UNIVERSIDAD REY JUAN CARLOS

**INGENIERÍA INFORMÁTICA**

Escuela Técnica Superior de Ingeniería Informática

**Curso académico 2011-2012**

**Proyecto Fin de Carrera**

# SOLVER
# Un agente que te enseña a resolver problemas de matemáticas

Autor: Daniel Sánchez Martín
Tutor: Diana Pérez Marín

# AGRADECIMIENTOS

En primer lugar, quisiera que mis primeras palabras de agradecimiento fuesen para Diana Pérez Marín, tutora de este proyecto, gracias por toda la dedicación, interés y ayuda que ha mostrado durante la realización de este proyecto.

Deseo hacer un agradecimiento especial a mis dos sobrinos, Francisco Javier y Alicia, y al resto de personas que han participado y han hecho posible las pruebas de aceptación realizadas en la aplicación. Extiendo este agradecimiento a toda mi familia, por estar ahí siempre, por aguantarme en los malos y buenos momentos, por su apoyo incondicional, gracias.

Por último, a todas aquellas personas que han intervenido en mi formación durante estos largos años de universidad, muchas gracias.

# RESUMEN

El uso de las tecnologías de la información ha pasado a formar parte de nuestra vida cotidiana en casi cualquier actividad diaria, este proyecto fin de carrera se centra en las posibilidades que ofrece las tecnologías de la información en el ámbito de la enseñanza.

En concreto, el objetivo es el desarrollo de un sistema informático de tipo Agente Conversacional Pedagógico (ACP) que permita ayudar al alumno en su proceso de aprendizaje de problemas y ecuaciones matemáticas.

La aplicación está dividida en dos partes, la parte del profesor y la parte del alumno. La parte del profesor está diseñada para que profesores de matemáticas definan problemas que puedan ser parametrizables. La parte del alumno está orientada a un sistema ACP para que el alumno pueda interactuar paso a paso y resolver los problemas planteados por el profesor.

El sistema ha sido construido mediante una arquitectura de tres capas (datos, negocio, presentación), desarrollado en el lenguaje Java. Se ha utilizado la biblioteca gráfica SWING. Se ha implementado un analizador léxico, sintáctico y semántico mediante JFlex y CUP. Se ha dispuesto MySQL como sistema gestor de base de datos para gestionar la información de la aplicación, además se ha empleado Hibernate que facilita el mapeo de atributos entre una base de datos relacional y el modelo de objetos de una aplicación.

Se han realizado pruebas de aceptación con seis niños de edades comprendidas entre 10 y 13 años con unos resultados satisfactorios, se ha obtenido una puntuación media de 7,5 sobre 10 en los cuestionarios de satisfacción con preguntas relacionadas con la utilidad, valoración o usabilidad de la aplicación.

# ÍNDICE


1 Introducción ... 1
2 Objetivos ... 4
2.1 Estudio de alternativas ... 4
2.1.1 Tipo aplicación ... 5
2.1.1.1 Aplicación de escritorio ... 5
2.1.1.2 Aplicación web ... 5
2.1.1.3 Justificación tipo aplicación elegida ... 6
2.1.2 Paradigmas de programación ... 7
2.1.2.1 Paradigma Declarativo ... 7
2.1.2.1.1 Paradigma Funcional ... 8
2.1.2.1.2 Paradigma Lógico ... 9
2.1.2.2 Paradigma Imperativo ... 9
2.1.2.3 Paradigma Orientado a Objetos ... 10
2.1.2.4 Justificación Paradigma elegido ... 11
2.1.3 Lenguajes Orientados a Objetos ... 12
2.1.3.1 Historia ... 12
2.1.3.2 Actualidad ... 13
2.1.3.2.1 Python ... 16
2.1.3.2.2 C++ ... 16
2.1.3.2.3 Java ... 17
2.1.3.2.4 Justificación del lenguaje de programación elegido ... 18
2.1.4 Gestor de Bases de Datos ... 19
2.1.4.1 Oracle ... 21
2.1.4.2 PostgreSQL ... 21
2.1.4.3 MySQL ... 22
2.1.4.4 Justificación SGBD elegido ... 23
2.1.5 Metodología Empleada ... 25
2.1.6 Plataforma de desarrollo ... 26
2.2 Historia Agentes Conversacionales Pedagógicos ... 27
3 Descripción Informática ... 30
3.1 Análisis de Requisitos ... 30
3.2 Casos de Uso ... 32
3.3 Diseño Arquitectura del Sistema ... 34
3.3.1 Servidor ... 35
3.3.2 Cliente ... 36
3.4 Diseño Base de Datos ... 36
3.5 Aplicación Java ... 46
3.5.1 Capa Datos ... 49
3.5.2 Capa Negocio ... 50
3.5.3 Capa Presentación ... 60
4 Pruebas ... 71
4.1 Pruebas unitarias ... 71
4.1.1 Pruebas de caja blanca ... 71
4.1.2 Pruebas de caja negra ... 74
4.1 Pruebas de integración ... 75
4.2 Pruebas de validación ... 77
4.3 Pruebas de aceptación ... 85
5 Conclusiones y Trabajos Futuros ... 89

Bibliografía ..... 90
Anexo A: Manual de Instalación ..... 92
Anexo B: Manual de usuario profesor ..... 96
Anexo C: Manual de usuario alumno ..... 102
Anexo D: Cuestionario de satisfacción ..... 104

# ÍNDICE DE FIGURAS

Figura 1.1 – Ejemplo Interfaz ..... 2
Figura 2.1 – Ejemplo Programación Funcional ..... 8
Figura 2.2 – Ejemplo Programación Lógica ..... 9
Figura 2.3 – Ejemplo Programación Imperativa ..... 10
Figura 2.4 – Ejemplo Programación Orientada a Objetos ..... 11
Figura 2.5 – Índice TIOBE Diciembre 2010 [6] ..... 14
Figura 2.6 – Lenguajes de programación del concurso Google AI Challenge ..... 14
Figura 2.7 – Gráficos sobre la popularidad de los lenguajes de programación ..... 15
Figura 2.8 – Modelo de desarrollo en espiral ..... 26
Figura 2.9 – Clip Office ..... 28
Figura 3.1 – Caso de Uso Alumno ..... 33
Figura 3.2 – Caso de Uso Profesor ..... 34
Figura 3.3 – Componentes del sistema ..... 35
Figura 3.4 – Modelo Entidad-Relación ..... 37
Figura 3.5 – Entidad Alumno ..... 38
Figura 3.6 – Entidad Profesor ..... 39
Figura 3.7 – Entidad Curso ..... 39
Figura 3.8 – Entidad Problema ..... 40
Figura 3.9 – ERM detalle ..... 42
Figura 3.10 – ERM relaciones ..... 42
Figura 3.11 – tProfesores ..... 43
Figura 3.12 – tAlumnos ..... 43
Figura 3.13 – tCursos ..... 43
Figura 3.14 – tHistorico ..... 43
Figura 3.15 – tProblemas ..... 44
Figura 3.16 – tInstrucciones ..... 44
Figura 3.17 – tParametros ..... 44
Figura 3.18 – tCondiciones ..... 44
Figura 3.19 – Modelo relacional estático ..... 45
Figura 3.20 – Tabla tAlumnos ..... 45
Figura 3.21 – Tabla tProfesores ..... 45
Figura 3.22 – Tabla tCursos ..... 45
Figura 3.23 – Tabla tProblemas ..... 46
Figura 3.24 – Tabla tInstrucciones ..... 46
Figura 3.25 – Tabla tParametros ..... 46
Figura 3.26 – Tabla tCondiciones ..... 46
Figura 3.27 – Tabla tHistorico ..... 46
Figura 3.28 – Arquitectura Aplicación Java ..... 47
Figura 3.29 – Explorador de paquetes ..... 49
Figura 3.30 – Clase Talumnos generada ..... 50
Figura 3.31 – Ejemplo expresión matemática ..... 51
Figura 3.32 – Tokens ..... 52
Figura 3.33 – Árbol sintáctico ..... 52
Figura 3.34 – Pila ..... 53
Figura 3.35 – Pila 2 ..... 53
Figura 3.36 – Pila 3 ..... 54
Figura 3.37 – Pila 4 ..... 54
Figura 3.38 – Pila 5 ..... 54

Figura 3.39 – Gramática ..... 57
Figura 3.40 – rellenarPila ..... 59
Figura 3.41 – Prototipo pantalla login ..... 61
Figura 3.42 – Pantalla login ..... 61
Figura 3.43 – Prototipo pantalla inicio del usuario alumno ..... 61
Figura 3.44 – Pantalla inicio del usuario alumno ..... 61
Figura 3.45 – Prototipo pantalla inicio del usuario profesor ..... 62
Figura 3.46 – Pantalla inicio del usuario profesor ..... 62
Figura 3.47 – Prototipo pantalla datos generales ..... 62
Figura 3.48 – Pantalla datos generales ..... 62
Figura 3.49 – Prototipo pantalla alumnos del usuario profesor ..... 63
Figura 3.50 – Pantalla alumnos del usuario profesor ..... 63
Figura 3.51 – Prototipo pantalla problemas del usuario profesor ..... 63
Figura 3.52 – Pantalla problemas del usuario profesor ..... 64
Figura 3.53 – Prototipo pantalla problema del usuario profesor ..... 64
Figura 3.54 – Pantalla problema del usuario profesor ..... 64
Figura 3.55 – Prototipo pantalla instrucción del usuario profesor ..... 65
Figura 3.56 – Pantalla instrucción del usuario profesor ..... 65
Figura 3.57 – Prototipo pantalla problema del usuario alumno ..... 65
Figura 3.58 – Prototipo pantalla problema del usuario alumno ..... 66
Figura 3.59 – Caso práctico 1 de 3 ..... 66
Figura 3.60 – Caso práctico 2 de 3 ..... 67
Figura 3.61 – Caso práctico 3 de 3 ..... 67
Figura 3.62 – Icono ayuda ..... 68
Figura 4.1 – Método procesar ..... 72
Figura 4.2 – Grafo de flujo ..... 72
Figura 4.3 – Interrelación componentes ..... 76
Figura 4.4 – RF1, RF2, RF5 ..... 77
Figura 4.5 – RF3, RF8 ..... 78
Figura 4.6 – RF4 ..... 78
Figura 4.7 – RF6, RF9 ..... 78
Figura 4.8 – RF10, RF12 ..... 79
Figura 4.9 – RF11, RF12 ..... 79
Figura 4.10 – RF14, RF15 ..... 80
Figura 4.11 – RF16, RF17 ..... 80
Figura 4.12 – RF18, RF19 ..... 81
Figura 4.13 – RF20 ..... 81
Figura 4.14 – RF21 ..... 82
Figura 4.15 – RF23, RF24 ..... 82
Figura 4.16 – RF25 ..... 83
Figura 4.17 – RF26, RF27 ..... 83
Figura 4.18 – RF28 ..... 84
Figura 4.19 – RF29 ..... 84
Figura 1 – Instalación MySQL 1 ..... 92
Figura 2 – Instalación MySQL 2 ..... 93
Figura 3 – Configuración MySQL 1 ..... 93
Figura 4 – Configuración MySQL 2 ..... 94
Figura 5 – Configuración MySQL 3 ..... 94
Figura 6 – Configuración MySQL 4 ..... 94
Figura 7 – Configuración MySQL 5 ..... 95

Figura 8 – Configuración MySQL 6 ..... 95
Figura 9 – Ejecutar scriptBBDD.sql ..... 95
Figura 10 – Pantalla acceso profesor ..... 96
Figura 11 – Pantalla inicio profesor ..... 96
Figura 12 – Pantalla datos profesor ..... 96
Figura 13 – Pantalla cursos ..... 97
Figura 14 – Pantalla cursos ..... 97
Figura 15 – Pantalla alumnos ..... 98
Figura 16 – Pantalla alumnos ..... 98
Figura 17 – Pantalla problemas ..... 99
Figura 18 – Pantalla problema ..... 99
Figura 19 – Icono manual ..... 100
Figura 20 – Manual pantalla problema ..... 100
Figura 21 – Pantalla ecuaciones ..... 100
Figura 22 – Pantalla ecuación ..... 101
Figura 23 – Pantalla acceso alumno ..... 102
Figura 24 – Pantalla inicio alumno ..... 102
Figura 25 – Pantalla datos personales ..... 103
Figura 26 – Pantalla histórico alumno ..... 103
Figura 27 – Pantalla resolver problema ..... 103

# ÍNDICE DE TABLAS

Tabla 2.1 – Comparativa Aplicación escritorio vs Aplicación Web ..... 6
Tabla 2.2 – Comparativa de los paradigmas de programación..... 11
Tabla 2.3 – Comparativa de los lenguajes de programación ..... 19
Tabla 2.4 – Comparativa SGBD..... 24
Tabla 4.1 – Casos de prueba caja blanca ..... 74
Tabla 4.2 – Casos de prueba caja negra..... 75
Tabla 4.3 – Información usuarios pruebas aceptación ..... 86
Tabla 4.4 – Estadísticas pruebas de aceptación usuarios..... 86
Tabla 4.5 – Resultados Cuestionarios Satisfacción ..... 87

# ABREVIATURAS

ACP: Agente Conversacional Pedagógico

POO: Programación Orientada a Objetos

SGBD: Sistema Gestión de Base de Datos

SGBDR: Sistema Gestión de Base de Datos Relacional

MSDE: Microsoft Desktop Engine

DBA: Administrador de Base de Datos

JDBC: Java Database Connectivity

ODBC: Open Database Connectivity

IDE: Entorno de Desarrollo Integrado

GUI: Graphical User Interface

JFC: Java Foundation Classes

AWT: Abstract Window Toolkit

RIA: Rich Internet Applications

ER: Entidad Relación

ERM: Modelo Relacional extendido

SQL: Structured Query Language

ORM: Mapeo Objeto Relacional

CASE: Computer Aided Software Engineering

GPL: General Public License

API: Application Programming Interface

LGPL: Lesser General Public License

ESO: Educación Secundaria Obligatoria

EGB: Educación General Básica

# 1 Introducción

Los grandes avances en las tecnologías de la información en las últimas décadas han dado lugar a cambios en casi todos los aspectos de nuestra vida diaria. La enseñanza y el aprendizaje han seguido también su propio camino de cambios y evolución. A día de hoy se cuestiona cada vez más el sistema tradicional de enseñanza y se plantean diferentes sistemas computacionales que tienen como finalidad apoyar el proceso de enseñanza y en particular el aprendizaje de los estudiantes.

Frente a los nuevos retos que nos plantea la educación hoy en día, se han implementado un gran número de aplicaciones y herramientas software de enseñanza asistida por ordenador, estas herramientas permiten repasar y estudiar conceptos dando mayor responsabilidad al alumno, pudiendo ir al ritmo que desee. Si bien existen importantes y numerosos desarrollos, todavía quedan algunos temas no resueltos, debido a problemas asociados que no son fácilmente perceptibles como pueden ser la ausencia de interacción, la sensación de aislamiento o falta de motivación.

Para evitar esto, el sistema que se desarrolla en este trabajo dispone de un agente conversacional pedagógico con una interfaz que tiene como objetivo ser agradable e intuitivo, y permitir al alumno interactuar paso a paso en la resolución de los problemas.

El proyecto se centra en la asignatura de matemáticas que tradicionalmente ha sido considerada como una de las asignaturas con un alto fracaso académico. Muchos estudiantes encuentran dificultades a la hora de resolver problemas matemáticos, cada persona debe dedicarle un tiempo de práctica diferente hasta conseguir los resultados esperados, los agentes conversacionales pedagógicos facilitan la posibilidad de repaso individualizado y flexible.

El aumento de formación tradicional y a distancia, el fomento del uso de las tecnologías de la información junto con las diferentes experiencias y resultados positivos en el uso de agentes para la enseñanza, auguran que estos sistemas sean considerados una herramienta fundamental en el proceso de enseñanza en los próximos años.

El agente conversacional diseñado, llamado Solver, permita al alumno interactuar paso a paso con el agente y resolver ecuaciones y problemas matemáticos adaptados a su nivel. La Figura 1.1 muestra un ejemplo de la interfaz del programa.

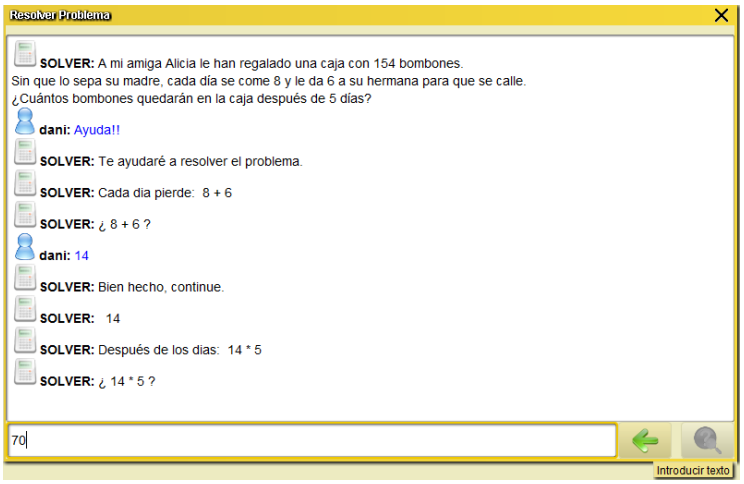


*Figura 1.1 – Ejemplo Interfaz*

En la aplicación se distinguen dos perfiles de usuarios bien diferenciados: profesor y alumno.

Por un lado, los usuarios con el perfil de profesor deben tener la posibilidad de añadir, editar y eliminar problemas, pudiendo definir los enunciados, parámetros y los pasos que deberá dar el alumno para la resolución del problema. De forma similar, también debe poder añadir, editar y eliminar las “plantillas” para definir ecuaciones matemáticas. Estas opciones se deben realizar bajo una interfaz intuitiva y no compleja, siendo consciente que el profesor puede no tener un alto conocimiento de informática.

Por otro lado, los usuarios con el perfil de alumno podrán interactuar con el agente para ir resolviendo paso a paso los problemas y ecuaciones que han sido definidas por el profesorado para su nivel. El agente conversacional debe ser amigable y debe “simular” en todo momento la presencia del profesor.

El presente documento se distribuye en 5 grandes secciones:

- **Introducción:** En esta sección, el principal propósito ha sido que el lector entienda el contexto del trabajo y describir el alcance del documento.
- **Objetivos:** Presenta los objetivos generales del proyecto y se analizan las distintas alternativas disponibles.
- **Descripción informática:** Se realiza una amplia descripción del sistema informático, requisitos, análisis, diseño y algoritmo del trabajo realizado junto con un caso práctico ilustrativo.
- **Pruebas**: Se describen las pruebas llevadas a cabo para probar el correcto funcionamiento y aceptación del programa.
- **Conclusiones y trabajo futuro:** Se analizan las conclusiones obtenidas y se proponen posibles líneas de investigación.

# 2 Objetivos

El objetivo principal del proyecto es el desarrollo de un agente conversacional, Solver, que sirva de apoyo al alumno, pudiendo lograr disminuir las dificultades a la hora de abordar y proceder a resolver un problema matemático o resolver un cálculo complejo.

Los objetivos que se desean cumplir mediante Solver son los siguientes:

- Facilitar un proceso de repaso flexible e individualizado.
- Fomentar el trabajo autónomo del alumno.
- Evitar la posible sensación de aislamiento, conservando el contacto con Solver.
- Proporcionar una herramienta al profesor que genera problemas de matemáticas para asistirle en el proceso de enseñanza.

## *2.1 Estudio de alternativas*

A la hora de desarrollar un sistema software se considera de obligada necesidad analizar varios aspectos tecnológicos y sus alternativas disponibles en el mercado para construir una aplicación que cumpla con los requisitos planteados.

Para ello, debemos contemplar los distintos tipos de aplicaciones que se pueden implementar, emplear un paradigma de programación y un sistema gestor de base de datos que nos proporcionen, faciliten y simplifiquen las tareas. Además de utilizar cualquier otra herramienta que nos ayude en la construcción del sistema.

### 2.1.1 Tipo aplicación

Las aplicaciones software que existen en el mercado se pueden dividir en 2 grandes grupos:

- Aplicaciones de escritorio
- Aplicaciones web

Cada una de ellas tiene sus ventajas e inconvenientes, a continuación pasaremos a analizarlas.

#### *2.1.1.1 Aplicación de escritorio*

Son aplicaciones para ser instaladas, configuradas y utilizadas en ordenadores de escritorio. Esta aplicación implica la existencia de una relación entre procesos que solicitan servicios (clientes) y procesos que responden a estos servicios (servidores). Estos dos tipos de procesos pueden ejecutarse en el mismo procesador o en distintos.

Una ventaja de este tipo de aplicaciones es la separación de las funciones según su servicio, permitiendo situar cada función en la plataforma más adecuada para su ejecución. Habitualmente las aplicaciones de escritorio se ejecutan en una red local, de modo que no es necesario el uso ni la dependencia de internet [1].

#### *2.1.1.2 Aplicación web*

Una aplicación web es una aplicación software que se codifica en un lenguaje soportado por los navegadores web. Los usuarios acceden a ella a través de internet.

Una de las ventajas clave del uso de aplicaciones web es que el problema de gestionar el código en el cliente se reduce drásticamente, todos los cambios se realizan cambiando el código que resida en el

servidor web. Se evita la inconsistencia de versiones, no existen clientes con distintas versiones de la aplicación. Otra ventaja es la independencia de la plataforma, sólo se necesita un navegador y acceso a internet para ejecutar la aplicación [1].

Por último mencionar la existencia de aplicaciones que requieren la instalación de software especial en el navegador para poder utilizarse, estas aplicaciones se denominan Aplicaciones de Internet Ricas (RIA).

#### *2.1.1.3 Justificación tipo aplicación elegida*

Las aplicaciones de escritorio llevan en el mercado mucho más tiempo que las aplicaciones web, las cuales suelen centrar su ejecución en el lado del servidor. Por esa razón, las herramientas de desarrollo están más evolucionadas y depuradas para el desarrollo de aplicaciones de escritorio en la creación interfaces de usuario o conectar con bases de datos, permitiendo al desarrollador ahorrar tiempos.

La Tabla 2.1 muestra una comparativa de los dos tipos de aplicaciones descritas anteriormente.

| | **Aplicación Escritorio** | **Aplicación Web** |
|---|---|---|
| Acceso Internet | No | Si |
| Actualización aplicación | Cada cliente | Solo servidor |
| Dificultad personal de desarrollo | Baja | Alta |
| Experiencia personal previa | Alta | Baja |

*Tabla 2.1 – Comparativa Aplicación escritorio vs Aplicación Web*

Finalmente y tras analizas las características de cada aplicación, y teniendo en cuenta que uno de los objetivos principales es que los usuarios puedan repasar los ejercicios en sus respectivas casas haciendo uso de su ordenador personal. Se considera innecesario un sistema web que dependa de la conexión a Internet y se decide implementar una aplicación de escritorio.

### 2.1.2 Paradigmas de programación

Un paradigma de programación es un estilo de programación que provee y determina la visión que el programador tiene de la ejecución de un programa. Es un modelo básico de diseño y desarrollo de programas, que permite producir programas con unas directrices específicas [2].

Hoy en día, el uso de la computación es extremadamente diverso, dando lugar a distintas aplicaciones en múltiples áreas que han requerido centenares de lenguajes de programación, cada uno con propósitos y formas distintas. No obstante, la mayoría de estos lenguajes se pueden agrupar según la "filosofía" (o forma) para resolver los problemas [3,4]:

- Describen el problema por solucionar.
- Describen los pasos por seguir para encontrar la solución.

Todos los paradigmas son igual de válidos y cado uno tendrá sus ventajas y sus inconvenientes. Dependiendo del caso en el que nos encontremos, el sentido común y la experiencia nos dirá cual es el paradigma más apropiado a cada situación.

Aunque existe un gran número de paradigmas, se va a proceder a explicar brevemente los más utilizados e importantes actualmente.

#### *2.1.2.1 Paradigma Declarativo*

Un lenguaje declarativo se construye estableciendo hechos, reglas, restricciones, ecuaciones, transformaciones u otras propiedades que debe tener el conjunto de valores que constituyen la solución.

A partir de esta información, el sistema debe de ser capaz de derivar un esquema de evaluación que nos permita computar una solución, es

decir, no existe una descripción paso a paso de cómo llegar a la solución.

Dentro del paradigma declarativo se puede distinguir el paradigma funcional o el paradigma lógico.

