

Ingeniería Técnica en Informática de Gestión
Curso 2007-2008

Proyecto de fin de carrera

Traza de programas en ambientes colaborativos de aprendizaje con computación móvil

Autor: Miguel Ángel Domínguez Coloma
Tutor: Maximiliano Paredes

Agradecimientos

- A mi padre por iniciarme en el mundo de la informática cuando era pequeño.
- A mi madre por darme apoyo constante durante todos mis años de estudio.
- A mi hermana por salir conmigo durante los malos momentos.

Índice general

1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	1
2. Fundamentos	4
2.1. .NET Framework	4
2.2. .NET Framework Remoting	6
2.3. .NET Compact Framework	14
2.4. MySQL	17
2.5. Servicios web	18
2.6. GDB Debugger	20
3. Desarrollo de la solución	23
3.1. Requisitos	23
3.2. Análisis	32
3.3. Diseño	37
3.4. Implementación y pruebas	47
4. Conclusiones	49
4.1. Conclusiones	49
4.2. Trabajo futuro	49
Bibliografía	51
A. Instalación	52
A.1. Instalación desde el CD	53
B. Base de datos	55
C. Guía rápida	58
C.1. Programa del profesor	58
C.2. Programa del alumno	63

Índice de figuras

2.1. Creación de código MSIL a partir del código fuente.	5
2.2. Conversión del código MSIL a código nativo.	6
2.3. Serializable: el objeto se serializa de un dominio a otro.	7
2.4. Marshal-by-Reference: el objeto se instancia en su propio dominio.	7
2.5. Singleton: Todos los clientes comparten la misma instancia remota.	8
2.6. SingleCall: Cada llamada del cliente crea un nuevo objeto sin permanencia del estado.	9
2.7. CAO: Cada cliente puede instanciar varios objetos remotos distintos.	9
2.8. Proxy: Creación de dos nuevos objetos para comunicarse con el servidor.	10
2.9. Servicios web: petición de un servicio de un cliente a un servidor.	18
3.1. Esquema general del sistema.	24
3.2. Casos de uso del programa del profesor.	28
3.3. Casos de uso que se implementan en el servicio web.	28
3.4. Casos de uso del programa del alumno.	29
3.5. Diagrama de estado para la elección del proyecto.	30
3.6. Diagrama de estado para dibujar en la pizarra.	31
3.7. Diagrama de estado para el envío de mensajes a grupos.	31
3.8. Diagrama de estado de la depuración de un proyecto.	32
3.9. Estereotipos	33
3.10. Diagrama de colaboracion del flujo principal (Registro)	33
3.11. Diagrama de colaboracion del flujo alternativo I (Registro)	33
3.12. Diagrama de colaboracion del flujo alternativo II (Registro)	34
3.13. Diagrama de colaboracion del flujo principal (Elección proyecto)	34
3.14. Diagrama de colaboracion del flujo alternativo I (Elección proyecto)	34
3.15. Diagrama de colaboracion del flujo principal (Dibujar formas)	35
3.16. Diagrama de colaboracion del flujo alternativo I (Dibujar formas)	35
3.17. Diagrama de colaboracion del flujo alternativo II (Dibujar formas)	35
3.18. Diagrama de colaboracion del flujo alternativo III (Dibujar formas)	35
3.19. Diagrama de colaboracion del flujo principal (Enviar mensajes)	36
3.20. Diagrama de colaboracion del flujo alternativo I (Enviar mensajes)	36
3.21. Diagrama de colaboracion del flujo alternativo II (Enviar mensajes)	36
3.22. Diagrama de colaboracion del flujo principal (Depurar la traza)	36
3.23. Diagrama de paquetes	39
3.24. Diagrama de clases para registrar al alumno.	40
3.25. Diagrama de clases para obtener una entrada de un proyecto.	40
3.26. Diagrama de clases para dibujar formas.	41
3.27. Diagrama de clases para enviar mensajes.	42
3.28. Diagrama de clases para depurar la traza y final.	43

3.29. Esquema de la base de datos.	44
3.30. Servicios desde el punto de vista del profesor.	45
3.31. Servicios desde el punto de vista del alumno.	46
C.1. Ventana principal del programa del profesor.	58
C.2. Creando un nuevo usuario.	59
C.3. Creando un nuevo grupo.	59
C.4. Creando un nuevo proyecto.	59
C.5. Añadiendo meta-información a un nuevo proyecto.	60
C.6. Añadiendo nuevas entradas al proyecto.	61
C.7. Ventana de edición de un proyecto.	62
C.8. Compilando y depurando un proyecto.	62
C.9. Ventana de registro en la PDA.	63
C.10. Elección de un proyecto remoto.	63
C.11. Ventana principal para dibujar y manejar la tabla de variables.	64
C.12. Modificando el valor de una variable en la tabla de variables.	64
C.13. Creando una suposición sobre el contenido de una variable.	65
C.14. Mandando un mensaje al resto de alumnos.	65
C.15. Cambiando el idioma del programa.	66
C.16. Cambiando las propiedades de la conexión.	66

Índice de tablas

2.1. Diferencia entre los canales estándar que ofrece .NET Remoting.	11
3.1. Requisitos para el programa de administración.	25
3.2. Requisitos para el programa servidor.	25
3.3. Requisitos para el cliente para PDA.	26

1. INTRODUCCIÓN

1.1 Motivación

Durante la lectura de esta memoria se demostrará una aplicación de depuración colaborativa entre alumnos de una misma clase.

Dado el avance de las nuevas tecnologías y las comunicaciones móviles, las personas tienen al alcance la comunicación entre ellas sin estar presencialmente juntas. Además han surgido nuevas formas de comunicación: chats, pizarras remotas, etc. Estamos en una época en la que casi cada persona posee un teléfono móvil o PDA.

Así pues, dadas estas tecnologías y nuevos usos de comunicación, precisamos del estudio de nuevas formas de educación basados en tecnologías móviles. En este trabajo de fin de carrera se quiere apoyar y facilitar el aprendizaje en aspectos básicos de programación, en concreto el concepto de ámbito y visibilidad de variables. Los alumnos de una clase (u otros grupos) podrán comprobar en tiempo real la depuración de un programa enviándose mensajes, razonando sobre los resultados de las variables de los programas antes de que las instrucciones de código se ejecuten y que todos estos cambios puedan verse en todo el grupo. Los alumnos tendrán la posibilidad de ver sus propios errores, corregir los de los demás y ayudarse mutuamente, como si se tratase de una gran mente colaborativa.

Los teléfonos móviles PDAs están al alcance de todos. Hemos elegido esta tecnología ya que dispone de dispositivos pequeños con acceso a Internet y que nos ofrecen todas las características necesarias para crear este proyecto.

1.2 Objetivos

Se requiere de una aplicación que permita el aprendizaje, de forma colaborativa, mediante la depuración de código fuente de aplicaciones. Esta herramienta ayudará a los alumnos a ejecutar paso a paso el código fuente de forma que puedan comprobar los cambios en los valores de las variables. Además, el alumno podrá interactuar dinámicamente con el código, creando marcas visibles en el código, aportando valores sobre las variables en el proceso de aprendizaje colaborativo y estableciendo comentarios en él.

El profesor necesitará disponer de una aplicación que le ayude en la enseñanza colaborativa. En este caso, la enseñanza en programación básica. Por ello, necesitaremos disponer de una aplicación que nos muestre la depuración de un programa. Mediante la depuración, un alumno podrá tener en cuenta, en tiempo real, los cambios realizados en las variables y entender el significado de los cambios en éstas.

En la mayoría de los casos, la depuración de un programa supone tener el programa ejecutable en un ordenador. Por ello, el programa deberá haber sido compilado y en el proceso de depuración deberá estar ejecutándose. Para la elaboración de esta práctica necesitaremos omitir todos estos pasos y dar a disponer al alumno un medio en el que pueda moverse a través del código sin disponer físicamente del programa. De esta manera,

no solo podrá ejecutar la depuración en cualquier sistema, sino que podrá hacerlo de una forma segura y colaborativa.

Para la depuración de estos programas se elegirán dispositivos pequeños que un alumno pueda tener en clase. Así pues, se elegirán dispositivos PDAs, ya que están al alcance de cualquier persona en la actualidad, son fáciles de usar y disponen de las tecnologías de comunicación requeridas para un proceso de educación colaborativa entre todos los dispositivos de una aula.

Así pues, el proceso que se seguiría en un aula mediante la aplicación sería el siguiente: 1) el profesor propone un programa del que posee el código fuente; 2) usa su programa para compilar y crear una traza de depuración; 3) asigna esa traza a un conjunto de alumnos; 4) los alumnos utilizan sus dispositivos PDA para depurar colaborativamente el programa, ayudándose de herramientas de dibujo que le proporciona su aplicación; 5) los alumnos reciben mensajes en sus dispositivos de otros compañeros: los mensajes pueden ser texto, dibujos y otro tipo de ayudas.

1.2.1 Programa del profesor

El programa del profesor es una aplicación de escritorio para Windows en la que podrá administrar cómodamente los alumnos, los grupos y los proyectos.

Se deberá administrar cómodamente los proyectos y los alumnos. De esta forma podremos tener una seguridad en el acceso. El profesor podrá asignar proyectos a clases de alumnos, los cuales podrán seleccionar entre los que están disponibles.

Los alumnos tendrán un número de acceso y contraseña para poder evitar así la suplantación de identidad. También se deberá Otro de los requisitos consistirá en la comunicación mutua entre alumno por vías diferentes, siempre y cuando participen en un mismo proyecto. De esta forma, podrán colaborar entre ellos, ayudándose y retándose.

El proyecto será la unidad básica de nuestro programa y contendrá toda la información que necesitamos para depurar un proyecto y obtener otra información de ésta. Los proyectos tendrán asignados un código fuente que podrá ser modificado mediante una interfaz. Podrán ser compilados y se podrá revisar los errores en la compilación. Así pues, se precisa una IDE de programación que permita realizar estas funciones. La depuración deberá crear trazas de depuración que podrán ser guardadas en una base de datos para la posterior utilización de éstas en tiempo real. El mecanismo de guardar la traza de depuración de forma estática nos permite algunos mecanismos que no nos sería posible utilizar en una depuración común: volver hacia atrás y comprobar valores en las variables. Otro de los objetivos es la creación de distintos estados de cómputo, es decir, dado un proyecto, generar trazas diferentes según su entrada estándar, ya que ésta puede repercutir en el estado de las variables y la traza de la depuración.

Todos estos proyectos y sus trazas deberán ser guardados en una base de datos para poder hacer uso de ésta mediante otras aplicaciones o tener la posibilidad de guardar los datos de forma remota. Será posible la utilización de la herramienta del profesor en varios ordenadores, y todos ellos podrán administrar los proyectos con la misma fuente de información: una base de datos, por ejemplo.

1.2.2 Programa del alumno

El programa del alumno sólo deberá encargarse de recoger los programas ya construidos y depurarlos. Para ello usará un dispositivo PDA que podrá usar también para enviar mensajes a los demás usuarios.

El alumno podrá ser capaz de depurar un programa local o remoto. Los programas locales no precisan de conexión a Internet y el programa estará guardado en memoria. Podrá hacer los mismos usos que si se tratase de un programa remoto, pero no podrá usar sistemas de comunicación como chat o envío de otros mensajes a otros usuarios.

Los programas remotos son el objetivo principal de la aplicación para los alumnos. Mediante éstos podrá comunicarse con los demás alumnos de un grupo. Podrá utilizar una pizarra virtual que compartirán todos, depurar en tiempo real a la par de otros alumnos, mandar un mensaje para anunciar que se dispone a modificar éste y chatear. Todos éstos mensajes se administrarán mediante un servidor administrado por el profesor.

El programa poseerá un sistema de registro para que los alumnos no puedan acceder a proyectos no autorizados o usar cuentas de otros alumnos.

Se creará una pizarra en la cual podremos dibujar figuras básicas como la línea, la cruz o la elipse. Todo ello con una gama de colores básicos. También podrá escribir texto en la pizarra. El objetivo de ésta será crear pequeñas anotaciones en el código o recibir las anotaciones de otros alumnos.

Otra de las características básicas de esta aplicación es el uso de una tabla de variables en las que podrá comprobar el cambio de éstas. Normalmente, cuando usamos aplicaciones de escritorio para la depuración, podemos usar el ratón por encima de las variables para comprobar su estado actual. También podemos crear pequeñas ventanas en las que podemos hacer un seguimiento de las variables. Estas características de aplicaciones de escritorio no se pueden desarrollar en dispositivos pequeños como las PDAs que tienen una interfaz de la que poseemos un espacio limitado. Así pues, la idea de crear una pequeña tabla en la que aparezcan las variables, los valores y sus cambios según el tipo de línea se ajusta a las propiedades de la PDA. En esta tabla podremos modificar los valores de las celdas, pudiendo así comprobar si los valores asignados a las variables son los correctos y aprender de nuestros errores.

El programa tendrá la posibilidad de presentar la interfaz de usuario en otros idiomas, inglés o español, que ayudará a alumnos de diferentes nacionalidades a personalizarlo cómodamente.

2. FUNDAMENTOS

2.1 .NET Framework

Microsoft.NET es el conjunto de nuevas tecnologías en las que Microsoft ha estado trabajando durante los últimos años con el objetivo de obtener una plataforma sencilla y potente para distribuir el software en forma de servicios que puedan ser suministrados remotamente y que puedan comunicarse y combinarse unos con otros de manera totalmente independiente de la plataforma, lenguaje de programación y modelo de componentes con los que hayan sido desarrollados. Ésta es la llamada plataforma .NET, y a los servicios antes comentados se les denomina servicios Web.

Para crear aplicaciones para la plataforma .NET, tanto servicios Web como aplicaciones tradicionales (aplicaciones de consola, aplicaciones de ventanas, servicios de Windows NT, etc.), Microsoft ha publicado el denominado kit de desarrollo de software conocido como .NET Framework SDK, que incluye las herramientas necesarias tanto para su desarrollo como para su distribución y ejecución y Visual Studio.NET, que permite hacer todo lo anterior desde una interfaz visual basada en ventanas.

El concepto de Microsoft.NET también incluye al conjunto de nuevas aplicaciones que Microsoft y terceros han (o están) desarrollando para ser utilizadas en la plataforma .NET. Entre ellas podemos destacar aplicaciones desarrolladas por Microsoft tales como Windows.NET, Hailstorm, Visual Studio.NET, MSN.NET, Office.NET, y los nuevos servidores para empresas de Microsoft (SQL Server.NET, Exchange.NET, etc.)

2.1.1 Common Language Runtime

El Common Language Runtime (CLR) es el núcleo de la plataforma .NET. Es el motor encargado de gestionar la ejecución de las aplicaciones para ella desarrolladas y a las que ofrece numerosos servicios que simplifican su desarrollo y favorecen su fiabilidad y seguridad. Posee de un modelo de programación consistente y sencillo, es decir, es un modelo de programación orientado a objetos, en lo que se diferencia de algunas librerías DLL que ofrecen funciones globales. La programación con .NET Framework abstrae al programador de conceptos de sistemas operativos como el registro de Windows, GUIDs, HRESULTS, IUnknown, etc.

Una de las nuevas ventajas de esta plataforma es la desaparición del concepto conocido como «infierno de las DLL» en las que librerías DLL deben ser sustituidas por nuevas versiones. A su vez, ciertos programas que comparten estas librerías en común pueden tener problemas de convivencia. En .NET las nuevas versiones de las DLLs pueden coexistir con las viejas.

El CLR actúa como una máquina virtual, encargándose de ejecutar las aplicaciones diseñadas para la plataforma .NET. Es decir, cualquier plataforma para la que exista una versión del CLR podrá ejecutar cualquier aplicación .NET. Microsoft ha desarrollado versiones del CLR para la mayoría de las versiones de Windows. Asimismo, Microsoft

está acordando portar las librerías a otros sistemas como Linux. Además, desde cualquier lenguaje para el que exista un compilador que genere código para la plataforma .NET es posible utilizar código generado para la misma usando cualquier otro lenguaje tal y como si de código escrito usando el primero se tratase.

El CLR incluye un recolector de basura que evita que el programador tenga que tener en cuenta cuándo ha de destruir los objetos que dejen de serle útiles. Este recolector es una aplicación que se activa cuando se quiere crear algún objeto nuevo y se detecta que no queda memoria libre para hacerlo. En ese caso, buscará en la memoria dinámica de la aplicación para liberar objetos en desuso.

El CLR posee detecta el correcto tipado de los objetos, con esto, a parte de ayudar al programador durante la compilación, aseguramos que los procesos no puedan acceder a código de otros puesto que no pertenecen al mismo tipo. De igual forma, podemos controlar de manera efectiva las excepciones.

El CLR permite que excepciones lanzadas desde código para .NET escrito en un cierto lenguaje se puedan capturar en código escrito usando otro lenguaje, e incluye mecanismos de depuración que pueden saltar desde código escrito para .NET en un determinado lenguaje a código escrito en cualquier otro.

Otra de las ventajas del CLR es la interoperabilidad con código antiguo, es decir, desde el código escrito para la plataforma .NET podemos tener acceso a objetos COM.

2.1.2 Microsoft Intermediate Language

Ninguno de los compiladores para la plataforma .NET produce código máquina para la CPU. Estos compiladores generan un lenguaje intermedio llamado MSIL (*Microsoft Intermediate Language*) que puede ser interpretado por el CLR, como podemos ver en la figura 2.1.

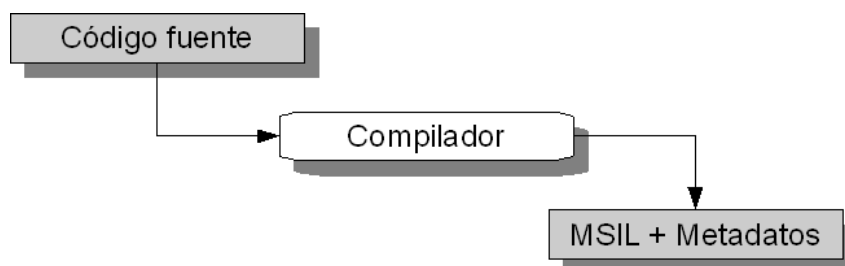


Figura 2.1.: Creación de código MSIL a partir del código fuente.

La principal ventaja del MSIL es la facilitación de la ejecución multiplataforma, ya que se pueden crear CLR para diferentes sistemas sin cambiar el código inicial. Sin embargo, puesto que la CPU no puede ejecutar código directamente del MSIL, se deberá convertir a código nativo de la CPU. Para ello, el CLR tiene un componente llamado *jitter* (*Just-In-Time*) que se encarga de esta tarea. En diferencia a lenguajes como Java, el *jitter* compilará a código nativo una única vez durante el proceso de la ejecución del programa y no interpretará constantemente el código del lenguaje intermedio cuando lo necesita. Podemos ver el proceso del *jitter* en la figura 2.2.

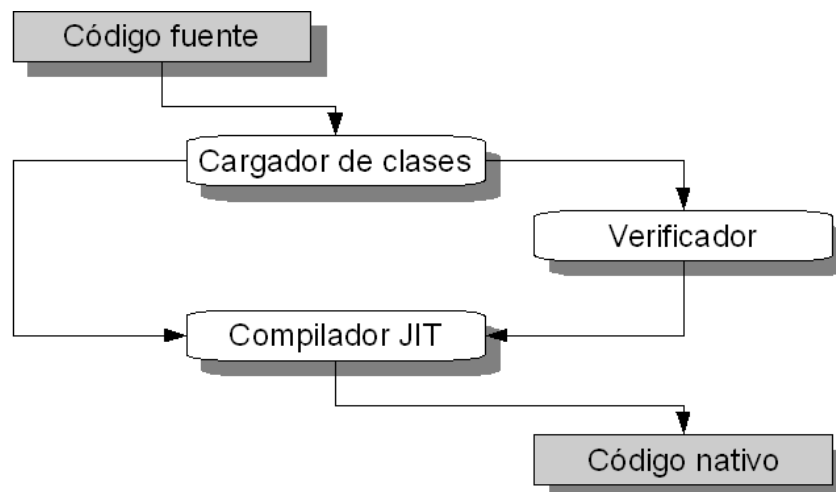


Figura 2.2.: Conversión del código MSIL a código nativo.

2.1.3 Ensamblados

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

2.2 .NET Framework Remoting

.NET Framework Remoting es un marco de trabajo que nos ayuda a crear fácilmente aplicaciones distribuidas y nos permite interactuar entre objetos a través de dominios de aplicación distintos.

Su arquitectura se basa en la transparencia del uso de objetos remotos, de esta manera podremos hacer uso de éstos como si se manejaran de manera local. Nos aporta distintos mecanismos de compartición, canales personalizados, serialización, tiempo de vida y formateadores tanto en la implementación de la aplicación como durante las peticiones.

2.2.1 Objetos remotos

Unos de los puntos más importantes de Remoting es la ocultación y la transparencia a la hora de hacer uso de los objetos remotos. Para ello disponemos de los siguientes mecanismos:

- **Serialización**

Es el acto de representar el estado de un objeto en una secuencia de bits. Esta secuencia debe ser capaz de deserializarse pudiendo recuperar el estado del objeto.

En .NET Remoting, podemos serializar un objeto y enviarlo a través de un canal usando el atributo *[Serializable]* (figura 2.3).

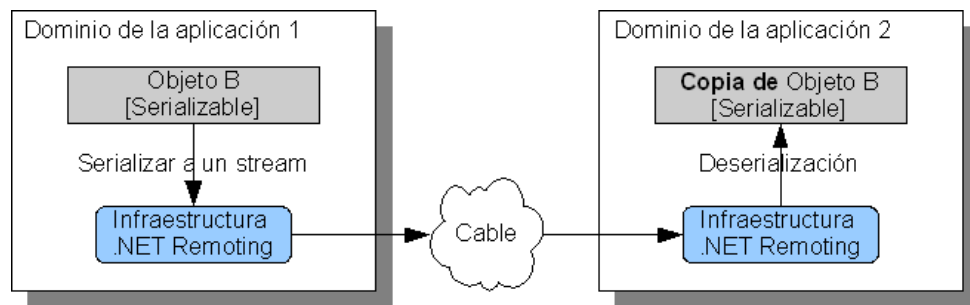


Figura 2.3.: Serializable: el objeto se serializa de un dominio a otro.

