



**Universitat Autònoma
de Barcelona**

Generación de interfaces para aplicaciones por línea de comandos

Memoria del proyecto
de Ingeniería Técnica en
Informática de Sistemas

realizado por

Marta Río García

y dirigido por

Gonzalo Vera Rodríguez

Escuela de Ingeniería

Sabadell, Septiembre de 2011

El abajo firmante, **Gonzalo Vera Rodríguez**,
profesor de la escuela de ingeniería de la UAB,

CERTIFICA:

Que el trabajo al que corresponde la presente memoria ha sido realizado bajo su dirección
por

Marta Río García

Y para que conste firma la presente.
Sabadell, Septiembre de 2011

Firmado: Gonzalo Vera Rodríguez

FICHA DEL PROYECTO

Título del proyecto:

Generación de interfaces para aplicaciones por línea de comandos

Autora: Marta Ríó**Fecha:** Septiembre 2011**Tutor:** Gonzalo Vera**Titulación:** Ingeniería Técnica en Informática de Sistemas

Palabras clave

• Català:

- XML - HTML
- ASP.NET - C#

• Castellà:

- XML - HTML
- ASP.NET - C#

• Anglès:

- XML - HTML
- ASP.NET - C#

Resumen del proyecto

• Català:

Generació dinàmica de interfícies web basades en fitxers descriptius XML per al control de la parametrització complexa i execució de programes per línia de comandes. La necessitat sorgeix amb l'aplicació mlcoalsim, utilitzada pels investigadors de la UAB, la parametrització de la qual requereix la edició manual d'un fitxer de text amb una sintaxi complicada i feixuga. Amb la generació d'interfícies web es pretén ajudar als usuaris en la correcta parametrització i execució d'aplicacions com mlcoalsim.

• Castellà:

Generación dinámica de interfaces web basadas en ficheros descriptivos XML para el control de la parametrización compleja y ejecución de programas por línea de comandos. La necesidad surge con la aplicación mlcoalsim, utilizada por investigadores de la UAB, cuya parametrización requiere la edición manual de un fichero de texto la sintaxis del cual es complicada y pesada. Con la generación de interfaces web se pretende ayudar a los usuarios en la correcta parametrización y ejecución de aplicaciones como mlcoalsim.

• Anglès:

Dynamic generation of web interfaces based on XML descriptive file to control the configuration and execution of complex programs by command line. The need arises mlcoalsim application used by UAB researchers, whose parameterization requires manually editing a text file which syntax is complicated and heavy. With the generation of web interfaces are intended to assist users in that the parameters and running applications as mlcoalsim.

ÍNDICE

| | | |
|-------|---|----|
| 1 | Introducción | 1 |
| 1.1 | Antecedentes y motivación..... | 1 |
| 1.2 | Objetivo | 1 |
| 1.3 | Estado del arte..... | 2 |
| 1.4 | Estructura de la memoria..... | 3 |
| 2 | Estudio de viabilidad | 4 |
| 2.1 | Introducción | 4 |
| 2.2 | Contexto | 4 |
| 2.3 | Objetivo | 5 |
| 2.4 | Objeto..... | 5 |
| 2.4.1 | Perfil de usuario | 6 |
| 2.5 | Diagrama del sistema | 7 |
| 2.6 | Posibles alternativas..... | 8 |
| 2.7 | Planificación | 8 |
| 2.7.1 | Diagrama de Gantt | 11 |
| 2.8 | Evaluación de riesgos y contingencias | 11 |
| 2.9 | Análisis de costes – beneficios | 12 |
| 2.9.1 | Recursos técnicos | 12 |
| 2.9.2 | Recursos humanos | 13 |
| 2.9.3 | Beneficio..... | 14 |
| 2.10 | Conclusión | 16 |
| 3 | Análisis..... | 17 |
| 3.1 | Sistema actual | 17 |
| 3.2 | Requisitos funcionales..... | 19 |
| 3.3 | Requisitos no funcionales..... | 21 |
| 3.4 | Marco tecnológico..... | 21 |
| 3.5 | Arquitectura de la aplicación..... | 25 |
| 4 | Diseño..... | 26 |
| 4.1 | Interfaz dinámica..... | 26 |
| 4.2.1 | Diseño XML..... | 26 |
| 4.2.2 | Diseño interfaz | 28 |
| 4.2 | Framework | 29 |
| 4.3.1 | XMLManager | 30 |
| 4.3.2 | FileManager..... | 31 |
| 4.3.3 | ValidatorManager..... | 32 |

| | | |
|-------|---|----|
| 4.3.4 | Gramática | 32 |
| 5 | Implementación Y TEST | 35 |
| 5.1 | Interfaz dinámica..... | 35 |
| 5.1.1 | Formulario | 36 |
| 5.1.2 | Diseño..... | 38 |
| 5.1.3 | Validación de parámetros | 39 |
| 5.2 | Gestión de archivos | 42 |
| 5.3 | Test | 44 |
| 5.4 | Puesta en marcha..... | 45 |
| 6 | Conclusiones..... | 49 |
| 6.1 | Seguimiento del proyecto | 49 |
| 6.2 | Consecución de objetivos y posibles mejoras | 49 |
| 6.3 | Valoración personal..... | 51 |
| 7 | Bibliografía y referencias..... | 52 |
| 7.1 | Libros | 52 |
| 7.2 | Referencias en internet..... | 52 |

ÍNDICE DE FIGURAS

| | | |
|------------|--|----|
| Figura 1. | Esquema donde se puede ver la petición al servidor para su ejecución..... | 7 |
| Figura 2. | Esquema donde se puede ver la petición de descarga de ficheros..... | 7 |
| Figura 3. | Diagrama de Gantt | 11 |
| Figura 4. | Ejecución por línea de comandos de la aplicación mlcoalsim..... | 17 |
| Figura 5. | Caso de uso correspondiente la carga de un formulario desde archivo local..... | 20 |
| Figura 6. | Caso de uso correspondiente a creación de archivo. | 20 |
| Figura 7. | Figura. Caso de uso correspondiente al download del archivo resultado. | 20 |
| Figura 8. | Evolución Visual Studio | 22 |
| Figura 9. | Capas del lenguaje Mono | 24 |
| Figura 10. | Arquitectura Mono | 24 |
| Figura 11. | Arquitectura aplicación | 25 |
| Figura 12. | Diagrama Framework..... | 30 |
| Figura 13. | Formulario | 35 |
| Figura 14. | Diagrama clase XMLManager | 37 |
| Figura 15. | Fila generada a partir del XML | 38 |
| Figura 16. | Icono de información del parámetro | 38 |
| Figura 17. | Formulario con cambio de estilo..... | 39 |
| Figura 18. | Diagrama clase XMLManager y ValidatorManager | 40 |
| Figura 19. | Iconos validación. | 42 |
| Figura 20. | Diagrama de la clase FileManager. | 43 |
| Figura 21. | Carga de archivos. | 43 |
| Figura 22. | Estructura de carpetas. | 48 |

1 INTRODUCCIÓN

1.1 Antecedentes y motivación

Actualmente en varios departamentos de investigación en el ámbito universitario se utilizan aplicaciones por línea de comandos para realizar estudios y simulaciones. Estas aplicaciones han sido desarrolladas por los propios investigadores para facilitar su tarea y automatizar algunos procesos de cálculo cuyo coste es muy elevado.

El principal problema que conlleva el uso de estas aplicaciones es la gran cantidad y complejidad de los parámetros que necesitan. Éstos son de múltiples formatos y además pueden existir dependencias entre ellos lo que convierte la necesidad de establecer estos parámetros en una tarea larga y complicada, no exenta de errores al realizarse de forma manual. Es en la resolución de esta problemática donde se ha decidido centrar este proyecto, tomando como ejemplo una de estas aplicaciones: mlcoalsim.

La principal motivación de este proyecto ha sido la de crear una aplicación dinámica que permita facilitar la tarea de introducir los parámetros necesarios para la ejecución de la aplicación mlcoalsim.

Al tratarse de una aplicación dinámica, se vio que esta podría utilizarse también para otras aplicaciones con igual problemática, simplemente modificando los parámetros requeridos para adaptarlos a las necesidades de cada aplicación. Este planteamiento ha hecho más complicado todo el desarrollo, ya que, aunque se partía de la idea de un "formulario dinámico" ha sido necesario dotarlo de un dinamismo más grande, convirtiendo este objetivo en el centro motivador del proyecto.

1.2 Objetivo

El objetivo de este proyecto es realizar un interfaz dinámica y atractiva que facilite la introducción de los parámetros necesarios para la ejecución de aplicaciones por línea de comandos. La interfaz será una página web en forma de formulario donde se podrán introducir los parámetros necesarios y donde se realizarán las validaciones necesarias para la correcta ejecución.

Como se ha comentado anteriormente el principal objetivo será dotar de dinamismo y flexibilidad esta web, para poder adaptarla a diferentes aplicaciones y para facilitar la inclusión de nuevos parámetros teniendo en cuenta posible nuevos avances en el estudio de las teorías y cálculos utilizados.

El desarrollo de este proyecto se va centrar en una aplicación que cuenta con todas las necesidades expuestas: mlcoalsim, aplicación utilizada para realizar simulaciones genéticas.

1.3 Estado del arte

Desde hace años la informática está indisolublemente unida a las interfaces gráfica, ya que los sistemas gráficos han ocasionado grandes consecuencias en la industria del software y el hardware. Las interfaces gráficas surgen de la necesidad de hacer los ordenadores más accesibles para el uso de los usuarios comunes ya que anteriormente eran necesarios conocimientos técnicos para usos más específicos de los ordenadores personales. Esta limitación fue salvada gracias al desarrollo de los entornos gráficos que permitieron que las personas pudieran acceder a un ordenador sin tener que pasar por el proceso de aprender a manejar un entorno bajo línea de comandos.

En los últimos años se ha avanzado mucho en el diseño de interfaces, haciéndolas cada vez más atractiva e intuitivas. Sin embargo, existen todavía muchas aplicaciones que funcionan por línea de comandos con todo lo que ello implica sobretodo en la complejidad de conocer y revisar los parámetros para una correcta ejecución. Todas estas aplicaciones pueden adaptarse a un interfaz gráfica, ya las tecnología actuales permiten establecer diferentes capas independientes, aislando la parte de la interfaz del código que ha de ejecutarse. Lógicamente esto conlleva un trabajo específico de desarrolladores para realizar la adaptación específica para cada aplicación, lo que implica un coste económico más elevado.

La solución que se propone en este proyecto está basada en la versatilidad de los archivos XML y la capacidad de interpretarlos a través de código. El dinamismo del que se quiere dotar a la aplicación se establece en este archivo XML, donde se establecerán los parámetros, el tipo, las dependencias e incluso el estilo que se quiere aplicar a la web. El framework o código se encargará de leer este XML y realizar las acciones que en este se le indican a la vez que construye una interfaz con los parámetros indicados. Una vez establecido el formato del XML está podrá adaptarse a otras aplicaciones de similar funcionamiento, lo que permite más agilidad y facilidad para crear esta interfaces, siendo necesario únicamente el diseño del XML.

La aplicación mlcoalsim es totalmente específica para el estudio del genoma. Como alternativa a esta existen otras aplicaciones que pueden usar los usuarios para realizar las simulaciones necesarias como por ejemplo: msABC, SIMCOAL2, y ms.

Todas han sido diseñadas desde departamentos universitarios y utilizan el mismo tipo de simulaciones. Existen más aplicaciones con la que obtener este tipo de resultados pero que realizan otras funciones más sofisticadas con similares aproximaciones. Esto implica todavía más especialización en su uso, más tiempo de procesado para obtener resultados similares que quizás no compensen este tiempo extra dedicado.

Las tres aplicaciones indicadas (msABC, SIMCOAL2, y ms) que coinciden en tipos de usuarios, tienen un funcionamiento muy similar a nuestra mlcoalsim. Todas se basan en línea de comandos y en la introducción manual de los parámetros, uno por uno o leyendo de un fichero de entrada. Quizá donde existan más diferenciales en la salida, en cómo se muestran los resultados obtenidos ya que algunos utilizan soluciones más gráficas que otras.

De momento ninguno cuenta con una interfaz web a la que se pueda acceder con una simple conexión a internet.

Como principal objetivo de este proyecto, se plantea no sólo la creación de una página web, sino de dotarla de flexibilidad y dinamismo. Para aplicaciones de este tipo es importante la introducción de nuevos parámetros, para adaptarla a nuevos avances en el estudio de estas teorías genéticas.

1.4 Estructura de la memoria

La documentación del proyecto se estructura en capítulos organizando y esquematizando los puntos clave más importantes que la caracterizan.

En el primer capítulo haremos una introducción al proyecto. En el segundo capítulo se realizará el Estudio de Viabilidad estableciendo el objetivo particular del proyecto así como la planificación temporal del mismo, hasta concluir si el proyecto es viable. El tercer capítulo tratará el Análisis, así que se realizará un estudio en profundidad de las necesidades actuales y se especificarán los requerimientos del sistema propuesto. Durante el cuarto capítulo se describirá el diseño técnico del proyecto. En el quinto capítulo se mostrarán los resultados de la nueva funcionalidad así como las pruebas que se han realizado para garantizarla. Las conclusiones y mejoras se reflejarán en el sexto punto.

A modo de complemento bibliográfico en el séptimo apartado se recopilarán las referencias bibliográficas de consulta y otros recursos utilizados.

2 ESTUDIO DE VIABILIDAD

2.1 Introducción

En este capítulo se va a realizar el estudio de viabilidad del proyecto tanto a nivel funcional, económico y temporal. Para ello se realizará una pequeña introducción sobre la solución propuesta y pasaremos a analizarla desde diferentes puntos de vista.

La aplicación mlcoalsim (Multilocus Coalescent Simulations) es una aplicación utilizada por el Departamento de Ciencia Animal y de Alimentos de la UAB y dirigida a los investigadores interesados en la evolución molecular del ADN de diferentes genomas. El programa, diseñado por un investigador del departamento, utiliza diferentes teorías para simular datos genéticos según diferentes modelos evolutivos. A partir de estas simulaciones es posible obtener la distribución empírica de una serie de estadísticas a través de una amplia gama de módulos evolutivos. Estas distribuciones se pueden utilizar para poner a prueba hipótesis evolutivas a partir de datos experimentales.