#### *2.1.2.1.1 Paradigma Funcional*

El objetivo del paradigma funcional (es una variedad de la programación declarativa) es conseguir lenguajes expresivos y matemáticamente elegantes, en los que no sea necesario bajar al nivel de la máquina para describir el proceso llevado a cabo por el programa. Concibe la computación como una evaluación de funciones matemáticas y evita utilizar datos que vayan cambiando de valor a lo largo de la ejecución del programa. Un 'programa' consiste en la definición de una o más funciones.

Existen dos grandes categorías de lenguajes funcionales: los funcionales puros y los híbridos. La diferencia entre ambos estriba en que los lenguajes funcionales híbridos son menos dogmáticos que los puros, al admitir conceptos tomados de los lenguajes procedimentales, como las secuencias de instrucciones o la asignación de variables. La Figura 2.1 muestra un ejemplo de programación funcional.

Ejemplos: Lisp, Haskell

```
predecesor(x)= x-1, si x>0
sucesor(x)= x+1
suma(x,0)= x
suma(x, y)= sucesor(suma(x, predecesor(y)))

?- suma(3,2)
```

*Figura 2.1 – Ejemplo Programación Funcional*

#### *2.1.2.1.2 Paradigma Lógico*

Los programas implementados siguiendo este paradigma (es una variedad de la programación declarativa) suelen definir hechos, que son considerados ciertos siempre, y reglas, que no explican cómo se tiene que solucionar un problema, sino qué relaciones hay entre las entidades que se modelan. La ejecución consiste en la demostración de hechos sobre las relaciones por medio de preguntas.

Este paradigma está ligado históricamente al lenguaje llamado Prolog, siendo éste el primer lenguaje de programación lógico, el más conocido y utilizado. La Figura 2.2 muestra un ejemplo de programación lógica.

Ejemplo: Prolog

```
Mujer(Rosa)
Mujer(Marta)
Mujer(Laura)
Padres(Rosa, Carlos, Pilar)
Padres(Marta, Carlos, Pilar)
Padres(Laura, Carlos, Pilar)
Hermanas(X, Y):- mujer(X), mujer(Y), padres(X, P, M), padres(Y, P, M)

?- hermanas(Rosa, Marta)
?- hermanas(Rosa, X)
```

*Figura 2.2 – Ejemplo Programación Lógica*

### *2.1.2.2 Paradigma Imperativo*

El código del programa está dividido en diferentes bloques llamados funciones o procedimientos. Cada uno de estos bloques se encarga de realizar una tarea específica y puede ser llamado varias veces desde otros puntos del programa. Este tipo de programación permite ver un programa como la ejecución de una serie de tareas complicadas, lo cual disminuye la complejidad de la programación. Está basado en la idea de "divide y vencerás". La Figura 2.3 muestra un ejemplo de programación imperativa.

Ejemplo: C, BASIC

```
leer(x)
leer(y)
resultado = x + y
escribir(resultado)
```

*Figura 2.3 – Ejemplo Programación Imperativa*

### *2.1.2.3 Paradigma Orientado a Objetos*

Un programa está formado por un conjunto de objetos independientes que interactúan entre ellos. Cada objeto contiene su propio código y sus datos, tiene la capacidad de recibir peticiones (mensajes de otros objetos) para hacer tareas, procesar datos y enviar peticiones (mensajes) a otros objetos [4].

La programación orientada a objetos (POO) aporta un nuevo enfoque a los retos que se plantean en la programación estructurada cuando los problemas a resolver son complejos. Al contrario que la programación procedimental que enfatiza en los algoritmos, la POO enfatiza en los datos. En lugar de intentar ajustar un problema al enfoque procedimental de un lenguaje, POO intenta ajustar el lenguaje al problema. La idea es diseñar formatos de datos que se correspondan con las características esenciales de un problema. Los lenguajes orientados combinan en una única unidad o módulo, tanto los datos como las funciones que operan sobre esos datos. Tal unidad se llama objeto. Si se desea modificar los datos de un objeto, hay que realizarlo mediante la función miembro del objeto. Ninguna otra función puede acceder a los datos. Esto simplifica la escritura, depuración y mantenimiento del programa. Existen cuatro características básicas en un sistema orientado a objetos:

- Encapsulamiento
- Herencia
- Polimorfismo

- Abstracción

El paradigma orientado a objetos domina actualmente la computación, tanto a nivel de programación, análisis, diseño o bases de datos. La Figura 2.4 muestra un ejemplo de programación orientada a objetos.

Ejemplos: C++, Java, Smalltalk

```
class Coche {
 // variables de clase
 int velocidad;

 // métodos de la clase
 void acelerar(int nuevaVelocidad) {
  velocidad = nuevaVelocidad;
 }
```

*Figura 2.4 – Ejemplo Programación Orientada a Objetos*

### *2.1.2.4 Justificación Paradigma elegido*

La Tabla 2.2 realiza una comparativa de los paradigmas descritos anteriormente.

| Paradigma | | Términos que maneja |
|---|---|---|
| Declarativo | Funcional | funciones |
| | Lógico | hechos y reglas |
| | Imperativo | procedimientos |
| | Orientado a Objetos | clases y objetos |

*Tabla 2.2 – Comparativa de los paradigmas de programación*

El paradigma de programación actualmente más utilizado es el de la Programación Orientada a Objetos (POO), debido a su "filosofía" de ver cualquier aspecto del lenguaje como un objeto del mundo real, siendo ésta la forma en que normalmente los seres humanos vemos las cosas.

La manera más adecuada para construir nuestro sistema software, es descartar el paradigma imperativo (no ver el sistema como una serie de procedimientos y algoritmos) y plantear nuestro sistema en términos de alumnos, profesores, problemas y las relaciones entre ellos, es decir, los principios de la POO (objetos, clases y relaciones entre ellos), tanto a nivel de programación como a nivel de base de datos.

### 2.1.3 Lenguajes Orientados a Objetos

Antes de analizar los posibles lenguajes orientados a objetos disponibles en el mercado, se va a proceder a realizar un breve repaso sobre la historia de los lenguajes, resaltando cómo distintos lenguajes han ido contribuyendo, soportando nuevas características y definiendo conceptos importantes de lo que hoy se engloba en el paradigma orientado a objetos.

#### *2.1.3.1 Historia*

Un cierto número de lenguajes de programación han contribuido a la evolución de los actuales lenguajes orientados a objetos, empezando con LISP en los años 50. LISP es un lenguaje de inteligencia artificial que introdujo el concepto de ligadura dinámica y la ventaja de un entorno de desarrollo interactivo en la evolución de los lenguajes orientados a objetos. El lenguaje Simula, desarrollado en los años 60 como lenguajes para programar simulaciones, contribuyó con el concepto de clase y los mecanismos de herencia. La abstracción de datos, en forma de tipos de datos abstractos, fue introducida en los años 70, inicialmente en los lenguajes académicos como CLU y posteriormente de forma más comercial en lenguajes populares como Ada y Modula-2 [5].

La fuerza principal que impulsó el desarrollo comercial de los lenguajes orientados a objetos fue el proyecto Smalltalk, donde se

describió el concepto de clase. Estos principios ya fueron incluidos en Simula-67. A pesar de este movimiento temprano hacia los lenguajes orientados a objetos, solamente se lograron pequeñas incursiones en la comunidad de la programación general.

Con la aparición de C en los años 80 como lenguaje de programación extremadamente popular en todas las plataformas y su ampliación orientada a objetos, C++, fueron razón fundamental para la atención creciente hacia la propia programación orientada a objetos. Finalmente su dominación fue consolidada gracias al auge de las interfaces gráficas de usuario, para las cuales la programación orientada a objetos está particularmente bien adaptada.

Entre 1990 y 1994 Sun Microsystems desarrollo Java, pensado originalmente como un reemplazo de C++, orientado a objetos y destinado a dispositivos embebidos. Posteriormente fue reorientado hacia su aplicación en la Web. Finalmente, Microsoft tomando ideas de C++ y Java y como parte de su estrategia comercial .NET desarrollo C#.

#### *2.1.3.2 Actualidad*

En la actualidad, la lista de lenguajes orientados a objetos es bastante amplía (C++, Objective C, Java, Smalltalk, Eiffel, Ruby, Python, OCAML, Clips, Visual Basic .NET, C#, Simula, etc.)

Para poder tomar una decisión correcta sobre que lenguaje de programación adoptar para comenzar a construir un nuevo sistema software, un buena medida puede consistir en basarse en el índice TIOBE, mostrada en la Figura 2.5, que calcula la popularidad de los lenguajes de programación en base al número de ingenieros, cursos, proveedores, etc.

| Position Dec 2010 | Position Dec 2009 | Delta in Position | Programming Language | Ratings Dec 2010 | Delta Dec 2009 | Status |
|---|---|---|---|---|---|---|
| 1 | 1 | = | Java | 17.999% | +0.94% | A |
| 2 | 2 | = | C | 16.076% | -0.21% | A |
| 3 | 4 | ↑ | C++ | 9.014% | -0.16% | A |
| 4 | 3 | ↓ | PHP | 7.511% | -2.26% | A |
| 5 | 6 | ↑ | C# | 6.687% | +0.43% | A |
| 6 | 7 | ↑ | Python | 6.482% | +1.30% | A |
| 7 | 5 | ↓↓ | (Visual) Basic | 5.118% | -2.66% | A |
| 8 | 13 | ↑↑↑↑↑ | Objective-C | 3.242% | +2.08% | A |

*Figura 2.5 – Índice TIOBE Diciembre 2010 [6]*

Existen otros criterios, más subjetivos, que permiten hacerse una idea de las tendencias de uso de los lenguajes actuales más populares. La Figura 2.6 corresponde con el resultado de un concurso de programación que organizó Google en 2010, en el que se muestra un gráfico de los lenguajes de programación más utilizados [7].

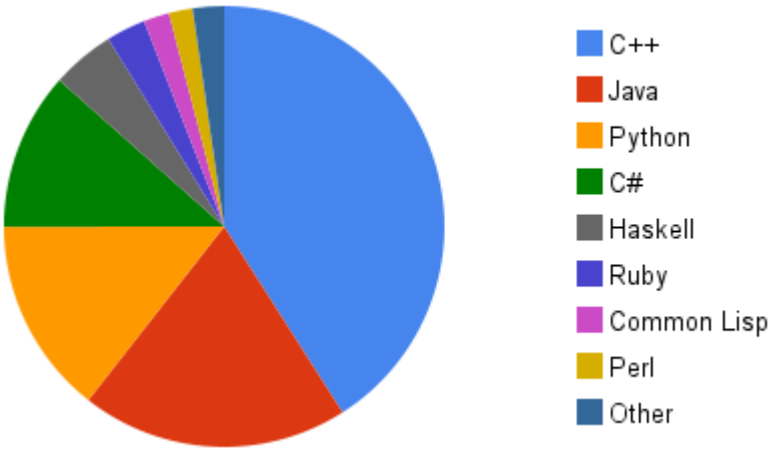


*Figura 2.6 – Lenguajes de programación del concurso Google AI Challenge*

Se puede tener también en cuenta los resultados proporcionados por el sitio web Langpop [8], Figura 2.7, sobre la popularidad de los lenguajes de programación. La popularidad de un lenguaje no se puede considerar como una buena medida, sino habría que intentar analizar que conceptos interesantes aportan estos lenguajes más populares y

porque han sido adoptados por la comunidad de programadores y en qué tipo de proyectos.

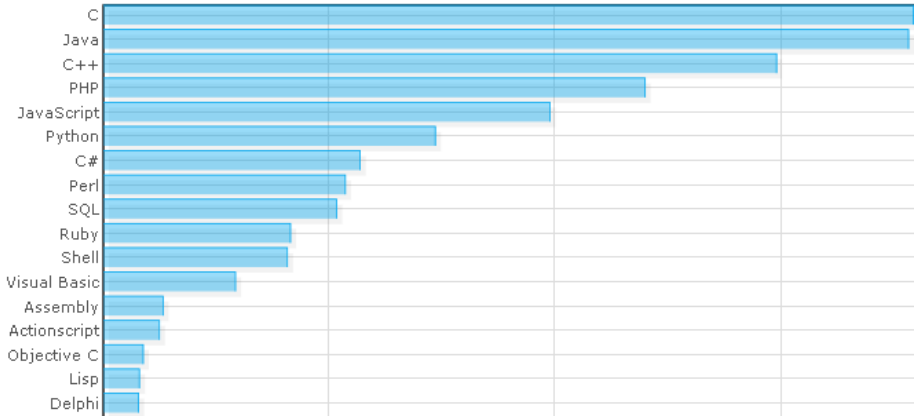


*Figura 2.7 – Gráficos sobre la popularidad de los lenguajes de programación*

Observando brevemente los diversos gráficos mostrados, descartando los lenguajes que no soportan el paradigma orientado a objetos y limitándose únicamente a lenguajes con fuerte presencia y uso. La lista de posibles lenguajes se puede reducir a:

- C#
- Python
- C++
- Java

En primer lugar y debido al requisito de construir software multiplataforma, se puede descartar C# ya que está limitado al sistema operativo Windows (aunque hay una implementación llamada Mono que es multiplataforma, Microsoft no da soporte oficial).

Una vez reducida la lista inicial, a tan solo tres lenguajes, se va a analizar las principales características de cada uno de ellos.

#### *2.1.3.2.1 Python*

Python fue creado a finales de los ochenta por Guido van Rossum en los Países Bajos como un sucesor del lenguaje de programación ABC [9].

Python es un lenguaje interpretado, es decir no requiere ser compilado ni enlazado, basta con tener el intérprete para ejecutar las aplicaciones. Se trata de un lenguaje fuertemente tipado (no se requiere que se declare los tipos de las variables), minimalista (todo lo que no sirve no se usa, evita el uso de las llaves, no requiere punto y coma al final de las sentencias, etc.), es multiplataforma y multiparadigma (más que forzar a los programadores a adoptar un estilo particular de programación, permite utilizar varios).

Python es muy utilizado como un lenguaje para crear aplicaciones de consola o scripts pero también se pueden construir aplicaciones visuales complejas. Otra de las gran virtudes es la cantidad de alternativas existentes para el desarrollo de aplicaciones web, existen servidores de aplicaciones complejos y ligeros, plantillas para el desarrollo web y la posibilidad de usar Python embebido en un documento html. No es recomendado para programación a bajo nivel (kernel, drivers, etc.).

Finalmente, destacar que Python está en un proceso continuo de desarrollo por una gran cantidad de desarrolladores y en los últimos años ha experimentado un importantísimo aumento del número de programadores y empresas que lo utilizan.

#### *2.1.3.2.2 C++*

C++ es un lenguaje de programación de propósito general desarrollado en los AT&T Bell Laboratories a principios de los 80 por Bjarne Stroustrup. Es una extensión del exitoso lenguaje de

programación C con la intención de incorporar mecanismos para la manipulación de objetos [10].

C++ es un lenguaje de programación extremadamente largo y complejo, permite la programación de bajo nivel y alto nivel. Ha sido tradicionalmente el lenguaje empleado para escribir código de alto rendimiento en aplicaciones científicas, gráficos 3D, etc. Además también permite realizar aplicaciones con interfaz gráfica que se integran de forma óptima y resultan muy portables entre distintos sistemas operativos.

Al igual que C, en el diseño de C++ primó sobre todo la velocidad de ejecución de código, se trata de un lenguaje compilado. Se citan a continuación diversas características del lenguaje de programación C++:

- Funciones miembro
- Plantillas
- Clases abstractas
- Espacios de nombres
- Excepciones
- Herencia múltiple
- Sobrecarga de operadores

Finalmente, destacar el extraordinario éxito, importancia y uso que C++ ha experimentado desde su creación hasta nuestros días. Una de las claves de su éxito es debido a ser un lenguaje de propósito general que se adapta a múltiples situaciones.

#### *2.1.3.2.3 Java*

Java es un lenguaje de programación orientado a objetos que fue creado por James Gosling, Patrick Naughton, Chis Warth, Ed Frank y Mike Sheridan en Sun Microsystem en 1991. Fundamentado en C++, el

lenguaje Java se diseño para ser pequeño, sencillo y portátil a través de plataformas y sistemas operativos, tanto a nivel de código fuente como binario, lo que significa que los programas Java, pueden ejecutarse en cualquier computadora que tenga instalada una máquina virtual de Java [11].

La sintaxis de Java se deriva en gran medida de C++. Pero a diferencia de éste, Java fue construido desde el principio para ser completamente orientado a objetos. Todo en Java es un objeto (salvo algunas excepciones) y todo en Java reside en alguna clase. Uno de los objetivos en su diseño era conseguir eliminar parte de la complejidad de C++ (eliminación de la aritmética de punteros, no hay herencia múltiple, administra automáticamente la memoria, dispone de un recolector de memoria, es robusto, etc.).

Todo esto, hace de Java un excelente lenguaje para programar aplicaciones (de escritorio y web) pero poco apropiado para hacer sistemas operativos o administradores de memoria.

A nivel académico Java es un lenguaje de programación utilizado en una gran parte de instituciones de estudios superiores y goza de gran popularidad y uso en la actualidad.

#### *2.1.3.2.4 Justificación del lenguaje de programación elegido*

La Tabla 2.3 muestra una comparativa realizada donde se detalla las características, ventajas, desventajas e implicaciones que supondrían elegir cada uno de los lenguajes de programación propuestos, teniendo en cuenta los requisitos del sistema software que se desea implementar.

| | Python | C++ | Java |
|---|---|---|---|
| **Uso previsto** | General, Scripting, Aplicación, Web | General, Aplicación | Aplicación, Web |
| **Paradigmas** | Orientado a objetos, Imperativo, funcional, reflectivo, orientado a aspectos | Orientado a objetos, Imperativo | Orientado a objetos, Imperativo, reflectivo |
| **Modelo de ejecución** | Interpretación | Compilación | Compilación e Interpretación |
| **Modelo de tipo de datos** | Dinámico y fuerte | Estático y débil | Estático y fuerte |
| **Influencias** | Perl, ABC | C, Simula, Algol 68 | C++ |
| **Introducción** | 1990 | 1979 | 1996 |
| **IDE** | ActivePython | Dev-C++ | Eclipse, NetBeans |
| **GUI** | Tkinter, WxPython, PyQT, PyGTK | wxWidgets | AWT, SWING |
| **Interés personal** | Alto | Medio | Alto |
| **Experiencia previa** | Baja | Media | Media |

*Tabla 2.3 – Comparativa de los lenguajes de programación*

Después de realizar esta comparación, considerando los requisitos del sistema y dando gran importancia a las preferencias personales, conocimientos previos, ayuda y soporte disponible, facilidad de desarrollo y de aprendizaje, etc. el lenguaje elegido para desarrollar el sistema software ha sido Java.

### 2.1.4 Gestor de Bases de Datos

El sistema software que se desea construir debe manejar y almacenar diversos datos de forma persistente, por lo que se hace necesario el uso de una base de datos.

Entre la base de datos física y los usuarios del sistema, se necesita lo que se denomina un Sistema de Gestión de Base de Datos (SGBD). En la actualidad existen numerosos sistemas gestores de base de datos y criterios para su clasificación [12].

Se pueden clasificar según el modelo de datos en que está basado el SGBD:

- Relacional (SGBDR): Representa a la base de datos como una colección de tablas.

- Orientado a Objetos: Define a la base de datos en términos de objetos, sus propiedades y sus operaciones.

- Objeto-relacional: Sistemas relacionales con características de los orientado a objetos.

- Jerárquico: Representa los datos como estructuras jerárquicas de árbol.

Se pueden clasificar dependiendo del número de usuarios a los que da servicio:

- Monousuario

- Multiusuario

También pueden clasificarse según el número de sitios en los que está distribuida la base de datos:

- Centralizado: La base de datos y el SGBD están almacenados en una sola computadora.

- Distribuido: La base de datos y el software SGBD están distribuidos en múltiples sitios conectados por una red.

A continuación y considerando que el mercado de gestores de bases de datos es bastante grande y ofrece demasiadas alternativas a la hora de

elegir un software en que confiar. Se revisarán como alternativas de gestores de bases de datos, una propuesta comercial como es Oracle y dos propuestas de software libre como son PostgreSQL y MySQL.

#### *2.1.4.1 Oracle*

Oracle es un sistema de gestión de bases de datos relacional desarrollado por Oracle Corporation. Es considerado como uno de los sistemas de base de datos más completos, destacando su soporte de transacciones, estabilidad, escalabilidad y soporte multiplataforma.

Oracle es el proveedor mundial líder de software para administración de información, la segunda empresa de software y dispone de una enorme cuota de uso en el mercado.

Oracle a partir de la versión 10g Release 2, cuenta con 6 ediciones:

- Oracle Database Enterprise Edition (EE)
- Oracle Database Standard Edition (SE)
- Oracle Database Standard Edition One (SE1)
- Oracle Database Express Edition (XE)
- Oracle Database Personal Edition (PE)
- Oracle Database Lite Edition (LE)

La única edición gratuita es la Express Edition, que es compatible con las demás ediciones de Oracle Database 10gR2 y Oracle Database 11g. Esta edición gratuita tiene ciertas limitaciones, sólo puede utilizar un procesador del servidor donde está instalada, un máximo de 1GB de RAM y tiene limitado el almacenamiento a 4GB de datos de usuario.

#### *2.1.4.2 PostgreSQL*

PostgreSQL es un sistema de gestión de base de datos relacional orientada a objetos y publicado bajo licencia libre. El desarrollo de

PostgreSQL no es manejado por una empresa y/o persona, sino que es dirigido por una comunidad de desarrolladores que trabajan de formas desinteresadas, altruistas, libres y/o apoyadas por organizaciones comerciales.

Algunas de sus principales características son, entre otras:

- Sistema de alta concurrencia
- Amplia variedad de tipos nativos
- Se pueden ejecutar funciones, escritas en distintos lenguajes, en el servidor.

Para complementar el motor de datos y las utilidades de PostgreSQL, se suele hacer uso de herramientas externas creadas especialmente para este motor.

#### *2.1.4.3 MySQL*

MySQL es un sistema de gestión de base de datos relacional, multihilo, multiusuario y con una gran presencia en el mercado. En enero de 2008 fue comprada por Sun Microsystems y ésta a su vez en abril de 2009 fue comprada pro Oracle Corporation.

MySQL es desarrollado como software libre en un esquema de licenciamiento dual, se ofrece bajo licencia libre pero para aquellas empresas que quieran incorporarlo en productos privativos deben comprar una licencia específica que les permita este uso. Además de la venta de licencias privativas, la compañía ofrece soporte, servicios y posee el copyright de la mayor parte del código.

Inicialmente MySQL carecía de elementos considerados esenciales en las bases de datos relacionales, tales como integridad referencial y transacciones. A pesar de ello, atrajo a los desarrolladores de páginas

web por su simplicidad. Poco a poco MySQL ha incorporado estos elementos y entre las características disponibles en las últimas versiones se puede destacar, entre otras:

- Disponibles en gran cantidad de plataformas y sistemas
- Procedimiento almacenados, disparadores, cursores, vistas actualizables
- Replicación
- Múltiples motores de almacenamiento
- Agrupación de transacciones de varias conexiones

Entre los productos y servicios ofrecidos por MySQL, destacan:

- MySQL Enterprise: incluye MySQL Enterprise Server, monitoreo de la red MySQL, servicios de consulta y soporte de producción.
- MySQL Cluster
- MySQL Drivers para JDBC, ODBC y .Net
- MySQL Tools: incluye MySQL Administrator, MySQL QueryBrowser y MySQL Migration Toolkit

Según cifras del fabricante, existen funcionando en la actualidad más de seis millones de copias de MySQL, lo que supondría en términos de uso ser la mayor herramienta de base de datos.