■ Marshal-by-Reference

Algunas circunstancias nos impiden que la máquina que quiere hacer uso de este objeto remoto pueda utilizar serialización, ya que ésta necesitará la implementación del objeto para deserializarlo. .NET Remoting define una clase llamada *MarshalByRefObject* de la que podemos derivar cualquier clase para convertirla en remota. De esta forma la máquina que haga uso de ésta, implementará un *proxy* que represente esta clase y sus acciones de forma transparente (figura 2.4).

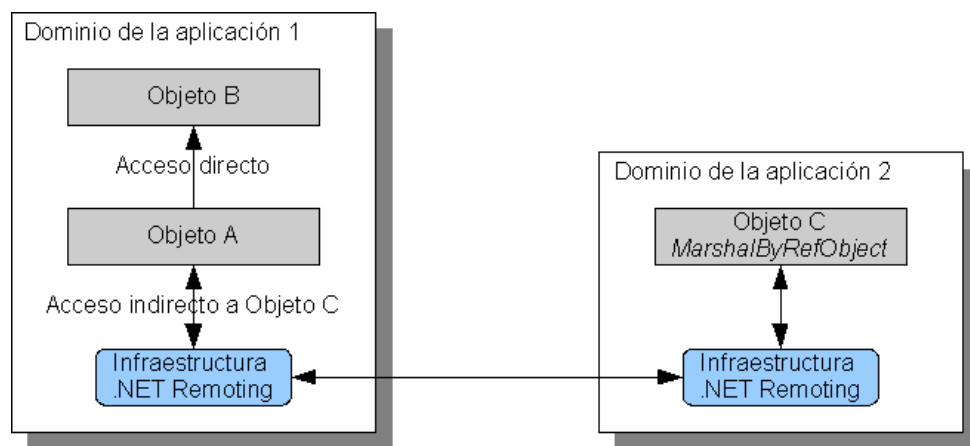


Figura 2.4.: Marshal-by-Reference: el objeto se instancia en su propio dominio.

■ Context-Bound

A veces podemos querer limitar el acceso a los objetos remotos dependiendo del contexto en el que nos encontremos. Así pues, dependiendo de éste, podremos acceder al objeto remoto de una forma u otra. Este tipo de objeto remoto se extiende del anterior y normalmente es usado en transacciones en el que el objeto solo puede ser usado por un dominio a la vez.

2.2.2 Activación de los objetos

La arquitectura de .NET Remoting proviene de algunos mecanismos de publicación de los objetos facilitando URI (*Uniform Resource Identifier*). De esta forma, el servidor podrá procesar las llamadas remotas de una manera conocida y configurable. Existen dos grandes grupos de configuraciones proporcionadas: activadas por el servidor (SAO) y activadas por el cliente (CAO).

■ Activación mediante el servidor (SAO)

Indica que el servidor es responsable de instanciar nuevos objetos remotos y publicarlos desde una conocida dirección. De este modo, el servidor podrá controlar la creación de nuevas instancias y permitir el uso de los objetos remotos. La activación mediante el servidor proviene de dos diferentes semánticas:

● Singleton

El servidor solo creará una instancia del objeto que podrá ser activada en cualquier momento. Esta instancia será usada por todos los clientes de forma secuencial y compartirán el mismo estado entre llamadas (figura 2.5).

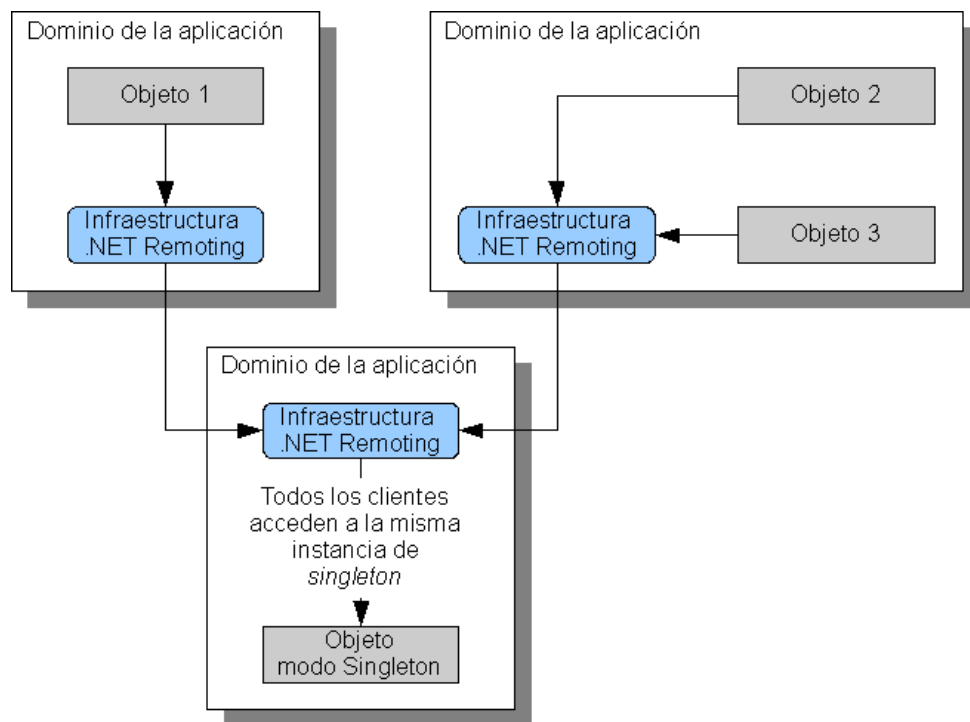


Figura 2.5.: Singleton: Todos los clientes comparten la misma instancia remota.

● SingleCall

El servidor creará una instancia del objeto cada vez que el cliente invoque un método de éste. De esta forma, el estado de la instancia no permanecerá entre dos llamadas distintas (figura 2.6).

■ Activación mediante el cliente (CAO)

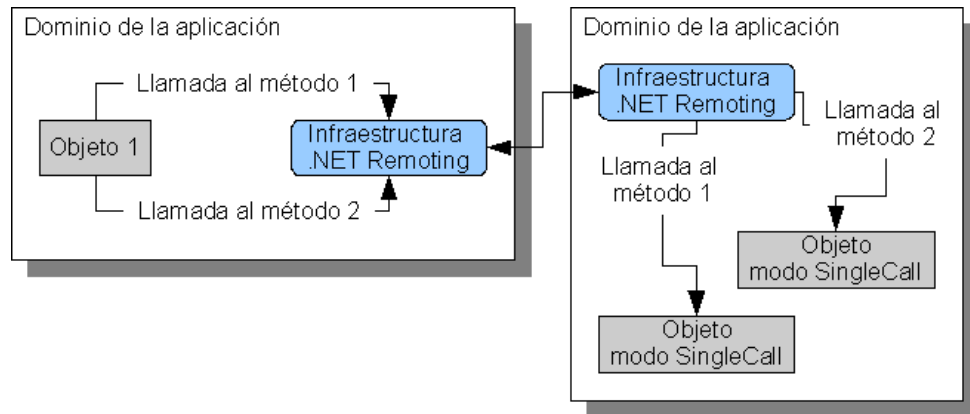


Figura 2.6.: SingleCall: Cada llamada del cliente crea un nuevo objeto sin permanencia del estado.

En algunos dominios es necesario que el cliente requiera de varias instancias a un objeto remoto. En este escenario, el servidor asignará una *URI* a cada instancia que necesite un cliente (figura 2.7).

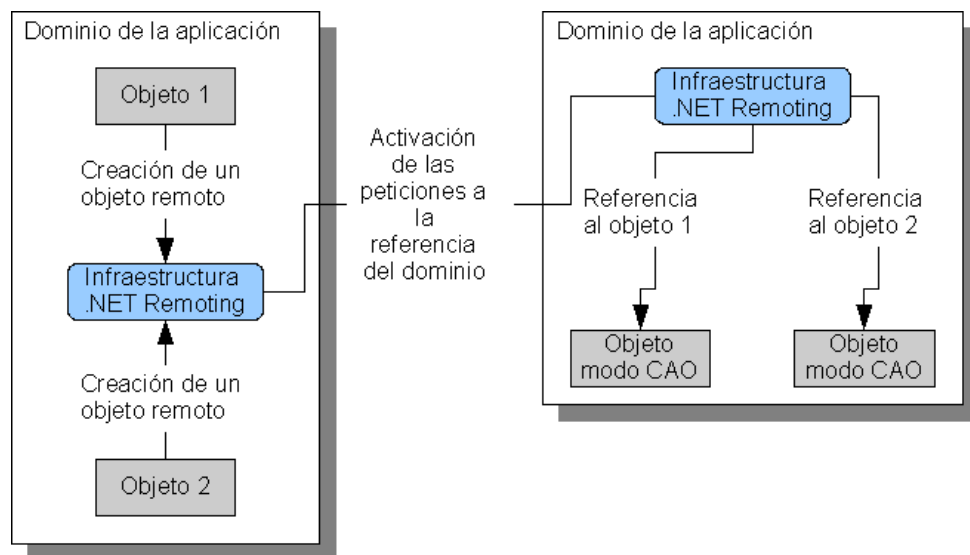


Figura 2.7.: CAO: Cada cliente puede instanciar varios objetos remotos distintos.

2.2.3 Tiempo de vida del objeto

.NET Remoting usa un exclusivo recolector de basura para sus objetos remotos. Un objeto pasa a ser procesado por el recolector de basura mediante unas condiciones predefinidas. Mediante los mecanismos ofrecidos por la plataforma, podemos administrar, controlar y configurar el tiempo de vida de los objetos. Este tiempo de vida dependerá del dominio en el que nos envolvemos. Por ejemplo, es posible querer restaurar el estado de un objeto pasado un día, o dadas un número concreto de peticiones por parte de los clientes.

Mediante la derivación de la clase *MarshalByRefObject* (véase 2.2.1), podremos sobrescribir el método *InitializeLifetimeServices* que controlará el tiempo de vida del objeto.

En caso de sobrescribir este método devolviendo una instancia nula, el objeto nunca pasará por el recolector de basura.

2.2.4 Comunicación de los clientes vía Proxies

Como vimos anteriormente en la figura 2.4 los objetos remotos tratados por el cliente son accedidos mediante una capa que la envuelve. Derivando una clase de *MarshalByRefObject* creamos dos nuevos objetos (figura 2.8), llamados *Proxies* que el cliente usa para interactuar con el servidor:

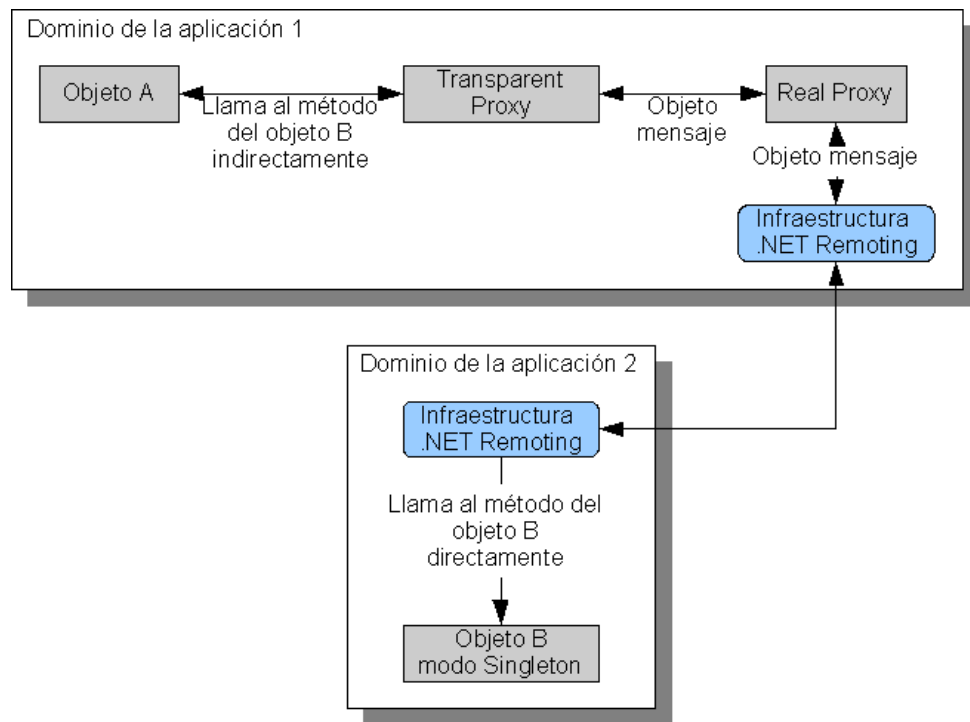


Figura 2.8.: Proxy: Creación de dos nuevos objetos para comunicarse con el servidor.

- ***Transparent Proxy* (Proxy transparente)**

De los dos nuevos objetos creados, este es el único al cual el cliente puede acceder. Este objeto mantiene la misma interfaz que el objeto real. De esta forma, el cliente interactúa con este objeto sin diferenciarlo del objeto remoto.

Una vez que el cliente hace llamadas a los métodos del *Transparent Proxy*, éste convierte esta llamada a un objeto mensaje y se lo envía al *Real Proxy*.

- ***Real Proxy* (Proxy real)**

El *Real Proxy* es el encargado de crear el mensaje correcto que el servidor pueda entender. Para esta tarea, hace uso de la infraestructura .NET Remoting y obtiene los serializadores de mensajes y canales.

2.2.5 Canales

.NET Remoting necesita serializar los objetos en mensajes y mandarlos a través de canales. Un canal en .NET Remoting es el mecanismo de comunicación que existe entre un cliente y un servidor. Estos canales provienen la multitud de protocolos de red que existen. La arquitectura ofrece dos tipos de canales estándar, TCP y HTTP, pero si estos canales no son suficientes, .NET Remoting ofrece la posibilidad de crear nuevos e incorporarlos en forma de pila.

■ Canal TCP

El canal TCP ofrecido por .NET Remoting ofrece una serialización binaria y la utilización de sockets TCP. Esta implementación hace que el mensaje a enviar ocupe muy poco espacio.

■ Canal HTTP

El canal HTTP utiliza cabeceras HTTP para enviar el mensaje. La serialización de la petición se realizará mediante SOAP, lo cual es ideal para la creación de servicios web.

Causa	TCP	HTTP
Rapidez	Es rápido, pues solo manda la información necesaria.	Es más lento, pues debe enviar un documento XML.
Seguridad	Un cortafuegos podría denegar el paso de información binaria.	Un cortafuegos podría permitir peticiones HTTP y con mensajes en XML.
Herramientas		Ofrece WSDL, con lo cual podremos usar servicios web.

Tabla 2.1.: Diferencia entre los canales estándar que ofrece .NET Remoting.

2.2.6 «Hola mundo» distribuido con .NET Framework

Durante esta sección implementaremos un ejemplo de un programa cliente y servidor. El servidor proporcionará un clase que tendrá un método llamado *HolaMundo()* y que devolverá la cadena de caracteres «Hola mundo». Lo primero que haremos será crear una clase que contenga dicho método:

```

1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 namespace Clases
6 {
7     // MarshalByRefObject hace que nuestra clase se comparta
8     // mediante Remoting.
```

```
9      public class ClaseRemota: MarshalByRefObject
10     {
11         public string HolaMundo()
12         {
13             Console.WriteLine("Se ha llamado a este metodo remotamente.");
14             return "Hola mundo";
15         }
16     }
17 }
```

Listado 2.1: Clase remota para nuestro ejemplo distribuido.

Vemos que hemos derivado de la clase *MarshalByRefObject* para indicarle que podremos usar esta clase remotamente.

El servidor lo configuraremos para que comparta esta clase mediante el puerto 1234 y se quede esperando a las peticiones.

```
1  // Archivo: Servidor.cs
2  // Biblioteca estandar y remoting
3  using System;
4  using System.Runtime.Remoting;
5
6  // Bibliotecas para usar un canal HTTP
7  using System.Runtime.Remoting.Channels;
8  using System.Runtime.Remoting.Channels.Http;
9  using Clases;
10
11 public class Servidor
12 {
13     public static void Main(string[] args)
14     {
15         // Creamos un nuevo objeto para el canal HTTP
16         // y le asignamos un puerto: 1234
17         HttpChannel chnl = new HttpChannel(1234);
18
19         // Le decimos a la maquina que nos reserve ese canal
20         ChannelServices.RegisterChannel(chnl);
21
22         // Opciones para la comparticion de la clase
23         RemotingConfiguration.RegisterWellKnownServiceType(
24             typeof(Clases.ClaseRemota),
25             "ClaseRemota.soap",
26             WellKnownObjectMode.SingleCall);
27
28         // Mensajes de espera en la pantalla
29         Console.WriteLine("Atendiendo las peticiones...");
30         Console.WriteLine("Pulse Enter para salir...");
31         Console.ReadLine();
32     }
33 }
```

Listado 2.2: Servidor que compartirá la clase remota.

El cliente contactará con el servidor y crearemos una instancia remota de la clase.

```

1 // Archivo: Cliente.cs
2 // Biblioteca estandar y remoting
3 using System;
4 using System.Runtime.Remoting;
5
6 // Bibliotecas para usar un canal HTTP
7 using System.Runtime.Remoting.Channels;
8 using System.Runtime.Remoting.Channels.Http;
9 using Clases;
10
11 public class Cliente
12 {
13     public static void Main(string[] args)
14     {
15         // Creamos un nuevo objeto para el canal HTTP
16         HttpChannel chnl = new HttpChannel();
17
18         // Le decimos a la maquina que nos reserve ese canal
19         ChannelServices.RegisterChannel(chnl);
20
21         // Opciones para la utilizacion remota de la clase
22         RemotingConfiguration.RegisterWellKnownClientType(
23             typeof(Clases.ClaseRemota),
24             "http://localhost:1234/ClaseRemota.soap");
25
26         // Ahora, usamos la clase remotamente (vemos que
27         // el estado del servidor cambia mostrando un mensaje).
28         ClaseRemota objeto = new ClaseRemota();
29         Console.WriteLine(objeto.HolaMundo());
30     }
31 }

```

Listado 2.3: Cliente que utilizará la clase remota.

Para comprobar que efectivamente hay una comunicación distribuida, hemos creado una traza dentro del método de la clase para que aparezca en el servidor si uno de los clientes utiliza el método. Así pues, los resultados obtenidos del cliente son:

Hola mundo

Y del servidor:

Atendiendo a las peticiones...
Pulse Enter para salir...
Se ha llamado a este metodo remotamente.

2.3 .NET Compact Framework

.NET Compact Framework (.NET CF) es una versión de .NET Framework diseñada para correr bajo Windows CE basados en dispositivos móviles y embebidos como PDAs, teléfonos móviles, controladores de factorías, sintonizadores de TV, etc. .NET CF usa algunas de las mismas librerías que .NET Framework, aunque algunas de ellas tuvieron que rediseñarse específicamente para los dispositivos móviles, como son los controles de los formularios.

Para poder cargar aplicaciones escritas en .NET CF, la plataforma debe tener instaladas las librerías de .NET Compact Framework. Existen varias versiones para los dispositivos móviles más comunes como Windows CE 4.1, Pocket PC, Smartphone, Windows Mobile. Las aplicaciones implementadas en .NET CF no pueden ser ejecutadas en un PC aunque los archivos binarios resultantes sean compatibles, ya que los programas poseen distintas firmas digitales. Ésta es una forma de evitar arrancar aplicaciones escritas en .NET Framework sobre dispositivos con .NET Compact Framework.

El principal motivo de la utilización de .NET Compact Framework fue la facilidad en la adaptación de los conocimientos de .NET Framework a .NET CF. Otra de las características importantes en .NET CF es la fácil utilización en la creación de interfaces para los programas. Visual Studio provee de una magnífica IDE para la creación de éstos y existe una gran documentación de esta plataforma en Internet. Además, en la actualidad se está portando la plataforma .NET CF a otros dispositivos móviles, en lugar de sólo a Windows Mobile, como a Symbian, lo que nos ayuda a reaprovechar los programas que construyamos durante este proyecto.

2.3.1 Diferencias entre .NET Framework y .NET Compact Framework

.NET Compact Framework implementa un 30 % de las librerías de .NET Framework. A continuación se describen algunas de las consideraciones que se tienen en cuenta a la hora de escribir programas para .NET Compact Framework.

- **Ensamblados y caché de ensamblados global:** De momento, .NET Compact Framework no admite ensamblados de varios módulos, pero sí que admite ensamblados satélite. Los ensamblados satélite se encargan de implementar recursos específicos del lenguaje en una aplicación.
- **Clases y tipos:** .NET Compact Framework admite un subconjunto de la biblioteca de clases de .NET Framework. Dicho subconjunto es apropiado para las aplicaciones diseñadas para ejecutarse en dispositivos con limitaciones de recursos y es semánticamente compatible con las clases del mismo nombre de .NET Framework.
- **Common Language Runtime:** Los Common Language Runtime de los dos entornos de Framework se benefician de la ejecución de código administrado, la compilación de código Just-In-Time (JIT) y la recolección de elementos no utilizados. Los dos son compatibles con la CLS (Common Language Specification).

Los dos marcos de trabajo tienen tipos primitivos integrados, así como otros tipos que pueden utilizarse y derivarse cuando se genera una aplicación.

El Common Language Runtime de .NET Compact Framework ocupa aproximadamente un 12 % del tamaño del Common Language Runtime de la versión completa de .NET Framework.

