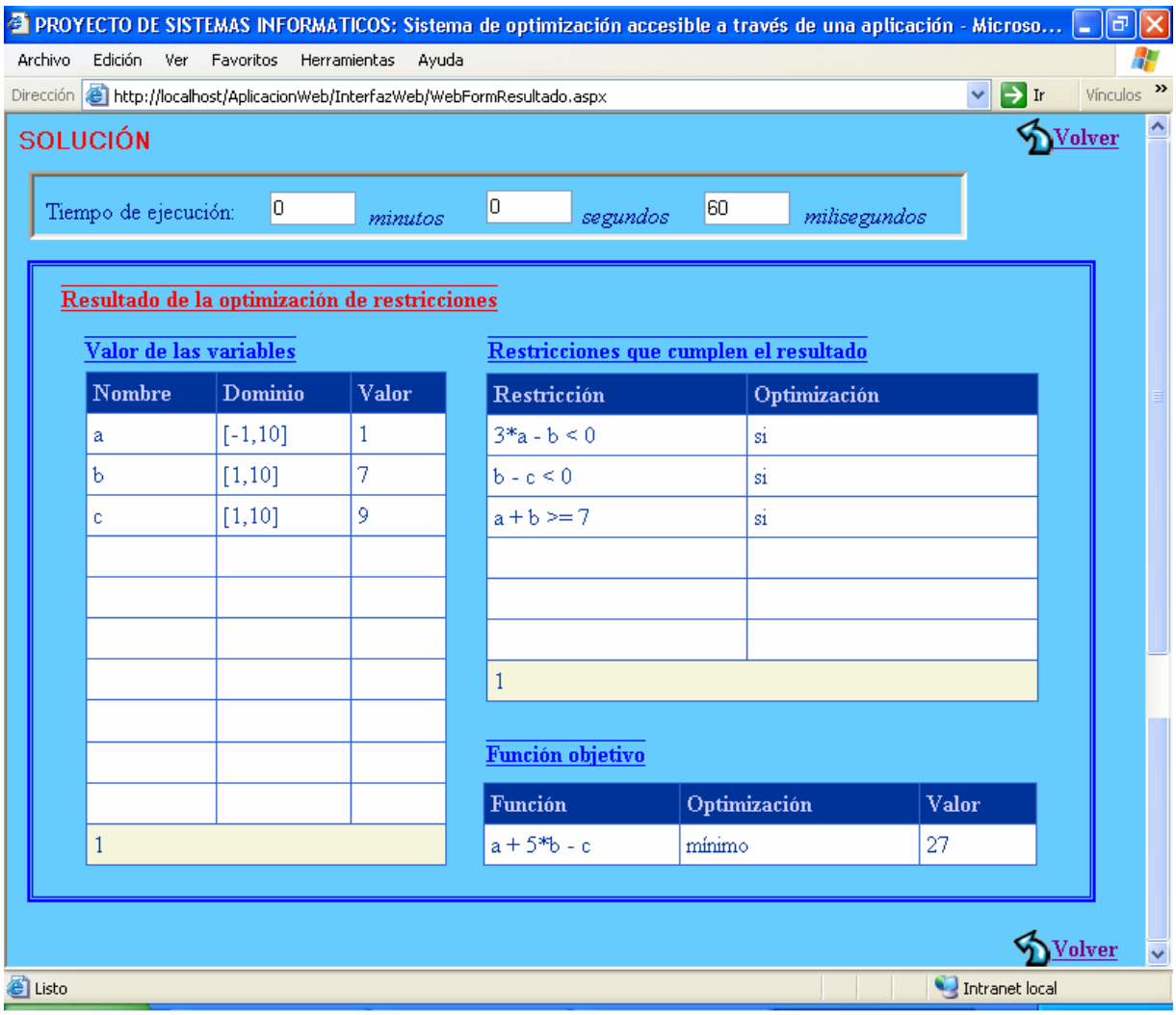


Figura 19.- Resultado ejecución directa

Muestra el tiempo que ha tardado en ejecutarse el algoritmo y muestra la información de la solución y del problema de restricciones de entrada en tres tablas, de igual manera que en las aplicaciones de formularios windows. En el caso de que hubiera muchas variables declaradas (más de 10) o muchas restricciones definidas (más de seis), las tablas relacionadas con las variables y con las restricciones permiten paginación, pudiendo acceder a la página por el índice dispuesto en el pie las tablas, respectivamente.

Si en la figura 18 cambiamos la forma de ejecución del algoritmo genético, seleccionando el radiobutton "Algoritmo Evolutivo (paso a paso)", y después pulsamos el botón resolver, la aplicación mostrará la figura 20. En ella esta representada la correspondencia entre el mejor individuo de la generación inicial y el valor de las variables en el paso 0 de la ejecución. Aquí también es válido todo lo explicado para las aplicaciones windows y servicio web.