2.2 Contexto

El entorno de utilización de la aplicación mlcoalsim se sitúa en departamentos de investigación universitaria, enfocadas al estudio de la evolución molecular y el genoma. El usuario final responde a un perfil muy claro, se trata de un investigador interesado en realizar una serie de simulaciones para poder llegar a conclusiones en su trabajo. Para estos usuarios el tiempo del que disponen para realizar las investigaciones es importante por lo que el tiempo de acceso a estos resultados debería ser más ágil. Son cálculos muy complejos basados en distribuciones genéticas. Ya que es difícil reducir el tiempo de estos cálculos, mediante este proyecto, se conseguirá que ahorren aquel dedicado a introducir los datos para el estudio.

El tiempo para crear un de estos archivo puede moverse entre los 15 minutos y las 2 horas, dependiendo del nivel de familiarización con el uso de la aplicación y la complejidad del fichero que deba realizarse. Los usuarios disponen de un manual de instrucciones que les indica cómo realizarlo, y aun así, es difícil asegurar que todos los datos introducidos son los correctos.

2.3 Objetivo

El objetivo de este proyecto es la realización de una interfaz gráfica para la aplicación ya existente, mlcoalsim, que actualmente funciona desde línea de comandos, con las dificultades de usabilidad que esto comporta. El proyecto se va a centrar en esta aplicación pero se ha planteado de forma que la solución pueda ser reutilizable para aplicaciones con características similares que también funcionan por línea de comandos.

El funcionamiento de mlcoalsim se basa en la gestión de un fichero de entrada que contiene los parámetros necesarios para cada tipo de simulación. Actualmente la creación de este fichero de entrada se realiza de forma manual, lo que dificulta y ralentiza su utilización.

La interfaz se realizará en formato de formulario web y la dotaremos de un dinamismo extra creando los controles para la introducción de parámetros en tiempo de ejecución desde un archivo XML. En este archivo se incluirán también las validaciones necesarias para cada parámetro así como las dependencias establecidas. La web se subirá a un servidor para de esta manera poder facilitar su distribución y también su mantenimiento. Esto permitirá al administrador de la aplicación ahorrar tiempo y dinero en el mantenimiento de la web, ya que, si necesita incluir un nuevo parámetro no necesitará rediseñar el código.

Cómo ampliación de este proyecto, se podría plantear el tratamiento de la salida de esta aplicación, del resultado, haciéndolo más gráfico y tratable para las personas que deben interpretarlo.

2.4 Objeto

Como punto de partida tenemos una aplicación diseñada por el departamento de Ciencia Animal y Alimentos, con usuarios potenciales dentro de la rama de investigación a la que pertenece. Este departamento colabora en el proyecto proporcionándonos los requerimientos y líneas a seguir para conseguir el objetivo planteado.

Dada la alta especialización de los usuarios en lo que respecta a la finalidad del programa, dicho código se mantendrá intacto, siendo en el caso de este proyecto algo totalmente transparente. Podríamos decir, que lo que en este proyecto se plantea es diseñar la "cáscara" de esta aplicación, que permitirá acceder a ella de forma más fácil e intuitiva.

Los usuarios podrán acceder a la aplicación desde cualquier ordenador con acceso a internet y pretende ser compatible con todos navegadores mayoritarios.

La interfaz interactuará directamente con la aplicación haciendo la llamada correspondiente y pasándole el fichero de entrada creado por el usuario. Esta interfaz contará con un sistema de validaciones para cada parámetro introducido, asegurándonos así que la aplicación

recibirá un fichero con datos coherentes, al menos en lo que al formato se refiere. La aplicación cuenta con un sistema propio de validación, dando un error si al realizar la ejecución el fichero de entrada no es correcto..

La web se diseñará basándonos en un sistema de Menús donde el usuario podrá gestionar el fichero de entrada: crear, editar, cargar y guardar. Contará con varios módulos para los diferentes tipos de parámetros definidos.

Para dotar de flexibilidad a esta web, se basará en un formulario cargado, en tiempo de ejecución, desde un archivo XML. Este archivo XML será la base del diseño de la web, y en él se incluirán los diferentes controles que la formarán y que permitirán la introducción de los parámetros.

Si en un momento el administrador necesita crear, eliminar o modificar algún parámetro sólo será necesario modificar este XML. Para que pueda realizar esta operación, se creará también una opción de menú que le permita hacerlo sin tener que modificar directamente el XML. En este XML se indicarán también las reglas de validación como por ejemplo validar el formato o comprobar que su valor es coherente.

2.4.1 Perfil de usuario

La aplicación está destinada a un usuario cuyo perfil se corresponde con un investigador de datos genéticos interesado en realizar unas simulaciones para obtener resultados aplicables a sus estudios.

Entre estos usuarios podríamos crear a partir de este proyecto dos tipos de usuarios: administrador y usuario. El usuario administrador, gran conocedor del funcionamiento interno de la aplicación tendría las siguientes funciones.

- Creación del XML para la publicación de la aplicación utilizando la interfaz propuesta. El diseño del XML implica seguir las pautas establecidas en este proyecto para la creación de los diferentes controles para recoger los parámetros y seguir la gramática propuesta para realizar las validaciones.
- Añadir o eliminar nuevos parámetros de entrada según se modifique la aplicación para añadir nuevas simulaciones.

El resto de usuarios se corresponden con aquellos que desean utilizar la aplicación web para la ejecución de la aplicación. Éstos utilizarían las opciones de la interfaz para crear el archivo de entrada y poder guardarlo. También podrían recuperar alguno ya creado para volver a utilizarlo o para modificarlo según las necesidades.

2.5 Diagrama del sistema

La arquitectura de la solución propuesta se basa en la interacción entre el servidor, donde se encuentra la aplicación, donde se realizarán los cálculos y donde se dejará la solución y la interfaz que interactúa con el usuario para la introducción de los parámetros.

En el siguiente esquema (Figura 1) podemos ver cómo el usuario, a través de la interfaz, genera el fichero de entrada y hace la petición al servidor para su ejecución:

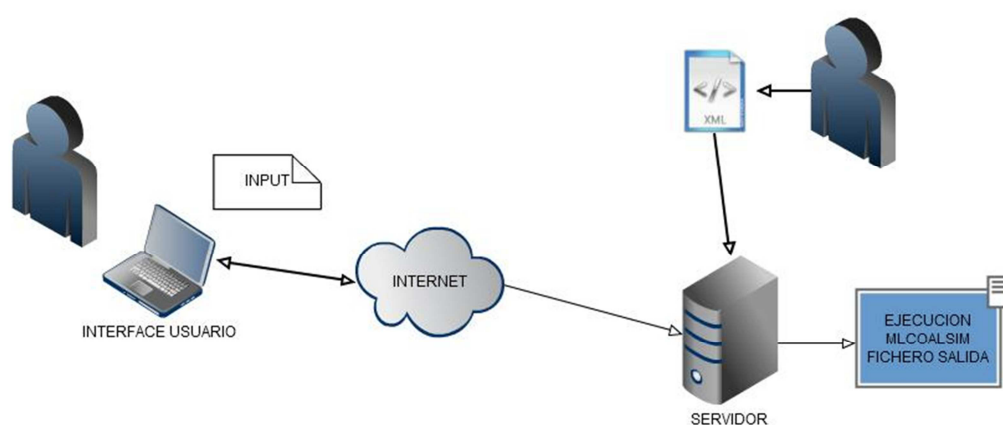


Figura 1. Esquema donde se puede ver la petición al servidor para su ejecución

En el siguiente (Figura 2) se puede observar la petición que hace el usuario, una vez ejecutada la aplicación y creado el archivo de salida resultado de la ejecución, poder descargárselo:

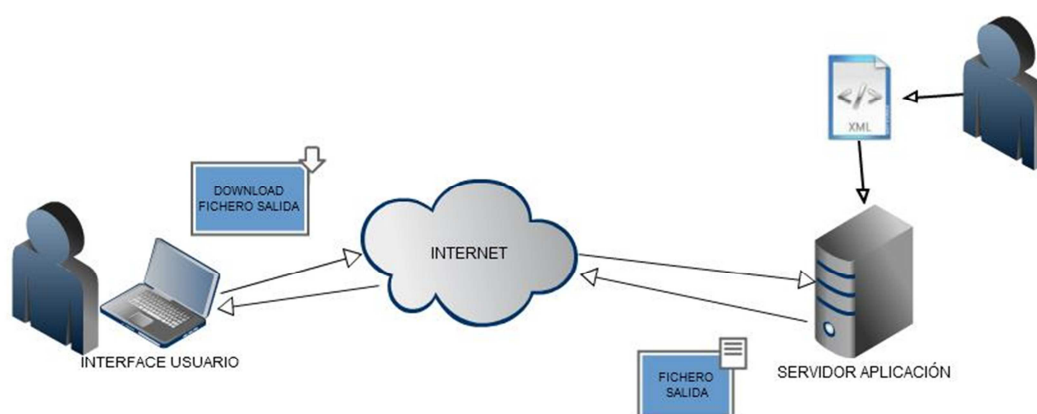


Figura 2. Esquema donde se puede ver la petición de descarga de ficheros

2.6 Posibles alternativas

Para el desarrollo de este proyecto podrían utilizarse diferentes entornos que permitan la creación de páginas web como Java, PHP, Ruby, etc. Todos ellos tienen sus pros y sus contras, defensores y detractores, y no entraremos en esta memoria en el debate sobre cual ofrece mejores resultados.

El principal y claro inconveniente de Visual Studio 2008, entorno que utilizaremos en este proyecto, es sin duda que sólo es compatible con entornos Microsoft. Sin embargo, me he decidido a utilizarlo ya que es el entorno con el que estoy más familiarizada, y el tiempo es oro. De esta manera con el tiempo que nos ahorraremos en formación y en la familiarización con otras formas de programación, podremos dedicarlo a mejorar, por ejemplo, la apariencia de la página web o a mejorar el rendimiento, dado que contamos con un plazo de tiempo determinado.

Tampoco hubiera sido posible la utilización de Visual Studio sino pudiéramos contar con el "Proyecto Mono". Esta herramienta provee del software necesario para desarrollar y ejecutar aplicaciones .NET tanto en el lado del cliente, como en el lado del servidor, sobre Linux., Solaris, Mac OS X, y Windows. Si no existiera, claramente hubiéramos tenido que optar por alguna otra de las opciones expuestas anteriormente ya que, como requerimiento no funcional inicial se establece que como servidor utilizaremos Linux. Este es un punto que también plantea alternativas ¿por qué no hacer correr la aplicación y la página web sobre un servidor Windows? Por lo visto se necesita una gran capacidad de cálculo para realizar las simulaciones y para asegurar un correcto funcionamiento en el caso de usuarios concurrentes, como sería el caso de subirlo a una web y parece que Linux ofrece una mejor respuesta calidad-precio para estas necesidades.

Y además es muy importante ofrecer una solución que se ajuste a la metodología de trabajo del departamento y a la filosofía del creador de la aplicación.

2.7 Planificación

En este punto se va a exponer el plan de proyecto, un aspecto muy importante a tener en cuenta. Es de esperar que se produzcan desviaciones dentro de este plan, pero lo importante es tenerlo siempre actualizado y poder actuar en consecuencia (realizar más horas si es necesario, cambiar plazos de entrega, contratación de más personal, etc.).

En este caso, al tratarse de un proyecto individual de final de carrera, yo seré el único recurso. Los días se definen como unidades de 4 horas ya que he de hacer compatible mi vida laboral con la elaboración de este proyecto.

A continuación expongo la planificación total del proyecto, dividida en los módulos principales:

- Documentación.

Durante esta fase se recogerá la información necesaria para el diseño de la interfaz. La documentación específica referente a la aplicación mlcoalsim ha sido facilitada por el investigador del departamento de Ciencia Animal de UAB que ha diseñado la aplicación. En ella se detalla uno por uno los parámetros que puede recibir la aplicación y sus características formales (tipo y dependencias).

También durante esta fase se pretende conocer más a fondo las tecnologías utilizadas: Mono y Visual Studio, para establecer los pasos que seguirá el desarrollo y las herramientas que podemos utilizar que mejor se adapten a nuestras necesidades.

- Análisis.

Mediante el análisis se realizará el estudio de viabilidad del proyecto, estableciendo una solución final a implementar y comprobando si esta será viable tanto a nivel económico como técnico. Es importante también establecer el tiempo que necesitaremos para implementar la solución y si este tiempo es asimismo viable.

Durante esta fase se establecerán también las diferentes partes en las que dividiremos el diseño final de la aplicación

- Diseño.

Se ha dividido esta fase en tres partes principales: diseño del XM y diseño de la interfaz.

1. Diseño XML.

Se definirá la forma de introducir los controles para la introducción de los parámetros. Será necesario establecer las reglas para que después se pueda adaptar a otras aplicaciones.

2. Diseño interfaz.

Aunque la interfaz será dinámica, es necesario establecer un diseño "base" sobre el que se dibujen los controles indicados en el XML. El menú de gestión de archivos formará parte de este diseño "base" ya que no dependerá del XML.

- Implementación y test.

Sin duda, la parte más complicada en cualquier desarrollo. Durante esta fase se implementará la funcionalidad básica de la aplicación: tratamiento del XML, gestión de archivos y gestión de validaciones.

- Redacción de la memoria.

La redacción de la memoria es una tarea que se va realizando durante los meses de trabajo en el proyecto para tener soporte escrito del trabajo realizado. Al final

del proceso debe realizarse una revisión para garantizar que el documento está completo y realizar la entrega final.

- Exposición oral.

Son días reservados para estructurar la exposición oral y para la preparación de los puntos más interesantes del proyecto a explicar.