- **Controles:** Los controles de formularios Windows Forms están específicamente diseñados para .NET Compact Framework.
- **Directorio actual:** La funcionalidad de un directorio actual no está presente en el sistema operativo Windows Embedded CE. Por lo tanto, .NET Compact Framework no admite los métodos *GetCurrentDirectory* y *SetCurrentDirectory*. .NET Compact Framework admite la propiedad *WorkingDirectory* de un objeto *ProcessStartInfo*. El ejecutable en ejecución no conserva, sin embargo, su contexto en los inicios y cargas de archivo subsiguientes.
- **Datos:** .NET Compact Framework proporciona una implementación del subconjunto de ADO.NET e incluye el proveedor de datos SQL Server Mobile. No se admite el espacio de nombres *System.Data.OleDb*.
- **Depurar en el símbolo del sistema:** .NET Compact Framework no admite el depurador Depurador de la línea de comandos de .NET Framework (*MDbg.exe*) que se incluye en .NET Framework 2.0. El antiguo CLR Debugger (*DbgCLR.exe*) se ha dejado de utilizar en la versión 2.0 de los dos entornos Framework.
- **Eventos:** .NET Compact Framework admite los eventos *GotFocus* y *LostFocus*, pero no es compatible con los eventos *Activated* y *Deactivated*.
- **Cadenas de descripción de excepciones**
 .NET Compact Framework proporciona las cadenas de mensajes de error de las excepciones en un archivo DLL independiente, denominado *System.SR.dll*, con el fin de ahorrar memoria.
- **Nombres de archivo y rutas de acceso:** Windows Embedded CE resuelve los nombres de archivo especificados sin la información de la ruta de acceso, como si estuvieran ubicados en el directorio raíz del dispositivo y no en el directorio de la aplicación. Para asegurarse de que las operaciones se realizan correctamente, se necesita especificar las rutas de acceso absolutas.
 .NET Compact Framework procesa las cadenas de identificador uniforme de recursos (URI) que llevan como prefijo *file://* de forma distinta que el entorno completo de .NET Framework. Una especificación relativa, como *file://myfile*, se resuelve como *\\myfile*. La cadena de URI *file:///myfile* (con tres barras diagonales) se resuelve como *\\myfile* en el directorio raíz.
 Puede obtener la versión de un ensamblado con la propiedad *Version*, pero su compatibilidad depende del fabricante del dispositivo y, por lo tanto, no está garantizada.
- **Entrada/salida (E/S):** A causa de las diferencias entre los sistemas operativos, existen restricciones y limitaciones en el modelo de E/S. .NET Compact Framework no proporciona notificaciones de cambios en los archivos.

Dado que la E/S en los dispositivos ocurre en la RAM, no es posible definir ni obtener acceso a los atributos de archivos y directorios.

- **Instalación y archivos CAB:** Puede usar archivos CAB y crear aplicaciones de Microsoft Windows Installer para distribuir sus aplicaciones.
- **Lenguajes:** .NET Compact Framework admite el desarrollo en Visual Basic y Visual C# pero, de momento, no admite el desarrollo en C++.
- **Math:** No todos los métodos matemáticos se admiten en todas las plataformas de dispositivos, pero se incluyen en la API por motivos de compatibilidad.
- **Memoria:** .NET Compact Framework está optimizado para sistemas alimentados por baterías y evita el uso intensivo de ciclos de RAM y de CPU.
- **Conexión de red:** .NET Compact Framework proporciona clases de asociación de datos por infrarrojos (IrDA) para establecer conexiones por infrarrojos y clases de escucha web para atender solicitudes HTTP en el dispositivo. Estas clases sólo están disponibles en .NET Compact Framework.
- **Código proxy:** .NET Compact Framework no admite todo el código generado por Herramienta Lenguaje de descripción de servicios Web (*Wsd.exe*).
- **Reflexión:** .NET Compact Framework no admite el espacio de nombres *System.Reflection.Emit*. En estos momentos, .NET Compact Framework no admite el comparador de igualdad (==) cuando compara objetos de reflexión, como *MethodInfo*, *FieldInfo*, etc.
- **Comunicación remota:** .NET Compact Framework no admite las librerías .NET Remoting de .NET Framework.
- **Mensajería segura:** .NET Compact Framework no admite los certificados y la autenticación de cliente mediante HTTPS. Se suele utilizar la autenticación básica.
- **Serialización:** Por cuestiones de tamaño y rendimiento, .NET Compact Framework no admite la serialización binaria mediante *BinaryFormatter* o la serialización SOAP mediante *SoapFormatter*.
No obstante, .NET Compact Framework proporciona compatibilidad con la serialización para transmitir datos de objetos por medio de SOAP en los servicios web XML y la serialización de conjuntos de datos para XML.
- **Tamaño:** .NET Compact Framework representa el 8 por ciento del tamaño del paquete redistribuible completo de .NET Framework. El tamaño en disco es un 50 % más pequeño debido a la compresión del sistema de archivos de Windows Embedded CE.
- **Sockets:** No se admiten todas las opciones de socket.
- **Threads:** Una aplicación .NET Compact Framework crea hasta cuatro subprocesos:
 - Un subproceso de aplicación principal.

- Un subproceso que se utiliza para controlar varios temporizadores y tiempos de espera que pueden programar las aplicaciones o el sistema.
 - Un subproceso que se utiliza para realizar un seguimiento de los cambios en las interfaces TCP/IP activas (que simulan el comportamiento de detección de medios).
 - Un subproceso que se utiliza para ejecutar los finalizadores de objetos.
- **Intervalos de tiempo:** La propiedad *Now* devuelve un valor con una precisión de segundos, no de milisegundos. Para obtener una medición más precisa, se utiliza la propiedad *TickCount*.
 - **Temporizadores:** No se admiten los métodos *Start* y *Stop* para un objeto *System.Timers.Timer*, pero puede iniciar y detener la temporización estableciendo la propiedad *Enabled* de un objeto *System.Windows.Forms.Timer* en *true* o *false*.
 - **Servicios Web:** El cliente de servicios web ejecuta directamente los ensamblados generados por *wsdl.exe*. No se debe utilizar *localhost* para crear un servicio web en el dispositivo, ya que *localhost* hace referencia al dispositivo que ejecuta la aplicación. En su lugar, se debe utilizar el nombre del equipo o su dirección IP.
 - **XML:** Debido a cuestiones de tamaño, .NET Compact Framework no admite la validación del esquema XML. No acepta el DOM (Modelo de objetos de documento) de XML.

2.4 MySQL

MySQL es un sistema de gestión de base de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones. MySQL AB (desde enero de 2008 una subsidiaria de Sun Microsystems) desarrolla MySQL como software libre en un esquema de licenciamiento dual.

Por un lado se ofrece bajo la GNU GPL para cualquier uso compatible con esta licencia, pero las empresas que quieran incorporarlo en productos privativos pueden comprar a la empresa una licencia específica que les permita este uso. Está desarrollado en su mayor parte en ANSI C.

Al contrario que proyectos como Apache, donde el software es desarrollado por una comunidad pública y el copyright del código está en poder del autor individual, MySQL es propiedad y está patrocinado por una empresa privada, que posee el copyright de la mayor parte del código.

Esto es lo que posibilita el esquema de licenciamiento anteriormente mencionado. Además de la venta de licencias privativas, la compañía ofrece soporte y servicios. Para sus operaciones contratan trabajadores alrededor del mundo que colaboran vía Internet. MySQL AB fue fundado por David Axmark, Allan Larsson, y Michael Widenius.

La versión de MySQL utilizada durante este proyecto fue la 5.0.51a.

2.5 Servicios web

Un servicio web (en inglés Web service) es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos. Las organizaciones OASIS y W3C son los comités responsables de la arquitectura y reglamentación de los servicios Web. Para mejorar la interoperabilidad entre distintas implementaciones de servicios Web se ha creado el organismo WS-I, encargado de desarrollar diversos perfiles para definir de manera más exhaustiva estos estándares.

La razón por la que se han extendido tantos los servicios web es por el uso del protocolo HTTP y el puerto 80. Cualquier aplicación, como un navegador, es capaz de hacer una petición mediante ese puerto y procesar la respuesta obtenida por el servicio web en XML. El uso de estándares abiertos también ha potenciado su uso.

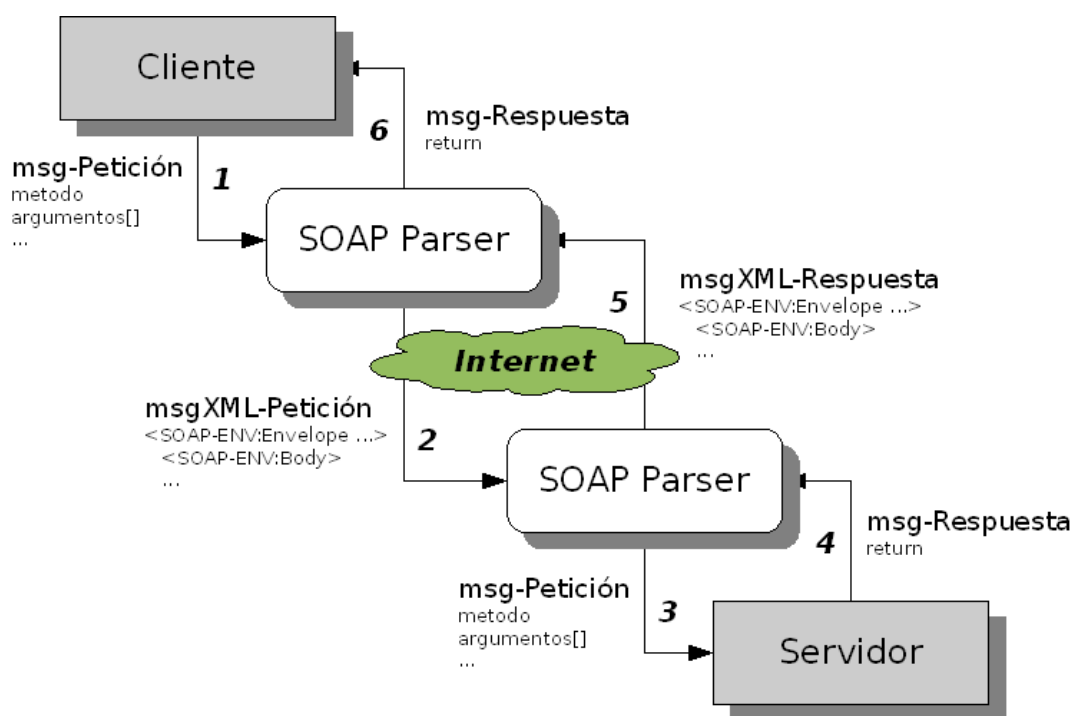


Figura 2.9.: Servicios web: petición de un servicio de un cliente a un servidor.

Los servicios web no son la forma más potente de comunicación entre aplicaciones, pero el uso de XML en los mensajes hacen que esta comunicación sea fácilmente entendible. Estándares como CORBA o RMI superan su rendimiento, ya que los mensajes no se basan en texto.

Los servicios web están basados en las siguientes tecnologías:

- **XML**

La base de los servicios web es el lenguaje XML. Los mensajes que se intercambian entre el cliente y el servidor tienen este formato.

■ Web Service Description Language (WSDL)

Es el lenguaje de la interfaz pública para los servicios Web. Es una descripción basada en XML de los requisitos funcionales necesarios para establecer una comunicación con los servicios Web.

■ Simple Object Access Protocol (SOAP)

Basado también en XML, permite la comunicación entre los clientes del servicio y el propio servicio. Define el formato XML que deberá utilizarse en las comunicaciones entre clientes y proveedor. Normalmente funciona sobre HTTP, aunque puede hacerlo sobre SMTP, MSMQ. Podemos ver un ejemplo de peticiones entre cliente y servidor en el siguiente código:

```

1 <SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
4   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
5   xmlns:clr="http://schemas.microsoft.com/clr/"
6   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
7 <SOAP-ENV:Body>
8   <i2:Suma id="ref-1"
9     xmlns:i2="http://schemas.microsoft.com/clr/nsassem/Calculo.
10      Calculadora/Calculo">
11     <izq xsi:type="xsd:double">2</izq>
12     <dcha xsi:type="xsd:double">2</dcha>
13   </i2:Suma>
14 </SOAP-ENV:Body>
15 </SOAP-ENV:Envelope>
16
17 <SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
18   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
19   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
20   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
21   xmlns:clr="http://schemas.microsoft.com/clr/"
22   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
23 <SOAP-ENV:Body>
24   <i2:SumaResponse id="ref-1"
25     xmlns:i2="http://schemas.microsoft.com/clr/nsassem/Calculo.
26      Calculadora/Calculo">
27     <return xsi:type="xsd:double">4</return>
28   </i2:SumaResponse>
29 </SOAP-ENV:Body>
30 </SOAP-ENV:Envelope>

```

Listado 2.4: Ejemplo de peticiones en SOAP

■ Universal Description Discovery and Integration (UDDI)

Permite la publicación de los servicios web, de forma que sean accesibles a otros clientes. Básicamente son directorios de servicios web, en los que los proveedores dan de alta sus servicios, y los clientes buscan servicios adecuados a sus necesidades.

2.6 GDB Debugger

Durante el desarrollo de este proyecto se utilizó el programa de código abierto GDB para la depuración de los programas y la obtención posterior de las trazas que utilizarán los alumnos. GDB o GNU Debugger es el depurador estándar para el sistema operativo GNU. Es un depurador que admite varias plataformas y admite varios lenguajes de programación como C, C++, PASCAL, Objective-C, Fortran, ADA, Assembly y actualmente JAVA y C# con Mono. Fue uno de los primeros programas escritos por Richard Stallman junto con el compilador GCC y gracias a colaboradores y empresas interesadas en su desarrollo es uno de los depuradores más potentes de la actualidad.

GDB es un potente depurador que da la posibilidad de modificar la ejecución del programa y controlar sus variables internas. Es capaz de depurar un programa aun habiendo estado ejecutándose anteriormente. Posee mecanismos de control de eventos para determinar cuando un programa se paró en caso de error y en que condiciones.

Es un depurador que se controla mediante una interfaz de línea de comandos, pero en la actualidad existen varias alternativas front-end como DDD, GDBtk/Insight o mediante Emacs.

GDB está licenciado bajo la licencia de software libre GNU GPL.

2.6.1 Ejemplo de depuración con GDB

Mediante el siguiente ejemplo, recorreremos paso a paso cada una de las instrucciones del código fuente. Además, en ciertas ocasiones, mostraremos el valor que las variables tienen en ese momento:

```

1 PROGRAM HelloWorld;
2
3 PROCEDURE Procedimiento();
4 VAR
5     A: INTEGER;
6     S: STRING;
7 BEGIN
8     A := 5;
9     S := 'Hello world into procedure.';
10
11     WriteLn(S);
12 END;
13
14 VAR
15     J: INTEGER;
16
17 BEGIN
18     J := 8;
19
20     Procedimiento();
21
22     WriteLn(J);
23 END.
```

Listado 2.5: Ejemplo del programa que depuraremos con GDB.

Así pues, una típica sesión con GDB para depurar este programa podría ser la siguiente (NOTA: los corchetes son comentarios durante el código, el símbolo «¿» representa entradas por teclado y las líneas sin símbolo son salidas de la consola):

```
[Compilacion del programa HolaMundo con FreePascal y
con opcion de depuracion con GDB]
>fpc -g HelloWorld.pas
Free Pascal Compiler version 2.2.2 [2008/08/03] for i386
Copyright (c) 1993-2000 by Florian Klaempfl
...

[Inicializamos el depurador]
>gdb HelloWorld.exe
[Nos situamos en la primera linea y arrancamos el programa]
>b main
Breakpoint 1 at 0x4013de: file HelloWorld.pas, line 18
> run
Starting program: C:\HelloWorld.exe

Breakpoint 1, main() at HelloWorld.pas:18
18      J := 8;
[Comprobamos las variables que hay en ese momento]
>info variables
All defined variables:

File HelloWorld.pas:
static J : SMALLINT;
...

[Recogemos el valor de la variable J]
>p J
$1 = 0

[Saltamos una linea]
>n
20      Procedimiento();

[Nos metemos dentro del procedimiento]
>s
PROCEDIMIENTO () at HelloWorld.pas:8
8      A := 5;

[Comprobamos las variables dentro del procedimiento]
>info locals
```

2. FUNDAMENTOS

```
A = -29436  
S = #00##030...
```

```
[Pasamos dos lineas hacia delante]
```

```
>n  
9      S := 'Hello world into procedure.';  
>n  
11     WriteLn(S);
```

```
[Comprobamos de nuevo los valores de las variables]
```

```
>info locals  
A = 5  
S = 'Hello world into procedure.'
```

```
[Salimos de la depuracion]
```

```
>q
```

3. DESARROLLO DE LA SOLUCIÓN

Durante esta sección desglosaremos las etapas que harán posible el desarrollo de la aplicación. Desde la etapa de requisitos, dónde recogemos las peticiones del cliente, continuando con un análisis que darán lugar a un esquema bien definido de la aplicación y terminando con el desarrollo del programa.

3.1 Requisitos

La etapa de requisito es una de las más importantes, pues recoge la funcionalidad del programa. Esta funcionalidad surge de la necesidad del cliente para realizar una tarea concreta. Es una etapa dinámica, pues durante el desarrollo pueden aparecer nuevos requisitos y la exclusión o sustitución de otros. Algunos de estos requisitos suelen ser, a priori, desconocidos por el cliente. Así pues, durante la implementación de la interfaz gráfica de las aplicaciones, surgen variados cambios y nuevas ideas por parte del cliente. Se puede decir que la etapa de requisitos es el contrato entre el desarrollador y el cliente.

Es importante crear tablas de requisitos y diagramas de casos de uso sin ambigüedad en el contenido, pues errores en esta fase pueden conllevar al arrastre de éstos durante las demás etapas con una difícil resolución, además de la contaminación del error en otros requisitos.

3.1.1 Escenario del problema

Se requiere implementar una herramienta que ayude al alumno a depurar un programa de forma colaborativa y distribuida. El profesor podrá poner a disposición de varios alumnos el código fuente de un programa y éstos podrán avanzar y retroceder en su traza. Además, habrá herramientas para que el alumno pueda intuir el valor de las variables del programa y la corrección de sus respuestas contra los valores reales. Para esta tarea se implementarán dos herramientas, una destinada al profesor en la que gestionará los programas y otra al alumno en la que podrá depurar el programa.

- **Generador y administrador de trazas del profesor**

El profesor dispondrá de una herramienta en la cual pueda administrar, compilar y generar trazas de los nuevos proyectos que añada. Un proyecto se compone de un código fuente y varios estados de cómputo. Un estado de cómputo es un estado inicial que se aplica a un programa haciendo que este se comporte de maneras diferentes durante su ejecución. El programa debe ser capaz de compilar el código asociado al proyecto según cada uno de sus estados de cómputo. Una vez terminada la compilación, deberá mostrar el resultado de ésta por pantalla. Además, la herramienta dispondrá de una fácil administración de todos los proyectos pudiendo crear grupos de usuarios que puedan acceder a ellos.

■ Editor de trazas colaborativo del alumno

El alumno dispondrá de una herramienta para una máquina Pocket PC que muestre el código fuente de un programa y un editor de las variables del mismo. El usuario requerirá de un mecanismo de autenticación para poder acceder a los proyectos. El alumno podrá recorrer las líneas de la traza del programa pudiendo observar los cambios que otros alumnos han generado sobre ésta.

En la figura 3.1 podemos ver un sencillo esquema de la propuesta del sistema distribuido. El programa del profesor podrá crear trazas del código fuente de sus programas. Una vez creadas las trazas, éstas serán grabadas en una base de datos. El profesor, mediante la administración del programa, podrá indicar al servidor que trazas pueden estar disponibles en Internet y que usuarios pueden recoger la información de éstas. Un usuario, desde una PDA, puede preguntar al servidor si es posible recoger una de esas trazas (es decir, el profesor le dio permiso desde su programa). Una vez recogida la traza, podrá interactuar con ella desde su aplicación en la PDA. A su vez, el servidor ofrecerá servicios para poder comunicarse con todas las PDAs que estén registradas en el servidor. Para que un cliente pueda interactuar colaborativamente con otro, deberá hacerlo a través del servidor, pues éste es el único que guarda las direcciones IPs de los usuarios.

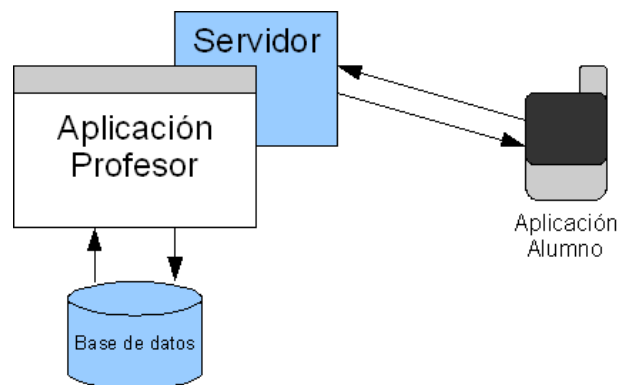


Figura 3.1.: Esquema general del sistema.

3.1.2 Especificación de requisitos

Una vez obtenida una descripción de la aplicación por parte del cliente, recogeremos los requisitos formalmente diferenciándolos entre requisitos funcionales, no funcionales y requisitos del hardware. Esta tabla no solo especifica el escenario del problema (sección 3.1.1) dado inicialmente por el cliente, sino requisitos que se han podido ser modificados (siempre y cuando fueron posibles las modificaciones) durante la etapa de análisis y desarrollo a petición del cliente.

Las tablas que expondremos posteriormente reflejan tres campos: el tipo de requisito, el nombre del requisito y la descripción de éste. Entre los requisitos que expondremos, podemos diferenciar tres tipos:

- RFi: Requisito funcional i.

- RNF*i*: Requisito no funcional *i*.
- RH*i*: Requisito hardware *i*.

Requisito	Nombre	Descripción
RF1	Base de datos	Cualquier dato, excluyendo la configuración del programa se almacenará en la base de datos.
RF2	Administración	El programa administrará proyectos, alumnos y grupos. Tendrá operaciones básicas de bases de datos.
RF3	Edición de proyectos	El programa podrá ser capaz de editar el código de los programas fuente y compilarlos para generar las soluciones.
RF4	Edición de alumnos y grupos	El programa será capaz de crear nuevos alumnos y grupos y modificar sus datos.
RNF1	Interfaz	Las interfaces deben adaptarse al profesor.
RNF2	Idioma	La interfaz deberá escribirse en inglés y español.
RNF3	Estados de cómputo Código externo	Cada proyecto podrá generar diferentes trazas según su estado de cómputo. Un estado de cómputo es un estado inicial que proporcionamos al programa para que tome rutas diferentes o cambios en sus variables durante su depuración. Los proyectos podrán obtener el código de ficheros externos.
RH1	Librerías	Se deberá poder utilizar un ordenador personal que pueda utilizar el framework .NET de Microsoft.
RH2	Acceso base de datos	El programa deberá tener acceso a la base de datos tanto localmente como remotamente.

Tabla 3.1.: Requisitos para el programa de administración.