#### *2.1.4.4 Justificación SGBD elegido*

La Tabla 2.4 muestra una comparativa de las características de los SGBD descrita anteriormente.

| | Oracle | MySQL | PostGreSQL |
|---|---|---|---|
| **Disponibilidad** | Media | Alta | Alta |
| **Formación Administrador** | Alta y cara formación | Cursos más baratos | Cursos más baratos |
| **Licencia y coste** | Comercial | GNU GPL | BSD |
| **Seguridad y fiabilidad** | Alta | Media | Media |
| **Peso en memoria** | Alta | Baja | Baja |
| **Popularidad** | Alta | Media | Baja |
| **Volumen de datos** | Alta | Media | Media |
| **Seguridad y copias de seguridad** | Alta | Media | Media |
| **Ventajas** | Es multiplataforma | Es multiplataforma | Es multiplataforma |
| | Es fácil de usar | Es gratuito | Es gratuito |
| | Es muy usado a nivel mundial | No necesita muchos requerimientos del sistema | Diseñado para ambientes de alto volumen |
| | | Se instala de manera muy sencilla | |
| **Desventajas** | Tiene un elevado precio | No es muy intuitivo | No es muy utilizado |
| **Interés personal** | Alto | Alto | Medio |
| **Experiencia previa** | Media | Media | Baja |

*Tabla 2.4 – Comparativa SGBD*

Los productos Oracle de edición gratuita tienen limitaciones, generalmente de carácter físico y están diseñadas con el propósito de introducir en sus sistemas a desarrolladores, estudiantes, DBAs, formadores, etc. Para poder garantizar la escalabilidad del sistema software, evitar depender en un futuro de un proceso de actualización a otras versiones más completas (con coste asociado) y habiendo disponibles en el mercado otras opciones validas, se considera oportuno descartar este SGBD.

Entre las otras dos opciones disponibles (PostgreSQL y MySQL), y aunque ambas podrían ser adecuadas, se decide utilizar MySQL debido principalmente a un mayor interés personal y conocimientos previos.

En la decisión también ha influido que MySQL disponga de una gran comunidad, con gran soporte en foros y documentación en la web.

### 2.1.5 Metodología Empleada

En este apartado se explica la metodología utilizada para la realización del proyecto.

La metodología empleada debe escogerse dependiente de la naturaleza del proyecto, de las herramientas que se van a utilizar, y de sus requisitos. Los modelos de ciclo de vida más extendidos son el modelo en cascada, el prototipado, el formal y el de espiral [13].

Este último, el modelo en espiral, es el empleado a lo largo del proyecto. Consiste en la realización de una serie de fases en las que se incluyen en cada una de ellas una funcionalidad nueva, hasta conseguir la versión final deseada. Consta de pocas etapas, las cuales se van realizando de una manera continua y cíclica.

Uno de los aspectos más resaltantes de este modelo es la interacción que se produce alrededor de la espiral, desarrollando sucesivas versiones del programa cada vez más completas. La Figura 2.8 muestra las cuatro fases habituales de la metodología en espiral:

- Planificar
- Determinar objetivos
- Valorar y reducir riesgos
- Desarrollar y validar

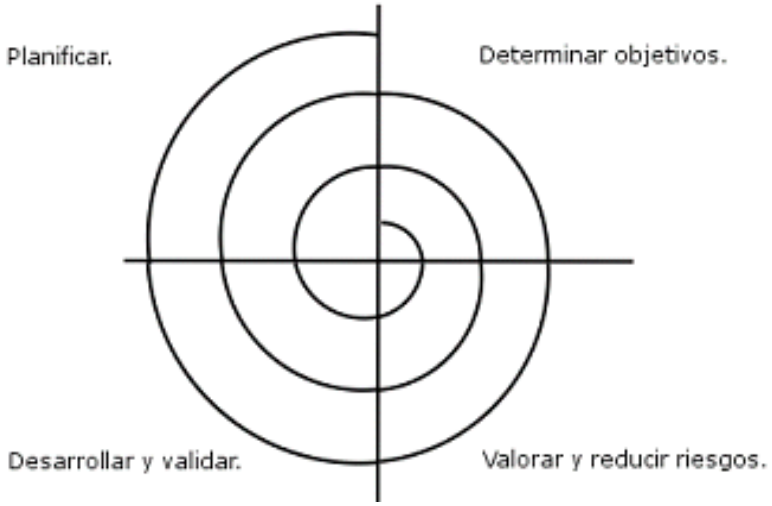


*Figura 2.8 – Modelo de desarrollo en espiral*

Distintas iteraciones que se han llevado a cabo durante el desarrollo del proyecto son:

- Implementar analizador expresiones matemáticas mediante JFlex y CUP.
- Implementar capa de datos mediante Hibernate.
- Implementar clases de negocio y presentación de las distintas pantallas de la aplicación (cursos, alumnos, profesores, etc.).

Cada una de las iteraciones tiene las mismas fases y cuando termina da un producto ampliado con respecto al ciclo anterior. Una diferencia importante, respecto a otros modelos, es que tiene en cuenta el concepto de riesgo. Un riesgo puede ser muchas cosas: requisitos no comprendidos, mal diseño, errores en la implementación, etc.

### 2.1.6 Plataforma de desarrollo

A continuación se describen los componentes del equipo empleado en el desarrollo del proyecto:

- Computadora
    - Hardware:

        - AMD Athlon II 2,10GHz
        - 4GB RAM
    - Sistema Operativo:
        - Windows 7
    - Software:
        - Eclipse 3.6: Entorno desarrollo integrado.
        - MySQL 5.1.52: Sistema gestión de base de datos relacional.
        - MySQL Workbench 5.2.30: Editor visual de base de datos MySQL.
        - Microsoft Word 2007: Procesador texto.
        - Dia 0.97: Editor de diagramas.

## 2.2 Historia Agentes Conversacionales Pedagógicos

Durante la década de 1990 se pone de moda una nueva palabra: el término "agente software" aparece en infinidad de campos como algo novedoso. No obstante los agentes software llevaban siendo un tema de investigación desde la década de 1970. También durante la década de 1990 llegan al gran público los interfaces gráficos de usuario. Surgiendo así lo que se da por llamar "agentes de interfaz" que pretenden convertir los ordenadores en asistentes. Sumado a que la mayor parte de la gente trata a los ordenadores como si fueran personas incluso aunque el interfaz no nos incite a ello. Por tanto, una mejora sustancial es aprovecharse de este hecho y personificarlos en "actores virtuales" que se muestren en el interfaz, y que reaccionan a las acciones del usuario [14].

El agente de interfaz más conocido es el clip de office, Figura 2.9, surgió por primera vez en Microsoft Office 97. Su objetivo era crear asistentes informáticos que tuvieran una representación en pantalla, y se comportaran como personas, tanto por la presencia visual como por el diálogo hablado.

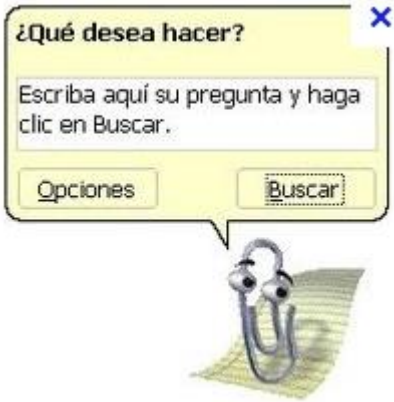


*Figura 2.9 – Clip Office*

Bajo este contexto surgen los "agentes pedagógicos conversacionales" especializados en una tarea concreta: ayudar al estudiante durante su aprendizaje en un entorno interactivo. Los primeros ACP eran muy limitados, sólo mostraban información, sin permitir interacción. En las siguientes décadas, se mejoró la animación, interacción y el contenido de la conversación.

Existen multitud de ejemplos [14]:

- **Herman the Bug**: Diseñado para enseñar conceptos sobre anatomía y fisiología botánica.
- **Cosmo**: El objetivo principal era el aumento de la credibilidad de este tipo de agentes.
- **Steve**: Agente animado diseñado específicamente para ayudar a uno o varios estudiantes a llevar a cabo labores físicas que sigan un procedimiento determinado.
- **Guilly**: Desarrollado para enseñar a niños a separar la basura.
- **Sam**: Diseñado para desarrollar la capacidad narrativa de niños en edad escolar.
- **Baldi**: Ejerce como tutor para entrenamiento del lenguaje inglés en niños con déficit auditivo o para adultos en su aprendizaje como una segunda lengua.
- **WhizLow**: Cuyo objetivo es servir para la enseñanza de la arquitectura de los computadores.

- **MRE**: Persigue la creación de un entorno de aprendizaje en el que los usuarios se encuentren inmersos en un mar de imágenes, sonidos y circunstancias lo más parecido al mundo real.
- **Jacob**: Agente pedagógico con capacidad de procesamiento de lenguaje natural que se ha utilizado para enseñar a resolver el problema de las torres de Hanoi.
- **Adri**: Sistema desarrollado para ayudar a aprender a tocar el piano.
- **Lahystotrain**: Proyecto desarrollado para entrenar a cirujanos en operaciones de laparoscopia e histeroscopia.

# 3 Descripción Informática

## 3.1 Análisis de Requisitos

Se procede a establecer una lista de los requisitos que debe cumplir el software, tanto a nivel funcional como no funcional. En nuestro caso, teniendo en cuenta que se trata de un proyecto fin de carrera, la toma de requisitos se ha basado principalmente en una reunión inicial llevada a cabo entre el alumno y la tutora.

El listado de los requisitos funcionales es:

- RF1: Al iniciar el sistema, se mostrará una pantalla de autentificación.
- RF2: Un alumno se podrá autentificar con un nombre de usuario y contraseña
- RF3: Si un alumno se autentifica correctamente, se accederá a una pantalla de inicio.
- RF4: Si un alumno se autentifica incorrectamente, se mostrará un mensaje de error.
- RF5: Un profesor se podrá autentificar con un nombre de usuario y contraseña
- RF6: Si un profesor se autentifica correctamente, se accederá a una pantalla de inicio.
- RF7: Si un profesor se autentifica incorrectamente, se mostrará un mensaje de error.
- RF8: La pantalla de inicio del alumno contendrá un menú con las siguientes opciones: modificar datos personales y salir.
- RF9: La pantalla de inicio del profesor contendrá un menú con las siguientes opciones: modificar datos personales, problemas y alumnos.
- RF10: En la pantalla de modificar datos personales del alumno, se podrán modificar los datos y guardar los cambios.

- RF11: En la pantalla de modificar datos personales del profesor, se podrán modificar los datos y guardar los cambios.
- RF12: La información mínima de cada alumno es: nombre de usuario, contraseña, nombre, apellidos, curso, edad.
- RF13: La información mínima de cada profesor es: nombre de usuario, contraseña, nombre, apellidos.
- RF14: Al seleccionar la opción de salir del alumno, se cerrará la aplicación.
- RF15: Al seleccionar la opción de salir del profesor, se cerrará la aplicación.
- RF16: En el apartado alumnos del menú profesor, se visualizará el listado de los alumnos.
- RF17: En la pantalla alumnos del menú profesor, se podrá añadir, modificar o eliminar alumnos.
- RF18: En la pantalla problemas del menú profesor, se visualizará el listado de los problemas.
- RF19: En la pantalla problemas del menú profesor, se podrá añadir, modificar o eliminar problemas.
- RF20: En la pantalla problema del menú profesor, se especificará el enunciado, los parámetros del problema y las instrucciones necesarias para solucionar el problema.
- RF21: En la pantalla problemas del menú alumno, se visualizará un histórico de los problemas resueltos por el alumno.
- RF22: En la pantalla problemas del menú alumno, se habilitará la opción de resolver un nuevo problema.
- RF23: En la pantalla problema del menú alumno, se mostrará el enunciado de un problema acorde a su curso, edad y su historial de problemas resueltos correctamente y erróneos.
- RF24: Al mostrar el enunciado de un problema en el apartado del menú alumno, el sistema solicitará la siguiente instrucción necesaria para resolver el problema.
- RF25: Al mostrar el enunciado de un problema en el apartado del menú alumno, el usuario podrá introducir el resultado final del problema.

- RF26: Al mostrar el enunciado de un problema en el apartado del menú alumno, el usuario podrá introducir la siguiente instrucción necesaria para resolver el problema.
- RF27: Al mostrar el enunciado de un problema en el apartado del menú alumno, el sistema indicará ayudas convenientes para que el usuario pueda resolver la siguiente instrucción.
- RF28: Si la instrucción introducida por el alumno en un problema es correcta, el sistema le solicitará la siguiente instrucción necesaria.
- RF29: Si la instrucción introducida por el alumno en un problema es errónea el sistema le mostrará consejos.

El listado de los requisitos no funcionales es:

- RNF1: El sistema debe ser multiplataforma.
- RNF2: El sistema debe tener una interfaz gráfica agradable e intuitiva.
- RNF3: El sistema debe responder en tiempo real, no produciéndose bloqueos ni pérdidas de tiempo innecesarias.

## 3.2 Casos de Uso

Los casos de uso son una secuencia de transacciones que son desarrollados por un sistema en respuesta a un evento que inicia un actor sobre el propio sistema.

Los diagramas de casos de uso se utilizan para ilustrar los requerimientos del sistema al mostrar cómo reacciona un sistema a eventos que se producen en el mismo. En este tipo de diagrama intervienen algunos conceptos nuevos:

- Actor: Es una entidad externa al sistema que se modela y que puede interactuar con él, un ejemplo de actor podría ser un usuario o cualquier otro sistema.
- Caso de Uso: Es una operación/tarea específica que se realiza tras una orden de algún agente externo, sea desde una petición de un actor o bien desde la invocación desde otro caso de uso.
- Asociación: Es el tipo de relación más básica que indica la invocación desde un actor o caso de uso a otra operación.
- Dependencia o Instanciación: Es una relación entre clases muy particular, en la cual una clase depende de otra.
- Generalización: Este tipo de relación está orientado para casos de uso y puede ser de "Uso" (se utiliza cuando se tiene un conjunto de características que son similares en más de un caso de uso) o de "Herencia" (se recomienda utilizar cuando un caso de uso es similar a otro).

En las Figuras 3.1 y 3.2 se presentan los diagramas de casos de usos Alumno y Profesor, respectivamente. Los diagramas usados son bastantes representativos, se muestran el actor y los posibles casos de usos accesibles.

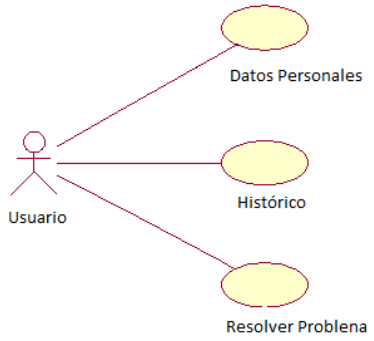


*Figura 3.1 – Caso de Uso Alumno*

El actor "Usuario" puede realizar los casos de uso: Datos Personales, Histórico y "Resolver Problema".

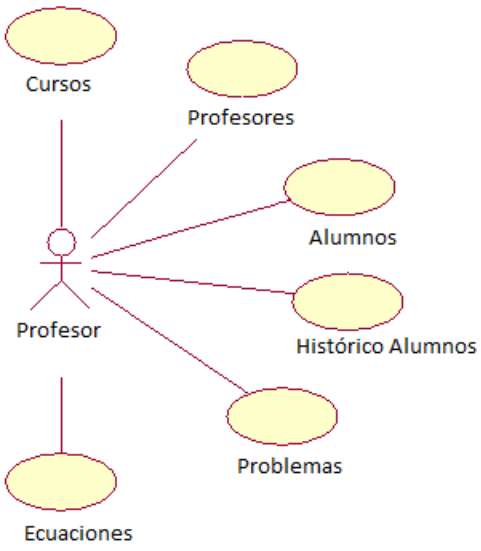


*Figura 3.2 – Caso de Uso Profesor*

El actor "Profesor" puede realizar los casos de uso: Curso, Profesores, Alumnos, Histórico Alumnos, Problemas y Ecuaciones.

## 3.3 Diseño Arquitectura del Sistema

La aplicación está implementada siguiendo una arquitectura de tres capas. Una capa de presentación que interactúa con el usuario, una capa de negocio que contiene la lógica del sistema y una capa de datos que se conecta con la base de datos ubicada en el servidor.

En la Figura 3.3 se muestra una primera visión global de los componentes del sistema, más adelante, se detallará con mayor profundidad los componentes del sistema.

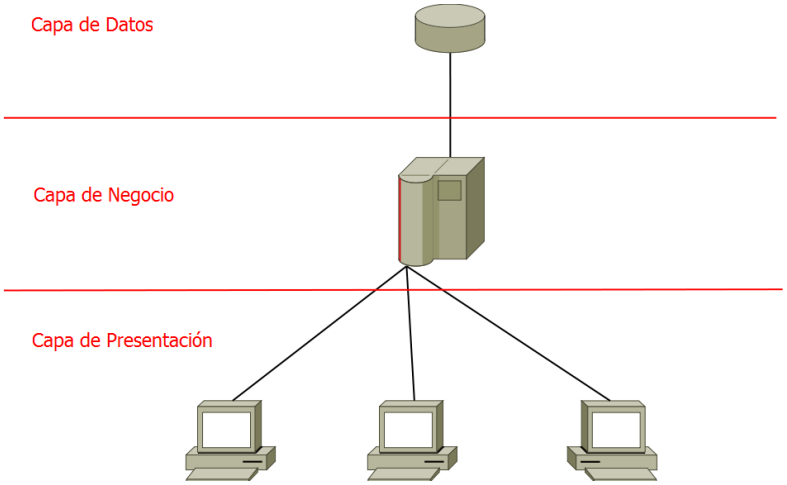


*Figura 3.3 – Componentes del sistema*

La fase de desarrollo y pruebas de este sistema se ha utilizado la misma computadora como servidor y como cliente.

### 3.3.1 Servidor

La computadora empleada en la fase de desarrollo tiene instalada la última versión estable de MySQL disponible en el momento del inicio del desarrollo del sistema, la versión 5.5.19 y se ejecuta sobre un sistema operativo Windows 7. Además, se ha optado por una instalación tipo "Developer Machine", para utilizar únicamente una pequeña parte de la memoria disponible de la computadora de desarrollo. En una puesta en producción del sistema, aunque no hay una configuración mínima necesaria para instalar MySQL [15] y se puede instalar en diversas plataformas, se aconseja disponer memoria, alta velocidad del microprocesador, buena velocidad de los discos duros y una instalación dedicada en un servidor, para garantizar un alto rendimiento del servidor MySQL.

Otra posibilidad, que queda fuera del alcance del proyecto debido a que el número de usuarios de la aplicación y el tamaño de la base de datos no justifica su necesidad, es la de disponer de varios servidores para lograr una base de datos distribuida. La carga actual y previsible

de este sistema no hace necesario contemplar estas posibilidades. En el anexo A se adjunta el manual de instalación MySQL.

### 3.3.2 Cliente

La computadora empleada en la fase de desarrollo tiene instalado el sistema operativo Windows 7 y la última versión estable de la máquina virtual de Java.

En una posible puesta en producción del sistema, se debería distribuir la aplicación (archivo con extensión JAR) en cada uno de las computadores donde se desea ejecutar la aplicación. Estas computadoras deberían tener conectividad con el servidor y tener instalada la máquina virtual de Java correspondiente.

## *3.4 Diseño Base de Datos*

Las bases de datos son componentes esenciales de los sistemas de información. Para el diseño de cualquier base de datos se debe optar por emplear algún tipo de modelo que describa los datos almacenados en ellas. Un modelo de datos es una colección de herramientas conceptuales que describen el tipo de datos, su semántica, las relaciones entre ellos y las restricciones que puedan tener. En este trabajo se utiliza uno de los modelos lógicos de datos más extendido llamado modelo entidad relación (ER).

El modelo ER es relativamente simple y fácil de emplear, posee una gran capacidad expresiva. Las técnicas que este modelo emplea sirven, básicamente, para especificar las necesidades de información de una aplicación y nos ofrecen, además, unos fundamentos sólidos sobre los que desarrollar un sistema apropiado y de alta calidad que soporte todo el peso de la información que se necesita, así como su posterior explotación [16].

En su forma más simple, la modelización semántica con esta técnica supone la identificación de los objetos de interés de nuestro sistema (que se denominan entidades), las propiedades relevantes de dichos objetos (que se denominan atributos) y cómo éstos se relacionan entre sí (relaciones o conexiones).

El modelo ER es extensamente usado durante el análisis de requerimientos. Debido a su simplicidad, es más fácil de entender por individuos no técnicos. Pruebas en el ambiente del mundo real han mostrado que es una efectiva herramienta de comunicación entre diseñadores de base de datos y usuarios finales. En este trabajo se ha realizado un modelo ER inicial, Figura 3.4, basándonos principalmente en la toma de requisitos. En este boceto se obtienen las principales entidades y relaciones que tendrán definitivamente la aplicación.

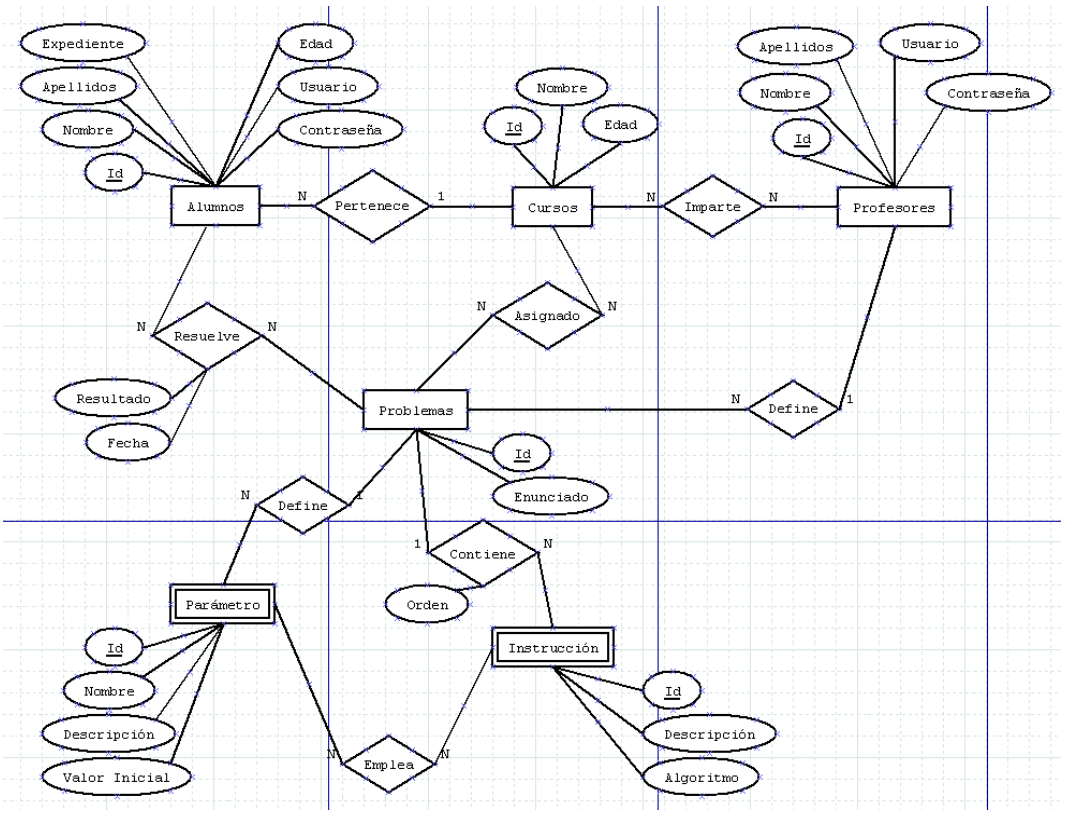


*Figura 3.4 – Modelo Entidad-Relación*

La entidad “Alumno”, Figura 3.5, en el que se detalla la información que sea guardar para cada alumno (expediente, apellidos, nombre, id, edad, usuario, contraseña). La relación “Pertenece”, en el que se estable que cada alumno pertenece a un solo curso y un curso está formado por N alumnos. La relación “Resuelve” en la que se guarda la fecha y el resultado y establece que un alumno puede resolver N problemas y a su vez un problema puede ser resuelto por N alumnos.