A continuación mostramos un tabla con el desglose de tareas y la horas dedicadas:

| Nombre de tarea | Duración | Comienzo |
|---|------------------|---------------------|
| Generación de interfaces | 185 horas | lun 15/11/10 |
| Fase documentación | 28 horas | lun 15/11/10 |
| Toma de requerimientos | 8 horas | lun 15/11/10 |
| Recopilación de información | 6 horas | vie 19/11/10 |
| Familiarización tecnologías utilizadas | 10 horas | mié 24/11/10 |
| Elección e investigación de soluciones aplicables | 4 horas | mié 01/12/10 |
| Fase de análisis | 13 horas | vie 03/12/10 |
| Estudio de viabilidad | 8 horas | vie 03/12/10 |
| Análisis y asimilación de los parámetros necesarios para el correcto funcionamiento de la aplicación. | 5 horas | jue 09/12/10 |
| Fase de diseño | 58 horas | mar 14/12/10 |
| Pruebas iniciales configuración dinámica del formulario web | 5 horas | mar 14/12/10 |
| Diseño XML | 12 horas | vie 17/12/10 |
| Diseño Interficie | 10 horas | sáb 25/12/10 |
| Diseño menú | 12 horas | sáb 01/01/11 |
| Diseño de los diferente módulos | 16 horas | mié 12/01/11 |
| Menú tratamiento fichero de entrada | 7 horas | mié 12/01/11 |
| Opción New | 2 horas | mié 12/01/11 |
| Opción Open | 2 horas | jue 13/01/11 |
| Opción Edit/Update | 2 horas | vie 14/01/11 |
| Resto opciones | 1 hora | lun 17/01/11 |
| Entrada de parámetros | 8 horas | mar 18/01/11 |
| Defining Basic Data | 2 horas | mar 18/01/11 |
| Evolution models and parameters | 2 horas | mié 19/01/11 |
| Simulation Parameters | 2 horas | jue 20/01/11 |
| Include Experimental Statistics in Simulations | 2 horas | vie 21/01/11 |
| Implementación y test | 48 horas | sáb 22/01/11 |
| Codificación tratamiento ficheros | 10 horas | sáb 22/01/11 |
| Codificación validación parámetros | 13 horas | sáb 29/01/11 |
| Codificación funcionalidad interfaz + mslcoalsim | 11 horas | jue 10/02/11 |
| Pruebas | 10 horas | vie 18/02/11 |
| Redacción de la memoria | 30 horas | vie 25/02/11 |
| Presentación del proyecto | 7 horas | vie 18/03/11 |

2.7.1 Diagrama de Gantt

A continuación mostramos el diagrama de Gantt resultante (Figura 3):

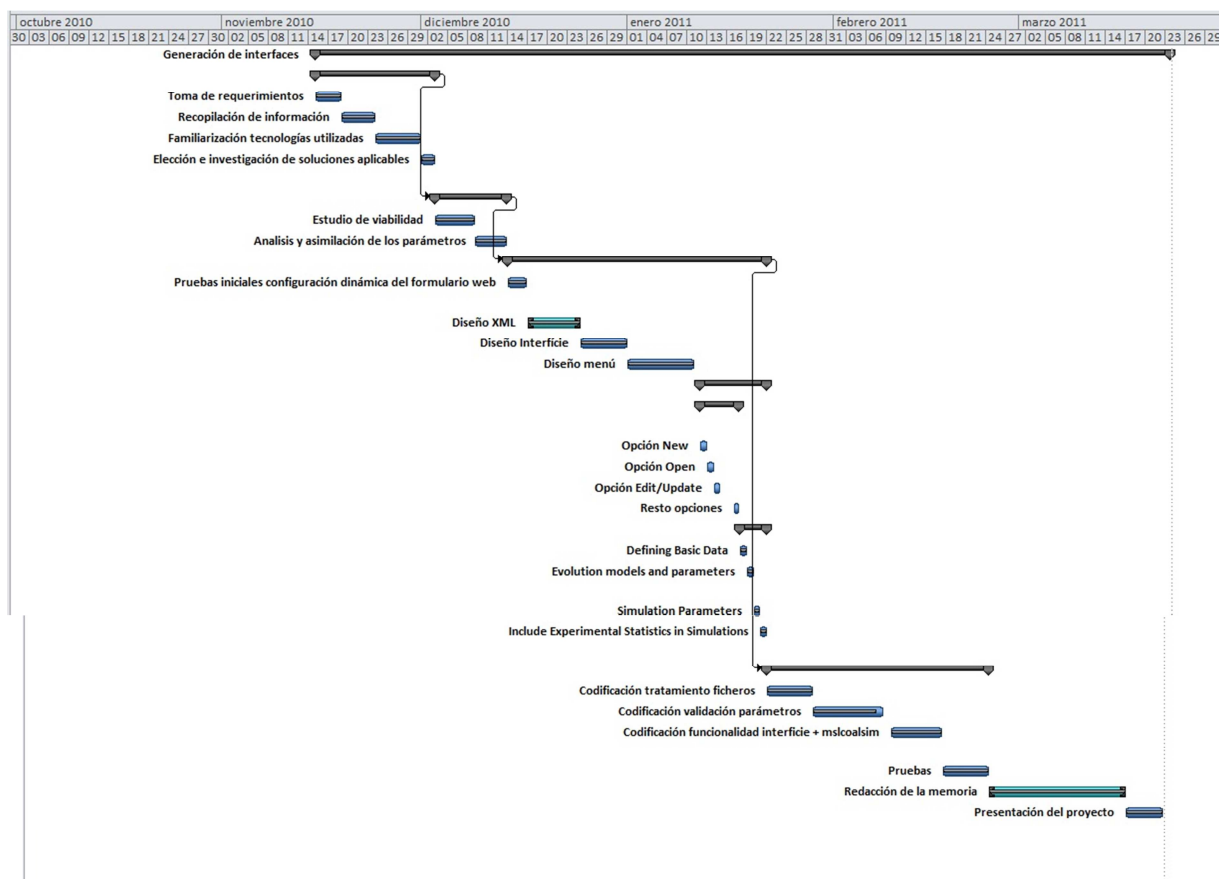


Figura 3. Diagrama de Gantt

2.8 Evaluación de riesgos y contingencias

Para establecer la viabilidad de un proyecto es importante evaluar los riesgos y contingencias y establecer alternativas en el caso de que se produzcan.

El principal riesgo que se plantea en el desarrollo de este proyecto fue la utilización de Mono para compilar el código realizado en Visual Studio y hacerlo compatible con Linux. A pesar de consultar toda la documentación disponible existía el riesgo que no todas las funcionalidades del Framework utilizado funcionaran de la forma esperada en Linux y no se podrá comprobar hasta que el código esté ya avanzado con al menos todas las

funcionalidades definidas e implementadas. En el caso que algo no funcionara como se espera deberíamos optar por MonoDevelop (<http://monodevelop.com/>), entorno que podría ejecutarse directamente sobre Linux con unas mínimas modificaciones sobre el código.

También se contempló como riesgo el no poder reflejar en el archivo que generaría el formulario toda la funcionalidad y flexibilidad necesaria. Esto fue un riesgo que hubo que asumir sin tener en cuenta más alternativas que la no consecución del objetivo principal. Si no se hubiera podido realizar, se hubiera tenido que crear una interfaz web adaptada totalmente a la aplicación mlcoalsim para al menos, solucionar el problema planteado inicialmente.

2.9 Análisis de costes – beneficios

Para establecer el coste tendremos en cuenta los recursos requeridos para la implementación de proyecto. Principalmente son de dos tipos: técnicos y humanos.

2.9.1 Recursos técnicos

En la siguiente tabla estableceremos los recursos técnicos que necesitamos para llevar a cabo la solución propuesta:

| Recurso | Coste Total (€) |
|--------------------|-----------------|
| Servidor Linux | 3.000 |
| Visual Studio 2008 | 0 |
| Mono | 0 |
| TOTAL | 3.000€ |

Para Visual Studio 2008 se dispone de licencias para uso del personal de investigación.

2.9.2 Recursos humanos

Se han establecido 4 tipos de recursos humanos:

| Nombre | Rol | Cost/h (€) | Horas dedicadas | Coste Total (€) |
|-----------|----------------------|------------|-----------------|-----------------|
| Marta_CP | Jefe de proyecto | 60,00 | 30 | 1.800 |
| Marta_AF | Analista Funcional | 50,00 | 46 | 2.300 |
| Marta_AP | Analista-Programador | 40,00 | 100 | 4.000 |
| Marta_TST | Testeador | 30,00 | 22 | 660 |
| TOTAL | | | | 8.100 € |

Detalle de distribución de las horas según los recursos:

| Perfil/Tarea | Horas dedicadas |
|---|-----------------|
| Jefe de proyecto | 30 |
| Toma de requerimientos | 8 |
| Recopilación de información | 6 |
| Fase de análisis | 8 |
| Estudio de viabilidad | 8 |
| Analista funcional | 46 |
| Familiarización tecnologías utilizadas | 5 |
| Elección e investigación de soluciones aplicables | 4 |
| Análisis y asimilación de los parámetros necesarios para el correcto funcionamiento de la aplicación. | 5 |
| Diseño XML | 5 |
| Diseño menú | 2 |

| | |
|---|-----|
| Codificación tratamiento ficheros | 5 |
| Codificación validación parámetros | 15 |
| Codificación funcionalidad interfaz + mslcoalsim | 5 |
| Analista Programador | 100 |
| Familiarización tecnologías utilizadas | 5 |
| Pruebas iniciales configuración dinámica del formulario web | 5 |
| Diseño XML | 25 |
| Diseño menú | 10 |
| Resto opciones | 4 |
| Codificación tratamiento ficheros | 8 |
| Codificación validación parámetros | 30 |
| Codificación funcionalidad interfaz + mslcoalsim | 8 |
| Pruebas iniciales | 5 |
| Testeador | 22 |
| Pruebas iniciales configuración dinámica del formulario web | 9 |
| Pruebas iniciales | 13 |

La inversión inicial para la implantación sería de $8.100 + 3.000 = \mathbf{11.100€}$.

2.9.3 Beneficio

Para calcular los beneficios obtenidos con la implantación de esta solución, hemos de estimar, por encima de todo, el tiempo que se ahorrarán sus usuarios, ya que no es una aplicación que nazca con espíritu comercial, sino para facilitarles el trabajo a estos investigadores.

El tiempo que ahorren con el uso de la aplicación podrán dedicarlo a lo realmente importante en su trabajo, el estudio de los resultados obtenido para poder llegar a conclusiones y poder avanzar en sus investigaciones.

Según estudios recientes la franja de salarios de los investigadores a nivel mundial puede situarse entre los 23.000 y los 40.000€ anuales, dependiendo principalmente del país donde se encuentren (<http://www1.salary.com/Animal-Scientist-I-Salary.html>). Nosotros vamos a establecer un sueldo anual de 30.000 € anuales, y a partir de este salario cuantificaremos las horas de ahorro que puede suponer la utilización de la nueva interfaz.

A continuación detallamos las diferentes tareas relacionadas con el uso de la aplicación y su gasto en horas cada vez que se utiliza. Se establece un precio por hora de 9 €.

| Tareas | Horas dedicadas antes | Horas dedicadas después | Horas ahorradas | Ahorro Total (€) |
|---|-----------------------|-------------------------|-----------------|------------------|
| Consulta manual de usuario | 3 | 0 | 3 | 27 |
| Creación fichero entrada | 2 | 0.5 | 1.5 | 13.5 |
| Detección/corrección errores en el caso de entrada incorrecta | 1 | 0 | 1 | 9 |
| | | | 5.5 h | 49.5 € |

El ahorro en horas de trabajo se traduce en 5.5h semanales (22h mensuales y 242 al año (año de 10 meses)) que suponen 49.5€ por investigador, cuya equivalencia anual sería de 242h (2.178€).

Hemos de sumar a estos costes la inversión previa en formación, que podríamos establecer en unas 30h más de las que necesitarán para el uso de la nueva aplicación.

También sumaremos como ahorro las horas de productividad, es decir, el departamento no "ahorrrará" esas horas, sino que éstas se convertirán en horas productivas para otras funciones más útiles de cara a la investigación.

Ahorro en horas de trabajo..... 2.178 €

Ahorro en formación..... 270 €

Ganancias en productividad 792 €

Total 3.240 €

Un departamento de investigación cuyos trabajadores utilicen para su trabajo la aplicación mlcoalsim, podrán ahorrar después de la implantación de la solución propuesta en este proyecto, durante el primer año 3.240€ por investigador. Si establecemos que un departamento de tamaño media tiene un total de 5 investigadores, el beneficio durante el primer año sería de 16.200 € por departamento.

Estableceremos el número de usuarios potenciales de la aplicación, teniendo en cuentas la facilidad de acceso y de uso de la que dispondrá, en unos 1.000 mundiales, durante el primer año (con expectativas de ampliarlos en años sucesivos) aunque en este proyecto nos centraremos únicamente en departamentos universitarios de ámbito español y el beneficio que acarrea por departamento.

Si a este beneficio le restamos los costes iniciales de desarrollo, obtenemos como beneficio total durante el primer año: $16.200\text{€} - 11.100\text{€} = 5.100\text{€}$

Como es normal el primer año, debido a los costes de desarrollo e implantación es habitual no obtener grandes beneficios, pero podemos ver que el gasto es asumible. El resto de años, sin tener en cuenta el aumento progresivo de usuarios y nuevas aplicaciones que puedas utilizar esta solución el beneficio sería incremental, al menos como mínimo, con los cálculos realizados anteriormente, supondría un ahorro de 242h por investigador en cada departamento.

2.10 Conclusión

El objetivo de este estudio es determinar si el desarrollo de este proyecto es viable para poder determinar si es seguro y eficaz realizarlo. Para ello hemos realizado una evaluación los recursos de los cuales disponemos, el impacto económico que supone, costes-beneficios y la planificación temporal. Hemos podido determinar que efectivamente el proyecto es viable, y que su desarrollo puede continuar con garantía de que se finalice y se obtengan los resultados esperados.

Quizá la cifra obtenida como beneficio el primer año no sea muy elevado, pero hemos de tener en cuenta, que tal y como se ha planteado, el coste de mantenimiento será bajo y los beneficios obtenidos serán los mismos cada año, o incluso más elevados si con esta solución se consigue llegar a más usuarios.