Requisito	Nombre	Descripción
RF1	Base de datos	El servidor deberá crear sus servicios basados en los datos contenidos en la base de datos.
RF2	Actualización	El servidor será sensible a cambios en la base de datos modificando sus servicios de forma correcta.
RNF1	Autenticación	Solo se podrán hacer uso de los servicios mediante un nombre de usuario y contraseña.
RNF2	Servicios web	Se deberá acceder a los servicios mediante servicios web usando WSDL y SOAP.
RH1	Librerías	Se deberá poder utilizar un ordenador personal que pueda utilizar el framework .NET de Microsoft.
RH2	Acceso base de datos	El programa deberá tener acceso a la base de datos tanto localmente como remotamente.

Tabla 3.2.: Requisitos para el programa servidor.

Requisito	Nombre	Descripción
RF1	Proyectos remotos y locales	Se podrán acceder a proyectos remotos vía internet y a proyectos locales contenidos en la memoria de la PDA.
RF2	Colaboración	Los cambios realizados en los proyectos remotos se deberán visualizar en las PDAs que estén usando el mismo proyecto remoto.
RNF1	Autenticación	Solo se podrán hacer uso de los servicios mediante un nombre de usuario y contraseña.
RNF2	Paneles en la PDA	La aplicación deberá contener un panel en la que se represente el código y en otra las variables.
RNF3	Zona de dibujo	En el panel del código se podrán dibujar varias figuras que se ofrecen pero con tamaños modificables.
RNF4	Limpieza de pantalla	El usuario podrá limpiar todos los dibujos de la pantalla.
RNF5	Operaciones de traza	Se deberán poder utilizar las operaciones básicas de traza: paso a paso, siguiente, atrás.
RH1	Librerías	Se deberá poder utilizar una PDA con Windows CE 2003 que pueda utilizar el framework .NET Compact de Microsoft.
RH2	Acceso a Internet	Deberá ser posible una conexión a Internet para poder acceder a proyectos remotos colaborativos.
RH3	Servidor	Deberá ser posible que la conexión permita la creación de un servidor en la PDA con un puerto concreto modificable.

Tabla 3.3.: Requisitos para el cliente para PDA.

3.1.3 Diagrama de casos de uso

Un Diagrama de Casos de Uso muestra la relación entre los actores y los casos de uso del sistema. Representa la funcionalidad que ofrece el sistema en lo que se refiere a su interacción externa. En el diagrama de casos de uso se representa también el sistema como una caja rectangular con el nombre en su interior. Los casos de uso están en el interior de la caja del sistema, y los actores fuera, y cada actor está unido a los casos de uso en los que participa mediante una línea.

Los elementos que pueden aparecer en un Diagrama de Casos de Uso son: actores, casos de uso y relaciones entre casos de uso.

■ Actores

Un actor es algo con comportamiento, como una persona (identificada por un rol), un sistema informatizado u organización, y que realiza algún tipo de interacción con el sistema. Se representa mediante una figura humana dibujada con palotes. Esta representación sirve tanto para actores que son personas como para otro tipo de actores.

■ Casos de Uso

Un caso de uso es una descripción de la secuencia de interacciones que se producen entre un actor y el sistema, cuando el actor usa el sistema para llevar a cabo una tarea específica. Expresa una unidad coherente de funcionalidad, y se representa en

el Diagrama de Casos de Uso mediante una elipse con el nombre del caso de uso en su interior. El nombre del caso de uso debe reflejar la tarea específica que el actor desea llevar a cabo usando el sistema.

■ Relaciones entre Casos de Uso

Un caso de uso, en principio, debería describir una tarea que tiene un sentido completo para el usuario. Sin embargo, hay ocasiones en las que es útil describir una interacción con un alcance menor como caso de uso. La razón para utilizar estos casos de uso no completos suele ser mejorar la comunicación en el equipo de desarrollo, el manejo de la documentación de casos de uso. Las relaciones entre estos y los casos de uso ordinarios pueden ser de los siguientes dos tipos:

- **Incluye**

Un caso de uso base incorpora explícitamente a otro caso de uso en un lugar especificado en dicho caso base. Se suele utilizar para encapsular un comportamiento parcial común a varios casos de uso.

- **Extiende**

Cuando un caso de uso base tiene ciertos puntos (puntos de extensión) en los cuales, dependiendo de ciertos criterios, se va a realizar una interacción adicional. El caso de uso que extiende describe un comportamiento opcional del sistema (a diferencia de la relación incluye que se da siempre que se realiza la interacción descrita).

Por problemas de espacio en la memoria, a continuación se analizarán los casos de uso más representativos del proyecto.

■ Diagrama de casos de uso para el programa del profesor

El profesor necesitará un programa que pueda administrar cómodamente la información. Este tipo de administración se basará en la creación de grupos, proyectos y alumnos.

Otra de las funcionalidades importantes del programa será la de la creación y generación de trazas en XML para que puedan ser accesibles fácilmente (véase 2.5).

Las tareas que el programa del profesor ofrecerá serán la creación de nuevos proyectos que contengan código fuente y trazas, la creación de nuevos alumnos y la creación de nuevos grupos. Todos estos tipos de objetos tendrán que ser administrados para poder asignar a los grupos los proyectos y los alumnos (Figura 3.2).

■ Diagrama de casos de uso para los servicios web

Los servicios web que se ofrecen son la base entre la comunicación entre los dispositivos y el servidor. Así pues, éstos nos darán acceso a la información de los proyectos, grupos de alumnos y trazas. También nos permitirán registrar nuestras IPs para informarnos de cualquier cambios. De este modo implementamos un sistema de eventos vía servicios web.

Los métodos más importantes del sistema serán los de la modificación de la traza de un proyecto, pudiendo continuar o retroceder en la traza. También se implementa un método genérico para incorporar nuevas funcionalidades basadas en mensajes: chat, peticiones de

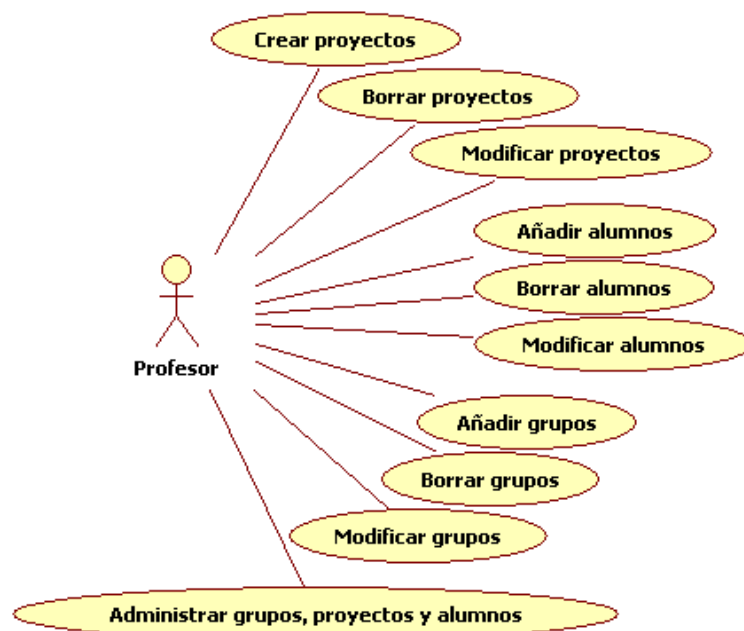


Figura 3.2.: Casos de uso del programa del profesor.

utilización, actualización de variables, etc. Este método nos ayudará a enviar mensajes al servidor siempre y cuando el servidor conozca el protocolo.

Los servicios web con los que se comunicará la PDA consistirán en el registro de los dispositivos, la extracción de información sobre los proyectos disponibles, la modificación de la traza de los proyectos actuales y el envío de mensajes para poder implementar otros protocolos (Figura 3.3).



Figura 3.3.: Casos de uso que se implementan en el servicio web.

■ Diagrama de casos de uso para el programa del alumno

El alumno, poseedor de una PDA, tendrá una aplicación con la que podrá depurar el código de una aplicación que podrá ser local o remota, usando el servidor se servicios web.

En el caso de que el alumno quiera acceder al servidor remoto necesitará ser poseedor de una cuenta de alumno con una contraseña.

Mediante la aplicación, el alumno podrá razonar y discutir colaborativamente sobre el programa, dibujando diversas formas en la pantalla de la PDA que serán transmitidas a los demás usuarios en caso de que utilice un proyecto remoto. El alumno también podrá enviar mensajes a los demás alumnos que estén compartiendo el mismo proyecto.

La depuración del programa se puede describir como dos tipos de acciones: depuración de la traza, en la que el alumno puede moverse línea a línea por el código; y la aportación de los alumnos sobre el valor de las variables en el proceso de aprendizaje colaborativo.

Así pues, el alumno, mediante el dispositivo, deberá poderse registrar, elegir proyectos (tanto on-line como localmente), dibujar formas, enviar mensajes a otros alumnos, depurar la traza del programa y escribir sus comentarios en una tabla editable de variables (Figura 3.4).

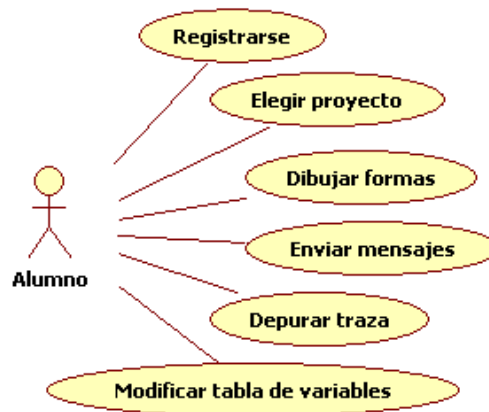


Figura 3.4.: Casos de uso del programa del alumno.

3.1.4 Diagramas de estado

Un Diagrama de Estados muestra la secuencia de estados por los que pasa bien un caso de uso, bien un objeto a lo largo de su vida, o bien todo el sistema. En él se indican qué eventos hacen que se pase de un estado a otro y cuáles son las respuestas y acciones que genera.

En cuanto a la representación, un diagrama de estados es un grafo cuyos nodos son estados y cuyos arcos dirigidos son transiciones etiquetadas con los nombres de los eventos. Un estado se representa como una caja redondeada con el nombre del estado en su interior. Una transición se representa como una flecha desde el estado origen al estado destino.

La caja de un estado puede tener 1 o 2 compartimentos. En el primer compartimento aparece el nombre del estado. El segundo compartimento es opcional, y en él pueden aparecer acciones de entrada, de salida y acciones internas. Una acción de entrada aparece en la forma *entrada/acción_asociada* donde *acción_asociada* es el nombre de la acción que se realiza al entrar en ese estado. Cada vez que se entra al estado por medio de una transición la acción de entrada se ejecuta. Una acción de salida aparece en la forma

salida/acción_asociada. Cada vez que se sale del estado por una transición de salida la acción de salida se ejecuta. Una acción interna es una acción que se ejecuta cuando se recibe un determinado evento en ese estado, pero que no causa una transición a otro estado. Se indica en la forma *nombre_de_evento/acción_asociada*.

Un diagrama de estados puede representar ciclos continuos o bien una vida finita, en la que hay un estado inicial de creación y un estado final de destrucción (finalización del caso de uso o destrucción del objeto). El estado inicial se muestra como un círculo sólido y el estado final como un círculo sólido rodeado de otro círculo. En realidad, los estados inicial y final son pseudoestados, pues un objeto no puede “estar” en esos estados, pero nos sirven para saber cuáles son las transiciones inicial y final(es).

■ Diagramas de estado para el programa del alumno

A continuación se mostrarán los diagramas de estado para el diagrama de casos de uso para el programa del alumno (sección 3.1.3).

■ Elección de un proyecto

La elección de un proyecto se representa en la figura 3.5. La elección de un proyecto deberá realizarse de una forma ordenada, pudiendo ser cancelada en cualquier momento. Un alumno deberá elegir un grupo para, a continuación, poder elegir un proyecto. Una vez elegido el proyecto podrá elegir una entrada de ese proyecto. En caso de querer elegir otro grupo o proyecto, el alumno tendrá la opción de volver hacia atrás para poder elegirlo.

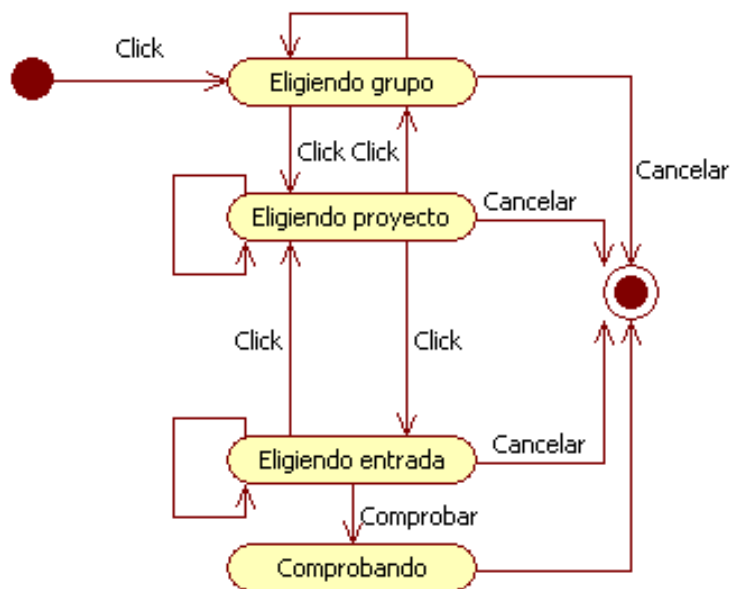


Figura 3.5.: Diagrama de estado para la elección del proyecto.

■ Dibujar formas

El proceso para dibujar una forma es simple y no dispone de muchos pasos. El usuario únicamente necesitará seleccionar la forma y arrastrarla a la pantalla para que pueda visualizarse. En cualquier momento el usuario puede cancelar esta operación y no dibujar nada. Una vez que se ha dibujado la figura, ésta se enviará a los demás usuarios para que puedan visualizarla de igual manera. Este proceso se muestra en la figura 3.6.

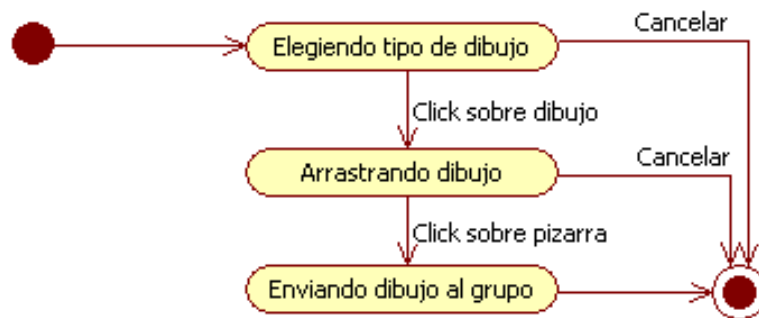


Figura 3.6.: Diagrama de estado para dibujar en la pizarra.

■ Enviar mensajes

Al igual que en el dibujo de las formas, el envío de mensajes (Figura 3.7) es simple. El usuario deberá escribir un mensaje que podrá cancelar en cualquier momento. Éste mensaje será transmitido a todos los alumnos que estén usando el mismo proyecto.

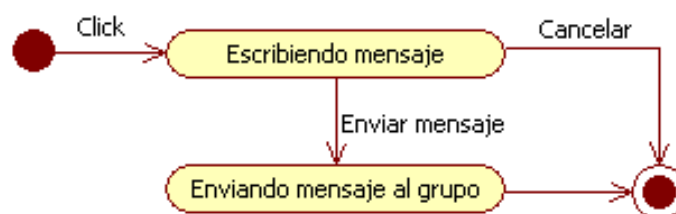


Figura 3.7.: Diagrama de estado para el envío de mensajes a grupos.

■ Depurar la traza

Aunque en el diagrama de casos de uso se ha dividido la visualización y aportación colaborativa sobre la tabla de variables y la depuración de la traza, en realidad estas dos acciones se consideran una modificación en la depuración. Así pues, vemos en la figura 3.8, como el usuario puede elegir entre modificar la traza haciendola retroceder o avanzar, mediante sus menús correspondientes, o modificar la tabla de variables eligiendo una variable y cambiando su valor. Todos estos datos que el alumno puede modificar, se enviarán a todos aquellos que estén usando el mismo proyecto, y así poder ver los cambios realizados.

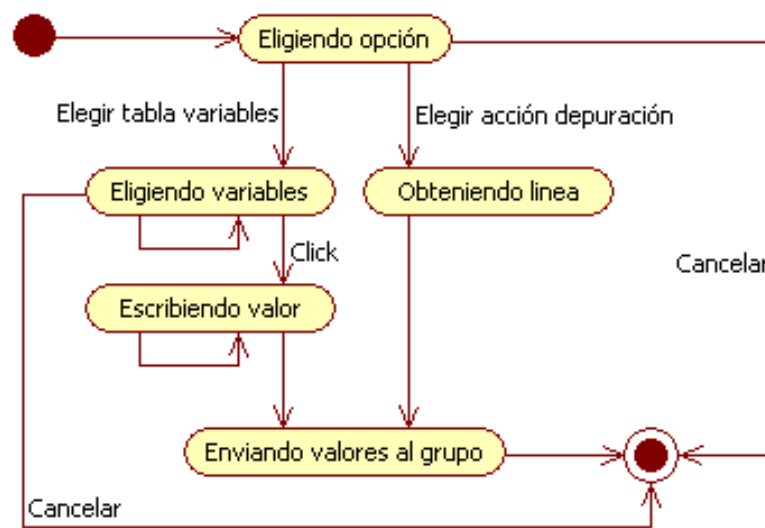


Figura 3.8.: Diagrama de estado de la depuración de un proyecto.

3.2 Análisis

En esta parte del trabajo se analizan los requisitos descritos en el apartado 3.1 los cuales se refinarán y estructurarán mediante un modelo de análisis. El modelo de análisis es una jerarquía de paquetes de análisis que contienen clases de análisis y realizaciones de casos de uso. El modelo de análisis refina los casos de uso como colaboraciones entre clasificadores:

- Clasificadores: clases de análisis, paquetes.
- Colaboraciones: realizaciones de los casos de uso.

Para entender esta definición es necesario también definir: un paquete de análisis como mecanismo para organizar los elementos del análisis; una clase de análisis como una abstracción de una o varias clases y/o subsistemas de diseño del sistema; una realización de caso de uso como una colaboración que describe cómo se lleva a cabo y se ejecuta un caso de uso determinado, en términos de las clases de análisis y de sus objetos de análisis en interacción.

Las clases de análisis encajan en uno de estos tres estereotipos básicos: clase de interfaz, clase de control o clase de entidad. Las clases de interfaz se utilizan para modelar la interacción entre el sistema y sus actores (es decir, usuarios y sistemas externos). Las clases de entidad se utilizan para modelar información que posee una larga vida y que es a menudo persistente. Por último, las clases de control representan coordinación, secuencia, transacciones y control de otros objetos y se usan con frecuencia para encapsular el control de un caso de uso en concreto (figura 3.9).



Figura 3.9.: Estereotipos

3.2.1 Diagramas de colaboración

■ Diagramas de colaboración para el programa del alumno

■ Registro

El trabajo que deberá realizar el programa en el registro consistirá en comprobar la correcta escritura en los campos del usuario y la contraseña antes de ser mandada a procesar. Una vez recogida esta información, se comprobará en la base de datos remota para confirmar la validación del usuario (figura 3.10).

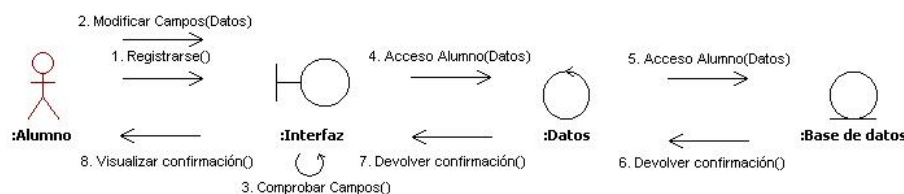


Figura 3.10.: Diagrama de colaboracion del flujo principal (Registro)

En caso de que los campos del alumno y la contraseña no hayan sido correctamente escritos, la interfaz no procesará estos datos y se quedará esperando hasta que se escriban correctamente (figura 3.11).

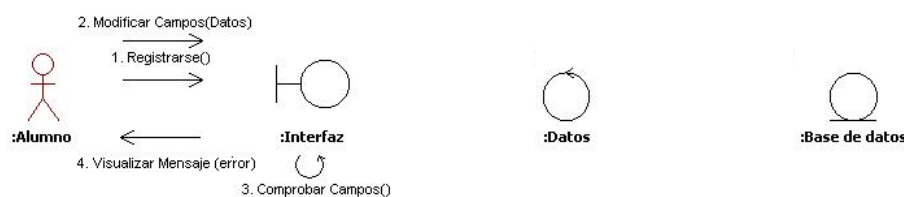


Figura 3.11.: Diagrama de colaboracion del flujo alternativo I (Registro)

En caso de que exista un error en la conexión, el proceso volverá al estado inicial pudiendo volver a escribir el usuario y la contraseña (figura 3.12).

■ Elección de un proyecto

Se creará una interfaz en la que se ofrezcan una lista de proyectos y sus entradas para la fácil elección por parte del usuario. En este caso, la comprobación de los campos no es necesaria, puesto que se tratan de listas tratadas por el control. Una vez elegido uno de los proyectos, se enviará la información a la base de datos, la cual se encargará de gestionar el proyecto asociado al usuario (figura 3.13).

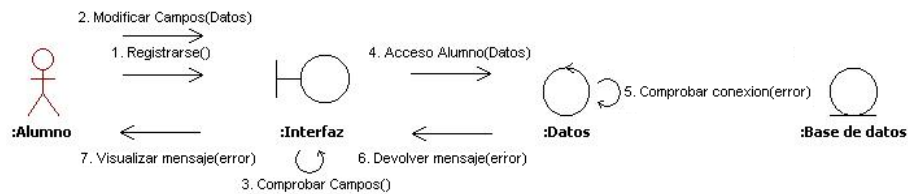


Figura 3.12.: Diagrama de colaboracion del flujo alternativo II (Registro)

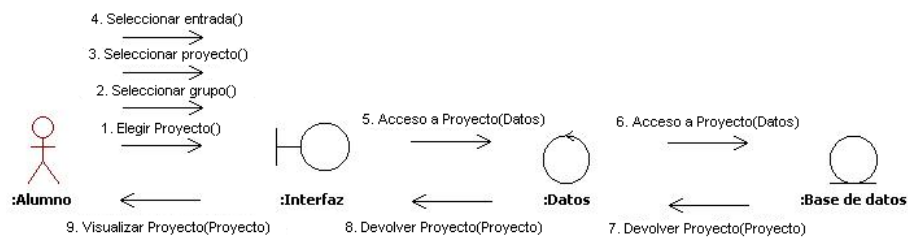


Figura 3.13.: Diagrama de colaboracion del flujo principal (Elección proyecto)

En caso de que haya una error en la conexión, el programa dejará intacta la interfaz de elección y se mostrará un aviso en pantalla. El usuario puede ser capaz de esperar a una correcta conexión o salir de esta interfaz (figura 3.14).