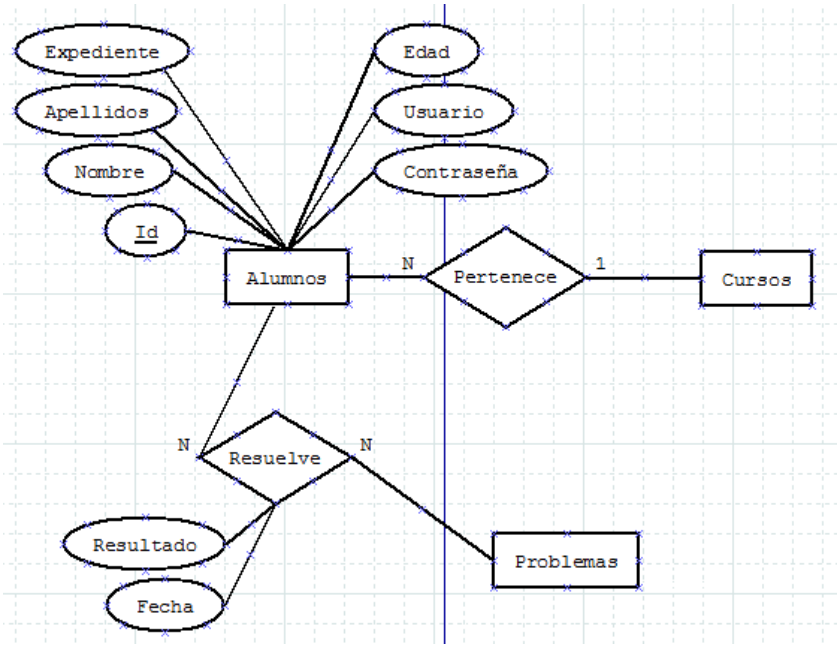


*Figura 3.5 – Entidad Alumno*

La entidad Profesor, Figura 3.6, en el que se detalla la información que se desea guardar para cada profesor (apellidos, nombre, id, usuario, contraseña). La relación “Imparte”, en el que se estable que cada profesor imparte un determinado curso y un curso puede ser impartido por N profesores. La relación “Define” que significa que un profesor puede generar N problemas y un problema es generado por un profesor.

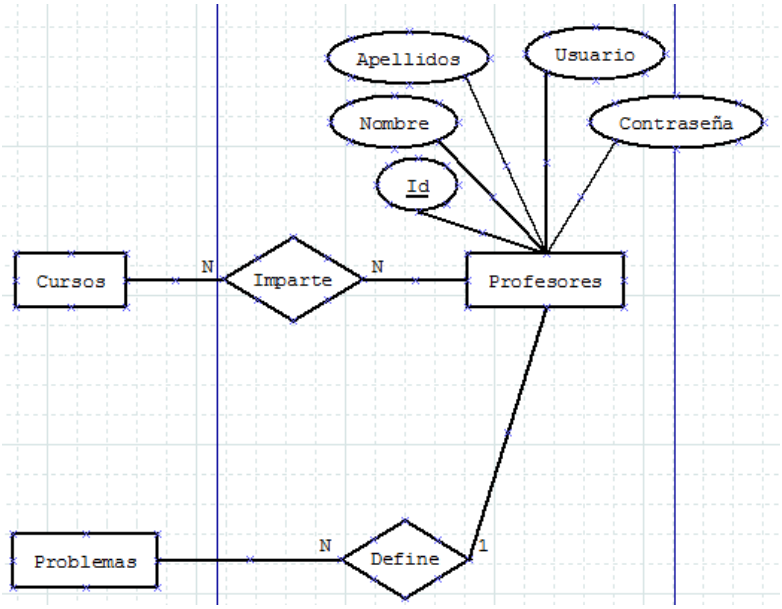


*Figura 3.6 – Entidad Profesor*

La entidad Curso, Figura 3.7, en el que se detalla la información que se desea guardar para cada curso (id, nombre, edad). Y se define además de las relaciones "Pertenece" y "Imparte" explicadas anteriormente la relación "Asignado" que especifica que a cada Curso se le asigna N problemas y un determinado Problema puede ser asignado a N Cursos.

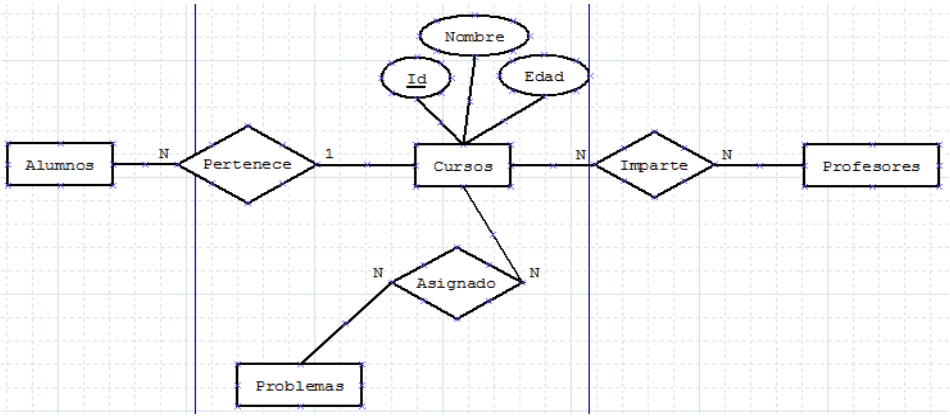


*Figura 3.7 – Entidad Curso*

Por último, una primera aproximación de la entidad "Problema", Figura 3.8, en el que se detalla la información que se desea guardar para cada problema (id, enunciado). Otras dos entidades débiles que dependen de la entidad "Problema", la entidad "Parámetro" y

“Instrucción”. Para cada problema se define N parámetros (relación “Define”) y un problema contiene N instrucciones en un determinado Orden (relación “Contiene”). Las instrucciones emplean N parámetros y un parámetro puede estar en N instrucciones (relación “Emplea”).

La información que se desea guardar para cada parámetro (id, nombre, descripción, valor inicial) y para cada instrucción (id, descripción, algoritmo).

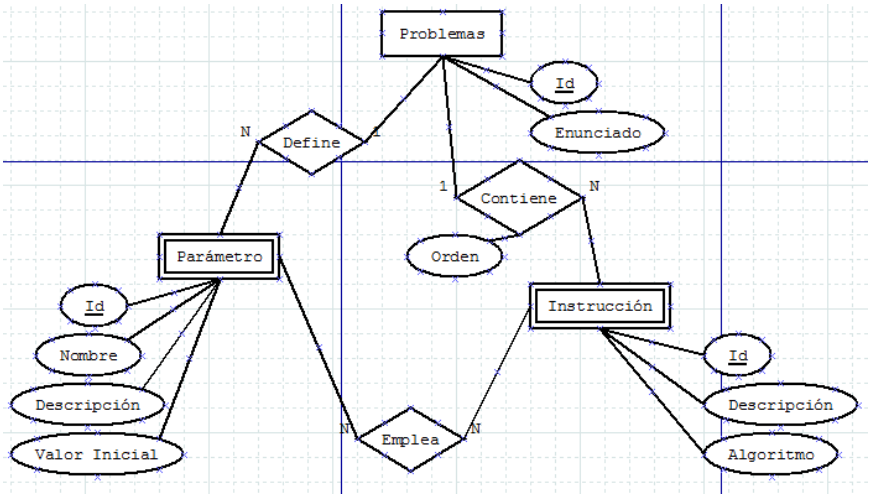


*Figura 3.8 – Entidad Problema*

A continuación, partiendo del resultado de la etapa de diseño conceptual expresado mediante el modelo ER, se transforma a un modelo relacional, aplicando una serie de reglas de transformación.

El modelo relacional ha sido, y es todavía, una pieza clave dentro del ámbito de las bases de datos. La gran mayoría de las base de datos existentes se ajustan en mayor o menor medida al modelo relacional. Además, la tendencia es que esta situación continúe, ya que los nuevos modelos de datos, tales como orientado a objetos, se plantean como extensiones o se basan en el modelo relacional. Fue desarrollado por E.F. Codd en 1970 y no tardó mucho tiempo en consolidarse como un nuevo paradigma en los modelos de base de datos [16].

Se utilizada MySQL Workbench como herramienta visual de diseño de base de datos. Los motivos principales por los que se ha utilizado esta herramienta son:

- Permite crear diagramas para un modelo relacional extendido (ERM, por sus siglas en inglés)
- Genera de forma automática, a partir del diagrama ERM, todas las sentencias de creación de tablas e índices que pueden ser ejecutadas en la base de datos MySQL.
- La simbología usada para los diagramas ERM es la que emplean la mayoría de las herramientas CASE en la actualidad.
- Su licencia es GPL y está disponible para varias plataformas.

El modelo relacional debe responder al reto de la orientación a objetos con la extensión del alcance del modelo relacional. El resultado de los esfuerzos, por lo general, es conocido como Modelo relacional extendido (ERM, por sus siglas en inglés). Se extiende el modelo relacional mediante procedimientos, objetos, versiones y otras nuevas capacidades. En el presente trabajo se ha decidido crear el diagrama ERM a través de la herramienta MySQL Workbench

En el ERM definitivo de la aplicación, Figura 3.9, se pueden observar las entidades relevantes así como sus interrelaciones, Figura 3.10 y propiedades.

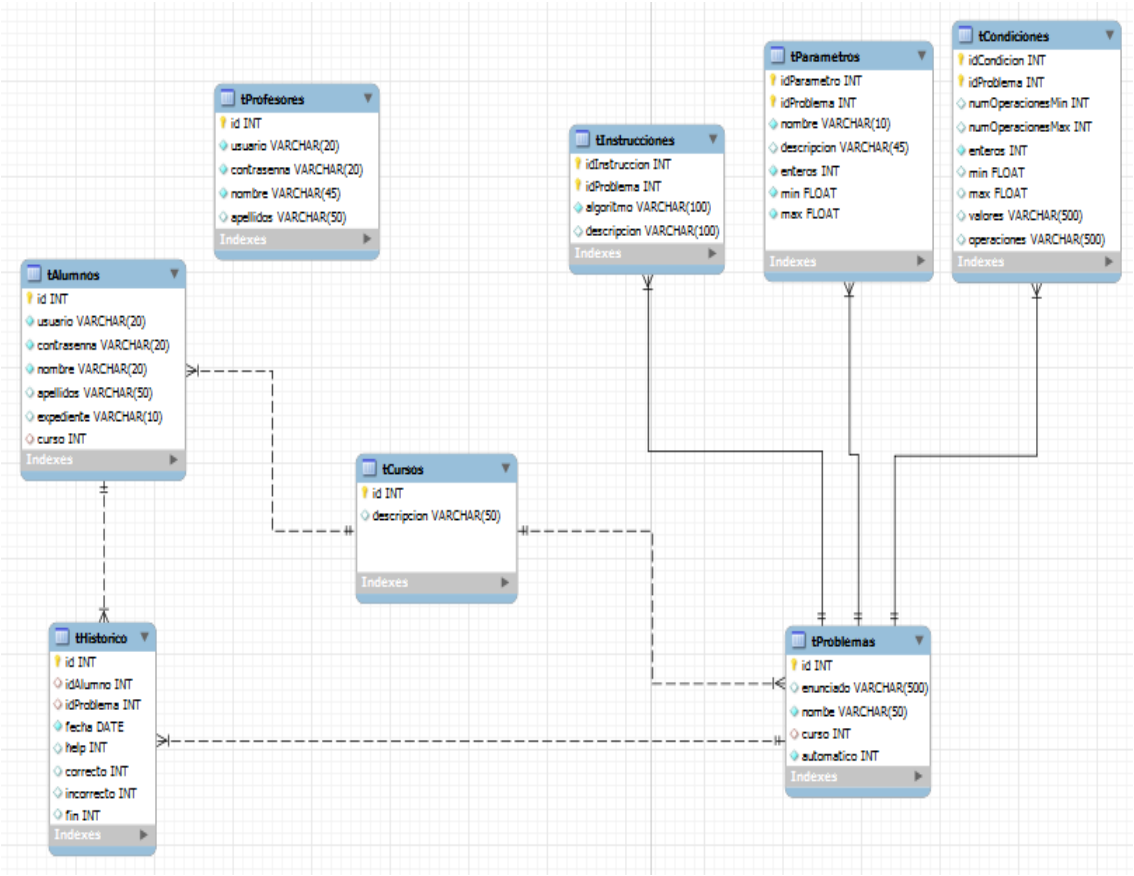


*Figura 3.9 – ERM detalle*

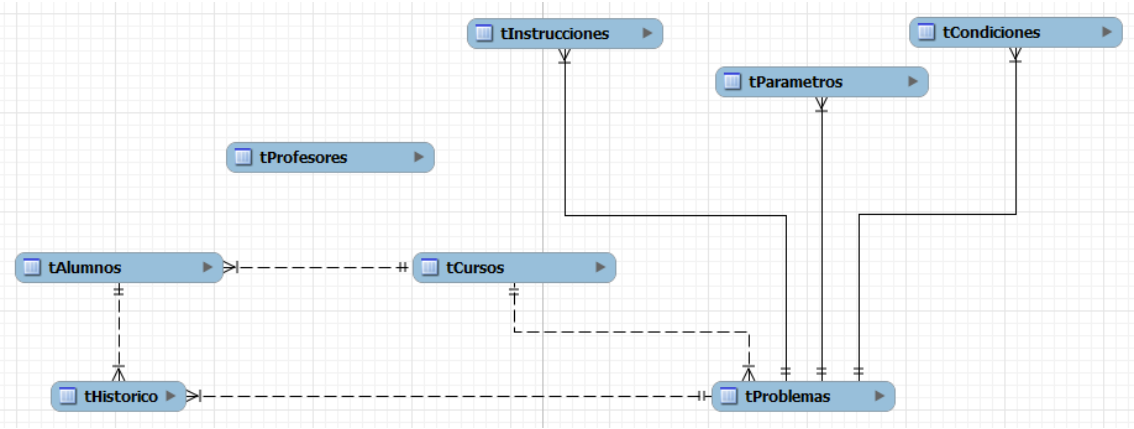


*Figura 3.10 – ERM relaciones*

Las figuras siguientes muestran cada una de las entidades creadas, Figuras 3.11, 3.12, 3.13, 3.14, 3.15, 3.16, 3.17 y 3.18.

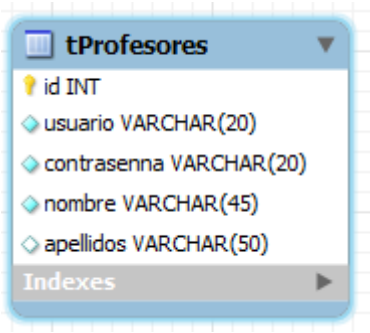


*Figura 3.11 – tProfesores*

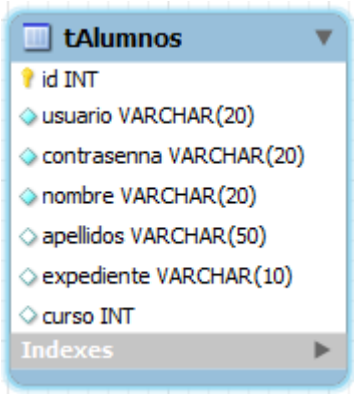


*Figura 3.12 – tAlumnos*

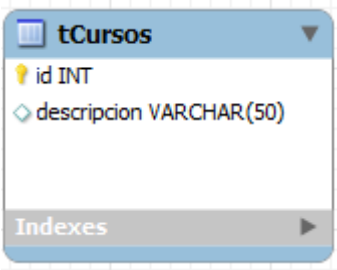


*Figura 3.13 – tCursos*

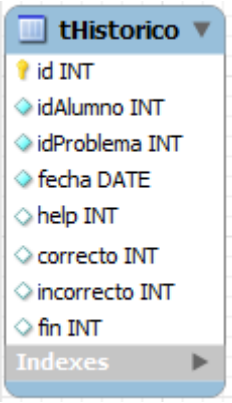


*Figura 3.14 – tHistorico*

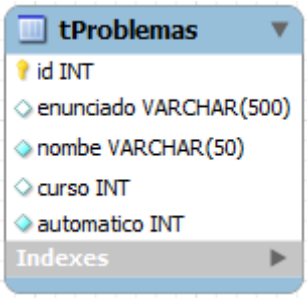


*Figura 3.15 – tProblemas*

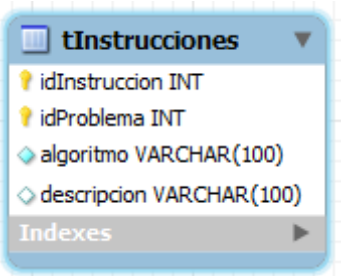


*Figura 3.16 – tInstrucciones*

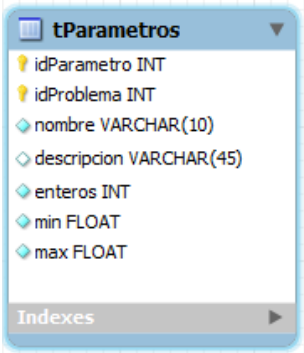


*Figura 3.17 – tParametros*

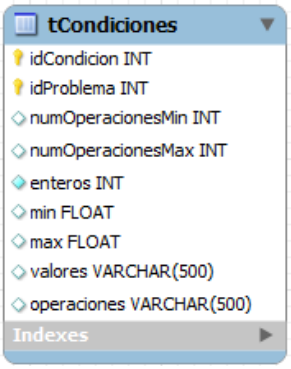


*Figura 3.18 – tCondiciones*

En la Figura 3.19 se muestra otra forma distinta de representar el modelo relacional estático.

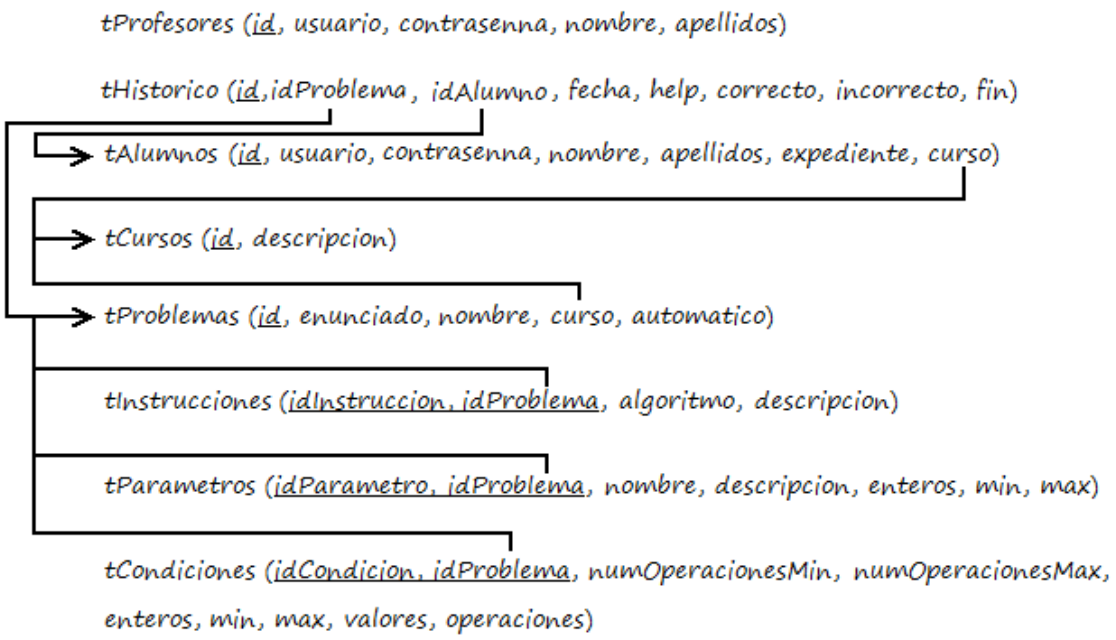


*Figura 3.19 – Modelo relacional estático*

Finalmente, se muestran en las Figuras 3.20, 3.21, 3.22, 3.23, 3.24, 3.25, 3.26 y 3.27 el contenido parcial de las tablas que componen la base de datos del proyecto.

| id * | usuario * | contrasenna * | nombre * | apellidos | expediente | curso |
|---|---|---|---|---|---|---|
| 1 | dani | 1 | Daniel | Sanchez Martin | 28921 | 1 |

*Figura 3.20 – Tabla tAlumnos*

| id * | usuario * | contrasenna * | nombre * | apellidos |
|---|---|---|---|---|
| 1 | admin | 123 | admin | |

*Figura 3.21 – Tabla tProfesores*

| id * | descripcion |
|---|---|
| 1 | 1 Primaria |
| 2 | 2 Primaria |
| 3 | 3 Primaria |
| 4 | 4 Primaria |

*Figura 3.22 – Tabla tCursos*

| id * | enunciado | nombe * | curso | automatico * |
|---|---|---|---|---|
| 1 | Adela tenía algun dinero ahorrado en la hucha. Por su cumpleaños, sus abuelos | Dinero hucha | 1 | 0 |
| 10 | Repaso de la tabla de multiplicar | Tabla de multiplicar | 1 | 1 |

*Figura 3.23 – Tabla tProblemas*

| idInstruccion * | idProblema * | algoritmo * | descripcion |
|---|---|---|---|
| 1 | 1 | Z = Y - X; | |

*Figura 3.24 – Tabla tInstrucciones*

| idParametro * | idProblema * | nombre * | descripcion | enteros * | min * | max * |
|---|---|---|---|---|---|---|
| 1 | 1 | X | | 1 | 10 | 20 |
| 2 | 1 | Y | | 1 | 20 | 40 |

*Figura 3.25 – Tabla tParametros*

| idCondicion * | idProblema * | numOperacionesMin | numOperacionesMax | enteros * | min | max | valores | operaciones |
|---|---|---|---|---|---|---|---|---|
| 3 | 10 | 1 | 1 | 1 | -1 | -1 | 0123456789 | * |

*Figura 3.26 – Tabla tCondiciones*

| id * | idAlumno * | idProblema * | fecha * | help | correcto | incorrecto | fin |
|---|---|---|---|---|---|---|---|
| 4 | 1 | 1 | 16/01/2012 0:00:00 | 0 | 0 | 0 | 0 |
| 5 | 1 | 1 | 16/01/2012 0:00:00 | 1 | 1 | 0 | 1 |
| 7 | 1 | 1 | 16/01/2012 0:00:00 | 1 | 1 | 0 | 1 |
| 8 | 1 | 1 | 16/01/2012 0:00:00 | 1 | 1 | 0 | 1 |
| 843 | 1 | 10 | 05/02/2012 0:00:00 | 0 | 0 | 0 | 0 |
| 844 | 1 | 10 | 05/02/2012 0:00:00 | 1 | 1 | 0 | 1 |

*Figura 3.27 – Tabla tHistorico*

## 3.5 Aplicación Java

En este apartado definimos, a grandes rasgos, la arquitectura a alto nivel del aplicativo Java que se desea construir, sus módulos y los frameworks utilizados.

Se ha implementado siguiendo una arquitectura de tres capas (capa presentación, negocio y datos). La ventaja principal de este estilo, es que el desarrollo se puede llevar a cabo en varios niveles y en caso de algún cambio sólo se ataca al nivel requerido sin tener que revisar el

restante. Además permite distribuir el trabajo de una aplicación, simplemente es necesario conocer la API que existe entre niveles. En dichas arquitecturas a cada nivel se le confía una misión simple, lo que permite el diseño de arquitecturas escalables (que pueden ampliarse con facilidad en caso de que las necesidades aumenten).

La Figura 3.28, muestra en un alto nivel, los componentes que forman cada capa y sus relaciones.

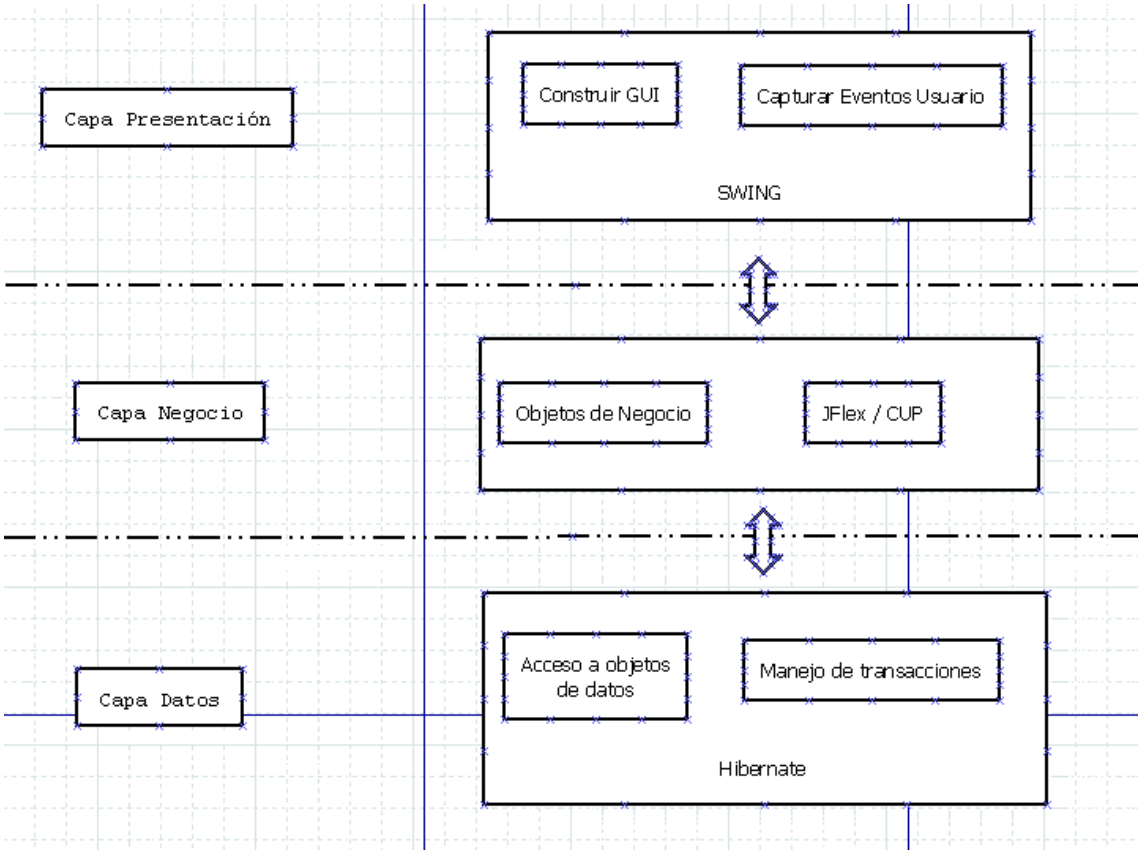


*Figura 3.28 – Arquitectura Aplicación Java*

La aplicación se va a implementar utilizando el IDE Eclipse, versión Helios. Todo el código fuente de la aplicación se va a desarrollar en un único proyecto de Eclipse, en la Figura 3.29 se muestra los distintos paquetes creados en el proyecto.