3 ANÁLISIS

Durante este capítulo vamos a realizar el análisis de problema que se ha planteado resolverse y cómo resolverlo. Se va a describir el funcionamiento del sistema actual y propondremos la mejora que vamos a desarrollar en este proyecto. Se van a exponer las especificaciones funcionales y no funcionales para describir que debe hacer el sistema y como debe hacerlo.

También presentaremos las opciones que nos ofrece el marco tecnológico para resolver los requerimientos establecidos y argumentar el por qué se van a utilizar.

3.1 Sistema actual

El uso de aplicaciones por línea de comandos está cada vez más en desuso debido a las mejoras en las tecnologías gráficas, los entornos de programación con mayor potencia y sobretodo la necesidad de llegar a más usuarios con interfaces fáciles de usar y que mejoren la usabilidad del producto.

La aplicación mlcoalsim es un buen ejemplo de este tipo de aplicaciones. Su funcionamiento consiste en realizar una llamada por línea de comandos, pasando como parámetros un fichero de entrada y otro de salida. En la figura 4 podemos ver su pantalla de ejecución:



Figura 4. Ejecución por línea de comandos de la aplicación mlcoalsim

La complejidad reside en el fichero de entrada que es el que contiene los parámetros para realizar una simulación en concreto. Un ejemplo sencillo del contenido de este tipo de ficheros sería el siguiente:

```
seed1 7354
print_matrixpol 0
print_neutttest 1
n_iterations 10000
n_loci 1
n_sites 1050
n_samples 25
npop 1
thetaw 0.05
recombination 0
```

Y uno más complejo:

```
seed1 5474
print_matrixpol 0
print_neutttest 2

n_iterations 1000
n_loci 4
n_sites 1050 560 2456 1000
n_samples 30 10 24 12
npop 1

recombination 10 0 30 56
thetaw 10 5 35 8.5
factorn_chr 1 1 1 1
no_rec_males 0

ifselection 0 0 1 0
pop_size 1E06
pop_sel 0 0 2E04 0
sel_nt 0 0 200 0
sinit 0 0 -0.01 0

TD_obs 1 -1.18213 -1.91896 -
2.19236 0.540555
FD_obs 1 -1.42122 -1.71123 -
1.57737 1.59608
FF_obs 0 -1.56732 -1.99677 -
2.09661 1.38405
Rm_obs 1 0 0 0 0

likelihood_line 1
TD_err 0.2
FD_err 0.2
Rm_err 0.01
```

Como puede comprobarse la relación de parámetros puede llegar a ser muy complicada. No sólo hay parámetros con diferente formato si no que existen dependencias entre ellos (si existe uno, debe informarse otro). Actualmente todos estos parámetros deben introducirse

manualmente, uno por uno y sin ninguna guía que nos indique si el valor y los parámetros que se necesitan son los correctos. Se dispone únicamente de la ayuda de un manual (elaborado por el creador de la aplicación) que orienta a los usuarios en la creación del fichero.

Al no poder saber con seguridad si el fichero creado es el correcto, pueden crearse ficheros erróneos o no coherentes, que darían error en la ejecución, aunque sin especificar donde está el error, o ficheros que no se ajustan a los parámetros que queremos estudiar, lo que implicaría obtener unos resultados erróneos para nuestra simulación.

También es importante destacar el tiempo de ejecución de la aplicación, es decir, el tiempo que tarda en crear el fichero de salida donde se vuelca la información de la simulación realizada. Dependiendo de los parámetros introducidos, de la complejidad de la simulación y de la potencia de la máquina donde se realiza puede llegar a tardar des de 1 minuto hasta 10 horas. Sería importante analizar cómo podría reducirse este tiempo de ejecución o como controlarlo de forma más efectiva, pero en este proyecto sólo nos vamos a centrar en el fichero de entrada para reducir el tiempo para el diseño.

3.2 Requisitos funcionales

Para describir las funcionalidades que ha de incluir la aplicación se describirán los requerimientos para el formulario dinámico, la gestión de archivos y la ejecución y control de entrada-salida

- Formulario dinámico
 - o Crear XML con toda la información necesaria para la implementación de la interfaz de la aplicación: formato, validaciones y dependencias.
 - o Diseño de una clase capaz de interpretar XML y generar un formulario web con funcionalidad completa.
 - o Crear un interfaz atractiva e intuitiva para el usuario.
- Gestión de archivos
 - o Leer archivos guardados en el sistema local del usuario e interpretarlos para llenar el formulario web.
 - o A partir de los parámetros introducidos en el formulario crear y guardar un archivo de forma local.
 - o Permitir realizar un download del archivo de salida creado en el servidor como resultado de la ejecución de la aplicación.

Estos requerimientos se han diseñado para cumplir los siguientes casos de uso:

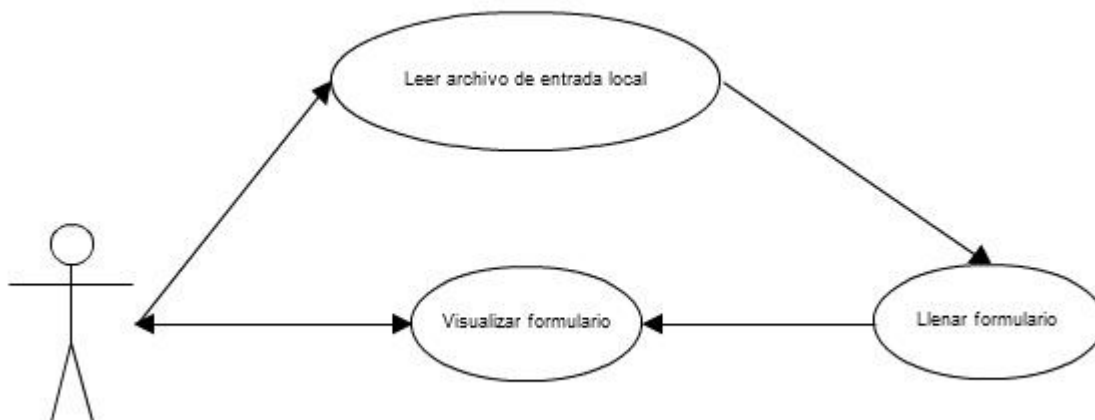


Figura 5. Caso de uso correspondiente la carga de un formulario desde archivo local.

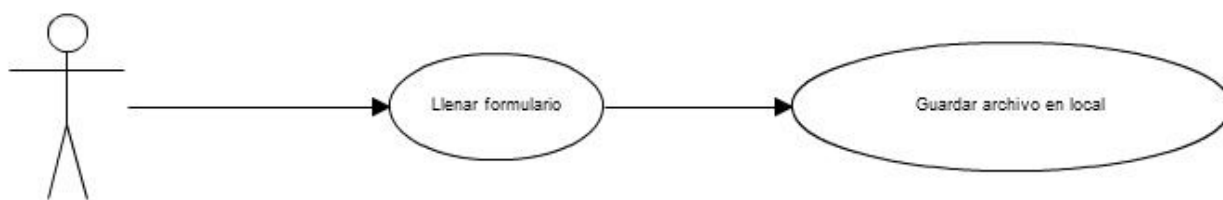


Figura 6. Caso de uso correspondiente a creación de archivo.

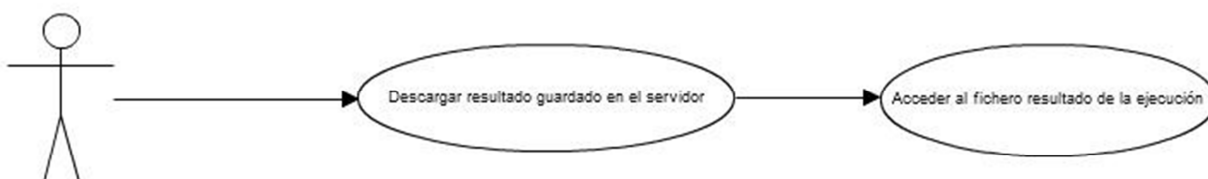


Figura 7. Figura. Caso de uso correspondiente al download del archivo resultado.

- Ejecución y control de entrada-salida
 - o Validación de los parámetros antes de crear archivo de entrada según se ha especificado en el XML.
 - o Control de errores durante la ejecución de la aplicación

- Informar al usuario del momento en que acaba la ejecución de forma correcta y de la creación del fichero de salida con los resultados.

3.3 Requisitos no funcionales

Existe un conjunto de especificaciones relacionadas con el entorno en el que debe funcionar la aplicación y que son necesarias en el cumplimiento de nuestros objetivos:

A continuación enumeraremos los requisitos no funcionales, relacionados con el entorno en que debe funcionar la aplicación y que son necesarias para cumplir nuestro objetivo:

- Interfaz intuitiva.
- Adaptación a los principales navegadores disponibles.
- Funcionamiento en Linux y mediante el servidor de páginas web Apache.
- Desarrollo en Visual Studio y Mono, por ahorro de tiempo en aprendizaje de un nuevo lenguaje.

3.4 Marco tecnológico

Para el desarrollo de esta propuesta se planteó utilizar dos herramientas: Visual Studio 2008 y Mono.

Visual Studio es un entorno de desarrollo integrado (IDE) para sistemas operativos Windows.

La versión 2008 incluye el Framework 3.5, que además de incluir nuevas funcionalidades aporta compatibilidad hacia atrás, lo que permite crear proyectos para múltiples plataformas .NET, es decir, la 2.0, 3.0 y 3.5, desde el mismo entorno. Podemos ver la evolución de Visual Studio el siguiente diagrama (Figura 8):

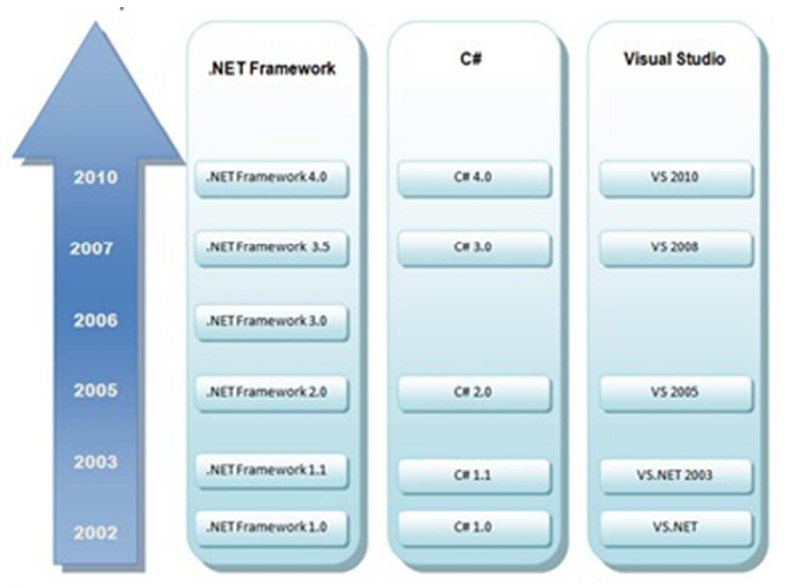


Figura 8. Evolución Visual Studio

Para la realización de nuestra aplicación utilizaremos C# .NET. C# es un lenguaje de programación que ha ido mejorando con el paso del tiempo, agregando nuevas e interesantes características en cada una de las versiones que han ido saliendo progresivamente con cada nueva entrega de Visual Studio. Sus principales características son las siguientes:

- Sencillez de uso.
- Modernidad.
- Orientado a objetos.
- Orientado a componentes.
- Recolección de basura.
- Seguridad de tipos.
- Instrucciones seguras.
- Unificación de tipos.
- Extensión de los operadores básicos.
- Extensión de modificadores.
- Eficiente.
- Compatible.

Además de las características expuestas, se ha elegido C# para el desarrollo de este proyecto por un cuestión eminentemente práctica, ya que es un lenguaje con el que estoy familiarizada lo que supone un ahorro de tiempo que se ha podido dedicar a otros aspectos.

La elección de Mono se debió en un principio a la necesidad de que la solución tenía que funcionar en Linux y preferiblemente ser desarrollada en Visual Studio, es decir, se tomó como alternativa más sencilla y menos costosa en tiempo a tener que aprender otro lenguaje de programación. Sin embargo, se ha convertido también en una parte importante del proyecto, ya que ha permitido investigar sobre las posibilidades de este proyecto de Novell que hace compatible la tecnología Microsoft con Linux, un "puente" entre dos mundos aparentemente opuestos.

Mono es el nombre de un proyecto de código abierto iniciado por Ximian y actualmente impulsado por Novell (tras la adquisición de Ximian) para crear un grupo de herramientas libres, basadas en GNU/Linux y compatibles con .NET. Los paquetes que componen la distribución de la plataforma Mono tienen un compilador C#, una máquina virtual (que permite ejecutar las aplicaciones), y un conjunto de librerías de clases que proporcionan miles de funciones.

Una vez escritas las aplicaciones se traducen a CIL (Common Intermediate Language), que es un lenguaje intermedio que no tiene particularidades de ninguna arquitectura. Una vez compilado en CIL la aplicación se traduce al lenguaje específico de la arquitectura final donde será ejecutado. Este sistema permite distribuir un único programa binario para todas las arquitecturas en vez de un programa específico para cada plataforma. Pero no menos es importante es la libertad de escoger cualquier lenguaje de programación o combinación de ellos y poder ejecutar la aplicación en cualquiera de las plataformas en las que Mono se encuentra disponible, entre las que se incluyen Intel, AMD64, SPARC, StrongArm y S390x. Mono actualmente proporciona las herramientas para crear aplicaciones para Linux (diversas distribuciones), Solaris, Windows, Mac/OS, y mainframes de IBM. A diferencia de los programas tradicionales que se ejecutan sobre el sistema directamente, los programas en la plataforma Mono se ejecutan sobre un entorno controlado de ejecución conocido como la máquina virtual.