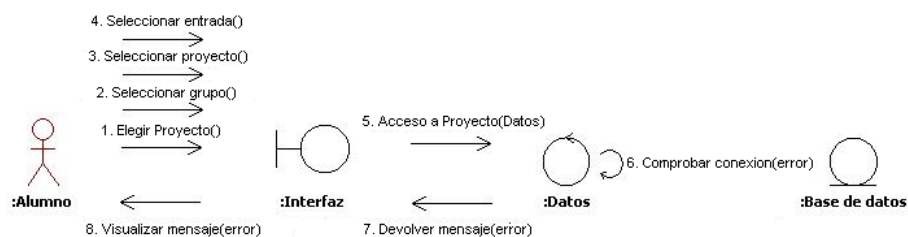


Figura 3.14.: Diagrama de colaboracion del flujo alternativo I (Elección proyecto)

■ Dibujar formas

Dibujar formas únicamente consistirá en pulsar el botón adecuado en la interfaz y utilizar el cursor en la pizarra para dibujar formas. Una vez dibujadas, se enviará información al servidor, el cual gestionará los datos y los enviará a los demás usuarios (figura 3.15).

Si el usuario arrastra el dibujo en un lugar fuera de la pizarra no se realizará ninguna acción, pues el usuario es capaz de interactuar con más partes de la interfaz (como los menús) (figura 3.16).

En caso de error en la conexión, el usuario podrá ser capaz de esperar a una correcta conexión (figura 3.17).

También tendrá la opción de trabajar de forma local (en la cual no es necesario enviar nuestras acciones al grupo) (figura 3.18).

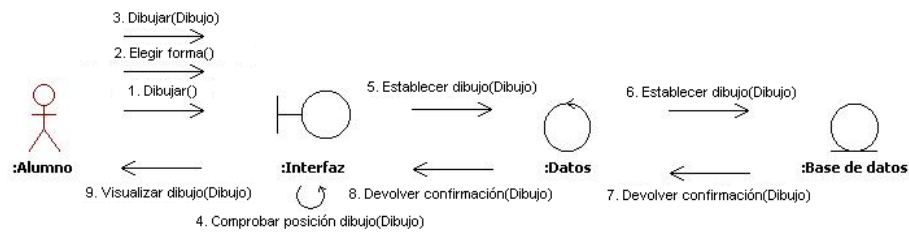


Figura 3.15.: Diagrama de colaboracion del flujo principal (Dibujar formas)

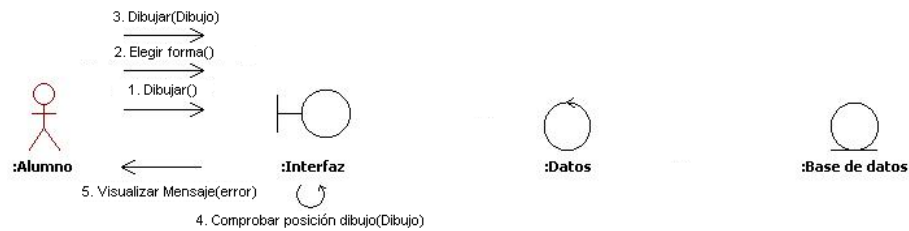


Figura 3.16.: Diagrama de colaboracion del flujo alternativo I (Dibujar formas)

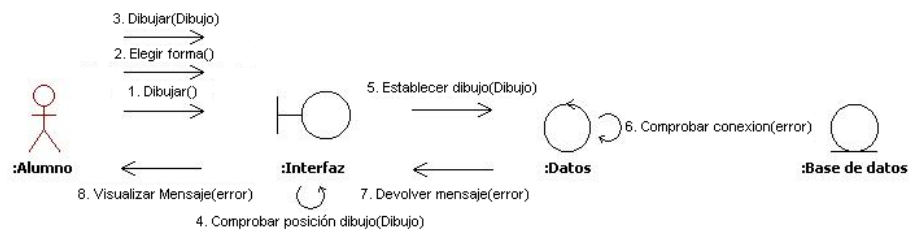


Figura 3.17.: Diagrama de colaboracion del flujo alternativo II (Dibujar formas)

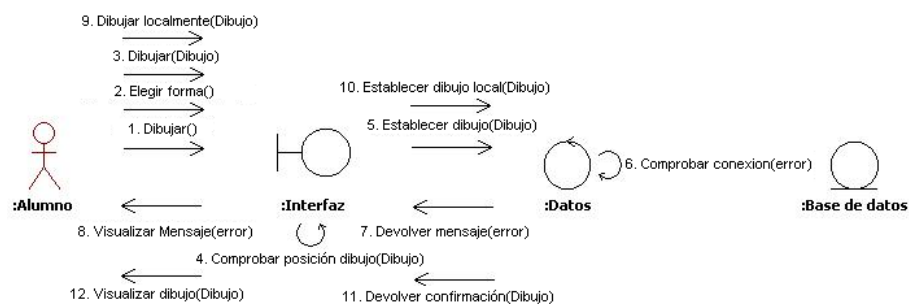


Figura 3.18.: Diagrama de colaboracion del flujo alternativo III (Dibujar formas)

■ Enviar mensajes

Enviar un mensaje consistirá en pulsar en el botón adecuado de la interfaz, escribir un mensaje y pulsar el botón de envío (figura 3.19).

En caso de que el mensaje esté vacío o sea erróneo, no se habilitará el botón de enviar y el usuario no podrá enviar el mensaje erróneo al grupo (figura 3.20).

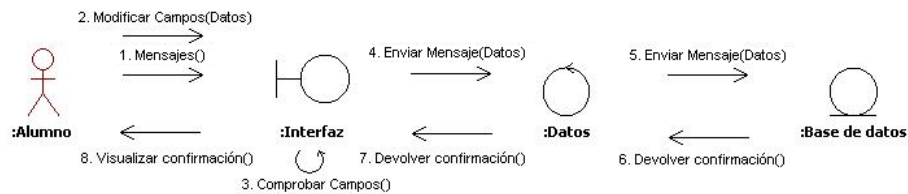


Figura 3.19.: Diagrama de colaboracion del flujo principal (Enviar mensajes)

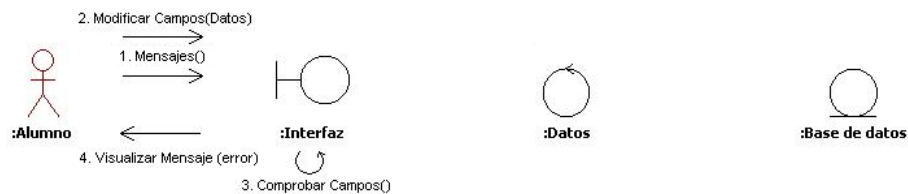


Figura 3.20.: Diagrama de colaboracion del flujo alternativo I (Enviar mensajes)

En caso de detectar un error, no se podrá trabajar localmente como en el caso anterior, pues el envío de mensajes es una operación totalmente colaborativa (figura 3.21).

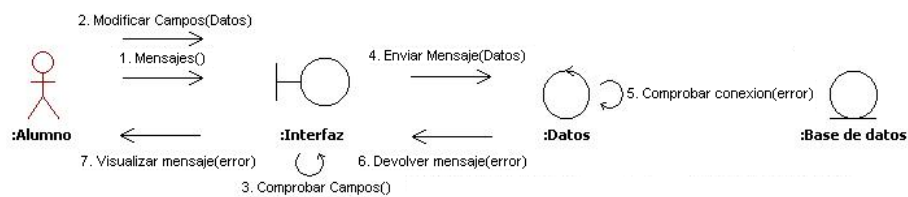


Figura 3.21.: Diagrama de colaboracion del flujo alternativo II (Enviar mensajes)

■ Depurar la traza

Depurar una traza consiste en pulsar los botones de avance y retroceso y cambiar los valores de las variables. Ya que los valores de las variables pueden tener cualquier valor, no se comprobará el valor de los campos en estos formularios (figura 3.22).

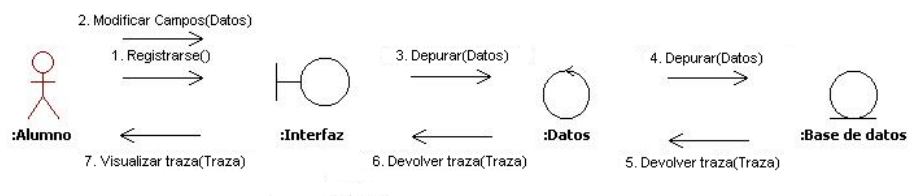


Figura 3.22.: Diagrama de colaboracion del flujo principal (Depurar la traza)

Al igual que en el caso de la figura 3.17 y 3.18, en el caso de pérdida de la conexión se podrá optar por trabajar localmente o esperar a que se reestablezca la conexión.

3.3 Diseño

A partir del análisis y la captura de requisitos, se llegó a la conclusión de que serán necesarias tres herramientas bien diferenciadas:

- Programa de administración para el profesor
- Programa para la depuración del alumno
- Servidor

3.3.1 La arquitectura de las aplicaciones desarrolladas en tres capas

Una parte importante del diseño de la aplicación es la elección de la arquitectura. En este proyecto se ha decidido realizar una arquitectura cliente/servidor.

La arquitectura de una aplicación es la vista conceptual de la estructura de ésta. Toda aplicación contiene código de presentación, código de procesamiento de datos y código de almacenamiento de datos. La arquitectura de las aplicaciones difieren según está distribuido este código. La arquitectura de tres capas propone la siguiente organización.

■ Capa de presentación

Esta capa soporta los servicios de presentación, los cuales proporcionan la interfaz necesaria para presentar información y reunir datos. También asegura los servicios de negocio necesarios para ofrecer las capacidades de transacciones requeridas e integrar al usuario con la aplicación para ejecutar un proceso de negocio. Los servicios de presentación generalmente son identificados con la interfaz de usuario, y normalmente residen en un programa ejecutable localizado en la estación de trabajo del usuario final. Aún así, existen oportunidades para identificar servicios que residen en componentes separados. El cliente proporciona el contexto de presentación, que generalmente puede ser un navegador web como Mozilla Firefox o Netscape Navigator, los cuales permiten ver los datos remotos a través de una capa de presentación HTML; o también una aplicación WIN32 o multiplataforma.

Mediante el uso de componentes, se separa la programación que da acceso a los datos en las bases de datos y de las aplicaciones del diseño de la interfaz. Esto ayuda a asegurar que los desarrolladores estén libres a la hora de escribir la lógica de negocios en componentes sin preocuparse acerca de cómo se muestra la salida. Recíprocamente, ésto da libertad a los diseñadores para usar herramientas familiares para modificar la interfaz. La capa de servicios de presentación es responsable de:

- Obtener información del usuario.
- Enviar la información del usuario a los servicios de negocios para su procesamiento.
- Recibir los resultados del procesamiento de los servicios de negocio.
- Presentar estos resultados al usuario.

■ Capa de negocio

Una tarea de negocios es una operación definida por los requerimientos de la aplicación, como introducir una orden de comprar o imprimir una lista de clientes. Las reglas de negocio (*business rules*) son políticas que controlan el flujo de las tareas. Los servicios de negocio son el «puente» entre un usuario y los servicios de datos.

Como las reglas de negocio tienden a cambiar más frecuentemente que las tareas específicas de negocio a las que dan soporte, son candidatos ideales para encapsularlas en componentes y separarlos de la lógica de la aplicación en sí.

El nivel de servicios de negocio es responsable de:

- Recibir la entrada del nivel de presentación.
- Interactuar con los servicios de datos para ejecutar las operaciones de negocio para los que la aplicación fue diseñada.
- Enviar el resultado procesado al nivel de presentación.

■ Capa de datos

La capa de datos se encuentra enlazada con la capa de negocio. El nivel de servicios de datos es responsable de:

- Almacenar los datos.
- Recuperar los datos.
- Mantener los datos.
- La integridad de los datos.

3.3.2 Estructura en paquetes

Para saber cómo han sido realizadas las distintas herramientas es importante conocer la estructura software que éstas tienen, de esta manera facilita su entendimiento. Debido a este motivo se va a comentar a continuación dicha estructura. El software se encuentra estructurado en paquetes que engloban a las clases de .NET Framework y .NET Compact Framework que tienen funcionalidad parecida, esto quiere decir, aquellas clases cuya función es realizar una misma tarea pero con objetivos distintos. De esta manera podemos encontrar cuatro grandes grupos que son los que exponemos a continuación:

- Paquete Manipulador: engloba a aquellas clases que se encargan de manipular los datos obtenidos en las consultas a la base de datos y procesarlos para su posterior visiaonado. El uso de este paquete puede ser optativo si la información obtenida de la base de datos no tiene que ser manipulada.
- Paquete Interfaz: en este paquete las clases almacenadas son aquellas que se encargan de generar la interfaz gráfica y solicitar los datos generados por las clases de otro paquete.

- Paquete Datos: en dicho paquete se almacena las clases que van a definir las estructuras de los datos que serán necesarias para operar dentro de la aplicación.
- Paquete Base de Datos: este paquete almacena las clases encargadas de hacer consultas con la base de datos.

La figura 3.23 representa esta jerarquía de paquetes.

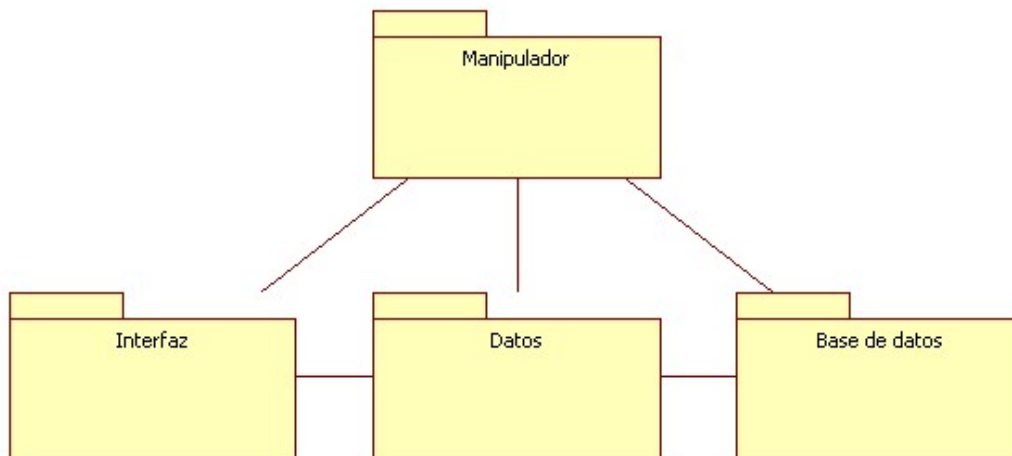


Figura 3.23.: Diagrama de paquetes

3.3.3 Diseño detallado

En el diseño se modela el sistema de manera que incluya la arquitectura y que soporte todos los requisitos (incluyendo los requisitos no funcionales y otras restricciones) que se han expuesto anteriormente. El principal resultado del diseño es el modelo de diseño, que se esfuerza en conservar la estructura impuesta en el modelo de análisis, y que sirve como esquema para la implementación. El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar. Se han llevado a cabo distintas fases correspondientes a las diferentes iteraciones realizadas. Y esto se describe en los apartados que siguen. Se hará referencia a clases identificadas en el modelo de análisis y aparecerán nuevas clases de diseño.

Al igual que en el apartado de análisis, se restringirán el número de diagramas dado al número de páginas de la memoria. Así pues, se analizará la herramienta del alumno y sus requisitos funcionales vistos anteriormente (véase 3.1.4).

■ Registro

Así pues, usando como modelo el programa del alumno y siguiendo la metodología del proceso unificado, es necesario la creación de los diagramas de clases de diseño en esta sección. En la figura 3.24 vemos representada como una de las interfaces del dispositivo

PDA (VisualDebugger) hace una petición de comprobación de un usuario para poder validar éste y usarlo durante las peticiones posteriores.

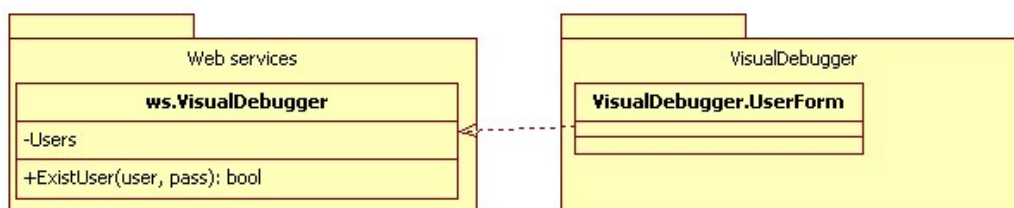


Figura 3.24.: Diagrama de clases para registrar al alumno.

■ Elección de un proyecto

Una vez hecha la petición de comprobación, se dará paso a la interfaz de selección de proyectos. Como vemos en la figura 3.25, el dispositivo deberá obtener una lista de grupos, proyectos y trazas relacionadas con el alumno que está usando el dispositivo. Así pues, la interfaz deberá tener como atributos el usuario y la contraseña y una vez realizada la transacción, deberá proporcionar los datos del proyecto a la interfaz principal.

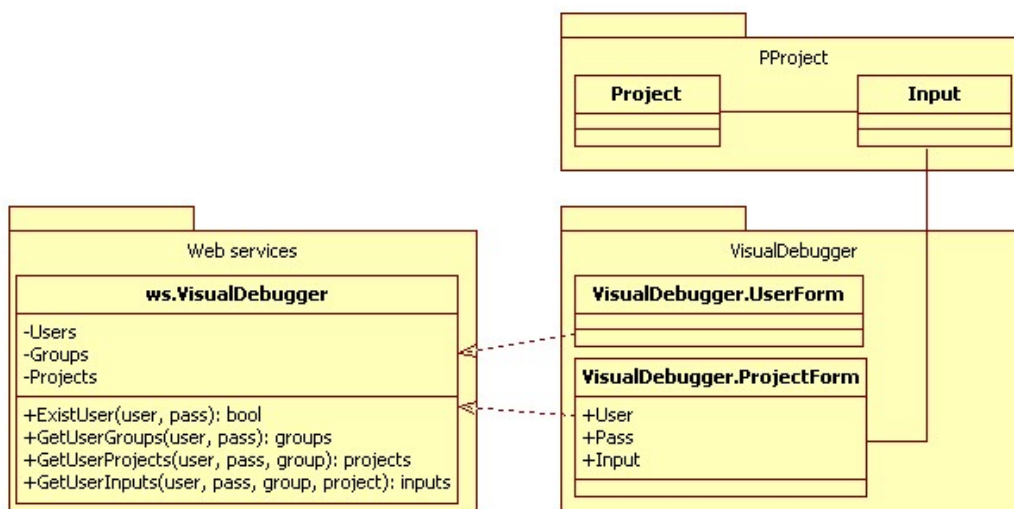


Figura 3.25.: Diagrama de clases para obtener una entrada de un proyecto.

■ Dibujar formas

En la figura 3.26 vemos representadas las clases necesarias para dibujar formas. Para ello necesitaremos de un objeto abstracto *Shape* que usaremos para crear nuevos tipos de formas. Estas formas contendrán los métodos necesarios para que dentro del objeto *DrawPanel*, que representa una pizarra, sea capaz de dibujarse. Otra peculiaridad de la pizarra es que es distribuida, así pues, se deberá informar al servidor de los cambios

producidos en ésta para que se contemplen en los demás dispositivos. Ya que el servidor no tiene la necesidad de saber que objeto es el que forman los dibujos, se puede serializar a XML para tratarlo como una cadena de texto, así nos evitamos implementar la clase *Shape* en el servidor y en el cliente.

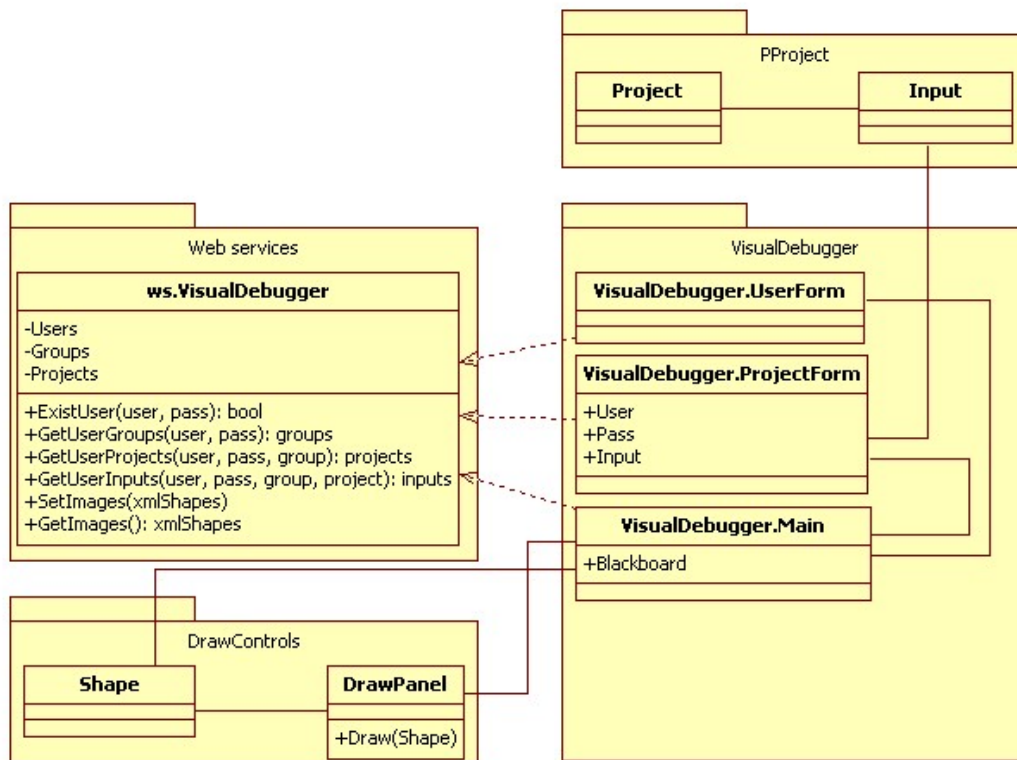


Figura 3.26.: Diagrama de clases para dibujar formas.