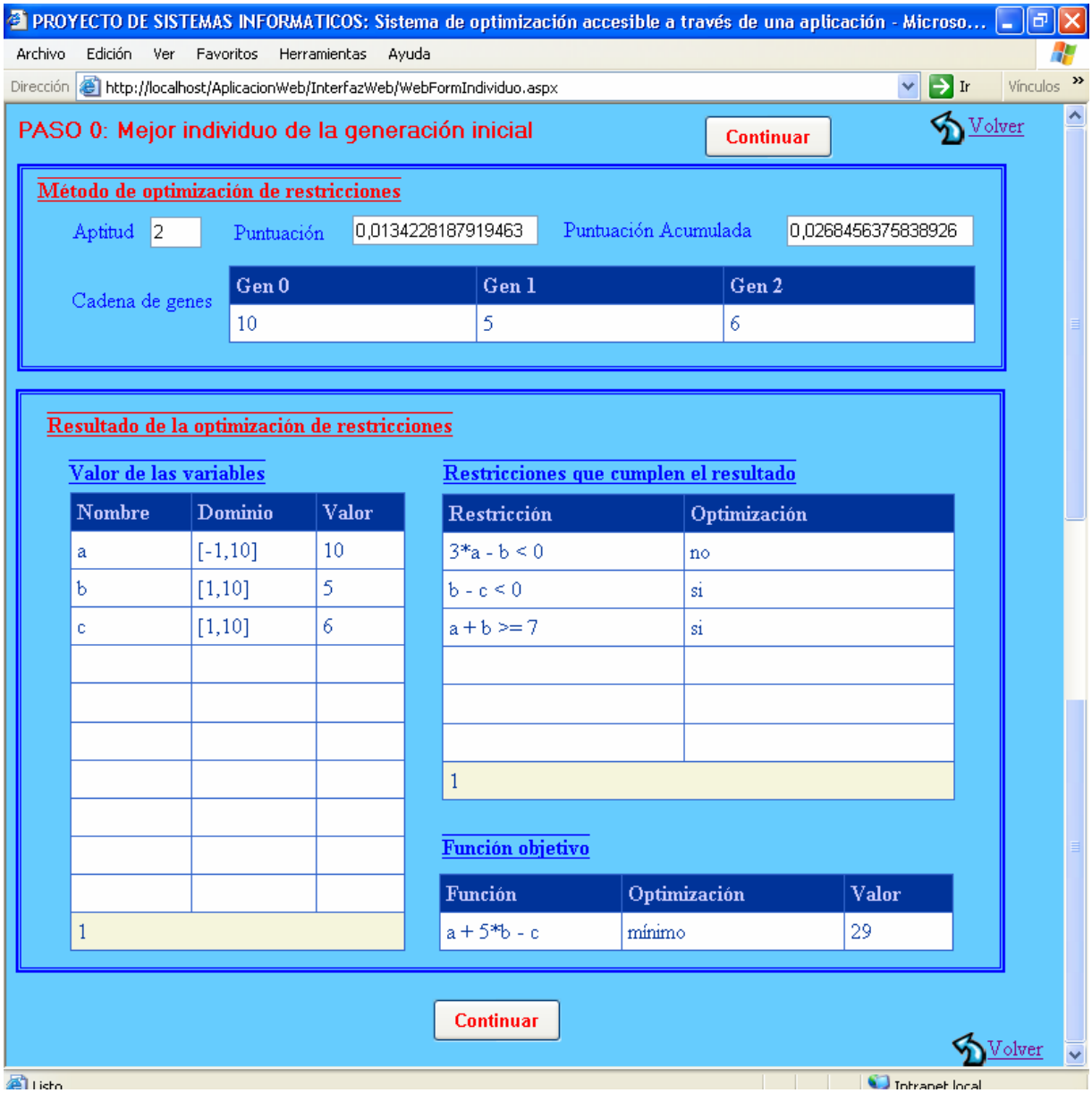


Figura 20.- Paso 0 de la ejecución paso a paso

Pulsando el botón "Continuar" situado al inicio y al final de la página de la figura 20, se mostrará el formulario del primer paso. Sucesivamente el segundo paso, tercer paso... así hasta el último paso reflejado en la figura 21, en la que ya no aparece el botón "Continuar". Se sigue el mismo mecanismo que el explicado en el apartado anterior para esta forma ejecución.