En los esquemas siguientes (figuras 9,10) se muestra las tres capas básicas de los lenguajes y el esquema de la arquitectura del proyecto mono:

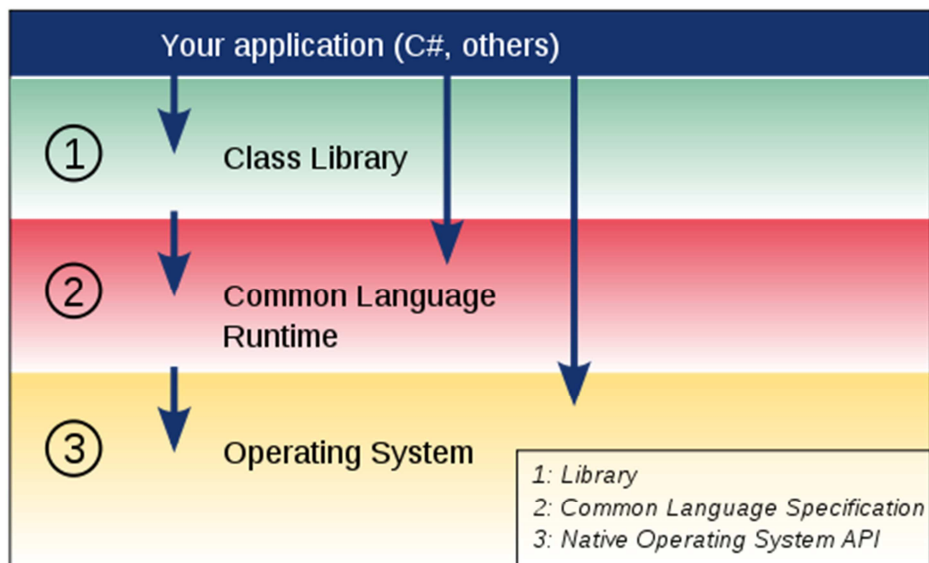


Figura 9. Capas del lenguaje Mono

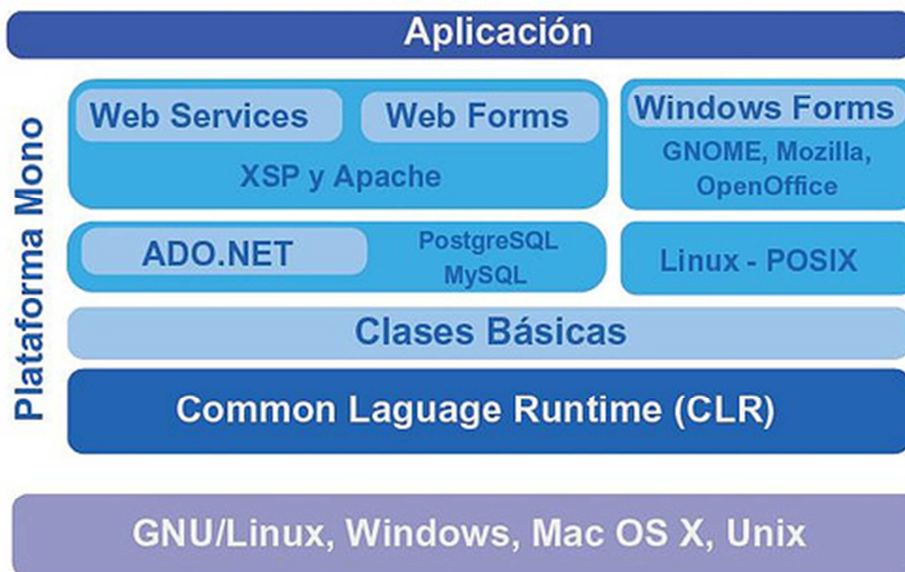


Figura 10. Arquitectura Mono

Como hemos visto, gracias a Mono, podremos utilizar todas las ventajas que nos ofrece .NET y C# para la creación de nuestra aplicación web.

Durante este apartado hemos podido ver las opciones que tenemos para solucionar el problema que se nos plantea. Disponemos de una aplicación con cálculos complejos que se ejecuta desde línea de comandos y que recibe como parámetro de entrada un fichero de

texto compuesto por diferentes parámetros que se deben introducir de forma manual. Esto conlleva problemas en el momento de escribir este fichero de entrada, ya que se ha de tener un conocimiento amplio de los parámetros que espera y no se puede realizar una validación previa a la ejecución. También se ha de tener en cuenta que estos parámetros pueden ampliarse si se amplía la funcionalidad de la aplicación en sí.

Como solución se plantea el diseño de una interfaz dinámica en forma de formulario web para que el usuario pueda introducir más fácilmente los parámetros requeridos para la ejecución de la aplicación. Este formulario será dinámico, con lo que podremos reutilizarlo tanto para introducir nuevos parámetros como para poder adaptarlo a aplicaciones por línea de comandos con funcionamiento similar. Este dinamismo lo proporcionará un fichero XML que se diseñara para cumplir con este objetivo y que será interpretado por la interfaz para crear el formulario con los parámetros indicados y las reglas de validación de éste.

En el siguiente capítulo detallaremos nuestra propuesta de diseño para la solución propuesta.

3.5 Arquitectura de la aplicación

La arquitectura de la aplicación está basada en el XML. La interfaz se genera a partir de este archivo y también a partir de lo establecido en éste se generan las validaciones de los parámetros a introducir.

A continuación mostraremos un diagrama de la arquitectura básica de la aplicación (Figura 11):

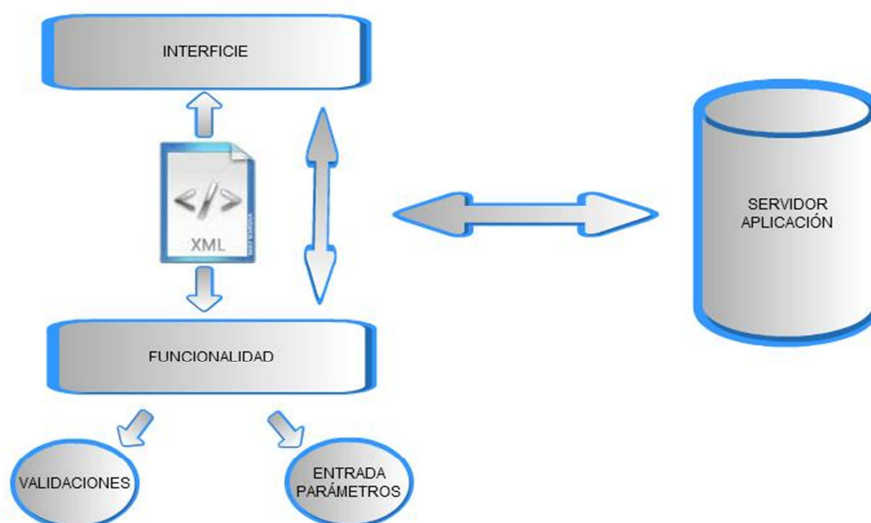


Figura 11. Arquitectura aplicación

4 DISEÑO

En el capítulo de diseño definiremos los componentes necesarios para crear la solución propuesta y que cumpla con el análisis descrito en el capítulo anterior. Los componentes serán:

- Interfaz dinámica, que contendrá la definición del XML donde se definen los campos del formulario y la lógica de sus validaciones. A partir de este XML se realizará el diseño de la página web.
- Framework, que contendrá las clases necesarias para el funcionamiento de la interfaz. Estas son tres: clase para la gestión del XML, gestión de ficheros y gestión de la gramática para las validaciones. También contendrá el módulo de ejecución mediante el cual la interfaz se relaciona con la aplicación en sí, ubicada en el servidor.

Además, realizaremos una revisión de la planificación actual del proyecto y especificaremos todas las posibles desviaciones que hayan podido aparecer a consecuencia del diseño.

4.1 Interfaz dinámica

En este apartado veremos cómo se ha definido el XML que dará forma a la interfaz dinámica. También mostraremos en que se ha basado el diseño de la página web.

4.2.1 Diseño XML

El archivo XML es el eje principal del proyecto, en el cual se basa la idea del formulario dinámico. A partir de lo especificado en el XML se creará el formulario para la introducción de los parámetros así como la lógica para la validación de estos.

En este archivo se puede dar forma al formulario. Permite introducir los elementos básicos: textbox, label, tablas (filas y columnas). Para cada uno de ellos se puede especificar propiedades básicas de estos campos y propiedades específicas para su funcionalidad.

A continuación se muestran las propiedades para cada uno de los elementos:

| | |
|--------------|--|
| TABLA | TYPE - Table |
| | ID – Identificador único |
| | CLASS – Estilo definido en el css |
| | TITULO – Título que mostrará la tabla |
| | WIDTH - Anchura de la tabla |
| | BORDER – Permite definir el borde de la tabla |
| CELDA | DISPLAY – Establece la visibilidad de la tabla |
| | ID – Identificador único |
| | WIDTH – Anchura de la celda |
| | CLASS – Estilo definido el en css |
| FILA | COLSPAN - Indica el número de columnas que ocupará la celda |
| | ID – Identificador único |
| | CLASS – Estilo definido el en css |
| | DISPLAY |

| | |
|--------------|--|
| LABEL | TYPE - Label |
| | ID – Identificador único |
| | TEXT – Texto que mostrará la etiqueta |

| | |
|---|---|
| TEXTBOX | TYPE - Textbox |
| | ID – Identificador único |
| | COLUMNS – Número de columnas dentro del textbox |
| | ROWS – Número de filas dentro del textbox |
| | MAXLENGTH – Longitud máxima del contenido |
| | TEXT – Contenido del textbox |
| | ENABLED – Indica si el campo está habilitado |
| | OUTPUT – Se establece el nombre que ha de tener en el archivo creado |
| | VALIDATIONS – Validaciones que ha de cumplir el contenido |
| CONT – Contador para los campos, necesario para su gestión | |

Pondremos un ejemplo de una parte de este XML para que se pueda ver la definición de estos parámetros.

```
<control type="Table" id="ID_TABLA" display="inline/none" titulo="TITULO" width="5" border=""
class="CSS_CLASS">
  <fila class="" ID="ID_FILA">
    <celda colspan="" width="5" class=""CSS_CLASS">
      <control type="Label" ID="ID_LABEL" Text="LABEL TEXT"/>
    </celda>
    <celda>
      <control type="TextBox" output="SALIDA" validations="int[1,32764]" ID="ID_TEXTBOX" cont="1"
Columns="1" Rows="1" MaxLength="10" Text="" Enabled="False"/>
    </celda>
    <celda>
      <control type="ImageInfo" ID="ID_IMAGE_INGO" cont="1"/>
    </celda>
    <celda display="none" id="tdError1">
```

```

    <control type="ImageError" ID="ID_IMAGE_ERROR" title="Información sobre el parámetro"/>
  </celda>
  <celda display="none" id="ID_FILA2">
    <control type="ImageValidate" ID="ID_IMAGE_VAL"/>
  </celda>
</fila>
</control>

```

También se han añadido controles tipo imagen, para mostrar al lado del Textbox y facilitar información del parámetro y si la validación ha sido correcta.

El parámetro `type` es donde se indica que tipo de control se va a crear excepto para los controles tipo fila y celda donde se indica los tags `<fila>` y `<celda>` directamente.

El elemento más complejo, por lo que respecta a su definición e interpretación es el Textbox, ya que es donde se han tenido que definir más parámetros para su correcta interpretación y personalización. A continuación vamos a proceder a explicar sus valores específicos.

Retomamos el ejemplo anterior, mostrando sólo la entrada del XML que dibujaría un textbox en el formulario:

```

<control type="TextBox" output ="param1" validations ="int[1,32764]" ID="ID_TEXTBOX" cont="1"
Columns="1" Rows="1" MaxLength="10" Text="" Enabled="False"/>

```

- Output.

En este parámetro se indica como ha de aparecer el parámetro en los fichero de salida o en los de entrada, es decir, en los que se crean y en los que se leen. Es importante ya que al ser aplicaciones que funcionan por línea de comandos esperan recibir unos parámetros en concreto y han de estar bien definidos.

- Validations

Aquí se indican el formato y las dependencias que ha de cumplir cada parámetro. Para ello se utiliza una gramática que definiremos más adelante.

- Cont

Este parámetro es un contador y se utiliza para la gestión de visibilidad. Existen controles que pueden ocultarse o hacerse visibles dependiendo de si es necesario o no (por ejemplo, en el caso de dependencias) y se controlan mediante este contador.

4.2.2 Diseño interfaz

Como hemos comentado en apartados anteriores el diseño de la interficie se basa en el XML. A partir de este se genera un formulario con los campos para introducir los parámetros necesarios.

Es un formulario básico pero gracias a que en el XML se puede incluir también las clases del css se le puede dar un formato más atractivo y personalizado.

Para la carga del formulario únicamente es necesario incluir en el código HTML de la página donde se desea cargar el formulario en siguiente código:

```
<div id="placeholder">  
  <asp:PlaceHolder ID="dinamico" runat="server"></asp:PlaceHolder>  
</div>
```

Se trata de un control web tipo div, que se utiliza en las página web para añadir contenido dentro de una división en la página, pudiendo tratarla de forma unitaria en lo que respecta a su diseño o visibilidad.

Todo el formulario indicado en el archivo XML se va añadiendo a este control quedando encapsulado dentro de él. Esto permite poder diseñar prácticamente un web sencilla desde cero y poder "incrustar" el formulario añadiendo este control div y la gestión de las clases del Framework que interactúan entre el XML y la interfaz.

En la misma página HTML se puede incluir la llamada al archivo css que dará formato a la página y también puede darlo al formulario. Para ello dentro del XML y para cada elemento web (label, textbox, tabla o fila, por ejemplo) existe el parámetro **class** donde su puede indicar que clase del css debe aplicarse.

4.2 Framework

En el apartado de Framework se explicarán las principales clases que intervienen en la funcionalidad de la interfaz. Cada una de ellas puede aislarse y reaprovecharse de forma individual siguiendo la filosofía de la programación por clases.

Las tres clases principales son XMLManager, ValidatorManager y FileManager.

En el siguiente diagrama (figura 12) podemos cómo interactúan estas clases con la interfaz y el archivo XML.