■ Enviar mensajes

El envío de mensajes es un procedimiento muy sencillo, como podemos ver en la figura 3.27. Únicamente necesitaremos obtener del servidor un método que tenga un argumento para el mensaje. También se deberá especificar hacia quién va dirigido. Por ello, el método deberá controlar el grupo, el proyecto y la entrada hacia quién va dirigido. De esta forma, todos los usuarios que estén participando en ese proyecto, recibirán dicho mensaje.

Además de usar éste método para mandar mensajes de chat, también se usará para crear nuevos protocolos de mensajes. Estos protocolos se basan en cadenas de texto que el cliente puede interpretar si conoce el protocolo. Por ejemplo, para mandar un mensaje a todo el grupo utilizaremos la expresión *PROTOCOL_CHAT*_*[Usuario]*_*[Mensaje]*. El cliente al recibir este mensaje, conocerá su significado y se procesará el mensaje de forma adecuada. Así pues, otros de los protocolos usados por el cliente son:

- *PROTOCOL_CHAT*_*[Usuario]*_*[Mensaje]*: Se utiliza para enviar un mensaje de chat.
- *PROTOCOL_EDITING*_*[Usuario]*: Indica que un usuario quiere tomar el control del programa e informa a todo el grupo.

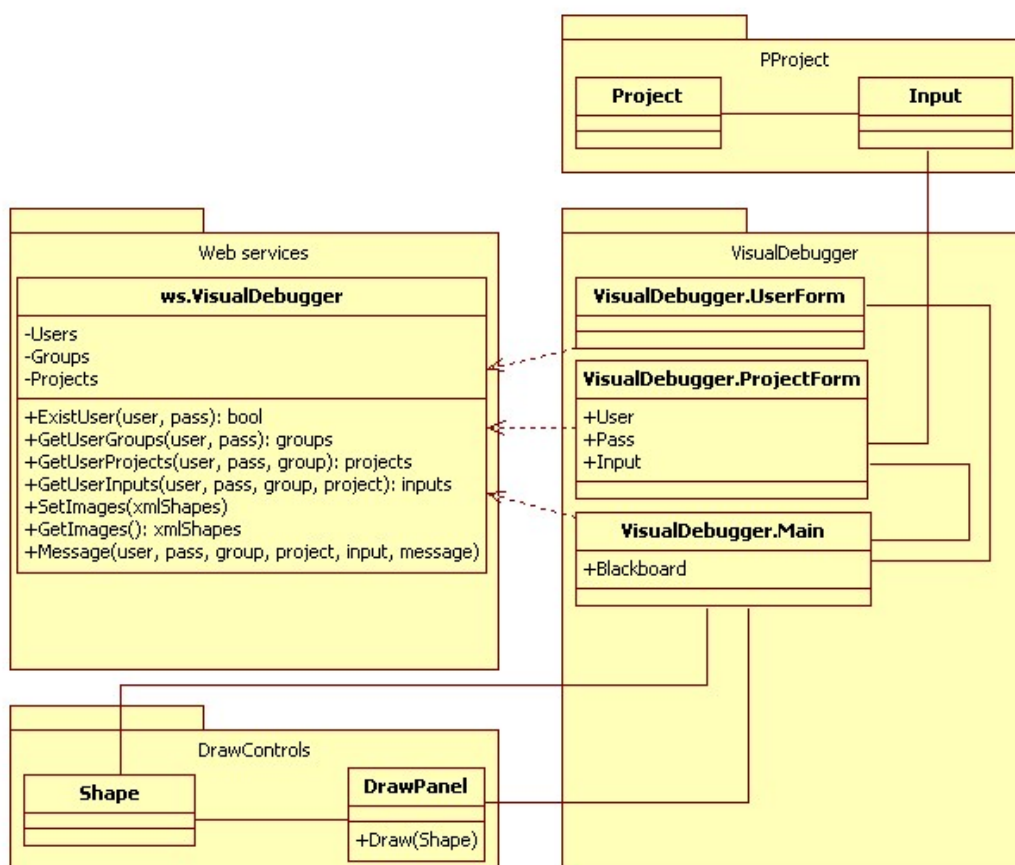


Figura 3.27.: Diagrama de clases para enviar mensajes.

■ Depurar la traza

La figura 3.28 representa los cambios necesarios para que el programa pueda depurar la aplicación. Con este, también podemos obtener una vista en su totalidad de las funcionalidades. El proceso de edición es el que utiliza más métodos, aunque muchos de ellos son muy parecidos. Así pues, deberemos tener métodos que nos indiquen la línea en la que actualmente se encuentra el estado del programa en depuración *ActualId*. Proyectos para avanzar o retroceder en la traza (*Back* y *Next*), con la capacidad de entrar internamente en los métodos o no. Y métodos que recojan el proyecto serializado en XML para poder manejar ciertas propiedades localmente. De esta forma, si nos desconectamos del servidor o queremos trabajar localmente con él, ya dispondremos de toda la información necesaria.

Dado que el cambio de variables no pertenece al estado original de un proyecto y se cambian en el estado interno del programa, éstas usaran el protocolo de mensajes para obtener los cambios de los demás alumnos.

3.3.4 Diseño de la base de datos

Durante el proceso se usó una base de datos MySQL. Nuestra base de datos posee información acerca de los usuarios, grupos y proyectos. La figura 3.29 representa el esquema de la base de datos. En el apéndice podremos encontrar las sentencias para crearla.

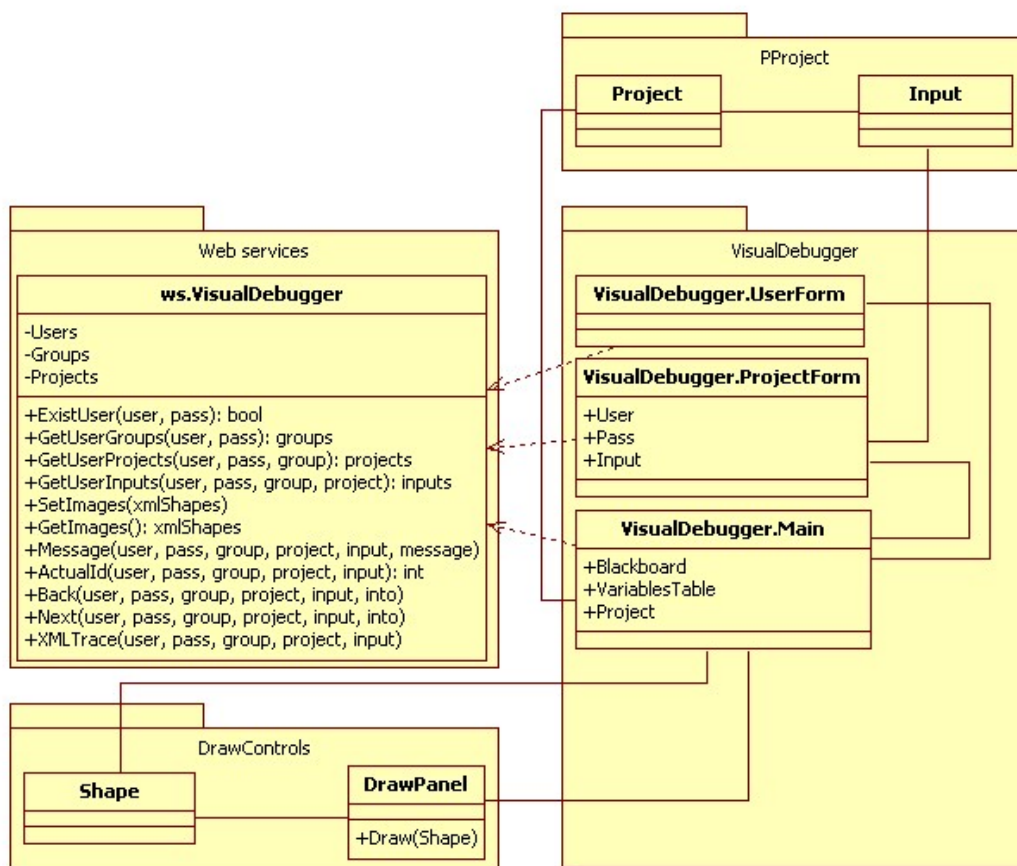


Figura 3.28.: Diagrama de clases para depurar la traza y final.

3.3.5 Arquitectura distribuida

Como hemos visto anteriormente, nuestro depurador colaborativo requiere de dos programas independientes: 1) un programa para el profesor, el cual administrará los proyectos y a los alumnos; 2) un programa para el alumno integrado en la PDA con el cual podrá depurar colaborativamente con otros alumnos. Así pues, se necesitará una arquitectura distribuida en la que los programas administrados por el profesor puedan ser accesibles desde los alumnos con cada uno de sus dispositivos y los alumnos puedan comunicarse entre ellos siendo capaces de observar a tiempo real los cambios que se produzcan en los proyectos.

Así pues, se vio en la necesidad de crear un servidor capaz de realizar dos tareas:

1. Administrar los proyectos y alumnos creados por el profesor.
2. Capaz de suministrar los proyectos a los alumnos y administrar los dispositivos para que se puedan comunicar entre ellos.

■ Trabajos del servidor

- Para el programa del profesor

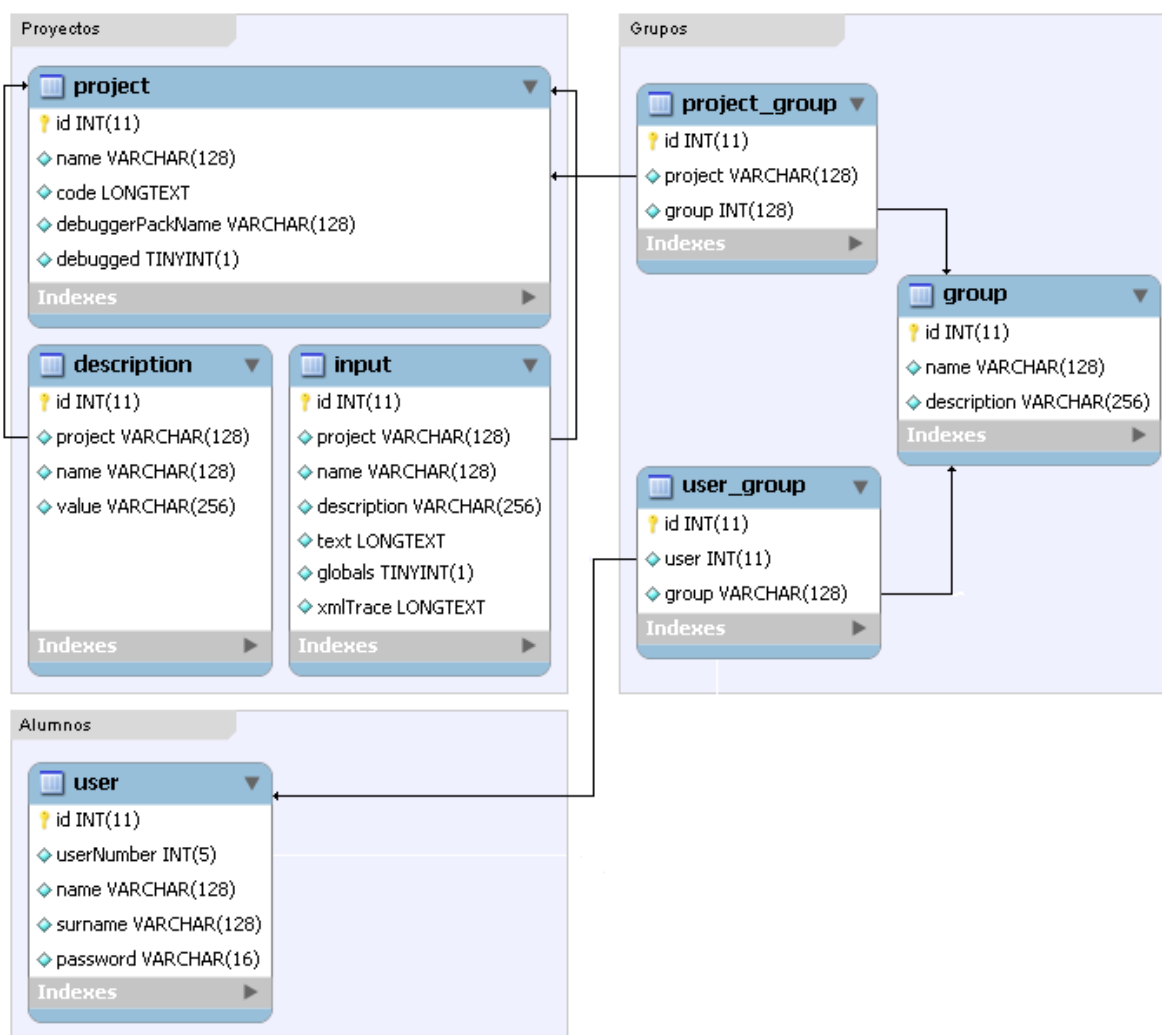


Figura 3.29.: Esquema de la base de datos.

Toda la información administrada por el profesor se escribe finalmente en unas bases de datos, la cual contiene información sobre grupos, alumnos y proyectos. El servidor debe ser capaz de recoger esta información y serializarla en objetos para el fácil uso con .NET Remoting (véase 2.2) y así poder crear un sistema distribuido transparente (figura 3.30).

La base de esta información son los grupos. Los grupos tienen un nombre y una descripción y pueden contener tanto alumnos como proyectos. Todo aquello contenido dentro de un grupo es visible dentro de éste. Así pues, un alumno asignado a un grupo es capaz de ver y usar todos los proyectos que están dentro de él. Podemos crear grupos que representen clases de alumnos (ej: «Metodología y Tecnología de la Programación, Gestión», «Estructura de Datos y de la Información, Sistemas»). Tanto los proyectos como los alumnos podrán pertenecer a diferentes grupos simultáneamente.

El trabajo del servidor en éste ámbito deberá ser la posibilidad de ofrecer de forma segura los proyectos asignados a los alumnos. Un alumno no podrá acceder a los

proyectos que no están en sus grupos ni asignarse maliciosamente a otros grupos.

El servidor también se encargará de guardar el estado de los proyectos mientras están depurándose. Guardará la línea en la que se encuentra depurándose el proyecto y a partir de ésta los dispositivos podrán ser capaces de recoger el valor de las variables.

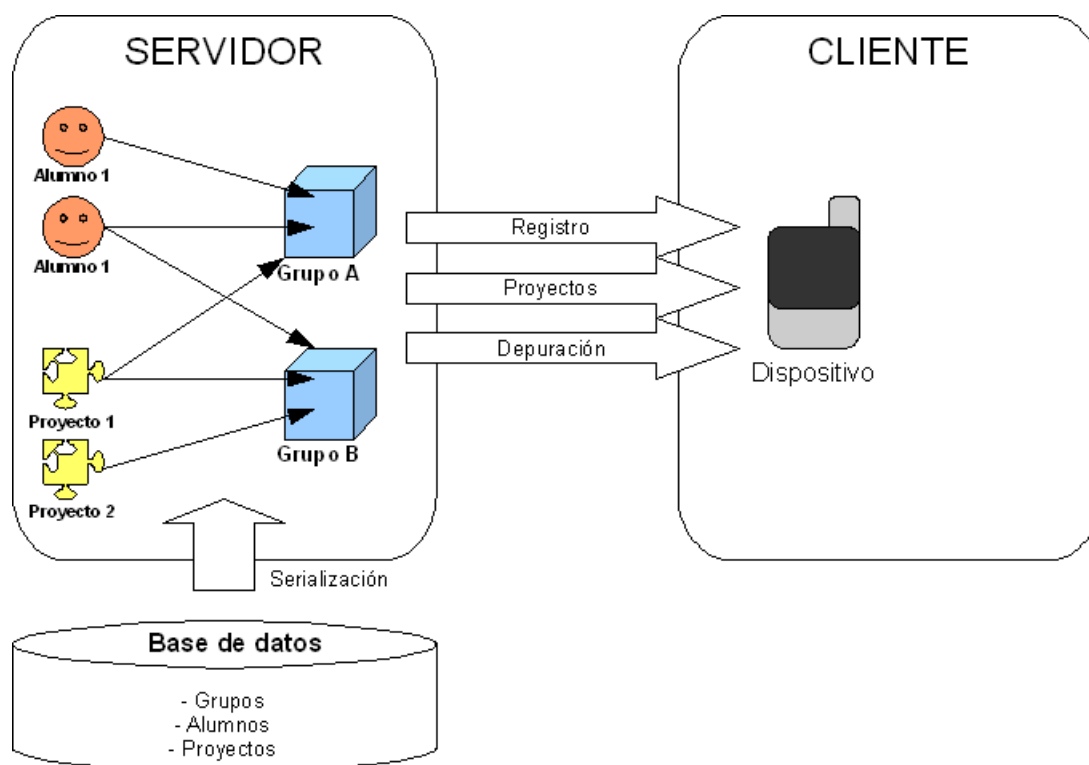


Figura 3.30.: Servicios desde el punto de vista del profesor.

- Para el programa del alumno

La punto principal de este trabajo será la colaboración entre alumnos. Como vimos anteriormente en 3.3.5, el servidor ofrece los proyectos y guarda el estado de los mismos. Ya que a un mismo proyectos se podrán unir diferentes alumnos, se deberá guardar la información de todos aquellos que están retocando éste y avisarlos de diferentes cambios que se puedan producir. Así pues nos basaremos en eventos que producirá el servidor (figura 3.31).

Ya que el uso de servicios web es unidireccional (pregunta y respuesta), no permite eventos. Así pues, el servidor también ofrecerá un servicio de cliente en el que tendrá registrada cada IP de cada dispositivo que se conecte a él y así poder informar a los dispositivos de nuevos sucesos. Los dispositivos, a parte de cliente, necesitarán de un servidor que puedan escuchar todos aquellos eventos provenientes del servidor. Así pues, tanto el servidor como el cliente tienen a su vez un cliente y un servidor que es usado únicamente para los eventos. Un cliente podrá enviar un evento a otro cliente enviando un mensaje primero al servidor. Éste lo procesará y buscará la dirección IP del/de los destinatarios como si de una agenda se tratase. Una vez encontrados, usará el servidor del dispositivo de destino para enviarle el mensaje.

El servidor, así pues, hace de puente entre varios servicios: chat, petición de uso del depurador y futuras implementaciones posibles.

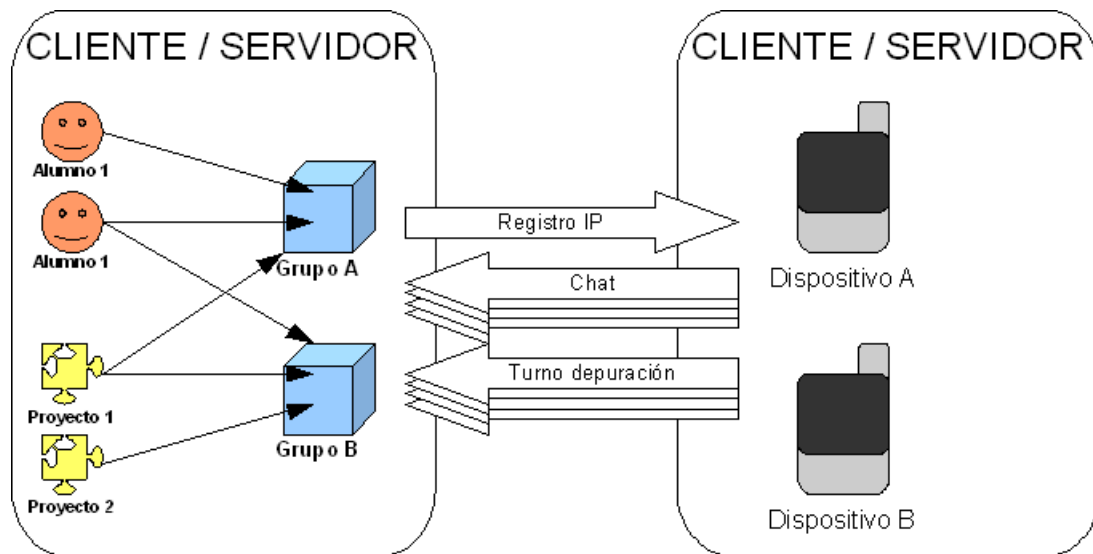


Figura 3.31.: Servicios desde el punto de vista del alumno.

■ Servicios web desde el servidor

Se ha usado .NET Remoting (véase 2.2) para crear un servidor que ofrezca servicios web. De esta forma, el programa que controla la base de datos y crea los objetos son fácilmente accesibles añadiendo las librerías de Remoting y ofreciendo ciertos servicios transparentemente. Los servicios web en formato WSDL pueden ser accesibles desde la página web:

`http:\\\\nombreDelDominio\\VisualDebugger.soap?wsdl`

Una descripción de los servicios que ofrece pueden verse a continuación:

- ExistUser(user, pass): bool
- GetUserGroups(user, pass): groups
- GetUserProjects(user, pass, group): projects
- GetUserInputs(uiser, pass, group, project): inputs
- SetImages(xmlShapes)
- GetImages(): xmlShapes
- Message(user, pass, group, project, input, message)
- ActualId(user, pass, group, project, input): int
- Back(user, pass, group, project, input, into)
- Next(user, pass, group, project, input, into)
- XMLTrace(user, pass, group, project, input)

■ Administración de IPs

Cualquier dispositivo que haga uso de alguno de estos servicios web con una autenticación correcta se registrará su IP automáticamente [1] y se adjudicará al proyecto al que se haya querido acceder. De esta forma entra en juego el cliente en el lado del servidor. El cliente posee una lista de IPs asociada a cada proyecto que esté actualmente en uso. Una vez que se realiza una llamada a cualquier función de un proyecto para su depuración, se usará la lista de IPs para mandar mensajes a todos aquellos dispositivos que estén escuchando. En caso de no poderse conectar a un dispositivo, se borrará de la lista y este dispositivo deberá hacer uso de la depuración para volver a conectarse. De esta forma eliminamos fácilmente dispositivos que ya no están en uso o se han querido desconectar.