Desde el punto de vista de la codificación, se va a seguir unas pautas de codificación básicas para intentar conseguir un código fuente fácil de leer y comprender:

- El nombre de las clases debe ser simple y descriptivo.
- Si el nombre de la clase es compuesto, la primera letra de cada palabra en mayúscula.
- Si el nombre del método es compuesto, la primera letra de la segunda palabra en mayúscula.
- El nombre de las variables se escriben en minúsculas.
- Si el nombre de la variable es compuesto, la primera letra de la segunda palabra en mayúscula.
- Una sola declaración de variables por línea.
- Declaración de variables al inicio de cada bloque.
- El nombre de las constantes se escriben en mayúsculas.
- Espaciado e indentación de instrucciones lógico y que facilite la lectura de código.
- Evitar instrucciones largas.
- Asignar nombres significativos y autoexplicativos a los identificadores y funciones.
- Uso de subrutinas en vez de repetitivas llamadas a funciones comunes.
- Evitar niveles de anidamiento profundo.

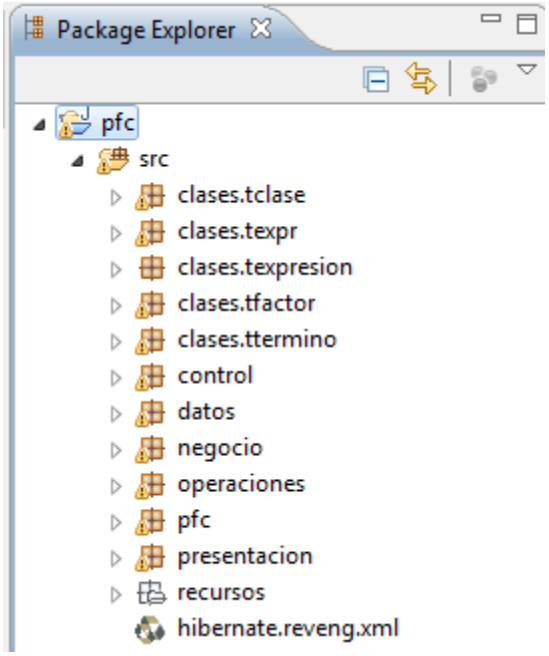


*Figura 3.29 – Explorador de paquetes*

### 3.5.1 Capa Datos

Uno de los principales requerimientos que debemos resolver en el desarrollo de una aplicación es la integración con una base de datos para poder guardar, actualizar y recuperar la información. Si bien, existen distintas alternativas para conseguirlo, se va a proceder a implementar, la capa de datos, mediante el uso del framework Hibernate.

Hibernate es una herramienta de Mapeo objeto-relacional (ORM) para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional y el modelo de objetos de la aplicación, mediante archivos declarativos (XML) que permiten establecer estas relaciones.

Los costes de aprendizaje de este framework en términos de tiempo se ven recompensados desde el primer día con el ahorro en tiempo de desarrollo y depuración, además de que nos permite centrar nuestros esfuerzos en desarrollar la funcionalidad de la aplicación.

En el proyecto Eclipse se ha creado un paquete llamado “datos” en el que se disponen de los archivos javabean generados automáticamente mediante Hibernate. Gracias a Hibernate y a su fácil integración con Eclipse, es posible generar fácil y rápidamente dichos archivos ante cualquier cambio en la estructura de la base de datos. La Figura 3.30 muestra una captura de pantalla de la clase javabean generada automáticamente y que hace referencia a la entidad Talumnos.

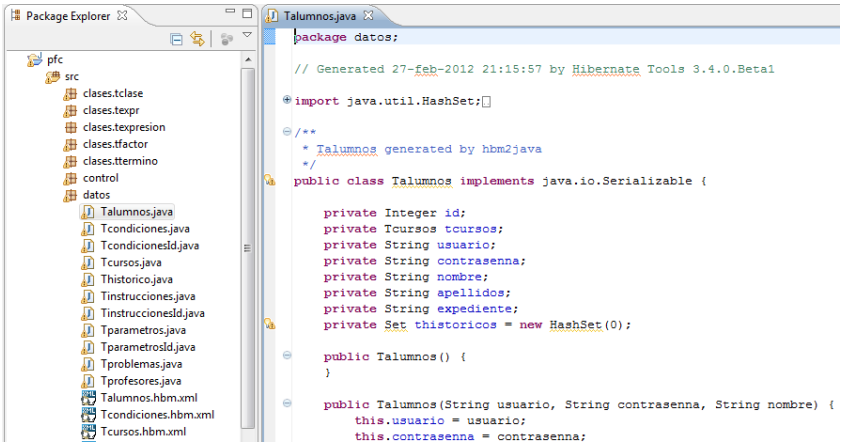


*Figura 3.30 – Clase Talumnos generada*

### 3.5.2 Capa Negocio

En esta capa se modelan las reglas del negocio, es decir, se establecen todos los procesos necesarios para poder responder ante una petición del usuario. Esta capa se sirve de las capa de datos para implementar las funcionalidades requeridas.

Esta capa se va a desarrollar, principalmente, utilizando los paquetes estándares que proporciona el lenguaje Java. El paquete “negocio” contiene las principales clases creadas con la lógica de la aplicación.

Gran parte de la lógica de la aplicación está relacionado con expresiones matemáticas, por lo que se hace necesario desarrollar funcionalidad capaz de realizar un análisis léxico, sintáctico y semántico de expresiones matemáticas. Para ello, vamos a utilizar

JFlex (analizador léxico para Java) y CUP (analizador sintáctico/semántico LALR para Java).

El agente conversacional debe poder analizar una expresión matemática, como el que se muestra en la Figura 3.31, y realizar los siguientes cálculos para poder interactuar correctamente con el usuario:

- Calcular el resultado de la expresión.
- Calcular una sugerencia para indicar al usuario la siguiente operación básica a realizar.

```
X= 3 + 2 * ( 4 / 1 );
```

*Figura 3.31 – Ejemplo expresión matemática*

En el ejemplo anterior, el valor final de X es 11 y la siguiente operación básica a realizar es: 4 / 1.

Las operaciones básicas que han sido consideradas en la aplicación son la suma, la resta, la multiplicación, la división y el paréntesis. Es decir, la aplicación está principalmente dirigida a niños en edad escolar.

Se va a detallar paso a paso los procesos necesarios que tiene que realizar nuestra aplicación. El primer paso es realizar un análisis léxico y comprobar los “tokens” validos. La Figura 3.32 muestra los tokens de nuestro ejemplo.

| | |
|---|---|
| X | Nombre |
| = | Igual |
| 3 | Número |
| + | Suma |
| 2 | Número |
| * | Multiplicar |
| ( | Paréntesis izquierdo |
| 4 | Número |
| / | Dividir |
| 1 | Número |
| ) | Paréntesis derecho |
| ; | Punto y Coma |

*Figura 3.32 – Tokens*

Una vez obtenidos los tokens se construye un árbol sintáctico, en este ejemplo el árbol sintáctico generado sería similar al de la Figura 3.33:

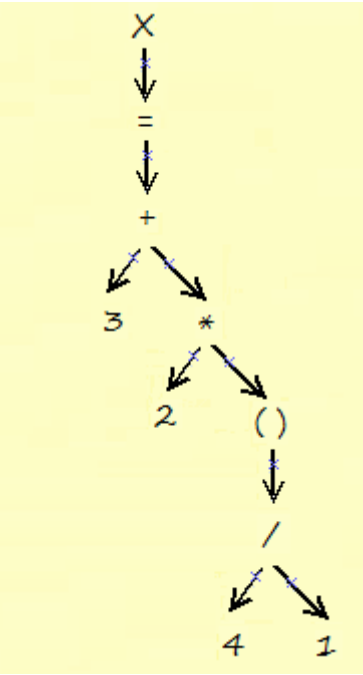


*Figura 3.33 – Árbol sintáctico*

Una vez construido el árbol sintáctico, se procede a recorrer el árbol desde el nodo raíz hasta las hojas de forma posorden y se va

rellenando una estructura de datos similar a una pila, añadiendo la información que se necesite para cada elemento. Una vez recorrido e insertado los tokens, el contenido de la pila sería similar al que se representa en la Figura 3.34.

| Posición | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| Token | Número | Número | Dividir | Paréntesis | Número | Multiplicar | Número | Sumar | Nombre |
| Sucesores | 0 | 0 | 2 | 1 | 0 | 2 | 0 | 2 | 1 |
| Valor | 1 | 4 | / | () | 2 | * | 3 | + | X |

*Figura 3.34 – Pila*

Una vez se dispone de esta estructura dinámica de datos correctamente cargada se pueden realizar los cálculos necesarios para la aplicación de forma rápida y eficiente recorriendo la pila.

Recorriendo el contenido de la pila, y con cierta lógica de negocio implementada en la aplicación, se puede obtener que la siguiente operación básica a realizar debe ser la división: “4/1”. Primero se lee de la pila el número “1”, después el número “4” y posteriormente la operación “dividir” (que necesita 2 operandos que ya han sido leídos, “1” y “4”). La Figura 3.35 muestra el contenido de la pila una vez realizada la división.

| Posición | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| Token | Número | Paréntesis | Número | Multiplicar | Número | Sumar | Nombre |
| Sucesores | 0 | 1 | 0 | 2 | 0 | 2 | 1 |
| Valor | 4 | () | 2 | * | 3 | + | X |

*Figura 3.35 – Pila 2*

De forma similar, recorriendo la pila se puede determinar que la siguiente operación básica a realizar será la resolución del paréntesis: “(4)”. Primero se lee de la pila un número y luego la operación paréntesis que necesita un operando. La Figura 3.36 muestra el contenido de la pila tras quitar el paréntesis.

| Posición | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Token | Número | Número | Multiplicar | Número | Sumar | Nombre |
| Sucesores | 0 | 0 | 2 | 0 | 2 | 1 |
| Valor | 4 | 2 | * | 3 | + | X |

*Figura 3.36 – Pila 3*

Siguiendo con la misma lógica, se lee de la pila un número, luego otro número y luego la operación multiplicar que necesita los dos operandos ya leídos, la siguiente operación básica a realizar es: “2*4”. La Figura 3.37 muestra el contenido de la pila después de realizar la multiplicación.

| Posición | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Token | Número | Número | Sumar | Nombre |
| Sucesores | 0 | 0 | 2 | 1 |
| Valor | 8 | 3 | + | X |

*Figura 3.37 – Pila 4*

La siguiente operación básica sería:”8+3”. La Figura 3.38 muestra el contenido de la pila después de realizar la suma.

| Posición | 0 | 1 |
|---|---|---|
| Token | Número | Nombre |
| Sucesores | 0 | 1 |
| Valor | 11 | X |

*Figura 3.38 – Pila 5*

Para finalizar con el ejemplo, la última operación que se lee de la pila es la asignación: “X=11”.

Ahora se va a proceder a describir los pasos realizados para conseguir la evaluación de expresiones matemáticas explicada en nuestro sistema informático.

Como se ha mencionado anteriormente, se usa JFlex para realizar el análisis léxico que constituye la primera fase a la hora de evaluar una expresión matemática. JFlex permite agrupar la expresión matemática en componentes léxicos (tokens) que son secuencias de caracteres que tienen un significado. Además todos los espacios en blanco y demás información innecesaria se elimina. También se comprueba que los símbolos validos se han escrito correctamente.

Como resumen y sin centrarnos en detalle en el funcionamiento de JFlex vamos a citar brevemente las tres partes en las que se basa el fichero de especificación léxica necesaria:

- "Usercode": En esta zona se declaran los "imports" que necesitará el código java que genera JFlex.

- "Options, Declarations and Macros": Aquí es donde incluiremos el código que queremos que se incluya en la clase generada.

- "Lexical Rules and Actions": En ésta sección se asigna a cada expresión regular las acciones que queremos que se lleven a cabo al encontrar un token que coincida con cada una de ellas.

JFlex define las reglas de reconocimiento de símbolos (tokens) a partir de expresiones regulares. Cuando un "token" es reconocido por uno de estos patrones de agrupamiento se le define una acción, por lo general está acción devuelve el tipo y valor. El archivo JFlex necesario para reconocer una expresión matemática, debe poder reconocer y devolver los siguientes tipos:

- Igual
- Sumar
- Restar
- Multiplicar
- Dividir

- Número
- Nombre
- Punto y coma
- Lparentesis
- Rparentesis

El fichero creado, siguiendo la sintaxis de JFlex, para reconocer las acciones descritas anteriormente ha sido el siguiente:

```
// Este es el archivo FLEX
%%
%public
/* Nombre de la clase generada para el analizador léxico */
%class scanner
/* Indicar funcionamiento autónomo */
%standalone
%unicode
/* Acceso a la columna y fila actual de analisis CUP */
%line
%column
/* Habilitar la compatibilidad con el interfaz CUP para el generador sintáctico */
%cup
%{
%}
punto = [.]
letra = [a-zA-Z]
digito = [0-9]
letra_digito = [a-zA-Z0-9]
cadena = [^']
espacio = [\ ]
comentario = [^\"]
%%
"="                     {return new Symbol (sym.IGUAL,  yyline, yycolumn, new String(yytext()));}
"+"                     {return new Symbol (sym.SUMAR,  yyline, yycolumn, new String(yytext()));}
"-"                     {return new Symbol (sym.RESTAR,  yyline, yycolumn, new String(yytext()));}
"*"                     {return new Symbol (sym.MULTIPLICAR,  yyline, yycolumn, new String(yytext()));}
"/"                     {return new Symbol (sym.DIVIDIR,  yyline, yycolumn, new String(yytext()));}
("-")?{digito}+({punto}{digito}+)?{return new Symbol (sym.NUMERO,  yyline, yycolumn, new String(yytext()));}
{letra}+     {return new Symbol (sym.NOMBRE,  yyline, yycolumn, new String(yytext()));}
";"                     {return new Symbol (sym.PUNTOYCOMA,  yyline, yycolumn, new String(yytext()));}
"("                     {return new Symbol (sym.LPARENTESIS,  yyline, yycolumn, new String(yytext()));}
")"                     {return new Symbol (sym.RPARENTESIS,  yyline, yycolumn, new String(yytext()));}
[\t\r\n\f]              {}
```

JFlex está diseñado para trabajar junto con CUP aunque también se puede utilizar junto con otros generadores de analizadores sintácticos o como una herramienta independiente. En este proyecto se va a utilizar JFlex junto con CUP.

CUP es una herramienta capaz de generar analizadores sintácticos, es decir, analiza las cadenas que aparecen en un fichero de entrada y

comprueba que son sintácticamente correctas con respecto a una gramática. El analizador sintáctico que generemos podrá, además del análisis, realizar diversas acciones según se reconozcan las cadenas. CUP sigue una forma de funcionamiento similar a la del JFlex, toma como entrada un fichero de especificación en el que se incluye una gramática y genera un analizador sintáctico escrito en Java.

En cuanto al contenido del fichero CUP se pueden diferenciar cinco partes:

- Especificaciones de “package” e “imports”.
- Componentes de código de usuario.
- Lista de símbolos de la gramática: terminales y no terminales
- Declaraciones de precedencia
- Especificación de la gramática

La Figura 3.39 representa una manera de representar la gramática necesaria para reconocer expresiones matemáticas de la forma:

X = 1 + 2 * 3;

Esta gramática evita ambigüedades (una gramática ambigua es la que ofrece varios árboles de análisis sintáctico para la misma cadena)

```
E -> X = E
E -> E ;
E -> E + E | E - E | T
T -> T * F | T / F | F
F -> ( E ) | N

,siendo X igual a Nombre
E igual a Expresión
T igual a Término
F igual a Factor
N igual Número
```

*Figura 3.39 – Gramática*

El fichero necesario para reconocer la gramática anteriormente descrita, siguiendo la sintaxis de CUP es el siguiente:

```
/** Este es el archivo CUP*/
import java_cup.runtime.*;

parser code {:
public void syntax_error(Symbol s) {
  System.out.println("Error de sintaxis en linea " + s.left + " y columna " + s.right);
}
public void unrecovered_syntax_error(Symbol s) throws java.lang.Exception
{
}
:};

/* Preliminaries to set up and use the scanner.  */

/* Terminals (tokens returned by the scanner). */
  terminal        IGUAL, SUMAR, RESTAR, MULTIPLICAR, DIVIDIR, NUMERO;
  terminal        LPARENTESIS, RPARENTESIS, PUNTOYCOMA, NOMBRE;

/* Non terminals */

  non terminal    tfactor tfactor;
  non terminal    ttermino ttermino;
  non terminal    texpr texpr;
  non terminal    texpresion texpresion;
  non terminal    tclase tclase;

/* Precedences */
  precedence left SUMAR, RESTAR;
  precedence left MULTIPLICAR, DIVIDIR;


  start with tclase;


/* The grammar */

  tclase ::= NOMBRE:n IGUAL texpresion:e {:RESULT=new tclase1(n,e); :} ;

  texpresion ::= texpr:e PUNTOYCOMA {:RESULT=new texpresion1(e); :} ;

  texpr ::= texpr:e SUMAR:op ttermino:t {:RESULT=new texpr1(e,op,t); :}
          | texpr:e RESTAR:op ttermino:t {:RESULT=new texpr2(e,op,t); :}
          | ttermino:t {:RESULT=new texpr3(t); :} ;
```

```
ttermino ::= ttermino:t MULTIPLICAR:op tfactor:f {:RESULT=new ttermino1(t,op,f); :}
        | ttermino:t DIVIDIR:op tfactor:f {:RESULT=new ttermino2(t,op,f); :}
        | tfactor:f {:RESULT=new ttermino3(f); :} ;

tfactor ::= LPARENTESIS:lp texpr:e RPARENTESIS:rp {:RESULT=new tfactor1(lp,e,rp); :}
        | NUMERO:n {:RESULT=new tfactor2(n); :} ;
```

Una vez generadas la clase “scanner.java” mediante JFlex y las clases “sym.java” y “parser.java” se añaden al proyecto de Eclipse. A continuación se adjunto una función de la aplicación en la que a partir de instrucción (por ejemplo: “X=3+2*(4/1);”) se rellena la estructura de datos “Pila”.

En esta función, Figura 3.40, se llama a la clase “scanner”, analizador léxico, posteriormente se llama a la clase “parser”, analizador sintáctico, pasándole como parámetro de entrada el resultado del analizador léxico. Finalmente, se obtiene el inicio del árbol sintáctico y se va recorriendo y rellenando la estructura pila.

```
private Pila rellenarPila(String instruccion){
    Pila pila = null;
    try
    {
        StringReader instr = new StringReader(instruccion);
        scanner s = new scanner(instr);
        parser p = new parser(s);
        tclase t = (tclase) p.parse().value;
        pila = t.Apilar();
    } catch (Exception e) {
        pila = null;
    }

    return pila;
}
```

*Figura 3.40 – rellenarPila*

### 3.5.3 Capa Presentación

Las principales funcionalidades que realiza la capa de presentación son:

- Recoger la información del usuario.
- Enviar esta información a la capa de negocio.
- Recoger los resultados de la capa de negocio.
- Presentar los resultados al usuario.

Esta capa se desarrolla utilizando Swing, biblioteca gráfica de Java. El paquete Swing es parte de la JFC (que provee facilidades para construir GUIs) en la plataforma Java.

Cada ejecutable Java tiene un objeto UIManager que determina el Look&Feel, es decir, la apariencia en pantalla y funcionamiento, que van a tener los componentes Swing de ese ejecutable: botones, cajas de texto, cajas de selección, listas, etc.

Una de las ventajas que representan las capacidades de Look&Feel, es poder crear un interfaz gráfico estándar y corporativo, independientemente de la plataforma en que se esté ejecutando.

Con Java es fácil cambiar el Look&Feel, basta con tener la librería adecuada y una sola línea de código llamando al método setLookAndFeel del objeto UIManager. En internet hay muchos sitios webs que facilitan estás librerías, en este proyecto hemos utilizado el Look&Feel "NimRod" [17] , disponible bajo licencia LGPL y que ofrece un interfaz gráfico agradable y acorde al tipo de aplicación desarrollado y usuarios a los que va destinado.

Todos los iconos utilizados en la aplicación han sido descargadas del sitio web Crystal Clear [18], disponibles bajo licencia LGPL.

A continuación, en las Figuras 3.41, 3.42, 3.43, 3.44, 3.45, 3.46, 3.47, 3.48, 3.49, 3.50, 3.51, 3.52, 3.53, 3.54, 3.55, 3.56, 3.57 y 3.58, se presentan los prototipos iniciales de las distintas pantallas de nuestro sistema que se realizaron en la toma de requisitos junto capturas de pantalla de la aplicación. En estos bocetos se intenta transmitir al usuario la configuración y disposición de los elementos que contendrá el sistema.