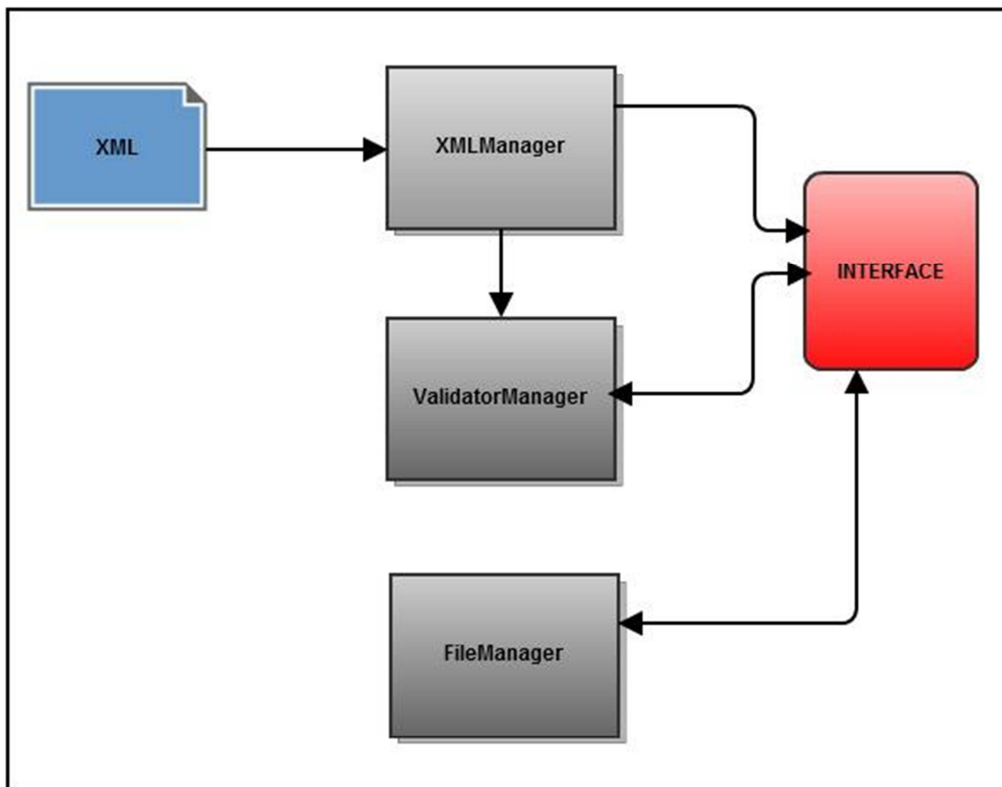


Figura 12. Diagrama Framework

4.3.1 XMLManager

Esta clase se encarga de interpretar el archivo XML en el que se base el diseño del formulario. Contiene toda la funcionalidad que se encarga de leer el contenido de este archivo e interpretarlo para dibujar el formulario y pasarle la información necesaria para el correcto funcionamiento.

Dependiendo del valor que lea en el parámetro *type* introduce en tiempo de ejecución en el formulario base el control web indicado, con los parámetros también leídos del XML. Además recoge los valores específicos como *validations* para pasárselos a la clase ValidatorManager donde se traducirá el contenido.

Para el ejemplo anterior:

```

<control type="Table" id="ID_TABLA" display="inline/none" titulo="TITULO" width="5" border=""
class="CSS_CLASS">
  <fila class="" ID="ID_FILA">
    <celda colspan="" width="5" class=""CSS_CLASS">
      <control type="Label" ID="ID_LABEL" Text="LABEL TEXT"/>
    </celda>
  </fila>
</control>

```

```

        <control type="TextBox" output ="input1" validations ="int[1,32764]" ID="ID_TEXTBOX"
cont="1" Columns="1" Rows="1" MaxLength="10" Text="" Enabled="False"/>
    </celda>
    <celda>
        <control type="ImageInfo" ID="ID_IMAGE_INGO" cont="1"/>
    </celda>
    <celda display="none" id="tdError1">
        <control type="ImageError" ID="ID_IMAGE_ERROR" title="Información sobre el
parámetro"/>
    </celda>
    <celda display="none" id="ID_FILA2">
        <control type="ImageValidate" ID="ID_IMAGE_VAL"/>
    </celda>
</fila>
</control>

```

La clase debería generar el siguiente código HTML:

```

<table id="ID_TABLE" border="0" style="DISPLAY:inline;">
    <tr id="ID_FILA">
        <td><span id="ID_LABEL">Label</span></td>
        <td>
            <input name="ValueNLoc" type="text" maxlength="10" size="1"
id="ID_TEXTBOX" onchange="Validate('ValueNLoc','1','int,1,32764,,,');" />
        </td>
        <td>
            
        </td>
        <td id="tdError1" style="DISPLAY:none;">
            </td>
        <td id="tdValidate1" style="DISPLAY:none;">
            
        </td>
    </tr>
</table>

```

4.3.2 FileManager

Esta clase es la encargada de la gestión de los archivos. El objetivo de la interfaz es crear un archivo de entrada con los parámetros necesarios para la ejecución de la aplicación. También permite cargar archivos ya creados para llenar el formulario y hacer modificaciones.

Para ellos se han creado las funcionalidades de Abrir y Guardar.

En el archivo XML se ha de especificar el nombre del parámetro en los ficheros para que, en el momento de leerlo o escribirlo se haga con el nombre que espera el programa. Este

nombre se indica en la propiedad **output** del textbox del XML. De esta manera se crea un fichero de texto plano con cada uno de los parámetros informados: nombre y valor. Y el mismo proceso se da en la lectura del fichero, leyendo cada parámetro con el nombre especificado y volcando el valor de este en el formulario.

4.3.3 ValidatorManager

Esta clase es la encargada de la interpretación de las validaciones introducidas en el XML.

Es importante que cada uno de los parámetros sea introducido en el formato correcto para que después pueda ejecutarse la aplicación sin errores.

En caso de la aplicación mlcoalsim, existen dependencias entre parámetros. Esto significa que dependiendo de si están o no informados y del valor que se les dé, otros parámetros deben estar o no informados y con valores en concreto.

Para gestionar esta necesidad, previendo que puede darse en otras aplicaciones de este tipo, se ha establecido una gramática para codificar estas dependencias así como el formato que deben tener. Para cada parámetro, en el control Textbox, donde se introduce el valor a validar, se especifica en el parámetro **validations** una expresión que la clase XMLManager le pasará a ValidatorManager para que la interprete.

Dentro de esta clase se desglosa la expresión, siguiendo un formato específico que debe respetarse para su correcta interpretación. Una vez recogidos los valores de pasan a funciones Javascript para que se realice la validación del lado del cliente.

En el siguiente apartado daremos una explicación de la lógica de la gramática establecida.

4.3.4 Gramática

En la sintaxis de la gramática se especifican dos cosas:

- Tipo de parámetro: entero, decimal, carácter o vector.
- Relaciones o dependencias con otros parámetros.

Se ha intentado diseñar una sintaxis sencilla para que no suponga mucha dificultad a la hora de introducirla en el XML. Aun así, como toda gramática de este tipo implica su dificultad y su tiempo en el momento de diseñarla. Como parte de la mejora del proyecto podría plantearse el diseñar un validador para esta gramática, para hacer más fácil el trabajo del diseñador el XML y poder asegurarnos que lo introducido es correcto.

A continuación se explicará cómo está hecha esta gramática.

Cómo hemos comentado antes en la sintaxis se especifican dos cosas:

Tipo_de_parametro#Relaciones

Ambas partes están separadas por el carácter "#".

Ahora pasaremos a desglosar cada una de estas dos partes:

- Tipo de parámetro

Tipo[valor_mínimo, valor máximo]

En el caso de que se trate de un vector:

Vector:tipo{tamaño_vector}[valor_mínimo, valor_máximo]

- Relaciones

Cada relación o dependencia estará separada por el carácter "/"

Dependencia/Dependencia/...

Y dentro de cada dependencia cada valor se separa con "+".

Cada valor se define de la siguiente manera:

Valor[<,>]:id_campo=valor_campo

La parte [,<,>] debe informarse en el caso de que sea necesario especificar que la dependencia dependa de un valor mayor o menor del indicado. En el caso de no especificarse, se establece relación de igualdad.

A continuación incluiremos algunos ejemplos :

- **Ejemplo 1:**

int[1,20]

En este caso se añade únicamente el tipo. Se trata de un entero (int) cuyo valor debe estar comprendido entre 1 y 20.

- **Ejemplo 2:**

vector:int{2*idCampoTexto}[1 50, 1 50]

En este caso también se indica únicamente el tipo. En este caso se trata de un vector de enteros cuya longitud es 2xValor contenido en TextBox con id idCampoTexto. Los valores han de estar comprendidos entre 1 y 50.

- **Ejemplo 3:**

int[1,20]#1:ID1+IDTextBox/2>:IDTextBo2x=1+ID2+IDTextBox3+IDTextBox3

En este caso ya se han incluido validaciones. Estas se establecen en la parte de la expresión está detrás de "#":

1:ID1+IDTextBox/2>:IDTextBo2x=1+ID2+IDTextBox3+IDTextBox3

Y cada dependencia separada por el carácter "/":

- **Dependencia 1:** 1:ID1+IDTextBox

En este caso las dependencias dependen del valor igual que 1 (1:). Al darse este valor deben activarse los campos cuyos identificadores se indican.

- **Dependencia 2:** 2>:IDTextBox2=1+ID2+IDTextBox3+IDTextBox3

Para este caso la dependencia depende la valores mayores o iguales a 2 (2>:). Aquí podemos ver cómo además de ver los identificadores de los campos a activar, se especifican valores concretos que han de tener algunos campos de texto (IDTextBox2=1).

5 IMPLEMENTACIÓN Y TEST

En este capítulo mostraremos el resultado final de la aplicación. Veremos el formato del formulario final así como partes del código que aclaran cómo funciona el conjunto de interfaz y framework.

5.1 Interfaz dinámica

Se ha creado una interfaz dinámica que permite al usuario de la aplicación la introducción de parámetros mediante un formulario.

En la figura 12 podemos ver el resultado final de la aplicación:

The screenshot displays the application's user interface. At the top left, there is a navigation bar with a red background containing the text "DEFINING BASIC DATA". Below this, a red header bar reads "Evolution Models and Parameters". The main content area is a table with the following parameters and their corresponding input fields:

| | | |
|-------------------------------|----------------------|-------------------|
| Variability | <input type="text"/> | ? |
| Heterogeneity (GAMMA DIST) | <input type="text"/> | ? |
| Modify Gamma Mean | <input type="text"/> | ? |
| Percentage of Invariant Sites | <input type="text"/> | ? |
| Multiple Hits | <input type="text"/> | ? |
| Ratio S/V | <input type="text"/> | ? |
| Time to Outgroup | <input type="text"/> | ? |
| Recombination/NT | <input type="text"/> | ? |
| Population mutation parameter | <input type="text"/> | ? |
| Heterogeneity (GAMMA DIST) | <input type="text"/> | ? |

At the top left of the interface, there is a button labeled "Open File" next to a text input field, followed by a button labeled "Examinar...". Below these elements is the label "Parámetros". On the right side, there is a vertical sidebar menu with a dark red background, containing the following items: "Home", "Application", "Help", "License", and "Contact".

Figura 13. Formulario

Vamos a dividir esta apartado en las dos partes principales de la interfaz: formulario y diseño.

5.1.1 Formulario

El dinamismo de esta interfaz viene dado por el XML que genera el formulario. Retomamos el ejemplo del anterior capítulo para poder ver con más detalle el funcionamiento:

```
<control type="Table" id="ID_TABLA" display="inline/none" titulo="TITULO" width="5" border=""
class="CSS_CLASS">
  <fila class="" ID="ID_FILA">
    <celda colspan="" width="5" class=""CSS_CLASS">
      <control type="Label" ID="ID_LABEL" Text="LABEL TEXT"/>
    </celda>
    <celda>
      <control type="TextBox" output ="input1" validations ="int[1,32764]" ID="ID_TEXTBOX"
cont="1" Columns="1" Rows="1" MaxLength="10" Text="" Enabled="False"/>
    </celda>
    <celda>
      <control type="ImageInfo" ID="ID_IMAGE_INGO" cont="1"
path="../Resources/images/info.png" title="Información sobre el parámetro"/>
    </celda>
    <celda display="none" id="tdError1">
      <control type="ImageError" ID="ID_IMAGE_ERROR" />
    </celda>
    <celda display="none" id="ID_FILA2">
      <control type="ImageValidate" ID="ID_IMAGE_VAL"/>
    </celda>
  </fila>
</control>
```

El código se repite a lo largo del XML, pudiendo introducir variaciones en el diseño, en la longitud de los campos o en tipo de validaciones.

La clase encargada de leer e interpretar el XML es XMLManager. A continuación mostramos un diagrama (figura 14) de la clase donde su pueden ver los métodos implementados.

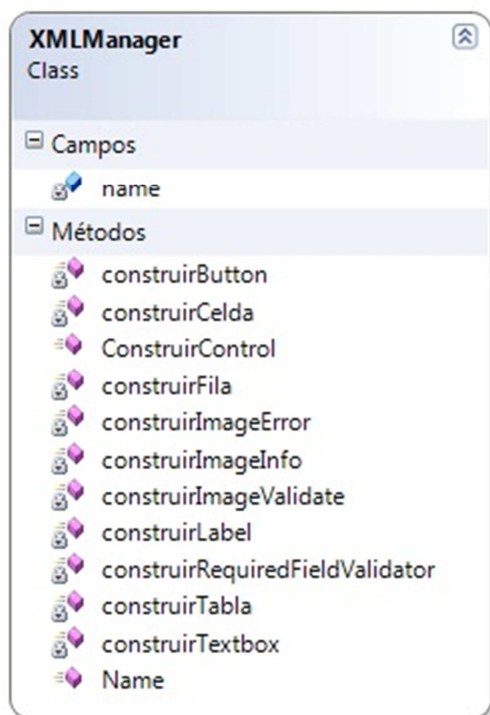


Figura 14. Diagrama clase XMLManager

En estos métodos se identifica el tipo de control que se desea insertar y se le asignan las propiedades indicadas en los campos del XML.