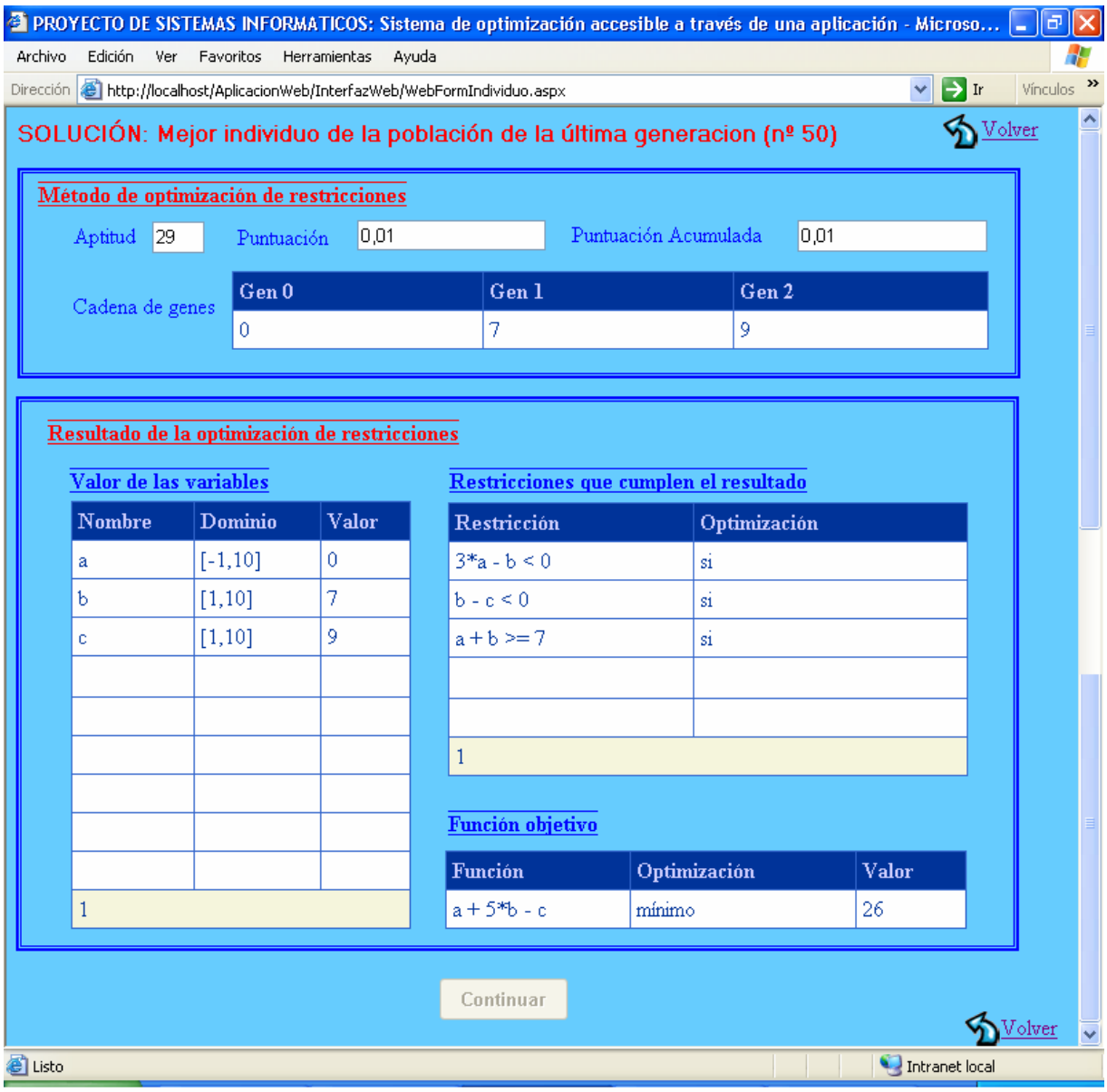


Figura 21.- Último paso de la ejecución paso a paso

Si en la figura 19, 20 ó 21 se pulsa el link "Volver", se regresa a la figura 14.

La aplicación se puede terminar en cualquier momento cerrando el navegador, sea cual sea la página en la que nos encontremos.

10.- **Bibliografía**

1. "A Platform for Web Services", Mary Kirtland,
Microsoft Developer Network
http.//msdn.microsoft.com

2. "Web Services: Beyond the Hype", Vaughan-Nichols,
IEEE Computer, Febrero 2002.

3. "XML, el lenguaje universal", Ramón Montero,
Manual formative, Acta 13, 1999.

4. ".NET is coming", Meyer B.,
IEEE Computer, Agosto 2001.

5. Microsoft .NET white papers.
http://www.microsoft.com/net/

7. "Microsoft .NET: Construyendo la 3ª generación de
Internet", Monográfico Byte Nº 3, MKM Publicaciones,
2001.

8. "Programming with Constraints. An introduction", K. Marriot and P.J. Stuckey.
MIT Press, 1998.

9. "The OPL Optimization Programming Language", Pascal Van Hentenryck,
MIT Press, 1999.

10. "Genetic Algorithms + Data Structures = Evolution Programs", Zbigniew
Michalewicz. Springer, 1999.

11. " Computer Algorthms, Introduction to Design and Analysis", Baase S.,
Addision Wesley.

12. "Program Construction and Verification", Backouse R. C.,
Prentice-Hall.