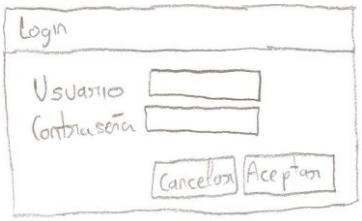


*Figura 3.41 – Prototipo pantalla login*

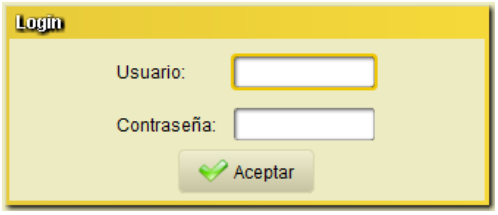


*Figura 3.42 – Pantalla login*

*Figura 3.43 – Prototipo pantalla inicio del usuario alumno*

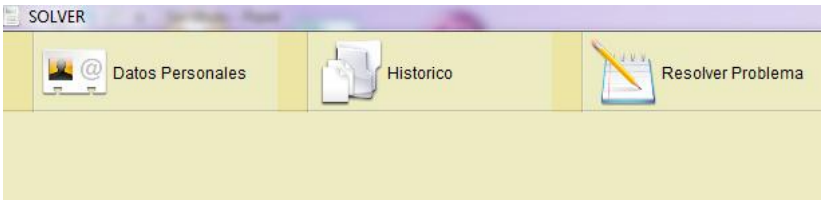


*Figura 3.44 – Pantalla inicio del usuario alumno*

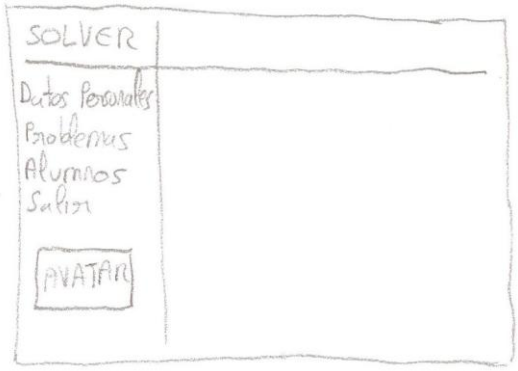


*Figura 3.45 – Prototipo pantalla inicio del usuario profesor*

*Figura 3.46 – Pantalla inicio del usuario profesor*

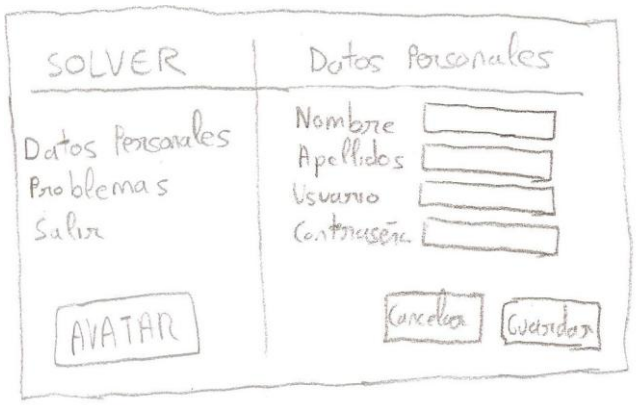


*Figura 3.47 – Prototipo pantalla datos generales*

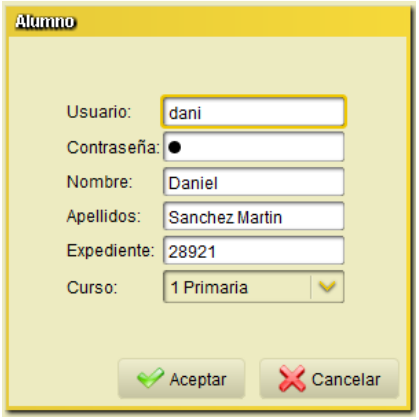


*Figura 3.48 – Pantalla datos generales*

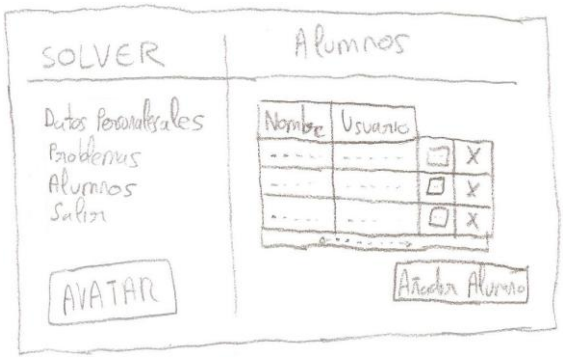


*Figura 3.49 – Prototipo pantalla alumnos del usuario profesor*

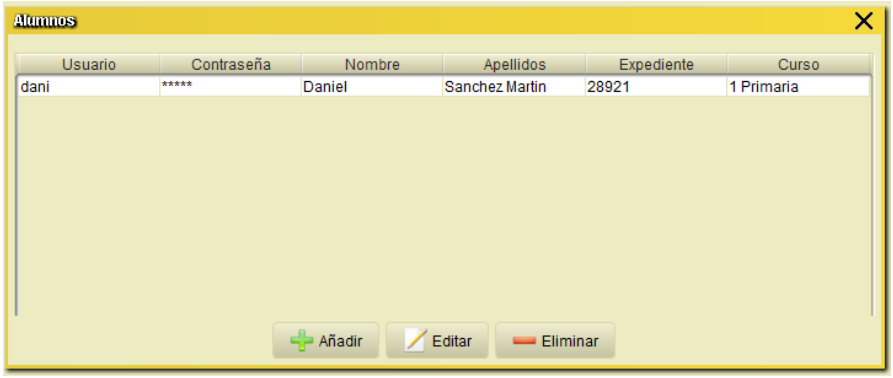


*Figura 3.50 – Pantalla alumnos del usuario profesor*

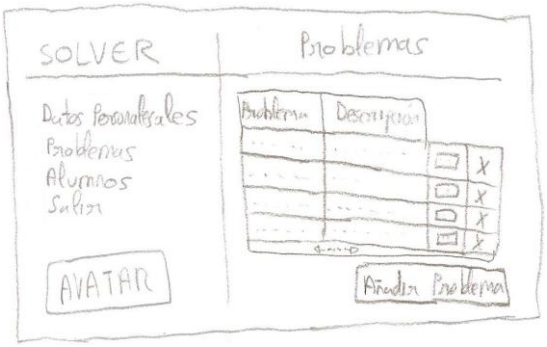


*Figura 3.51 – Prototipo pantalla problemas del usuario profesor*

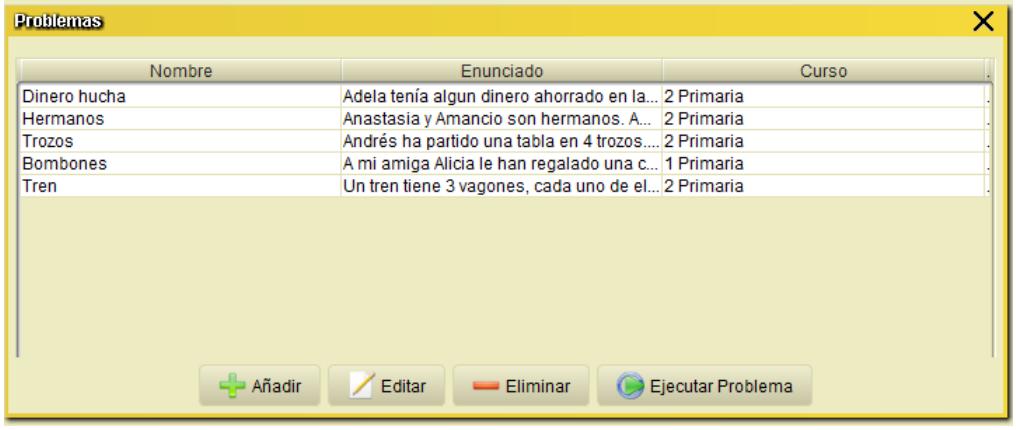


*Figura 3.52 – Pantalla problemas del usuario profesor*

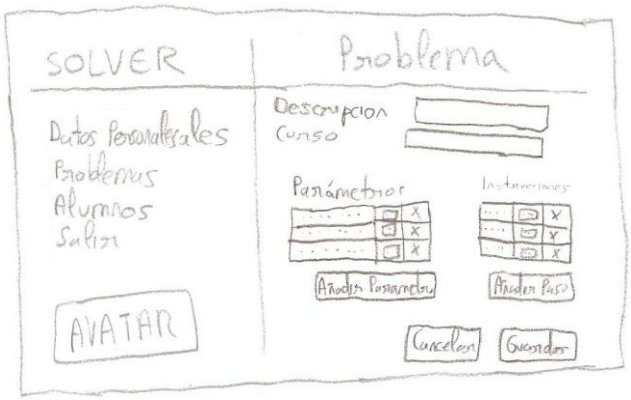


*Figura 3.53 – Prototipo pantalla problema del usuario profesor*

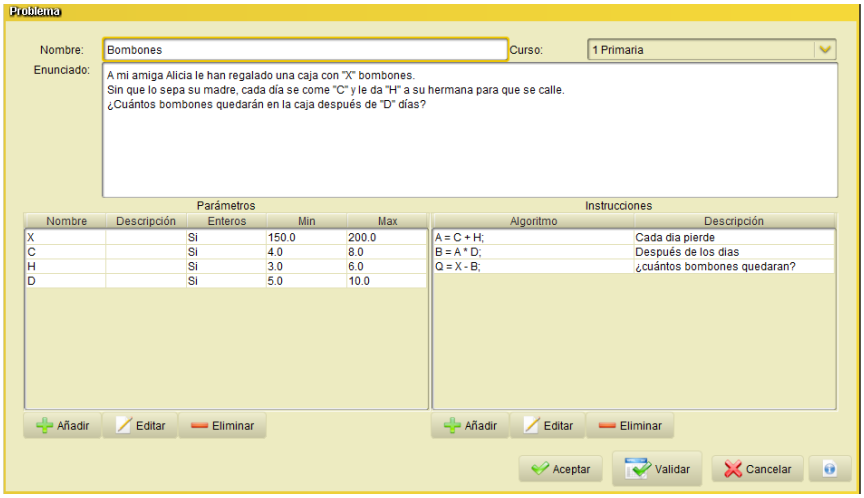


*Figura 3.54 – Pantalla problema del usuario profesor*

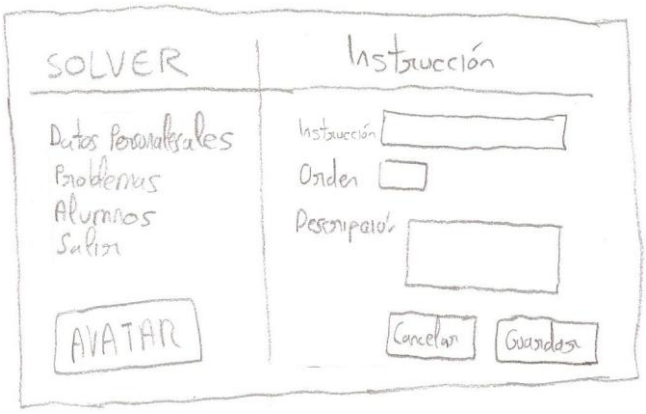


*Figura 3.55 – Prototipo pantalla instrucción del usuario profesor*

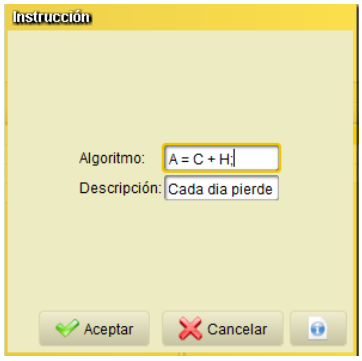


*Figura 3.56 – Pantalla instrucción del usuario profesor*

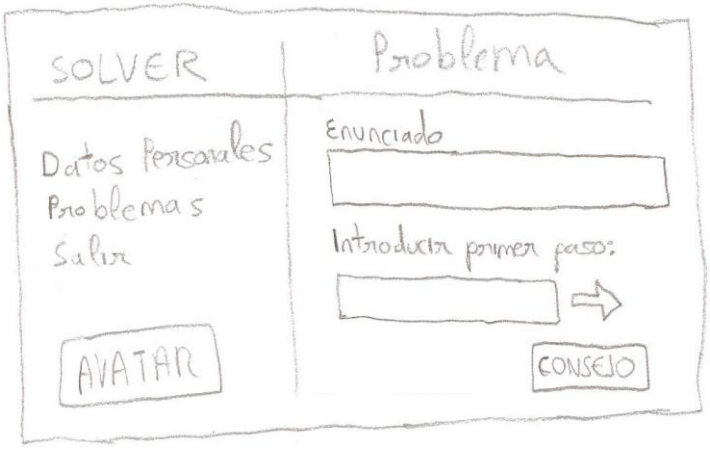


*Figura 3.57 – Prototipo pantalla problema del usuario alumno*

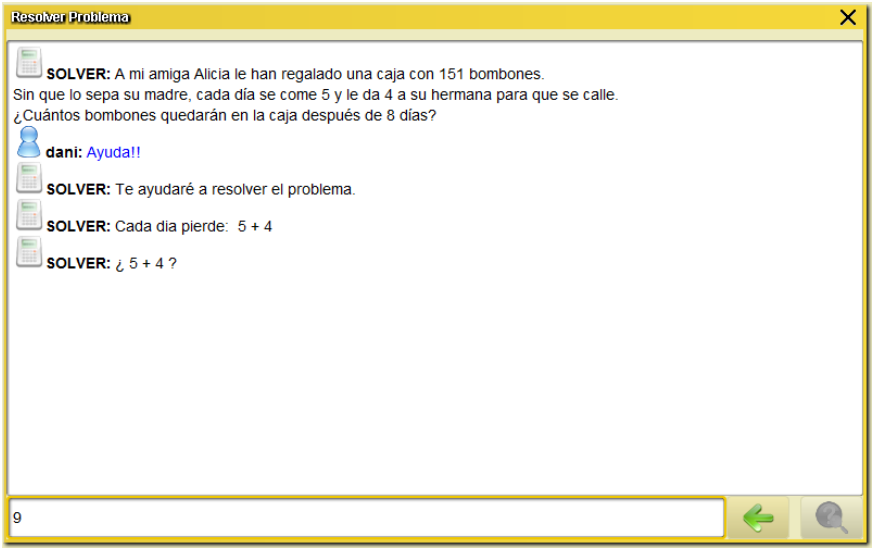


*Figura 3.58 – Prototipo pantalla problema del usuario alumno*

A continuación se explica detalladamente un caso práctico que muestra cómo el algoritmo funciona interaccionando con un usuario. Las Figuras 3.59, 3.60 y 3.61 muestran las capturas de pantalla de la resolución del problema.

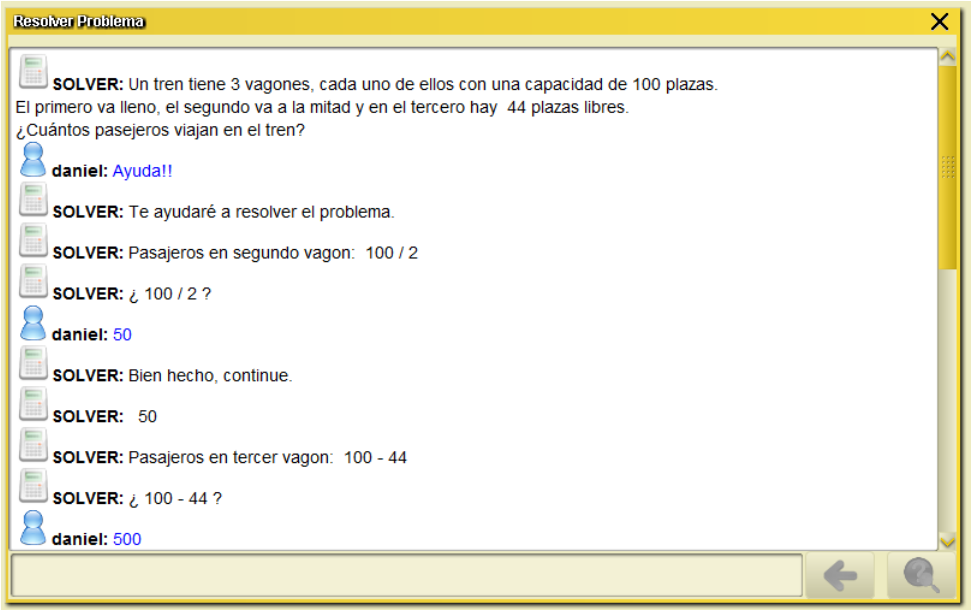


*Figura 3.59 – Caso práctico 1 de 3*

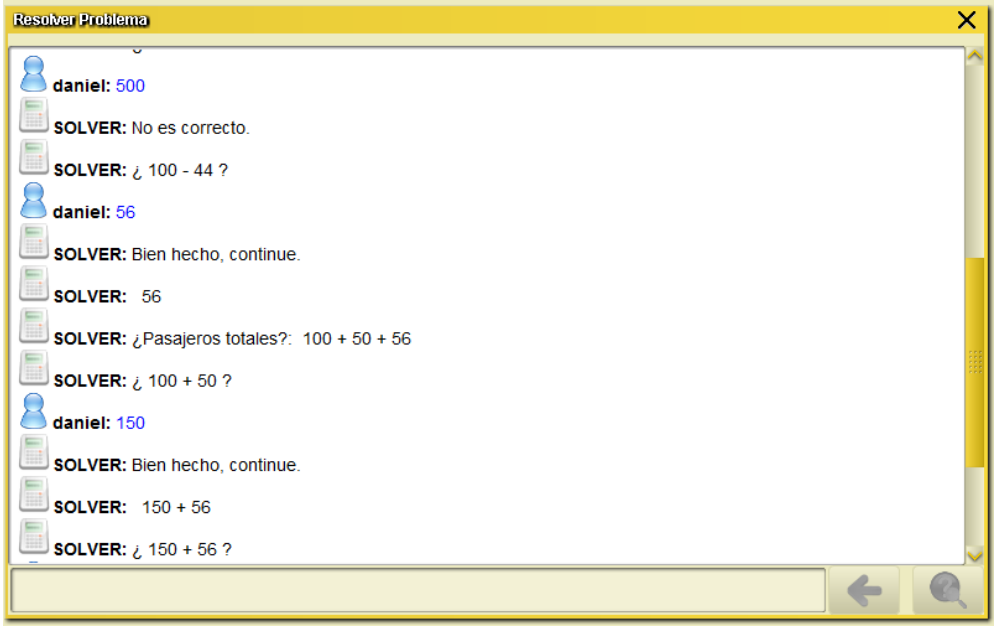


*Figura 3.60 – Caso práctico 2 de 3*

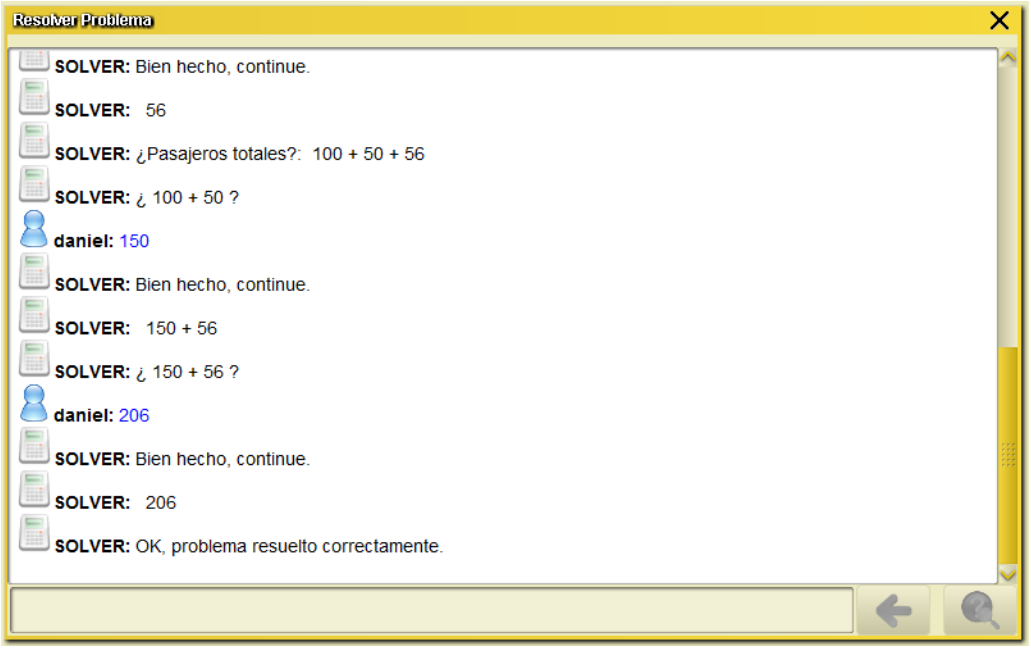


*Figura 3.61 – Caso práctico 3 de 3*

Al arrancar la aplicación Solver propone el enunciado del problema:

*SOLVER: Un tren tiene 3 vagones, cada uno de ellos con una capacidad de 100 plazas. El primero va lleno, el segundo va a la mitad y en el tercero hay 44 plazas libres. ¿Cuántos pasajeros viajan en el tren?*

El usuario solicita ayuda haciendo click en el icono de la parte inferior derecha de la pantalla, Figura 3.62.

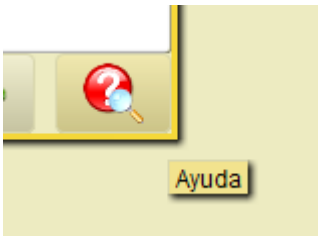


*Figura 3.62 – Icono ayuda*

Solver presenta al usuario la ecuación que define los pasajeros que hay en el segundo vagón. Y pregunta al usuario que los resuelva.

*SOLVER: Te ayudaré a resolver el problema.*
*SOLVER: Pasajeros en segundo vagón: 100 / 2*
*SOLVER: ¿ 100 / 2 ?*

El usuario resuelve correctamente la división.

*daniel: 50*

A continuación, Solver felicita al usuario por su respuesta. Plantea la ecuación que define los pasajeros que hay en el tercer vagón y solicita al usuario que lo resuelva.

*SOLVER: Bien hecho, continúe.*
*SOLVER: 50*
*SOLVER: Pasajeros en tercer vagón: 100 - 44*
*SOLVER: ¿ 100 - 44 ??*

El usuario responde incorrectamente.

*daniel: 500*

A continuación, Solver le comunica que la respuesta no es correcta y le vuelve a preguntar que realice la operación correctamente.

*SOLVER: No es correcto.*
*SOLVER: ¿ 100 – 44 ?*

El usuario realiza correctamente la resta.

*daniel: 56*

A continuación, Solver felicita al usuario por su respuesta. Plantea la ecuación que define los pasajeros totales que hay en el tren. Solicita al usuario que resuelva la primera suma de la ecuación.

*SOLVER: Bien hecho, continúe.*
*SOLVER: 56*
*SOLVER: ¿Pasajeros totales?: 100 + 50 + 56*
*SOLVER: ¿ 100 + 50 ?*

El usuario realiza correctamente la suma.

*daniel: 150*

A continuación, Solver felicita al usuario por su respuesta. Muestra como queda la ecuación con su respuesta y vuelve a preguntar al usuario que resuelve la segunda suma de la ecuación.

*SOLVER: Bien hecho, continúe.*
*SOLVER: 150 + 56*
*SOLVER: ¿ 150 + 56 ?*

El usuario realiza correctamente la suma.

*daniel: 206*

Solver le comunica al usuario que el problema ha sido resuelto correctamente.

*SOLVER: OK, problema resuelto correctamente.*

# 4 Pruebas

En este apartado se presentan las pruebas realizadas.

## *4.1 Pruebas unitarias*

El objetivo principal de las pruebas unitarias es detectar errores en cada uno de los módulos software al ser ejecutados independientemente del resto de componentes.

Estas pruebas se centran en la menor unidad de diseño del software, el módulo y suelen ser ejecutadas por el programador que construye el módulo. Se han considerados dos enfoques distintos, denominados caja blanca y caja negra

### 4.1.1 Pruebas de caja blanca

Las pruebas de caja blanca consisten en realizar pruebas para verificar que líneas específicas de código funcionan tal como está definido. Se comprueban los caminos lógicos del software en base a examinar trozos específicos del programa (bucles, sentencias de bifurcación, etc.)

El número de sentencias de un programa es finito, se podría diseñar un plan de pruebas de caja blanca que comprueba la totalidad de la aplicación. En la práctica, elaborar un proceso de pruebas completo puede ser excesivamente laborioso y costoso. En el presente trabajo se ha realizado un plan de pruebas de caja blanca únicamente sobre uno de los métodos más importantes del sistema, el método “procesar” de la clase “ResolverProblema”, encargado de devolver el mensaje de respuesta de Solver ante una entrada del usuario.

En la Figura 4.1 se adjunta el código fuente del método “procesar”.

```
public String procesar(String input, TipoUsuario tipo){
    String mensaje = "";
    if (!_ayuda){
        if (TipoBase.comparar(input, _solucion)){
            mensaje = "OK, problema resuelto correctamente.";
            _fin = true;
            _historico = _problemas.updateHistoricoFin(_historico, 1, tipo);
            _historico = _problemas.updateHistoricoCorrecto(_historico, tipo);
        }else{
            mensaje = "No es la solución correcta.";
            _historico = _problemas.updateHistoricoIncorrecto(_historico, tipo);
        }

    }else{
        if(TipoBase.comparar(input, _sugerenciaUltima.getSolucion())){
            mensaje = "Bien hecho, continue.";
            _historico = _problemas.updateHistoricoCorrecto(_historico, tipo);
            _pila.reducir(_sugerenciaUltima.getSolucion(),_sugerenciaUltima._posOperad
            _sugerenciaUltima = null;
            mensaje += "\n " +  _pila.getEnunciado(false);
        }
        else{
            mensaje = "No es correcto.";
            _historico = _problemas.updateHistoricoIncorrecto(_historico, tipo);
        }
    }

    return mensaje;
}
}
```

*Figura 4.1 – Método procesar*

El método del camino básico permite obtener una medida de la complejidad de un diseño procedimental. Esta medida puede ser usada como guía a la hora de definir un conjunto de caminos de ejecución, diseñando casos de pruebas que garanticen que cada camino se ejecuta al menos una vez. La Figura 4.2 representa el grafo de flujo asociado al código fuente del método "procesar", donde cada nodo representa una o más sentencias procedimentales, los rombos son los nodos predicados (condiciones simples) y las flechas (aristas) representan el flujo de control.