Para el ejemplo expuesto el código HTML generado sería el siguiente:

```
<table id="ID_TABLE" border="0" style="DISPLAY:inline;">
  <tr id="ID_FILA">
    <td><span id="ID_LABEL">Label</span></td>
    <td>
      <input name="ValueNLoc" type="text" maxlength="10" size="1"
id="ID_TEXTBOX" onchange="Validate('ValueNLoc','1','int,1,32764,,,');" />
    </td>
    <td>
      
    </td>
    <td id="tdError1" style="DISPLAY:none;">
      </td>
    <td id="tdValidate1" style="DISPLAY:none;">
      
    </td>
  </tr>
</table>
```

Esto se corresponde con la fila de una tabla, que contiene un etiqueta, un control para introducir texto y varias imágenes o iconos para facilita más información sobre el parámetro. Esta sería el control que se generaría a partir de este código HTML (figura 15).



Figura 15. Fila generada a partir del XML

De esta forma cada vez que se quiera o se tenga la necesidad de añadir un parámetro, únicamente será necesario añadir otra "fila" al XML, añadiendo las características de este parámetro según las necesidades.

Para aportar más información sobre los parámetros se añade un icono en el que se muestra un "tooltip". En el XML se puede especificar esta información así como la ruta para permitir poder ubicar la imagen en alguna otra carpeta e incluso cambiarla por otra para adaptarla a un diseño más personalizado.

```
<celda>
  <control type="ImageInfo" ID="ID_IMAGE_INFO" title="Información sobre el parámetro"
  cont="1" path="../Resources/images/info.png" / >
</celda>
```

Y se vería de la siguiente forma (figura 16):



Figura 16. Icono de información del parámetro

5.1.2 Diseño

El diseño de la página web en sí, es totalmente independiente del formulario generado a partir del XML. En este desarrollo se ha utilizado una plantilla css para poder realizar la pruebas y darle algo de forma a la web.

A continuación veremos un ejemplo en el que se puede ver como cambiando el css o algunas propiedades de éste cambia el aspecto del formulario:

The image shows a web application interface with a dark red theme. At the top left, there is a navigation bar with 'Open File' and 'Examinar...' buttons. Below this is a section titled 'Parámetros'. The main content area is a form with a dark red background and white text. The form is divided into several sections: 'DEFINING BASIC DATA', 'Evolution Models and Parameters', and 'Simulation Parameters'. Each section contains various input fields and labels. The right sidebar is also dark red and contains links for 'Home', 'Application', 'Help', 'License', and 'Contact'. At the bottom of the form, there are 'Save File', 'Examinar...', and 'OK' buttons.

Figura 17. Formulario con cambio de estilo

En este ejemplo se ha cambiado únicamente el color del formulario y de algunos labels, pero se podría cambiar totalmente de aspecto sin cambiar en nada su funcionalidad.

Como se ha explicado anteriormente, el formulario especificado en el XML se puede incrustar dentro de un diseño completamente diferente de página web, permitiendo así un diseño más personalizado y atractivo. Durante este proyecto no se ha priorizado el tema del diseño de la web ya que no era la prioridad, sino el hecho de permitir que el formulario que necesitamos para la introducción de parámetros sea lo más independiente posible del diseño para que pueda después ser adaptado a cada aplicación según convenga.

5.1.3 Validación de parámetros

El formulario creado permite además la validación de los parámetros en el momento de introducirlos. Las validaciones se especifican también dentro del XML.

```
<control type="TextBox" output ="input1" validations ="int[1,32764]" ID="ID_TEXTBOX" cont="1" Columns="1" Rows="1" MaxLength="10" Text="" Enabled="False"/>
```

La clase XMLManager recoge el parámetro **validations** y se lo pasa a la clase ValidatorManager para que interprete la gramática y cree la función javascript necesaria para la validación del lado del cliente. A continuación lo mostramos en un diagrama (Figura 18):

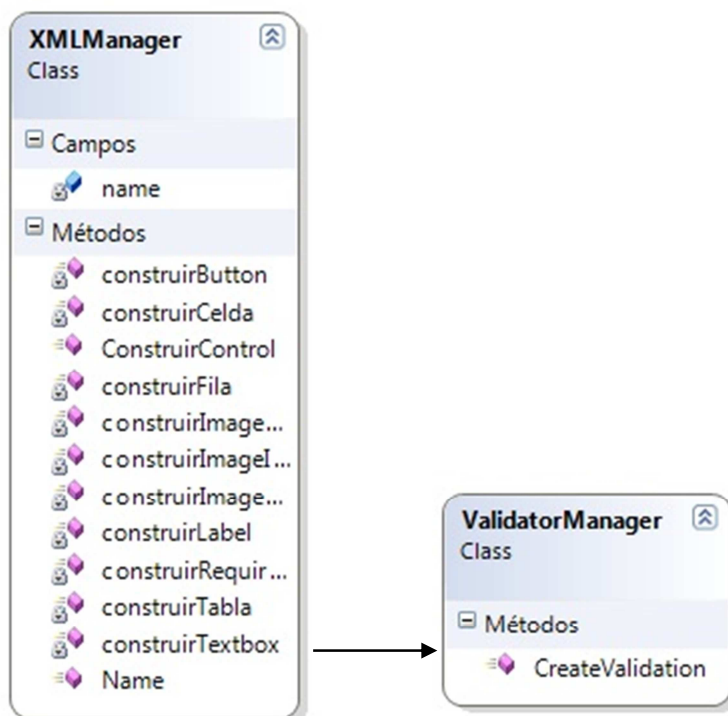


Figura 18. Diagrama clase XMLManager y ValidatorManager

Dentro del método *construirTextbox* de la clase XML se hace la llamada para traducir la gramática y después se incluye el evento *onchange* para la llamada a la función *Validate* del archivo javascript.

```

if (nodo.Attributes["validations"] != null &&
!nodo.Attributes["validations"].Value.Equals(""))
{
    ValidatorManager vm = new ValidatorManager();
    validations = vm.CreateValidation(nodo.Attributes["validations"].Value);
    txtGenerated.Attributes.Add("onchange", "Validate(" + txtGenerated.ID + ", " + cont + ", " +
validations + ");");
}
  
```

A la función javascript se le pasan tres parámetros:

1. Identificador del control.
2. Contador de controles.
3. Validaciones traducidas por la clase ValidatorManager.

Las validaciones que se pasan a javascript pueden ser sencillas, por ejemplo en las que sólo se necesita validar el tipo del parámetro:

```
Validate('ValueNLoc', '1', 'int,1,32764,,,');
```

Si es un vector se incluye también la longitud de este:

```
Validate('txtRegions', '3', 'vector, 1 3267, 1 3267,int,2*txtNlinkedLoci,')
```

O pueden ser más complejas, es el caso de parámetros que tienen dependencias entre si:

```
Validate('txtNlinkedLoci', '2', 'int,1,32764,,,1+False+False:TDDisplacement##+$txtNofSites##+$ /2+True+False:ValueNLoc+1$TDRRegions##+$txtRegions##+$txtNofSites##+$ /');"
```

Lo primero que hace la función javascript es dividir las validaciones, que le llegan separadas por comas, para poder tener el valor necesario para comprobar que lo especificado es correcto. Una vez indentificado el tipo, se hace una validación para cada uno de los principales y después se evalúan las dependencias:

```
function Validate(_id, _cont, _vals)
{
    //Se recogen los valores, separados por coma
    var element = document.getElementById(_id).value;
    var validators = _vals.toString().split(',');
    var tipo = validators[0];
    var value_ini = validators[1];
    var value_final = validators[2];
    var tipo_vector = validators[3];
    var tamaño_vector = validators[4];
    var dependencias = validators[5];
    //Se valida el Tipo
    //Tipo entero
    if (tipo == "int")
    {
        ValidateInt(_id, element, _cont);
    }
    //Tipo float
    if (tipo == "float")
    {
        ValidateFloat(_id, element, _cont);
    }
}
```

```

//Se evalua si el valor está en el rango indicado (para entero y float)
if (element < value_ini || element > value_final)
{
    noValid(_id, _cont);
    return;
}
//Tipo vector
if (tipo == "vector")
{
    ValidateVector(_id, element, _cont, tipo_vector, tamaño_vector, value_ini, value_final);
}

//Si el tipo es correcto, entonces se evalúan las dependencias
if (dependencias != "")
{
    EstablecerDependencias(dependencias, element, _id, _cont);
}
}

```

Cada vez que uno de los valores de los controles textbox es introducido se llama a esta función *Validate*. Si la validación es correcta (tipo y dependencias) se muestra la un imagen/icono para indicar que el parámetro introducido es correcto y una imagen/icono de error en el caso contrario, como se muestra en la siguiente imagen (Figura 19):







| DEFINING BASIC DATA | | | |
|---------------------|--------------------------------|---|---|
| Independent Loci | <input type="text" value="1"/> |  |  |
| Linked Loci | <input type="text" value="2"/> |  |  |
| Regions | <input type="text" value="1"/> |  |  |

Figura 19. Iconos validación.

5.2 Gestión de archivos

La implementación de la gestión de archivos se basa en la clase FileManager. En la figura 20 podemos ver su correspondiente diagrama:

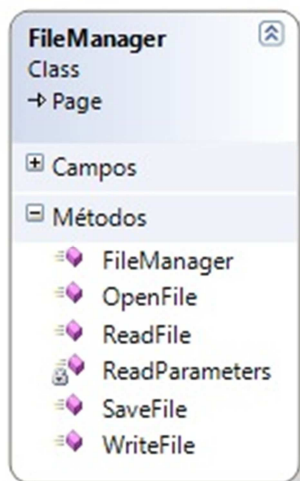


Figura 20. Diagrama de la clase FileManager.

Sus métodos principales son: abrir, leer, escribir y guardar.

1. Abrir (OpenFile)

Este método se ocupa de leer un fichero, siempre que cumpla el formato esperado. Lo recorre y guarda cada parámetro definido junto con su valor, para después identificar estos parámetros en el formulario.

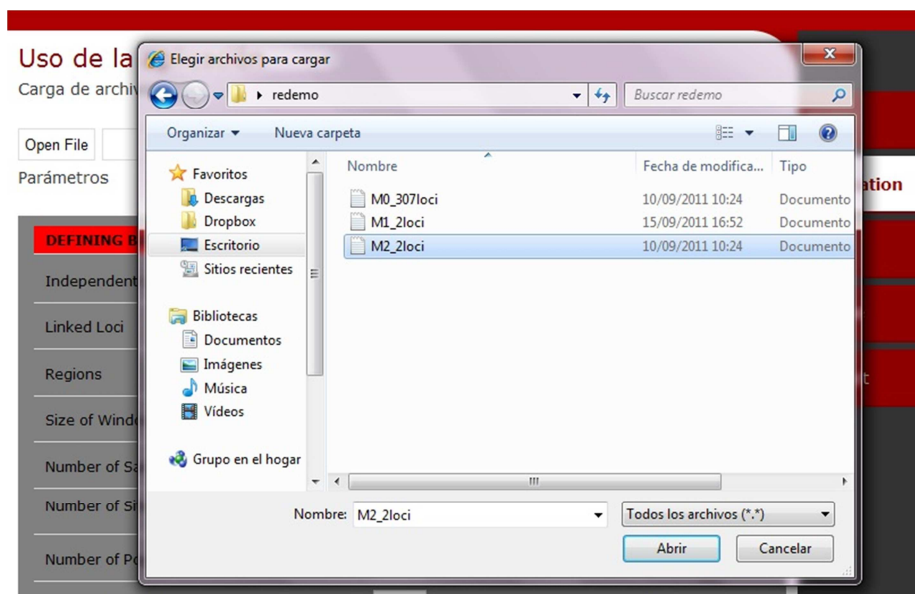


Figura 21. Carga de archivos.

2. Leer (ReadFile)

Aquí se recorre el formulario mirando la propiedad **output** definida en el textbox y llenando el formulario con los valores leídos en el archivo y guardados al abrir el archivo.

3. Escribir (WriteFile)

En este método se escriben los parámetros introducidos en el formulario en un archivo seleccionado previamente. Recorre el formulario recogiendo el nombre y el valor de todos los valores que se hayan informado.

4. Guardar (SaveFile)

Guarda el archivo creado en la función anterior en el disco duro local.

Al tratarse FileManager de una clase independiente, seguimos la misma filosofía de interfaz dinámica e independiente de la funcionalidad. Estas funciones pueden ser llamadas desde cualquier página tipo aspx, con la única condición de pasarle los parámetros necesarios. De esta forma se puede cambiar el diseño de los botones y adaptarlos al gusto del diseñador.

5.3 Test

En este apartado se van describir las principales pruebas que se han realizado para asegurarnos del correcto funcionamiento de la interfaz en lo que a funcionalidad se refiere.

Dado que el principal objetivo de este proyecto es permitir la introducción de parámetros en un formulario y su validación, las principales pruebas se han orientado a garantizar que se realiza de forma correcta.

A continuación listaremos algunas de las pruebas realizadas, los errores encontrados y las soluciones que se han implementado para solucionarlo:

1. Introducción de varios parámetros de tipo entero.

Se realiza la validación de forma correcta pero no se activa el GIF de validación. Para solucionarlo se establecen contadores para todos los parámetros de forma que se pueda relacionar el textbox con la imagen correspondiente.

2. Introducción de parámetros tipo string.

Se realiza la validación de forma correcta.

3. Introducción de parámetros tipo decimal.

- Valida correctamente decimales del tipo 0.00 (separador ".") pero no si se introducen con un separador ",".
 - Se modifica la validación para que en momento de validar cambie el "." por la "," escribiéndolo también así en el fichero.
4. Introducción de parámetros tipo vector.
- Se realiza la validación correctamente.
5. Comprobación de las dependencias.
- Surgen varios problemas, ya que al fichero javascript no le llegan los identificadores de forma correcta.
 - Se modifica la gramática y la clase que la interpreta para que le pase al javascript los identificadores en el formato correcto.
6. Validación de los parámetros al cargar formulario desde archivo.
- No se activa el evento onchange necesario para que se ejecute la función en javascript.
 - Se modifica la funcionalidad para llamar al evento desde el lado del servidor, de manera que, en el caso de carga de archivo no se hará ésta del lado del cliente.
7. Carga de diferentes archivos de ejemplo y creación de archivo a partir de este.
- Realiza bien la carga pero se identifican parámetros que faltan definir en el formulario que tampoco se vuelcan en el fichero creado.
 - Se incluyen en el XML.