■ Servidor en el dispositivo

En el lado de los dispositivos móviles, existe un servidor que escucha todas estas llamadas. Se han usado sockets TCP de .NET Compact Framework (véase 2.3) que poseen algunos problemas como se pudo ver en la descripción de la introducción. Aun así, .NET Compact no proporciona facilidades en cuanto a sockets y por ello se eligió TCP para asegurar el envío e los mensajes. Uno de los problemas que se han encontrado es la imposibilidad de parar el servidor mediante alguna llamada a algún método. Así pues, el dispositivo, una vez que quiere parar el servidor se mandará un mensaje a sí mismo y desde dentro del servidor se pedirá que deje de seguir escuchando. Una vez que uno de los mensajes se reciben desde el servidor, éste se serializará a su formato correcto para que el programa de depuración pueda interpretarlo correctamente y elija el protocolo conveniente: protocolo para chat, protocolo para pedir turno y otros que podrían ser implementados posteriormente.

3.4 Implementación y pruebas

Esta sección se describirá brevemente ya que la descripción de la implementación conlleva suficiente detalle y nuestra memoria tiene restringidas el número de hojas. Cabe decir que se utilizaron las tecnologías .NET Framework para el desarrollo de la aplicación del profesor y .NET Compact Framework para el desarrollo de la aplicación del alumno. Se utilizó el software libre GDB para la implementación de la depuración de nuestro proyecto aplicando una capa superior y accediendo a sus servicios (podemos ver un ejemplo de su uso en 2.6). Se utilizaron las API de .NET Remoting para la construcción del servidor utilizando clases remotas compartidas por un canal HTTP que nos permite compartir estas clases utilizando SOAP.

Las pruebas se realizaron con dos PDA y el ordenador personal del alumno. El tipo de pruebas que se realizaron se describe a continuación:

- Pruebas sobre la base de datos: se crearon clases de datos que envuelven las consultas, serializándolas en los objetos necesarios (cómo proyectos, alumnos y grupos). Se tuvieron en cuenta diversos problemas como la búsqueda de proyectos inexistentes, bases de datos vacías, campos con valores nulos y otros percances. Al tener esta capa de datos sobre la base de datos se pudo depurar cómodamente como un módulo.

- Problemas de interfaz. Se revisaron todos los objetos de las interfaces, comprobando sus eventos y creando un estado de interfaz para tener controladas las demás partes de la interfaz. Por ejemplo, en el caso de que un proyecto sea local, se inhabilita todos los comandos de interfaz referidos a comunicación remota.
- Problemas de comunicación. Se probaron, con satisfactorios resultados, todas las llamadas a los servicios web. En caso de no poder conectar con el servidor se da la posibilidad de trabajar en local o seguir esperando. La URL de la conexión se puede cambiar mediante el menu de opciones de los programas.
- Problemas de eventos. Los eventos nos ayudan a detectar clientes no conectados. Cada vez que se intenta mandar un evento a un cliente que no responde, el servidor supondrá que ese usuario no está conectado y así podremos ir actualizando la lista de clientes y sus IPs.

4. CONCLUSIONES

4.1 Conclusiones

Una vez realizadas las aplicaciones, hemos podido comprobar el cumplimiento de los requisitos iniciales, e incluso hemos adaptado el programa para mejorarlo a base de plugins (véase 4.2). Los profesores disponen de una aplicación en la que puedan crear, gestinar, borrar y modificar proyectos de código, grupos y usuarios de una manera fácil usando interfaces tan simples como el arrastre con el puntero de los objetos.

El alumno dispondrá de una aplicación en la que podrá depurar colaborativamente los proyectos con otros alumnos. Podrá enviar aportaciones, comentarios, etc. a otros miembros y dibujar en la pantalla con el lápiz de la PDA. Además dispondrá de una tabla de variables para ver sus valores en tiempo real.

Durante la realización del proyecto se han observado algunos problemas, generalmente relacionados con la escasez de API por parte de .NET Compact Framework en relación con .NET Framework. A continuación se enumeran algunos problemas:

- El uso de Threads en .NET Compact Framework imposibilita el paso de mensajes entre éstos. De esta manera se deberán cerrar estos procesos mediante eventos.
- Los pocos protocolos existentes para la creación de servidores en .NET Compact Framework. En un principio se decidió crear un servidor HTTP, pero dada la no existencia de una librería posteriormente creada por .NET, se obtuvo por usar un servidor usando el protocolo TCP mediante mensajes en XML.
- .NET Compact Framework no nos ofrece ningún componente para dibujar en la pantalla con el lápiz de la PDA. Se tuvo que crear un componente totalmente nuevo, parecido a un ListBox, pero que permitiera el dibujo de figuras.
- .NET Compact Framework no soporta Remoting, así pues, se creó un servidor en Remoting que ofreciera los servicios como servicios web para que la aplicación de la PDA pudiera “verlos”.
- .NET Compact Framework no soporta SOAP, así pues, todos los mensajes fueron serializados en XML a través de una API ofrecida por .NET Compact Framework.

4.2 Trabajo futuro

El punto clave de nuestro trabajo ha sido el desarrollo para la ampliación futura de la funcionalidad del programa. Así por ejemplo, se han usado servicios web para la comunicación entre clientes y el servidor. De esta manera podremos implementar distintos clientes en distintos lenguajes de programación.

4. CONCLUSIONES

Durante el desarrollo hemos creado un depurador para PASCAL, pero el programa está adaptado para la creación de nuevos plugins para nuevos lenguajes. Así pues, una de las ideas para la ampliación del programa sería nuevos envoltorios (plugins) para diferentes compiladores y generadores de trazas para lenguajes como C++, C#, Java, Haskell, Visual Basic, etc.

BIBLIOGRAFÍA

- [1] Remoting Server, Remoting Client IP [online]. URL: <http://www.z6688.com/info/35198-1.htm>.
- [2] GDB, The GNU Project Debugger [online]. URL: <http://www.gnu.org/software/gdb/gdb.html>.
- [3] Jesse Liberty, editor. *Programming C#, Fourth Edition*. O'Reilly Media, 2005.
- [4] .NET Framework y .NET Compact Framework [online]. URL: <http://msdn.microsoft.com>.
- [5] Daniel, editor. Programación Orientada a Objetos en C# .NET 2.0 [online]. URL: http://www.wikilearning.com/curso_gratis/programacion_orientada_a_objetos_en_c_net_2_0/11734.
- [6] Kim Williams Scoot McLean, James Naftel, editor. *Microsoft .NET Remoting*. Microsoft Press, 2002.
- [7] Xavier Ferré Grau, editor. Desarrollo Orientado a Objetos con UML [online]. URL: http://www.wikilearning.com/tutorial/desarrollo_orientado_a_objetos_con_uml/6321.
- [8] Wikipedia [online]. URL: <http://es.wikipedia.org>.
- [9] More .NET Compact Framework Woes [online]. URL: <http://www.zorched.net/2007/02/04/more-net-compact-framework-woes/>.

A. INSTALACIÓN

La instalación del programa del profesor requiere de un archivo XML básico en el que se especificará los datos de la conexión de la base de datos y otros campos. Este archivo tiene la siguiente estructura:

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <AppConfiguration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3   <DB_Hostname>localhost</DB_Hostname>
4   <DB_Database>visualdebugger</DB_Database>
5   <DB_Username>root</DB_Username>
6   <DB_Password>12345</DB_Password>
7   <Debuggers_URL />
8 </AppConfiguration>

```

Listado A.1: Ejemplo de los datos escritos en el archivo de configuración.

La primera vez que utilicemos el programa únicamente necesitaremos rellenar los campos *DB_Hostname*, *DB_Database*, *DB_Username* y *DB_Password*. Una vez inicializado el programa por primera vez, es posible que añada más información a este fichero. En cualquier caso, siempre se puede volver a retocar la información relacionada con la base de datos:

- *DB_Hostname*: nombre de la máquina dónde está alojada la base de datos.
- *DB_Database*: nombre de la base de datos que contiene las tablas necesarias para el programa (véase [B](#)).
- *DB_Username*: usuario con acceso a las tablas de la base de datos tanto en **escritura** como en **lectura**.
- *DB_Password*: contraseña del usuario.

NOTA: Una vez obtenido el programa deberá comprobarse que se haya el archivo *appConfiguration.xml* en la misma carpeta que el ejecutable principal, y que este archivo contiene como mínimo la información dada en el código anterior.

Otro archivo imprescindible en la instalación, es el archivo *Server.config*. Con este archivo, podremos configurar algunos parámetros del servidor, como son el puerto de escucha y la ruta URL. Éste deberá estar en la misma ruta que el ejecutable y deberá contener la estructura del código [A](#).

```

1 <!-- server -->
2 <configuration>
3   <system.runtime.remoting>
4     <application name="calcsrv">

```

```

5
6      <service>
7          <wellknown mode="Singleton" type="ServerClasses.VisualDebuggerServer,
8              Server" objectUri="VisualDebugger.soap" />
9      </service>
10
11      <channels>
12          <channel ref="http" port="1234">
13              <!-- support HttpChannel on port 999 -->
14              <serverProviders>
15                  <formatter ref="soap" />
16                  <!-- support SOAP encoding -->
17                  <provider ref="wsdl"/>
18                  <!-- support wsdl extraction -->
19                  <provider type="IPSink.ClientIPServerSinkProvider, Server"/>
20              </serverProviders>
21          </channel>
22      </channels>
23
24  </application>
25  </system.runtime.remoting>
26 </configuration>

```

Listado A.2: Ejemplo del archivo de configuración del servidor.

En este archivo solo podremos modificar dos campos:

- `port = "1234"`

Sólo podremos cambiar la propiedad «port» asignando el puerto de escucha del servidor.

- `objectUri = "VisualDebugger.soap"`

Cambiando la propiedad `objectUri` cambiamos el nombre de la dirección URL para acceder al servicio web.

Al cambiar estas dos propiedades, la dirección URL final quedaría de la siguiente manera.

`http://localhost:[port]/[objectUri]?wsdl`

En el ejemplo del código de configuración del servidor, la URL quedaría:

`http://localhost:1234/VisualDebugger.soap?wsdl`

A.1 Instalación desde el CD

Antes de instalar el programa del profesor en un PC, deberemos instalar los siguientes programas: FreePascal 2.2.2 y AppServ 2.5.10 que podemos encontrar en la carpeta `/instalacion/programas/`. La carpeta de destino en las instalaciones deberá ser la carpeta

por defecto. Una vez instalados los dos programas, se deberá abrir un navegador abrir la página `http : //localhost/phpmyadmin/`. Nos aparecerá un administrador de la base de datos en PHP. Escribiremos un nombre en el formulario llamado «Crear nueva base de datos» que corresponda con el nombre que deberemos poner en el campo *DB.Database* anteriormente citado. Una vez creada la base de datos, pulsaremos sobre la pestaña superior «SQL» y escribiremos las consultas de la base de datos que podemos encontrar en [B](#).

Para instalar el programa del profesor simplemente deberemos copiar la carpeta */instalacion/profesor/* en cualquier lugar del PC. También deberemos modificar los parámetros de los archivos vistos anteriormente. La instalación del programa de la PDA es igual de sencilla. Deberemos copiar los archivos de la carpeta */instalacion/alumno/* en cualquier lugar que deseemos en nuestra PDA.

Una vez posicionados los archivos en el ordenado o dispositivo, deberemos iniciar los programas desde los únicos ejecutables que hay en las carpetas. Para el caso del profesor, ejecutar el archivo «VisualDebugger Administrator.exe». En el caso del alumno desde la PDA, el archivo «VisualDebugger.exe».

B. BASE DE DATOS

El código a insertar en la base de datos es el siguiente.

```

1  -- phpMyAdmin SQL Dump
2  -- version 2.10.3
3  -- author: Miguel Angel Dominguez Coloma
4
5  SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
6
7  --
8  -- Base de datos: 'visualdebugger'
9  --
10
11  -----
12
13  --
14  -- Estructura de tabla para la tabla 'description'
15  --
16
17  CREATE TABLE description (
18    id int(11) NOT NULL auto_increment,
19    project varchar(128) NOT NULL,
20    'name' varchar(128) NOT NULL,
21    'value' varchar(256) NOT NULL,
22    PRIMARY KEY (id),
23    UNIQUE KEY project (project,'name')
24  ) ENGINE=MyISAM DEFAULT CHARSET=utf8;
25
26  -----
27
28  --
29  -- Estructura de tabla para la tabla 'group'
30  --
31
32  CREATE TABLE 'group' (
33    id int(11) NOT NULL auto_increment,
34    'name' varchar(128) NOT NULL,
35    description varchar(256) NOT NULL,
36    PRIMARY KEY (id),
37    UNIQUE KEY 'name' ('name')
38  ) ENGINE=MyISAM DEFAULT CHARSET=utf8;
39
40  -----
41

```

```

42 --
43 -- Estructura de tabla para la tabla 'input'
44 --
45
46 CREATE TABLE input (
47   id int(11) NOT NULL auto_increment,
48   project varchar(128) NOT NULL,
49   'name' varchar(128) NOT NULL,
50   description varchar(256) default NULL,
51   'text' longtext NOT NULL,
52   globals tinyint(1) NOT NULL default '1',
53   xmlTrace longtext,
54   PRIMARY KEY (id),
55   UNIQUE KEY project (project,'name')
56 ) ENGINE=MyISAM DEFAULT CHARSET=utf8;
57
58 -----
59
60 --
61 -- Estructura de tabla para la tabla 'project'
62 --
63
64 CREATE TABLE project (
65   id int(11) NOT NULL auto_increment,
66   'name' varchar(128) NOT NULL,
67   'code' longtext NOT NULL,
68   debuggerPackName varchar(128) NOT NULL,
69   debugged tinyint(1) NOT NULL default '0',
70   PRIMARY KEY (id),
71   UNIQUE KEY 'name' ('name')
72 ) ENGINE=MyISAM DEFAULT CHARSET=utf8;
73
74 -----
75
76 --
77 -- Estructura de tabla para la tabla 'project_group'
78 --
79
80 CREATE TABLE project_group (
81   id int(11) NOT NULL auto_increment,
82   project varchar(128) NOT NULL,
83   'group' varchar(128) NOT NULL,
84   PRIMARY KEY (id),
85   UNIQUE KEY project (project,'group')
86 ) ENGINE=MyISAM DEFAULT CHARSET=utf8;
87
88 -----
89
90 --

```