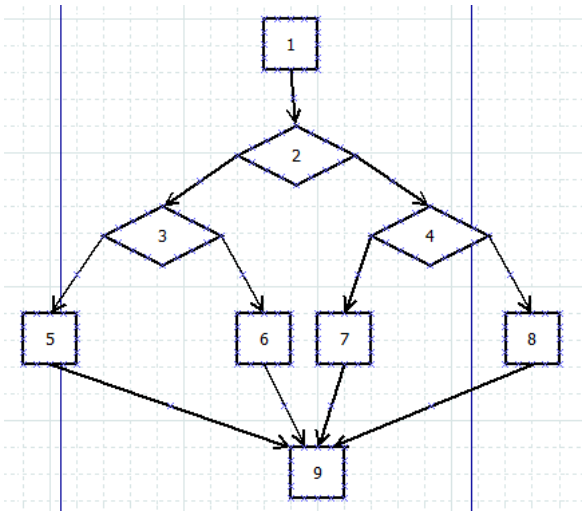


*Figura 4.2 – Grafo de flujo*

Una vez construido el grafo de flujo se va a calcular la complejidad ciclomática, métrica del software que proporciona una medida cuantitativa de la complejidad lógica de un programa. La complejidad ciclomática, V (G), de un grafo de flujo G, se puede definir de las siguientes formas:

V (G) = Aristas – Nodos + 2
V (G) = Nodos predicados +1

A partir del grafo de la Figura 4.2, la complejidad ciclomática sería 4:

V (G) = 11 – 9 + 2 = 4
V (G) = 3 +1 = 4

En el contexto del método de prueba del camino básico, el valor de la complejidad ciclomática define el número de caminos independientes de dicho programa, y por lo tanto, el número de casos de prueba a realizar. Un camino independiente es cualquier camino del programa que está constituido por lo menos por una arista que no haya sido recorrida anteriormente a la definición del camino. El número de caminos independientes es 4 y los caminos independientes son:

Camino 1: 1-2-3-5-9
Camino 2: 1-2-3-6-9
Camino 3: 1-2-4-7-9
Camino 4: 1-2-4-8-9

El último paso es construir los casos de prueba que fuerzan la ejecución de cada camino. Una forma de representar el conjunto de casos de pruebas es como se muestra en la Tabla 4.1.

| Número del camino | Caso de prueba | Resultado esperado y obtenido |
|---|---|---|
| 1 | _ayuda = FALSE<br>input = 10<br>_solucion = 10 | Ok, problema resuelto correctamente. |
| 2 | _ayuda = FALSE<br>input = 10<br>_solucion = 9 | No es la solución correcta. |
| 3 | _ayuda = TRUE<br>input = 10<br>_sugerenciaUltima.getSolucion() = 10 | Bien hecho, continúe. |
| 4 | _ayuda = TRUE<br>input = 10<br>_sugerenciaUltima.getSolucion() = 9 | No es correcto. |

*Tabla 4.1 – Casos de prueba caja blanca*

Finalmente, mencionar que el resultado obtenido en las pruebas ha sido igual al esperado.

### 4.1.2 Pruebas de caja negra

Las pruebas de caja negra se llevan a cabo sobre la interfaz del software y pretenden demostrar que el software funciona adecuadamente, es decir, que las entradas se aceptan de forma adecuada y que se produce una salida correcta. Estas pruebas no tienen en cuenta la estructura lógica interna del software. Las herramientas básicas son observar la funcionalidad y contrastar con la especificación.

En la Tabla 4.2 se adjuntan las pruebas de caja negra realizados en varias partes del sistema, el principal objetivo de las pruebas de caja negra realizadas es encontrar errores en la interfaz, comprobando la robustez de la aplicación.

| Pantalla | Entrada | Resultado esperado y obtenido |
|---|---|---|
| Datos Personales | Alumno = | Mensaje = 'Faltan introducir campos.' |
| Datos Personales | Alumno = 1 | Mensaje = 'Guardado correctamente.' |
| Alumnos | Editar sin tener ninguno registro seleccionado | Nada |
| Alumnos | Eliminar sin tener ninguno registro seleccionado | Nada |
| Resolver Problema | Introducir Texto = | Mensaje = 'Por favor, introduzca el siguiente paso!!! ' |
| Resolver Problema | Introducir Texto = 'Hola' | Mensaje = 'No es la solución correcta. ' |
| Añadir Curso | Descripción = | Mensaje = 'Faltan introducir campos.' |
| Añadir Problema | Nombre = | Mensaje = 'Faltan introducir campos.' |
| Validar Problema | Algoritmo = 'X = X' | Mensaje = 'El problema no está definido correctamente. ' |
| Añadir Alumno | Algoritmo = 'X = ' | Mensaje = 'El problema no está definido correctamente. ' |
| Añadir Alumno | Algoritmo = '2+3' | Mensaje = 'El problema no está definido correctamente.' |

*Tabla 4.2 – Casos de prueba caja negra*

## 4.1 Pruebas de integración

Las pruebas de integración parten de los componentes individuales previamente probados y tienen como objetivo descubrir errores que se pueden producir en la interacción entre los módulos.

En teoría, la combinación de componentes validos debería dar como resultado software en el que no se detecten errores, pero en la práctica hay múltiples ocasiones en las que las pruebas de unidad no detectan errores que sí se desvelan al ejecutar pruebas sobre el software integrado. Las interfaces entre los módulos suelen ser fuentes de errores de integración, pero también la definición de transacciones sobre las bases de datos o el acceso concurrente a estructuras de datos comunes [19].

A diferencia de los modelos tradicionales de desarrollo de software, desde el inicio del desarrollo se han llevado a cabos pruebas para verificar la correcta relación entre los módulos del sistema. De esta forma se ha evitado en la fase final de implementación del producto, detectar errores complejos en la interrelación de los módulos con su alto coste derivado de su corrección.

Una interrelación global de los componentes del sistema se muestra en la Figura 4.3, se representan los principales paquetes que se han construido en la aplicación y sus conexiones con otros paquetes.

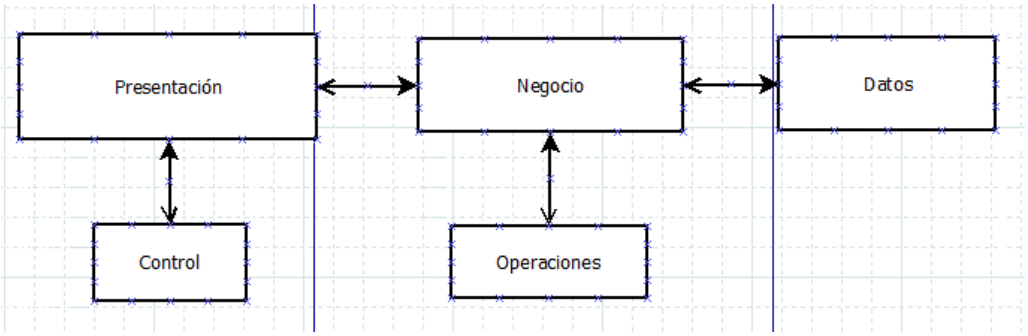


*Figura 4.3 – Interrelación componentes*

La interrelación de los tres paquetes (Presentación, Negocio y Datos) se ha probado continuamente desde el inicio del desarrollo. Se muestra la interrelación del paquete Negocio con el paquete Operaciones (contiene las clases que hacen referencia a cada operación básica). Por último la interrelación del paquete Presentación con el paquete Control (contiene todos los controles de usuario heredados a partir de otros y extendido su funcionalidad básica).

El buen funcionamiento de la aplicación depende en gran medida del funcionamiento correcto de las interrelaciones entre estos paquetes, por lo que se ha llevado a cabo un proceso continuo de pruebas y validación.

## 4.2 Pruebas de validación

En este apartado de va a detallar las pruebas de validación que se han realizado en la aplicación. Se van detallar con capturas de pantalla cómo se han cumplido los requisito establecidos inicialmente.

La Figura 4.4 muestra la pantalla que valida los requisitos RF1, RF2 y RF5.

- RF1: Al iniciar el sistema, se mostrará una pantalla de autentificación.
- RF2: Un alumno se podrá autentificar con un nombre de usuario y contraseña
- RF5: Un profesor se podrá autentificar con un nombre de usuario y contraseña

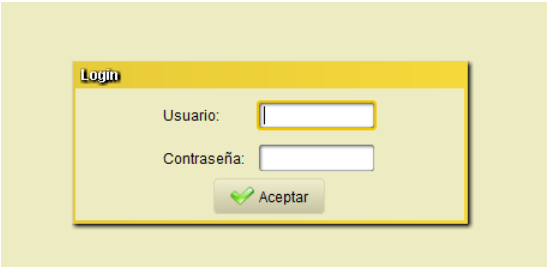


*Figura 4.4 – RF1, RF2, RF5*

La Figura 4.5 muestra la pantalla que valida los requisitos RF3, RF8 y RF22.

- RF3: Si un alumno se autentifica correctamente, se accederá a una pantalla de inicio.
- RF8: La pantalla de inicio del alumno contendrá un menú con las siguientes opciones: modificar datos personales y problemas.
- RF22: En la pantalla problemas del menú alumno, se habilitará la opción de resolver un nuevo problema.

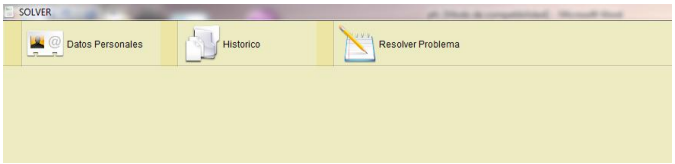


*Figura 4.5 – RF3, RF8*

La Figura 4.6 muestra la pantalla que valida los requisitos RF4 y RF7.

- RF4: Si un alumno se autentifica incorrectamente, se mostrará un mensaje de error.
- RF7: Si un profesor se autentifica incorrectamente, se mostrará un mensaje de error.

*Figura 4.6 – RF4*

La Figura 4.7 muestra la pantalla que valida los requisitos RF6 y RF9.

- RF6: Si un profesor se autentifica correctamente, se accederá a una pantalla de inicio.
- RF9: La pantalla de inicio del profesor contendrá un menú con las siguientes opciones: modificar datos personales, problemas, alumnos y salir.

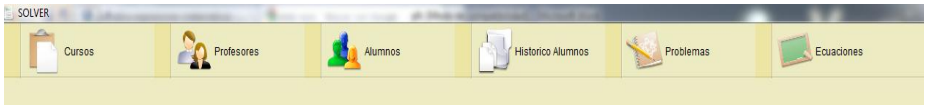


*Figura 4.7 – RF6, RF9*

La Figura 4.8 muestra la pantalla que valida los requisitos RF10 y RF12.

- RF10: En la pantalla de modificar datos personales del alumno, se podrán modificar los datos y guardar los cambios.
- RF12: La información mínima de cada alumno es: nombre de usuario, contraseña, nombre, apellidos, curso, edad.

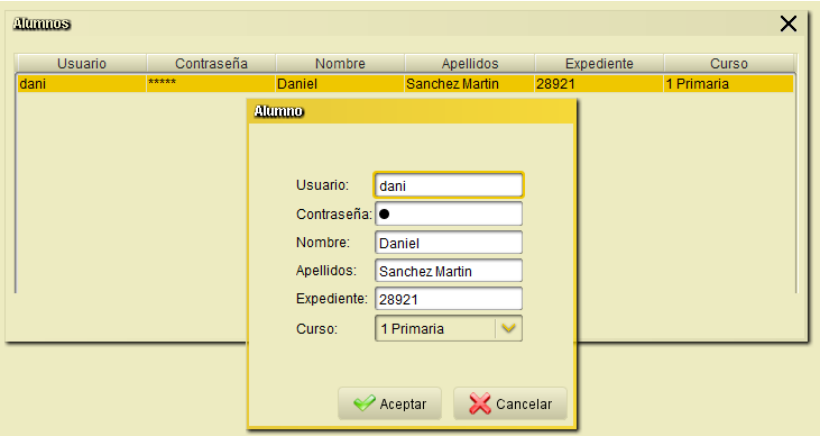


*Figura 4.8 – RF10, RF12*

La Figura 4.9 muestra la pantalla que valida los requisitos RF11 y RF13.

- RF11: En la pantalla de modificar datos personales del profesor, se podrán modificar los datos y guardar los cambios.
- RF13: La información mínima de cada profesor es: nombre de usuario, contraseña, nombre, apellidos.

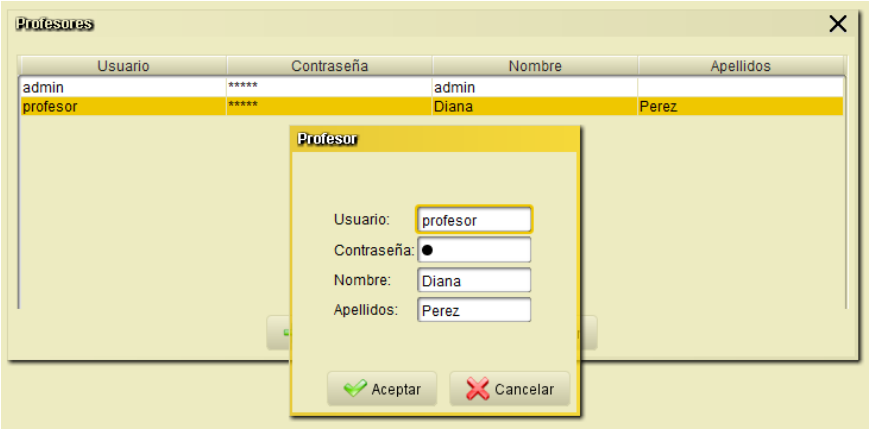


*Figura 4.9 – RF11, RF12*

La Figura 4.10 muestra la pantalla que valida los requisitos RF14 y RF15.

- RF14: Al seleccionar la opción de salir del alumno, se cerrará la aplicación.
- RF15: Al seleccionar la opción de salir del profesor, se cerrará la aplicación.

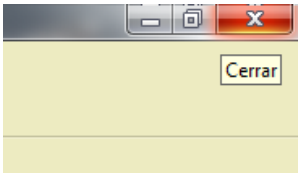


*Figura 4.10 – RF14, RF15*

La Figura 4.11 muestra la pantalla que valida los requisitos RF16 y RF17.

- RF16: En el apartado alumnos del menú profesor, se visualizará el listado de los alumnos.
- RF17: En la pantalla alumnos del menú profesor, se podrá añadir, modificar o eliminar alumnos.

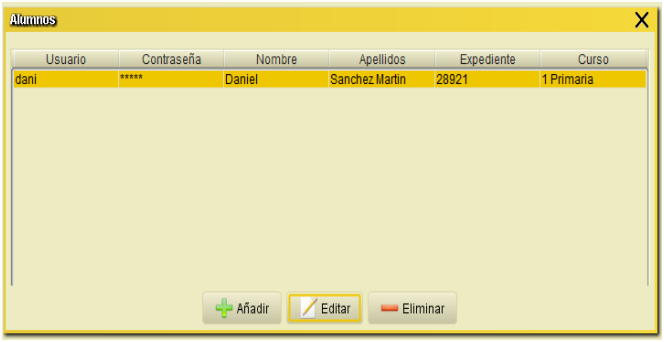


*Figura 4.11 – RF16, RF17*

La Figura 4.12 muestra la pantalla que valida los requisitos RF18 y RF19.

- RF18: En la pantalla problemas del menú profesor, se visualizará el listado de los problemas.
- RF19: En la pantalla problemas del menú profesor, se podrá añadir, modificar o eliminar problemas.

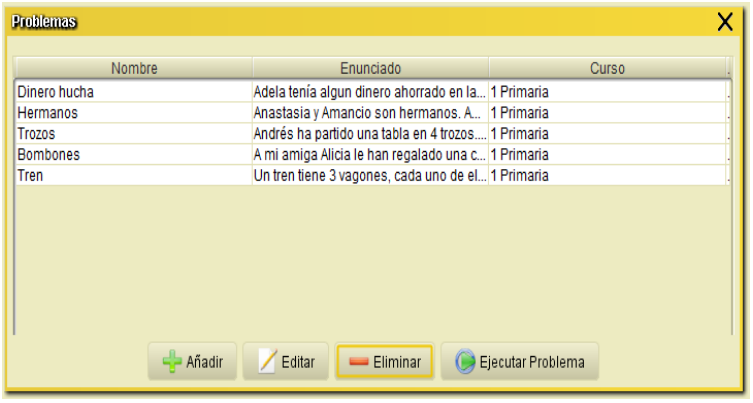


*Figura 4.12 – RF18, RF19*

La Figura 4.13 muestra la pantalla que valida el requisito RF20.

- RF20: En la pantalla problema del menú profesor, se especificará el enunciado, los parámetros del problema y las instrucciones necesarias para solucionar el problema.

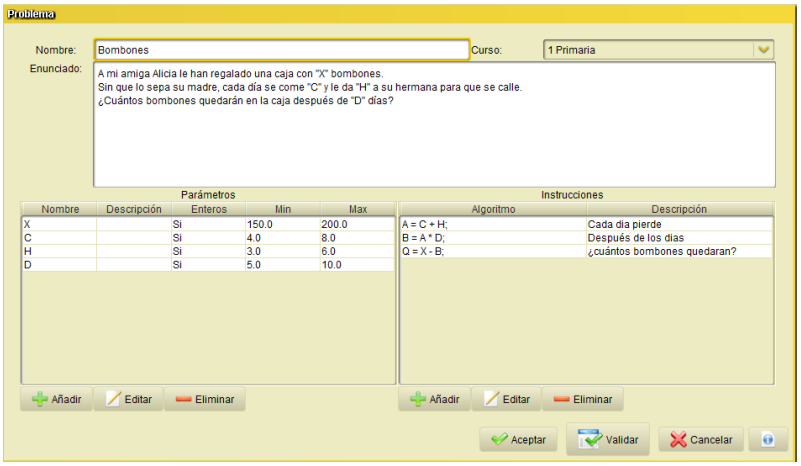


*Figura 4.13 – RF20*

La Figura 4.14 muestra la pantalla que valida el requisito RF21.

- RF21: En la pantalla problemas del menú alumno, se visualizará un histórico de los problemas resueltos por el alumno.

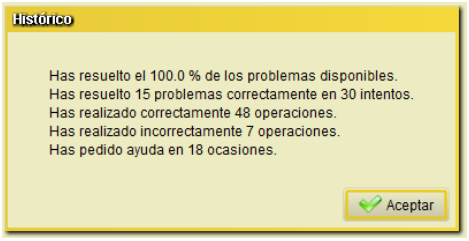


*Figura 4.14 – RF21*

La Figura 4.15 muestra la pantalla que valida los requisitos RF23 y RF24.

- RF23: En la pantalla problema del menú alumno, se mostrará el enunciado de un problema acorde a su curso, edad y su historial de problemas resueltos correctamente y erróneos.
- RF24: Al mostrar el enunciado de un problema en el apartado del menú alumno, el sistema solicitará la siguiente instrucción necesaria para resolver el problema.

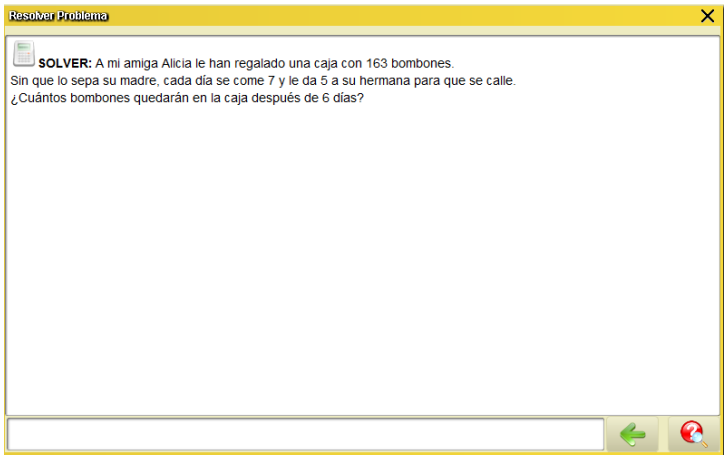


*Figura 4.15 – RF23, RF24*

La Figura 4.16 muestra la pantalla que valida el requisito RF25.

- RF25: Al mostrar el enunciado de un problema en el apartado del menú alumno, el usuario podrá introducir el resultado final del problema.

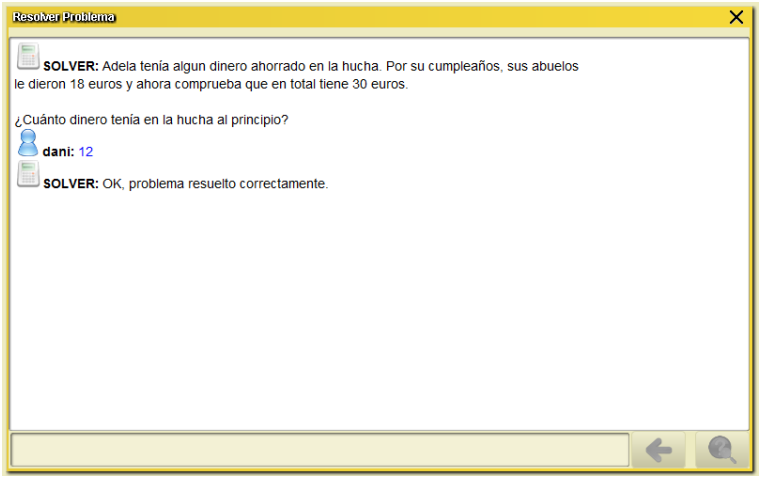


*Figura 4.16 – RF25*

La Figura 4.17 muestra la pantalla que valida los requisitos RF26 y RF27.

- RF26: Al mostrar el enunciado de un problema en el apartado del menú alumno, el usuario podrá introducir la siguiente instrucción necesaria para resolver el problema.
- RF27: Al mostrar el enunciado de un problema en el apartado del menú alumno, el sistema indicará ayudas convenientes para que el usuario pueda resolver la siguiente instrucción.

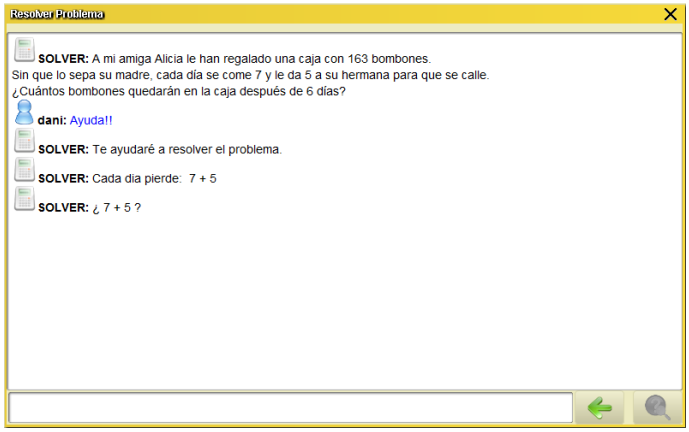


*Figura 4.17 – RF26, RF27*

La Figura 4.18 muestra la pantalla que valida el requisito RF28.

- RF28: Si la instrucción introducida por el alumno en un problema es correcta, el sistema le solicitará la siguiente instrucción necesaria.

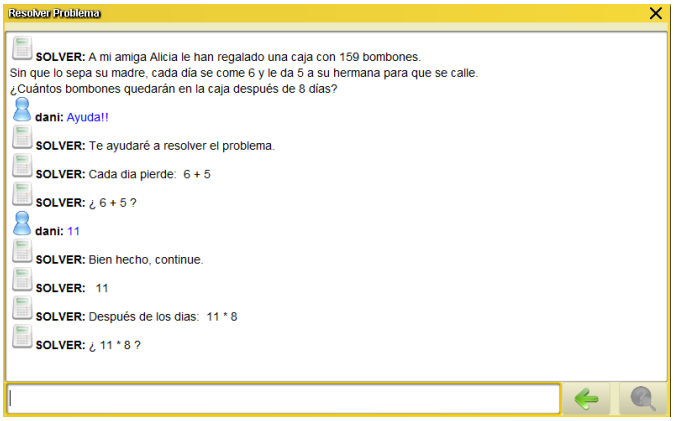


*Figura 4.18 – RF28*

La Figura 4.19 muestra la pantalla que valida el requisito RF29.

- RF29: Si la instrucción introducida por el alumno en un problema es errónea el sistema le mostrará consejos.