5.4 Puesta en marcha

Para la puesta en marcha de la aplicación es importante definir primero el entorno de ejecución.

El desarrollo de este proyecto se ha realizado en Windows (Visual Studio 2008, Framework 3.5) y probado en Linux mediante una máquina virtual donde se instaló OpenSuse.

En cambio el entorno de producción será un servidor Linux sobre el que debe instalarse la última versión de Mono (2.10.5) para que sea compatible con el framework utilizado. Mono además, es compatible con las principales distribuciones de Linux, de manera que podría

instalarse cualquiera de ellas. Para el funcionamiento de la web debe estar configurado el servidor de páginas web Apache

Para que funcionen en Linux los archivos ".dll" que se compilan en Windows es necesario, aparte de tener instalado Mono y Mono Server, realizar unas modificaciones en el fichero de configuración de Apache. En el caso de la distribución OpenSuse se trata del fichero:

etc/apache2/conf.d/**test.conf**

```
# MonoServerPath test "/opt/novell/mono/bin/mod-mono-server2"
# For Mono on openSUSE, uncomment the line below instead:
MonoServerPath MlCoalsimWeb "/usr/bin/mod-mono-server2"

# To obtain line numbers in stack traces you need to do two things:
# 1) Enable Debug code generation in your page by using the Debug="true"
#    page directive, or by setting <compilation debug="true" /> in the
#    application's Web.config
# 2) Uncomment the MonoDebug true directive below to enable mod_mono debugging
MonoDebug MlCoalsimWeb true

# The MONO_IOMAP environment variable can be configured to provide platform abstraction
# for file access in Linux. Valid values for MONO_IOMAP are:
# case
# drive
# all
# Uncomment the line below to alter file access behavior for the configured application
MonoSetEnv MlCoalsimWeb MONO_IOMAP=all
#
# Additional environment variables can be set for this server instance using
# the MonoSetEnv directive. MonoSetEnv takes a string of 'name=value' pairs
# separated by semicolons. For instance, to enable platform abstraction *and*
# use Mono's old regular expression interpreter (which is slower, but has a
# shorter setup time), uncomment the line below instead:
# MonoSetEnv test MONO_IOMAP=all;MONO_OLD_RX=1

MonoApplications MlCoalsimWeb "*/srv/www/webmlcoalsim"
<Location "/">
  Allow from all
  Order allow,deny
  MonoSetServerAlias MlCoalsimWeb
  SetHandler mono
  SetOutputFilter DEFLATE
  SetEnvIfNoCase Request_URI "\.(?:gif|jpe?g|png)$" no-gzip dont-vary
</Location>
<IfModule mod_deflate.c>
  AddOutputFilterByType DEFLATE text/html text/plain text/xml text/javascript
</IfModule>
</VirtualHost>
```

En el caso de otras distribuciones puede variar la ubicación del fichero.

Es necesario modificar el fichero para descomentar las líneas marcadas en negrita y añadir el nombre de la aplicación y la ruta donde se ubicaran los ficheros, así como otros parámetros necesarios para la correcta interpretación de todos los archivos.

Por otra parte también es necesario añadir al directorio de ficheros de configuración de Apache el siguiente fichero:

etc/apache2/conf.d/mod_mono.conf

```
# mod_mono.conf

# Achtung! This file may be overwritten
# Use 'include mod_mono.conf' from other configuration file
# to load mod_mono module.

<IfModule !mod_mono.c>
    LoadModule mono_module /usr/lib/apache2/mod_mono.so
</IfModule>

<IfModule mod_headers.c>
    Header set X-Powered-By "Mono"
</IfModule>

AddType application/x-asp-net .aspx
AddType application/x-asp-net .asmx
AddType application/x-asp-net .ashx
AddType application/x-asp-net .asax
AddType application/x-asp-net .ascx
AddType application/x-asp-net .soap
AddType application/x-asp-net .rem
AddType application/x-asp-net .axd
AddType application/x-asp-net .cs
AddType application/x-asp-net .vb
AddType application/x-asp-net .master
AddType application/x-asp-net .sitemap
AddType application/x-asp-net .resources
AddType application/x-asp-net .skin
AddType application/x-asp-net .browser
AddType application/x-asp-net .webinfo
AddType application/x-asp-net .resx
AddType application/x-asp-net .licx
AddType application/x-asp-net .csproj
AddType application/x-asp-net .vbproj
AddType application/x-asp-net .config
AddType application/x-asp-net .Config
AddType application/x-asp-net .dll
DirectoryIndex index.aspx
DirectoryIndex Default.aspx
DirectoryIndex default.aspx
```

Donde se añaden los tipos necesarios para el correcto funcionamiento las librerías .NET.

Una vez realizadas todas estas modificaciones se deber crear el directorio indicado en los .config.

./:/srv/www/webmlcoalsim

La estructura de carpetas es la siguiente (figura 22):

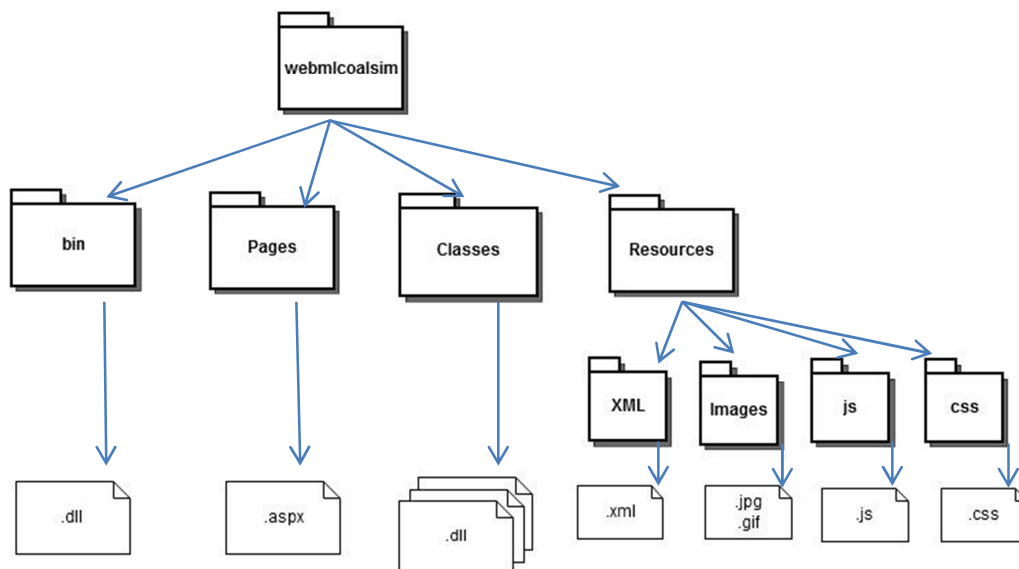


Figura 22. Estructura de carpetas.

Una vez realizadas todas estas acciones es necesario ejecutar un comando para la compilación de los ficheros .dll en Linux.

```
gmcs /t:library /out:bin/webmlcoalsim.dll -pkg:dotnet -r:System.Configuration -debug+ -recurse:*.cs
```

Y si es necesario añadir más librerías .dll se compilar añadiendo a la instrucción anterior el comando:

```
-r:*.dll
```

Una vez realizadas todas estas acciones ya podemos abrir en el navegador nuestra página web en local y configurar Apache para que se pueda acceder a ella desde internet.

6 CONCLUSIONES

Una vez mostrado el resultado final de la solución y finalizadas las pruebas, se realiza un balance con la planificación real del proyecto, observando las desviaciones con la planificación realizada en el apartado de "Diseño". También se va a realizar una valoración global de la experiencia y se plantean posibles mejores que podrían mejorar el resultado.

6.1 Seguimiento del proyecto

El seguimiento del proyecto permite controlar, a medida que se finalizan las tareas, en qué momento se encuentra el desarrollo del proyecto. También es importante en el momento de valorar la planificación hecha al final de la fase de diseño mostrando todas las tareas con su duración real en relación con la planificación, para que se pueda detectar en que fases el proyecto ha sufrido retrasos.

No ha habido desviaciones significativas en las primeras fases del proyecto: análisis y diseño. Las desviaciones más significativas se han dado en la tercera fase del proyecto: implementación y test. Estas desviaciones se han debido sobre todo a dos problemas. El primero ha sido un tema personal, ya que no he podido disponer de todo el tiempo que hubiera necesitado para realizar el proyecto en el tiempo planificado. El segundo ha sido la dificultad extra que ha supuesto para mí trabajar en Linux, debido a mi falta de experiencia en este sistema operativo y sobretodo en el servidor de páginas web Apache, lo que hay que sumar el funcionamiento de mono para poder ejecutar las librerías .NET. El tiempo dedicado a solventar esta parte del proyecto ha consumido el tiempo previsto para otras partes del proyecto afectando en especial al diseño "estético" de la interfaz de usuario.

También ha resultado costosa la parte de solución de errores, ya que en un principio no se previó la complejidad de todas las combinaciones de parámetros que se podían llegar a dar y al empezar las pruebas se vio que había varios casos que no se habían tenido en cuenta en el diseño.

Aun así se ha conseguido realizar el proyecto dentro del tiempo establecido para la realización de proyecto final de carrera.

6.2 Consecución de objetivos y posibles mejoras

Al final del proceso del desarrollo del proyecto se puede decir que se han conseguido los principales objetivos:

- ✓ Se ha desarrollado una interfaz dinámica, fácilmente ampliable y que facilita la tarea del usuario para la introducción de parámetros.
- ✓ Esta interfaz puede dar "cara" a las aplicaciones que se ejecutan por línea de comandos sin invertir mucho tiempo en el diseño de la web y pudiendo reaprovechar el código interno de estas aplicaciones.
- ✓ Se ha diseñado un archivo XML que puede ser implementado sin ser experto en creación de páginas web, aunque es necesario cierto conocimiento del formato de éste para poder dar la forma y funcionalidad deseada.
- ✓ Se ha comprobado la utilidad de Mono para compilar aplicaciones implementadas en Windows y que sean compatibles con otros sistemas operativos.

Una vez creadas las principales bases para el funcionamiento de esta interfaz es posible plantear posibles mejoras para ampliar la funcionalidad y mejorar el aspecto del formulario:

- Mejorar el diseño general de la interfaz:
 - Crear menú más ordenado y claro para la gestión de archivos.
 - Permitir introducir también mediante el archivo XML más opciones para mostrar los parámetros: ventanas tipo pop-up, listas desplegables e imágenes más modernas.

- Ampliar funcionalidad:
 - Crear sistema de gestión de usuarios.
 - Control más exhaustivo de errores y sistema de reintentos al conectar con el servidor.
 - Mejorar la seguridad de la web.

Está claro que la lista de posibles mejoras es extensa pero al estar sentadas las bases del diseño del archivo XML estas son fácilmente aplicables y no tienen por qué suponer mucho tiempo de trabajo para alguien experimentado en el diseño web.

6.3 Valoración personal

La elaboración de este proyecto ha sido una experiencia en la que he tenido la oportunidad de aprender mucho en todos los sentidos, tanto a nivel personal como profesional.

He podido comprobar la dificultad que implica el realizar un proyecto desde cero y realizar todas las fases necesarias para llevarlo a cabo con éxito además de aprender la necesidad de administrar bien el tiempo. Esto implica no sólo ser capaz de cumplir los tiempos establecidos sino también el saber reorganizarlo en el caso de que no se hayan podido cumplir.

Sin duda todo el esfuerzo merece la pena cuando una vez finalizado el proceso se ven los objetivos cumplidos y haciendo balance del tiempo dedicado puedo decir que este esfuerzo ha valido para tener más experiencia en programación, enfrentarme a nuevos retos de aprendizaje y ser capaz de superarlos.

7 BIBLIOGRAFÍA Y REFERENCIAS

7.1 Libros

Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. s.l. : Addison-Wesley Professional, 1994.

Mayo, Joseph. 2001. *C# Unleashed*. s.l. : Sams, 2001.

Bishop, Judith. 2007. *C# 3.0 Design Patterns*. s.l. : O'Reilly Media, 2007.

7.2 Referencias en internet

Accusoft. 2010. Empresa dedicada al desarrollo de SDKs para .NET 6 de Septiembre de 2010. <http://www.accusoft.com/>.

Block, Glenn. 2010. Artículo sobre PRISM y MEF en el blog del encargado de MEF.25 de Agosto de 2010. <http://www.sparklingclient.com/mef-and-prism/>.

Mono-opensuse. 2011. <http://en.opensuse.org/Mono>

Mono. 2011. Página sobre proyecto Mono (especificación de los archivo de configuración de Apache) <http://go-mono.com/config-mod-mono/>

Pozo, Pedro. 2010. Microsoft MVP en tecnología ASP .NET. <http://www.clikear.com/manuales/csharp/c10.aspx>

NET. 2008. Información sobre el framework .NET de Microsoft. 21 de Agosto de 2010. <http://www.microsoft.com/net/>.

Smith, Josh. 2010. MSDN Design Patterns. 25 de Agosto de 2010. <http://msdn.microsoft.com/en-us/magazine/dd419663.aspx>.

VS2008. 2010. Página oficial de Visual Studio 2008. 21 de Agosto de 2010. <http://www.microsoft.com/visualstudio/en-us>.

Wikipedia. 2010. Enciclopedia en Internet. 21 de Agosto de 2010. <http://en.wikipedia.org/wiki/>.

[http://wiki.gxtechnical.com/commwiki/servlet/hwiki?Proyecto+Mono,](http://wiki.gxtechnical.com/commwiki/servlet/hwiki?Proyecto+Mono)