```

91  -- Estructura de tabla para la tabla 'user'
92  --
93
94  CREATE TABLE 'user' (
95      id int(11) NOT NULL auto_increment,
96      userNumber int(5) NOT NULL,
97      'name' varchar(128) NOT NULL,
98      surname varchar(128) NOT NULL,
99      'password' varchar(16) NOT NULL,
100     PRIMARY KEY (id),
101     UNIQUE KEY userNumber (userNumber)
102 ) ENGINE=MyISAM DEFAULT CHARSET=utf8;
103
104  -----
105
106  --
107  -- Estructura de tabla para la tabla 'user_group'
108  --
109
110  CREATE TABLE user_group (
111      id int(11) NOT NULL auto_increment,
112      'user' int(11) NOT NULL,
113      'group' varchar(128) NOT NULL,
114      PRIMARY KEY (id),
115      UNIQUE KEY 'user' ('user','group')
116 ) ENGINE=MyISAM DEFAULT CHARSET=utf8;

```

Listado B.1: Estructura MySQL de la base de datos

C. GUÍA RÁPIDA

C.1 Programa del profesor

El programa del profesor nos servirá para administrar grupos, alumnos y usuarios. La figura C.1 nos muestra la ventana principal de la aplicación, la cual consta de dos etiquetas diferenciadas: *Administration* y *Project editor*.

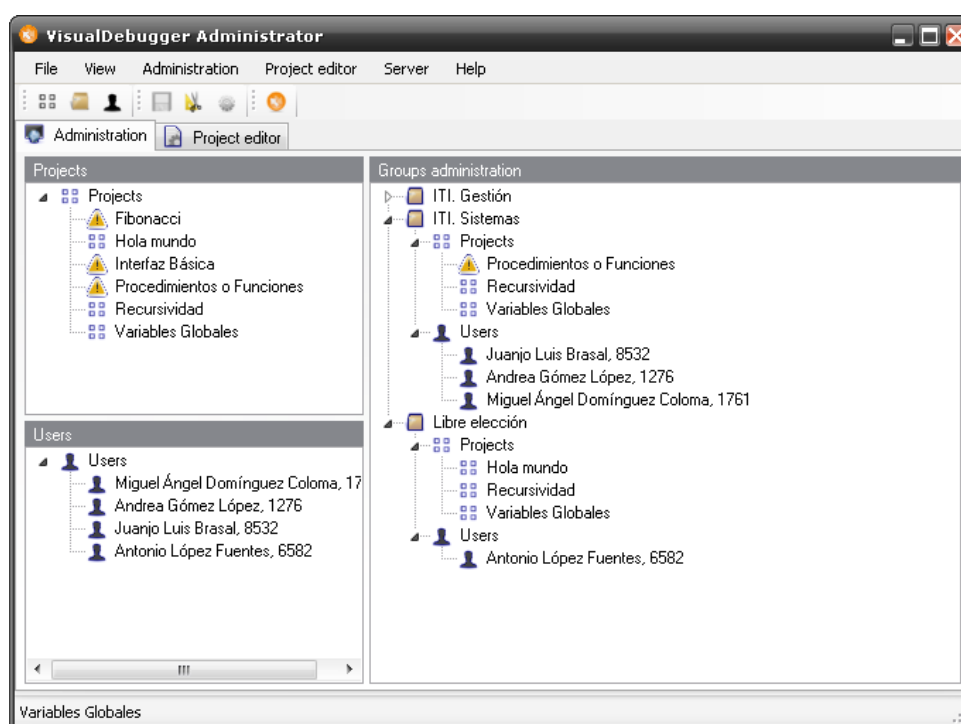


Figura C.1.: Ventana principal del programa del profesor.

- *Administration* nos servirá para crear nuevos grupos, alumnos y usuarios. Para crear éstos únicamente tendremos que pulsar sobre los botones de la barra de herramientas superior. La caja significa crear un nuevo grupo, la figura de una persona creará un nuevo usuario y los cuatro cuadrados creará un proyecto. Como podemos ver en las figuras C.2 y C.3 crear un grupo y un alumno nos llevará rellenar unos pocos campos. En el caso del alumno, la contraseña deberá ser la misma en los dos campos para poder continuar, pero no es obligatoria.

Para crear un proyecto deberemos de estar atentos a tres nuevas pantallas. Una vez que pulsamos el botón de añadir proyecto se nos presenta una ventana como la de la figura C.4. En ella deberemos asignar un nombre al proyecto, elegir uno de los depurador y compiladores disponibles en la aplicación y asignarle el código fuente

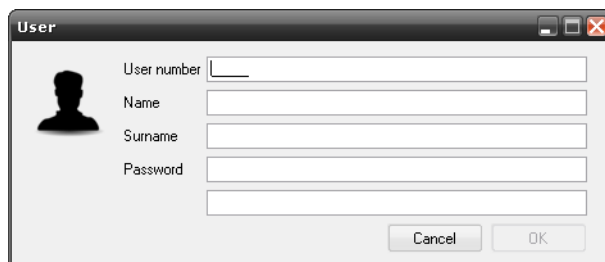

 A dialog box titled "User" with a silhouette icon on the left. It contains four text input fields: "User number", "Name", "Surname", and "Password". The "Password" field has a second, empty input field below it. At the bottom right are "Cancel" and "OK" buttons.

Figura C.2.: Creando un nuevo usuario.

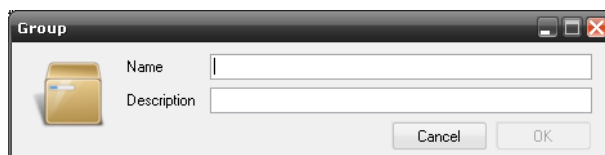

 A dialog box titled "Group" with a folder icon on the left. It contains two text input fields: "Name" and "Description". At the bottom right are "Cancel" and "OK" buttons.

Figura C.3.: Creando un nuevo grupo.

desde un archivo externo. Se podrá escribir, también, directamente el código en el editor como veremos más adelante.

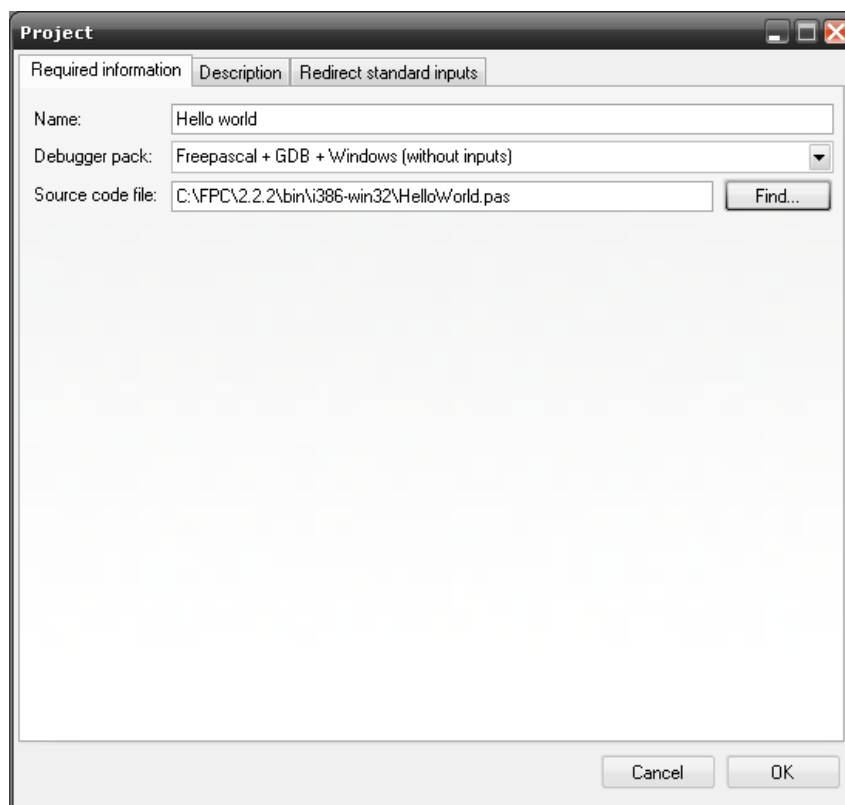

 A dialog box titled "Project" with three tabs: "Required information", "Description", and "Redirect standard inputs". The "Required information" tab is active. It contains three fields: "Name:" with the text "Hello world", "Debugger pack:" with a dropdown menu showing "Freepascal + GDB + Windows (without inputs)", and "Source code file:" with the text "C:\FPC\2.2.2\bin\i386-win32\HelloWorld.pas" and a "Find..." button. At the bottom right are "Cancel" and "OK" buttons.

Figura C.4.: Creando un nuevo proyecto.

La segunda ventana, mostrada en la figura C.5, en el proceso de creación de un proyecto nos ayudará a añadir metainformación al proyecto. Esta no es obligatoria en un proyecto, pero con ella el alumno podrá tener más datos sobre éste. Así pues,

se nos mostrará unos campos básicos al iniciar un nuevo proyecto, pero podremos borrarlos o añadir nuevos campos con el botón *Add*. El campo *Value* deberá contener alguna información, en otro caso el programa entenderá que no se necesita ese campo y no recogerá su información.

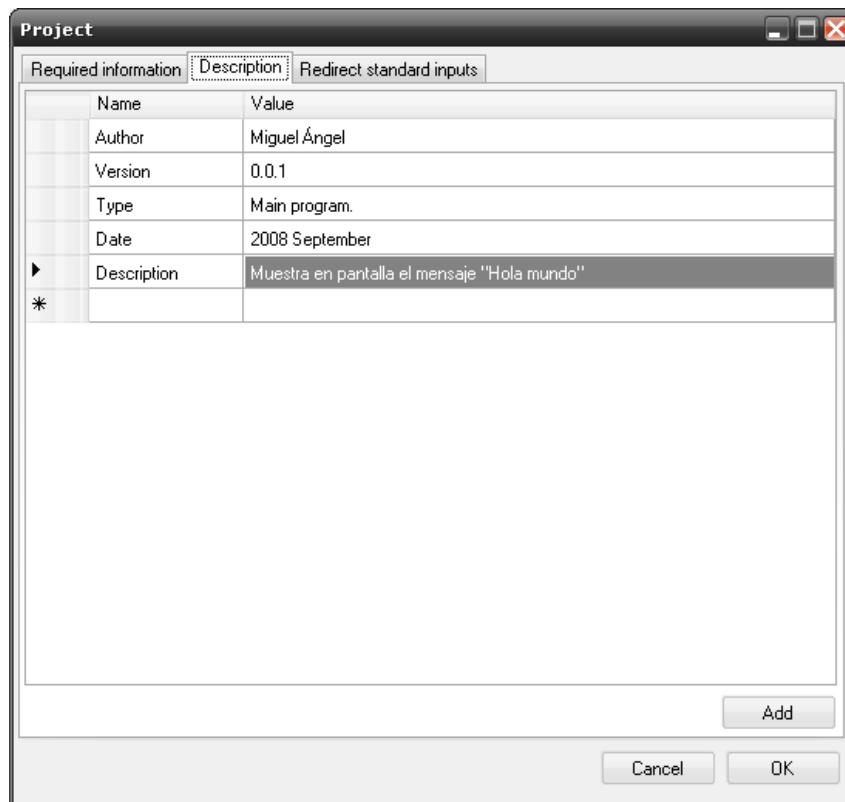


Figura C.5.: Añadiendo meta-información a un nuevo proyecto.

La tercera ventana, mostrada en la figura C.6, es la más compleja en la creación del proyecto, pero aun así fácil de manejar. Servirá para crear nuevas entradas estándar para el proyecto. Distinguimos dos partes: la superior *Add* y la inferior *Inputs*. La parte superior nos ayudará a crear nuevas entradas. Los datos que necesitaremos serán añadir un nombre, una descripción, un texto que podrá ser recogido desde un archivo o escrito a mano y la opción de mostrar las variables globales en la traza una vez que entremos dentro de procedimientos o funciones. Una vez rellenados estos datos, pulsaremos el botón *Add* y la nueva entrada pasará a la zona de abajo. La zona inferior nos ayudará a administrar las entradas. Podremos ver una lista en la que podremos elegir las entradas disponibles y cambiar sus datos en los campos. Una vez modificada una entrada, deberemos pulsar el botón *Modify*. En caso de querer borrar una entrada, pulsaremos el botón *Remove*.

En muchos programas no nos interesa la entrada estándar mediante teclado, por ello, el proyecto dispondrá de una entrada por defecto que no contendrá ningún valor.

Una vez que hemos creado proyectos, grupos y usuarios. Mediante el ratón arrastraremos los usuarios y los proyecto hacia los grupos. De esta forma, los proyectos

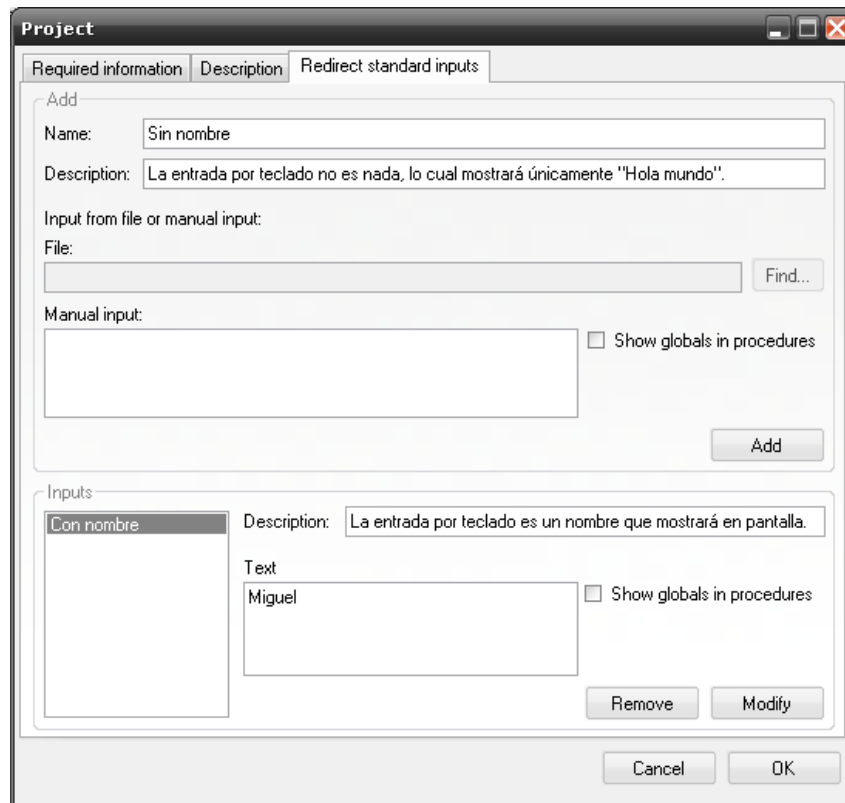


Figura C.6.: Añadiendo nuevas entradas al proyecto.

o usuarios quedarán asignados a los grupos. En caso de querer eliminar usuarios o proyectos de los grupos, pulsaremos con el botón derecho sobre ellos y elegiremos la opción *Exclude*. Esta acción no borra un proyecto, solo deja de ser asignado a un grupo.

Los proyecto que posean un icono en forma de «Aviso», serán los proyectos que necesitan ser compilados y depurados mediante la herramienta *Project editor* que veremos a continuación.

- *Project editor* es la pestaña que se encargará de editar, compilar y depurar el código fuente de un proyecto, como podemos ver en la figura C.7. Una vez que hemos seleccionado con el botón derecho sobre proyecto y hemos elegido la opción *Edit with «Project editor»*, éste pasará a ser editado en esta ventana. Esta ventana la encontraremos muy familiar, pues los IDE de programación utilizan algunas similares. Por un lado encontramos a la izquierda la descripción del proyecto. A la derecha encontramos un editor en el que podremos modificar el código o escribirlo de nuevo en caso de que no hubieramos seleccionado un archivo de texto. En la parte de abajo aparecerá la información de las entrada y la salida de consola que nos ayudará a saber si un programa se ha compilado correctamente, y de no serlo así, ver los errores o avisos.

Esta ventana posee tres botones en la barra de herramientas. El botón con forma de disquette guardará cualquier cambio que realicemos sobre el código o el proyecto. El segundo con un lápiz y una regla, nos mostrará la ventana de edición del proyecto

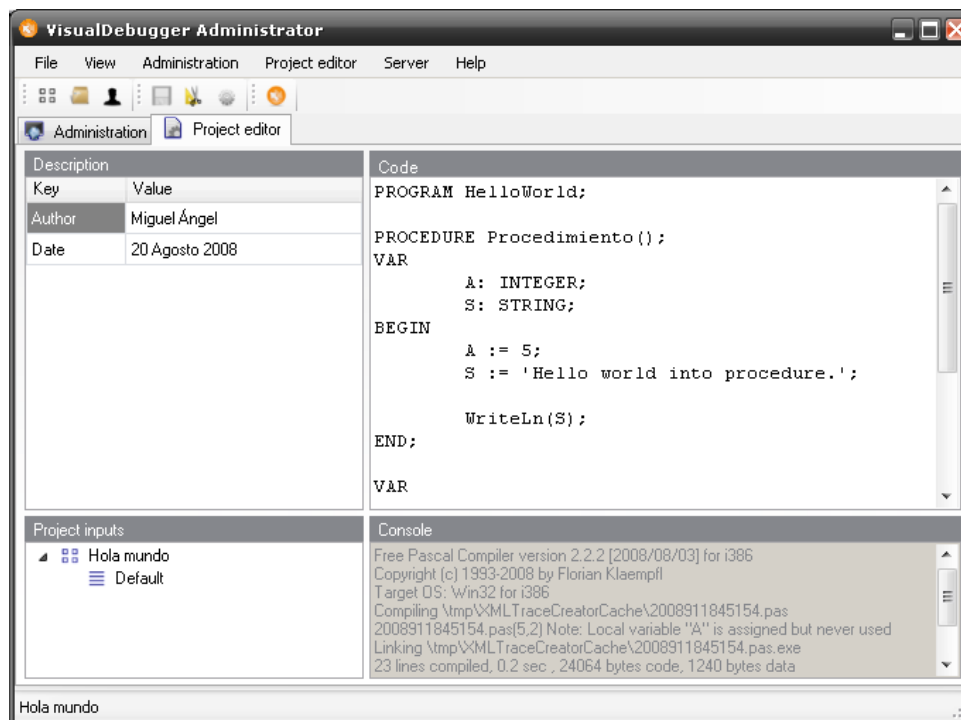


Figura C.7.: Ventana de edición de un proyecto.

al igual que la figura C.4, en la que la modificación será igual que en el apartado de creación de un nuevo proyecto. El tercer botón, en forma de ... será el encargado de compilar y depurar el programa como podemos ver en la figura C.8.

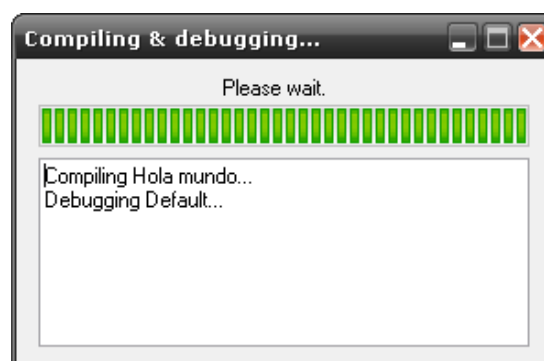


Figura C.8.: Compilando y depurando un proyecto.

Para finalizar, podremos observar un pequeño botón redondo con dos flechas en su interior. Este botón nos servirá para reiniciar el estado del servidor con las asignaciones de grupos que hay actualmente en el programa. Así pues, mientras se realizan cambios en los grupos, usuarios y proyectos, no afectará al estado actual del servidor y no se contemplarán estos cambios, se necesitará pulsar este botón para «avisar» al servidor.

C.2 Programa del alumno

El alumno dispondrá de una herramienta en la que podrá trabajar de forma remota o local. Ya que las dos formas son similares, explicaremos la forma remota, la cual tiene más funcionalidad.

C.2.1 Ejemplo del uso de un proyecto remoto

Una vez abierta la aplicación, se seleccionará el menú *File - Open remote project....* A continuación aparecerá una ventana de registro como la de la figura C.9 en la que deberemos escribir nuestro número de usuario y contraseña si poseemos.

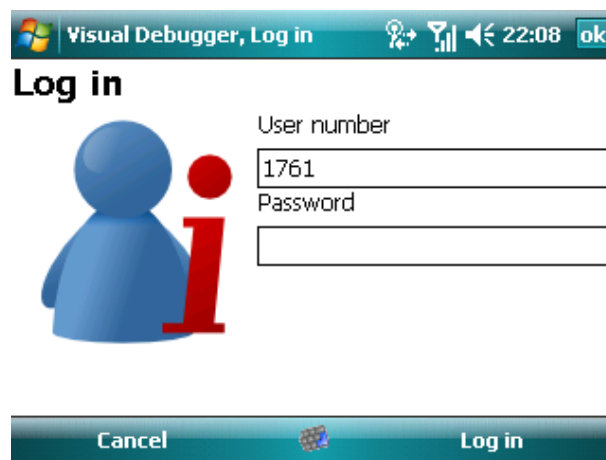


Figura C.9.: Ventana de registro en la PDA.

A continuación, una vez que la conexión a sido satisfactoria, se nos dará la posibilidad de elegir un grupo, proyecto y entrada. Como podemos ver en la figura C.10, según seleccionamos el grupo, el proyecto y la entrada, los campos se actualizan mostrándonos todas las posibles elecciones. Por ejemplo, al seleccionar un proyecto nos actualiza su lista de entradas.

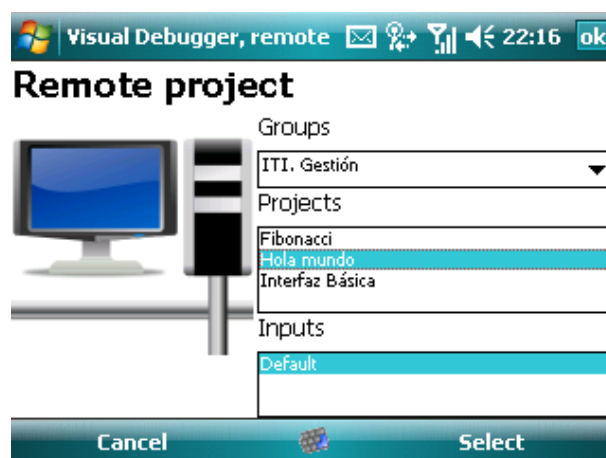


Figura C.10.: Elección de un proyecto remoto.

Una vez el proyecto se ha cargado (se ha recogido del servidor), aparecerá pintada de amarillo la línea por la que la depuración está en proceso. Podremos elegir entre los dibujos que aparecen en la parte izquierda de la barra de herramientas y pulsar sobre la pizarra para plasmarlos como podemos ver en la figura C.11.

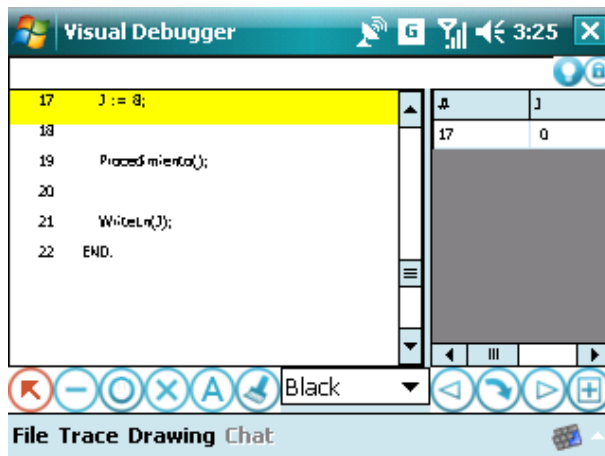


Figura C.11.: Ventana principal para dibujar y manejar la tabla de variables.

En esta misma ventana, la barra de herramientas nos ofrece a su izquierda botones para poder avanzar, retroceder o introducirse dentro de los procedimientos y funciones del código. Una vez usados estos comandos, la tabla de variables se irá actualizando con los nuevos valores. Si queremos actualizar el valor de un campo de la tabla de variables, únicamente tendremos que pinchar una vez sobre el campo y nos aparecerá un recuadro para la modificación de la variable como en la figura C.12.

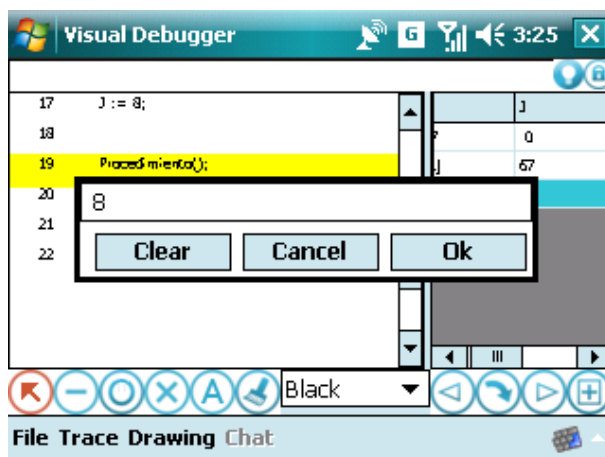


Figura C.12.: Modificando el valor de una variable en la tabla de variables.

Para incluir nuevas suposiciones sobre los valores de las variables, pulsaremos sobre el botón derecho con forma de «+». En la nueva ventana elegiremos una de las posibles variables que existen en el programa, el valor asignado a esa variable y en la línea en la que posiblemente tenga ese valor. Podemos ver un ejemplo de esta ventana en la figura C.13.

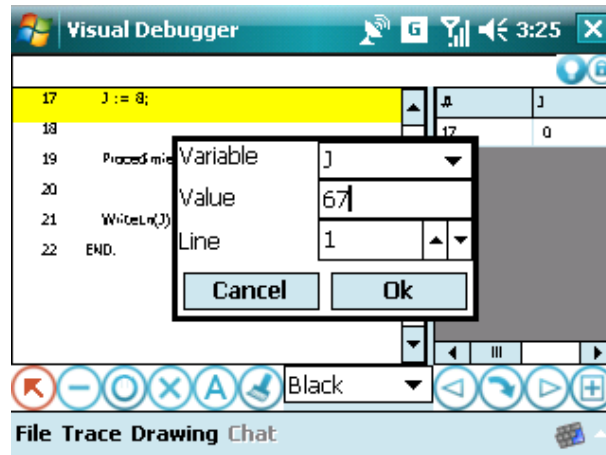


Figura C.13.: Creando una suposición sobre el contenido de una variable.

El programa dispone de un sistema de mensajes entre los usuarios. Para enviar un mensaje al resto de usuarios, escribiremos en la barra superior y pulsaremos el botón con forma de bombilla para enviar el mensaje como en la figura C.14. Para pedir el turno y avisar al resto de alumnos, tendremos que pinchar sobre el botón superior con forma de candado y el programa automáticamente enviará un mensaje con el nombre del alumno a todos los demás.

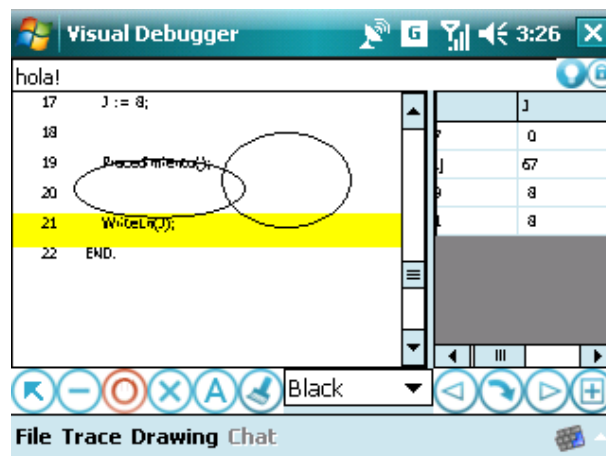


Figura C.14.: Mandando un mensaje al resto de alumnos.

El programa dispone de un menú de opciones para la modificación del idioma y conexión como podemos ver en las figuras C.15 y C.16.

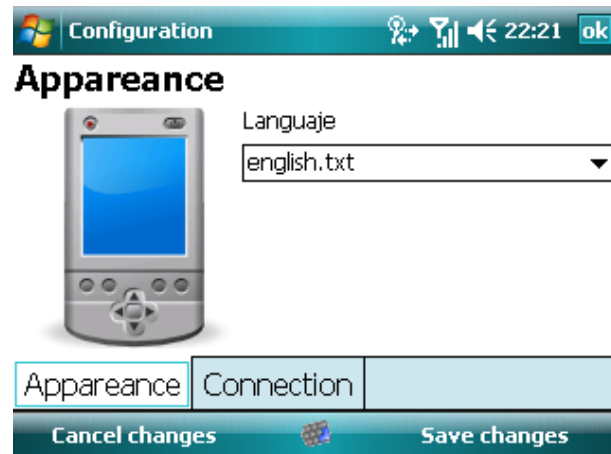


Figura C.15.: Cambiando el idioma del programa.

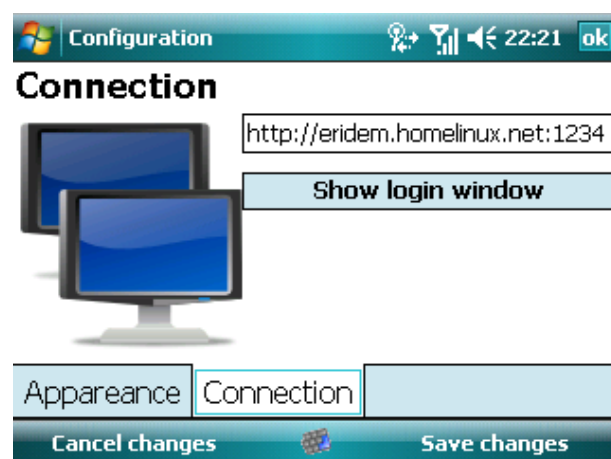


Figura C.16.: Cambiando las propiedades de la conexión.