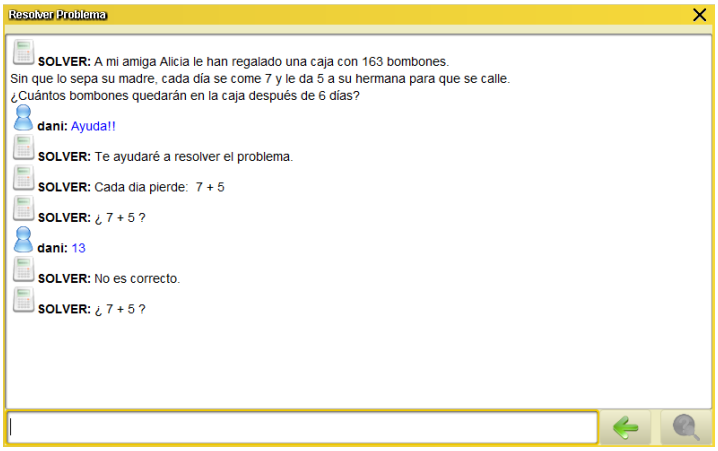


*Figura 4.19 – RF29*

Requisitos no funcionales:

- RNF1: El sistema debe ser multiplataforma
    - La aplicación ha sido probado en plataformas Windows (Windows XP, Windows 7) y en sistemas Linux (Ubuntu)

- RNF2: El sistema debe tener una interfaz gráfica agradable e intuitiva.
    - Los cuestionarios de satisfacción realizados por los usuarios, por norma general, verifican este requisito.

- RNF3: El sistema debe tener unos tiempos de respuesta adecuados.
    - El tiempo de respuesta ante cualquier acción realizada por el usuario es rápido (menos de 1 segundo). No se ha detectado en ninguna de las pruebas realizadas en la aplicación ninguna demora o espera de tiempo innecesario.

## 4.3 Pruebas de aceptación

Las pruebas de aceptación tienen como objetivo validar que el sistema cumple con el funcionamiento esperado y permitir al usuario que determine su aceptación, desde el punto de vista de su rendimiento y funcionalidad.

El grupo de usuarios que han probado la aplicación han sido 5 niños de 13 años de edad cursando primero de la ESO (Educación Secundaria Obligatoria) y una niña de 10 años de edad cursando cuarto de EGB (Educación General Básica). La Tabla 4.3 muestra la información necesaria de los usuarios. Todos estudian en un colegio/instituto público de un pueblo de la provincia de Ávila. Los resultados académicos de los usuarios son buenos, ninguno de ellos ha suspendido nunca la asignatura de matemáticas. Todos disponen de un nivel aceptable de conocimientos informáticos, habitualmente utilizan ordenadores personales.

| Id Usuario | Curso | Edad |
|---|---|---|
| 1 | 4 EGB | 10 |
| 2 | 1 ESO | 13 |
| 3 | 1 ESO | 13 |
| 4 | 1 ESO | 13 |
| 5 | 1 ESO | 13 |
| 6 | 1 ESO | 13 |

*Tabla 4.3 – Información usuarios pruebas aceptación*

Las pruebas han consistido en pedir a cada niño que probase la aplicación durante 15 minutos, analizando las acciones realizadas en la aplicación por el usuario y solicitando al niño que rellene un cuestionario de satisfacción, ver Anexo D.

Las pruebas de aceptación se han realizado sobre un conjunto de 8 problemas y 5 ecuaciones de un nivel acorde al curso de cada alumno. Se han definido los problemas consultando libros de matemáticas acordes al curso de cada usuario. La Tabla 4.4 se puede observar los problemas que han realizado los usuarios durante los 15 minutos de pruebas.

| IdAlumno | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Problemas resueltos correctamente | 6 | 9 | 10 | 10 | 8 | 9 |
| Operaciones correctas | 37 | 45 | 42 | 42 | 39 | 46 |
| Operaciones incorrectas | 12 | 7 | 10 | 5 | 9 | 8 |

*Tabla 4.4 – Estadísticas pruebas de aceptación usuarios*

La Tabla 4.5 muestra los resultados de los cuestionarios de satisfacción obtenidos. Se ha decidido elaborar un cuestionario de satisfacción bastante simple pero que permite recabar la información que se necesita: datos básicos (nombre, edad, curso), puntuar 6 preguntas con una valoración entre 0 y 10 y un apartado donde el usuario pueda exponer lo que considere oportuno sobre la aplicación (ningún usuario lo ha rellenado).

| Id Alumno | 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|---|
| ¿Le ha parecido fácil de utilizar la aplicación? | 5 | 7 | 8 | 6 | 7 | 6 | 6,5 |
| ¿Le ha parecido agradable la aplicación? | 7 | 8 | 8 | 7 | 7 | 8 | 7,5 |
| ¿Le ha parecido útil para practicar ejercicios? | 7 | 8 | 7 | 8 | 8 | 6 | 7,3 |
| ¿Le ha parecido útil para mejorar vuestras habilidades? | 8 | 5 | 5 | 6 | 5 | 7 | 6 |
| ¿Cree usted que este tipo de aplicaciones puedan tener futuro? | 9 | 10 | 8 | 9 | 9 | 8 | 8,8 |
| ¿Le gustaría utilizar esta aplicación en vuestras clases escolares? | 10 | 10 | 8 | 10 | 10 | 8 | 9,3 |
| | 7,67 | 8 | 7,33 | 7,67 | 7,67 | 7,17 | |

*Tabla 4.5 – Resultados Cuestionarios Satisfacción*

Las preguntas "¿Le ha parecido fácil de utilizar la aplicación?" y "¿Le ha parecido agradable la aplicación?" han tenido una puntuación media de 6,5 y 7,5 confirman que se ha conseguido una aplicación intuitiva y agradable.

Las preguntas "¿Le ha parecido útil para practicar ejercicios?" y "¿Le ha parecido útil para mejorar vuestras habilidades?" han tenido una puntuación media de 7,3 y 6 confirman que el agente conversacional realizado pueda ayudar positivamente al alumno en repasar y mejorar conceptos mediante la realización de problemas.

Las preguntas "¿Cree usted que este tipo de aplicaciones puedan tener futuro?" y "¿Le gustaría utilizar esta aplicación en vuestras clases escolares?" han tenido una puntuación media de 8,8 y 9,3 y confirman que las aplicaciones de tipo APC tienen una gran aceptación en niños en edad escolar.

Analizando el cuestionario también se puede concluir que los 6 usuarios han tenido una valoración media bastante buena a las preguntas solicitadas. Las puntuaciones media de cada usuario son 7.67, 8, 7.33, 7.67, 7.67 y 7.17 sobre un puntuación máxima de 10. La puntuación más pequeña obtenida ha sido un 5.

# 5 Conclusiones y Trabajos Futuros

El objetivo principal del proyecto se ha cumplido satisfactoriamente, se ha desarrollado un agente conversacional, Solver, que sirva de apoyo al alumno en la resolución de problemas y ecuaciones matemáticas. Se ha implementado una aplicación en dos partes bien diferenciadas. Por un lado, la parte específica para el profesorado, que permita definir problemas matemáticos que puedan ser parametrizables, también permite la generación de ecuaciones matemáticas automáticamente. Por otro lado, un agente conversacional, que permita al alumno interactuar paso a paso mediante Solver y resolver ecuaciones y problemas matemáticos adaptados a su nivel.

En el apartado de pruebas, se ha verificado que la aplicación cumple los requisitos planteados y el grupo de usuarios que han probado Solver han puntuado con una valoración media de 7,5 sobre 10 a distintas preguntas relacionados con la utilidad, valoración o usabilidad de la aplicación.

En muchos sentidos, la realización del presente trabajo ha supuesto para el autor un reto considerable, no sólo por lo que supone llevarlo a cabo, sino también por el esfuerzo requerido en aprender e integrar en el proyecto determinadas características relacionadas con los intereses del autor y la curiosidad surgida en la elaboración de este proyecto, como pueden ser: Swing, Hibernate, JFlex, Cup, etc.

En cuanto al trabajo futuro, se pretende permitir que el agente pueda realizar gestos y movimientos, pueda emitir sonidos, tener apariencia humana, relacionarse con el usuario en un lenguaje natural, etc. Finalmente, se quiere mencionar que también sería interesante desarrollar una versión web de Solver o incluso una versión para dispositivos móviles

# Bibliografía

**[1]** Sergio Luján Mora (2002). Programación de Aplicaciones Web: Historia, principios básicos y clientes web. Editorial Club Universitario.

**[2]** María Jesús Marco Galindo, Josep Ma Marco Simó, Josep Prieto Blázquez y Ramón Segret Sala (2010). Escaneando la informática. Editorial UOC.

**[3]** Jesús Javier Rodríguez Sala (2003). Introducción a la programación: teoría y práctica. Editorial Club Universitario.

**[4]** Fray León Osorio Rivera (2008). Lógica y programación orientada a los objetos: un inicio al desarrollo de software. Editorial ITM.

**[5]** Ann L.Winblad (1993). Software orientado a objetos. Ediciones Díaz de Santos.

**[6]** Disponible en web: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html. Consulta: 15 de diciembre de 2010.

**[7]** Disponible en web: http://ai-contest.com/. Consulta: 15 de diciembre de 2010.

**[8]** Disponible en web: http://langpop.com/. Consulta: 15 de diciembre de 2010.

**[9]** Mark Pilgrim (2004). Dive into Python. Editorial Apress

**[10]** Margaret A.Ellis, Bjarne Stroustrup (1994). C++ manual de referencia con anotaciones. Editorial Addison-Wesley.

**[11]** Sergio Augusto Cardona Torres, Sonia Jaramillo Valbuena, María Lili Villegas Ramírez (2008). Introducción a la Programación en Java. Editorial Elizcom.

**[12]** C.J. Date (2001). Introducción a los sistemas de bases de datos. Editorial Pearson Educación.

**[13]** A.Davis, E.Bersoff (1988). A Strategy for comparing alternative software development life cycle models. IEEE Transactions on Software Engineering.

**[14]** Pedro Pablo Gómez Martín (2007). Modelo de la enseñanza basada en casos: de los tutores inteligentes a los videojuegos.

**[15]** Cyril Thibaud (2006). MySQL 5: instalación, implementación, administración, programación. Editorial ENI.

**[16]** Olga Pons Capote (2005). Introducción a las bases de datos: El modelo relacional. Editorial Thomson

**[17]** Disponible en web: http://javootoo.com/. Consulta: 05 de mayo de 2012.

**[18]** Disponible en web: http://commons.wikimedia.org/wiki/Crystal_Clear. Consulta: 05 de mayo de 2012.

**[19]** Isabel Ramos Román, José Javier Dolado Cosín (2007). Técnicas cuantitativas para la gestión en la ingeniería software. Editorial: Netbiblo

# Anexo A: Manual de Instalación

A continuación se muestran los pasos necesarios para instalar y configurar MySQL.

Desde la sección descargas de la página oficial de MySQL (http://dev.mysql.com/downloads/mysql/) seleccionar el fichero correspondiente "MySQL Community Server" de la plataforma del equipo servidor donde se instalará el gestor de base de datos. En nuestro caso, el fichero descargado es el "Mysql-5.5.19-win32.msi".

Una vez descargado, proceder a su instalación y configuración (se adjuntan varios pantallazos). En ambos casos hay que seguir el asistente.

En la instalación, tras aceptar la licencia, Figura 1, hay que seleccionar "Typical" en la siguiente ventana, Figura 2, y el proceso de instalación comenzará.

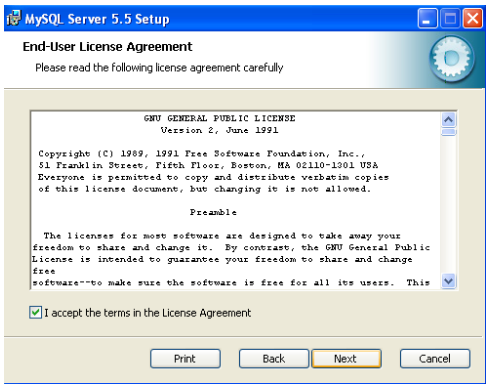


*Figura 1 – Instalación MySQL 1*

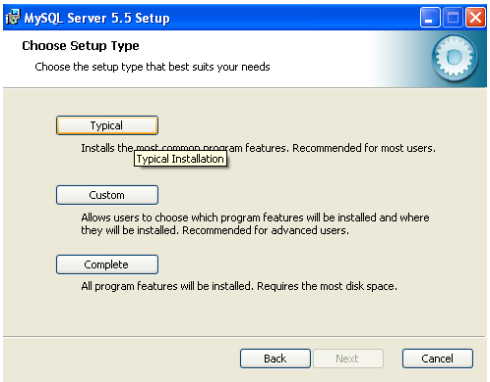


*Figura 2 – Instalación MySQL 2*

En la configuración hay que seleccionar las siguientes opciones en cada una de las pantallas, como se muestran en las Figuras 3, 4, 5, 6, 7 y 8: "Detailled Configuration", "Developer Machine", "Multifunctional Database", "Decision Support (DSS) / OLAP", "Enable TCP/IP Networking" y "Enable Strinct Mode", "Standart Character Set", "Install As Windows Services" y "Include Bin Directory in Windows Path", introducir una contraseña para el usuario root y MySQL estará configurado.

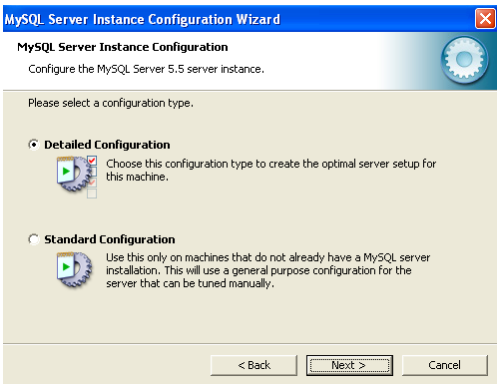


*Figura 3 – Configuración MySQL 1*

*Figura 4 – Configuración MySQL 2*

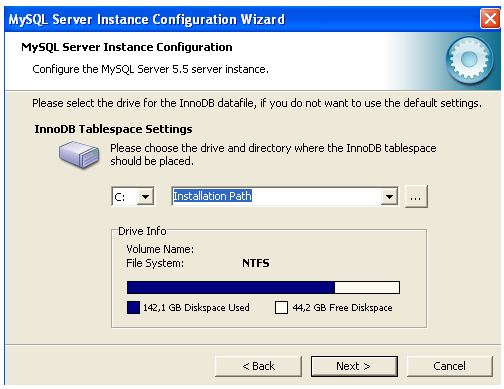


*Figura 5 – Configuración MySQL 3*

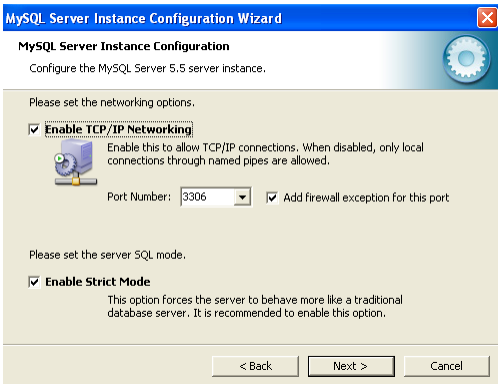


*Figura 6 – Configuración MySQL 4*

*Figura 7 – Configuración MySQL 5*

*Figura 8 – Configuración MySQL 6*

Una vez instalado y configurado MySQL. Se deberá de lanzar el script con las sentencias SQL para crear la base de datos con las tablas y registros iniciales de la aplicación. Para ello, ejecuta "MySQL 5.5 Command Line Client", tras introducir la contraseña, copia el contenido del fichero "scriptBBDD.sql" y pégalo, como se muestra en la Figura 9.

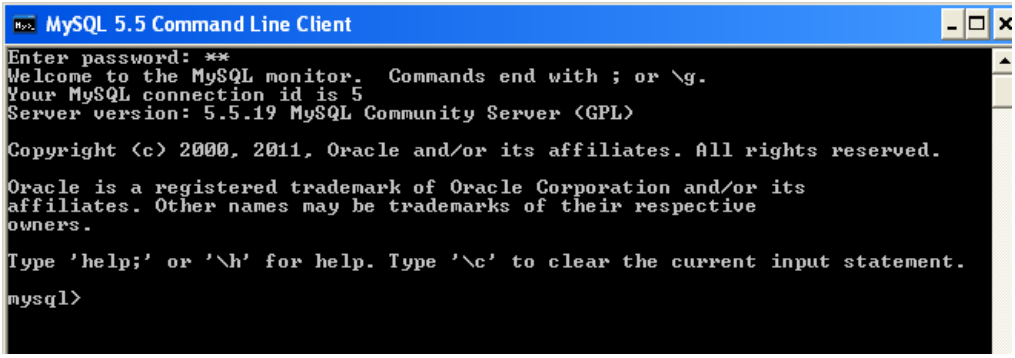


*Figura 9 – Ejecutar scriptBBDD.sql*

# Anexo B: Manual de usuario profesor

La Figura 10 muestra la pantalla de acceso a la aplicación.

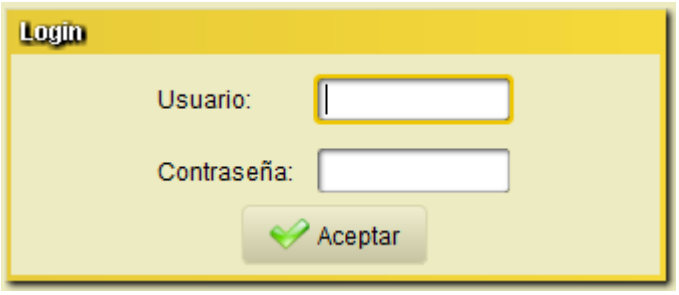


*Figura 10 – Pantalla acceso profesor*

En la Figura 11 se muestra la pantalla de inicio de la aplicación para los usuarios con perfil profesor. Desde esta pantalla se puede acceder a distintas pantallas como son: Datos Profesor, Cursos, Profesores, Alumno, Histórico Alumnos, Profesores y Ecuaciones.

*Figura 11 – Pantalla inicio profesor*

La Figura 12 muestra la pantalla con la información del profesor, pudiéndose modificar los datos que se deseen oportunos.

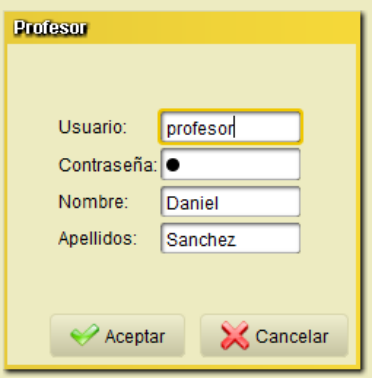


*Figura 12 – Pantalla datos profesor*

La Figura 13 muestra la pantalla con la información de los cursos disponibles. Se pueden añadir, editar y eliminar los cursos que se deseen oportunos.

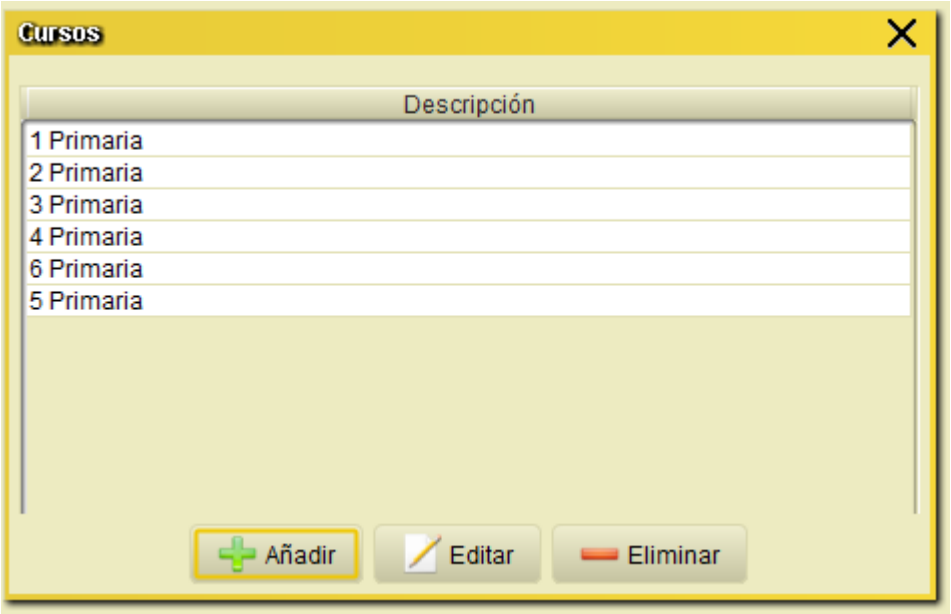


*Figura 13 – Pantalla cursos*

La Figura 14 muestra la pantalla con la información de los profesores disponibles. Se pueden añadir, editar y eliminar los profesores que se deseen oportunos, a excepción, del profesor 'admin' que se crea inicialmente y no se puede modificar ni eliminar.

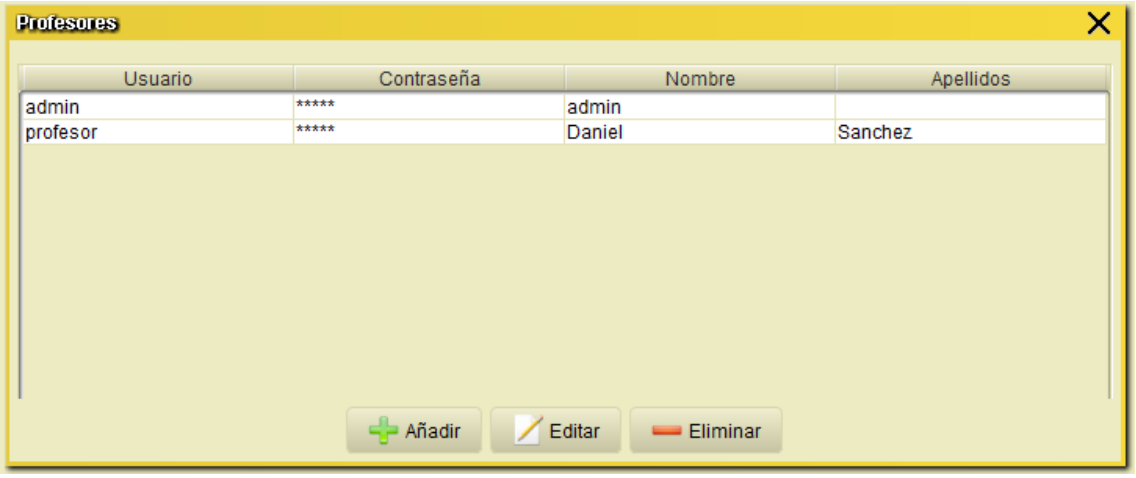


*Figura 14 – Pantalla cursos*

La Figura 15 muestra la pantalla con la información de los alumnos disponibles. Se pueden añadir, editar y eliminar los alumnos que se deseen oportunos.

*Figura 15 – Pantalla alumnos*

La Figura 16 muestra la pantalla histórico alumnos con información detallada sobre los acciones realizadas por cada alumno en cada ejercicio realizado.

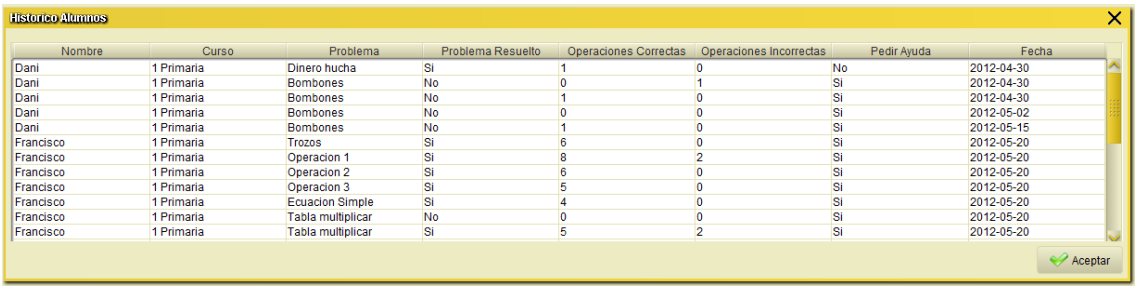


*Figura 16 – Pantalla alumnos*

La Figura 17 muestra la pantalla con la información de los problemas disponibles. Se pueden añadir, editar, eliminar y ejecutar los problemas que se deseen oportunos.

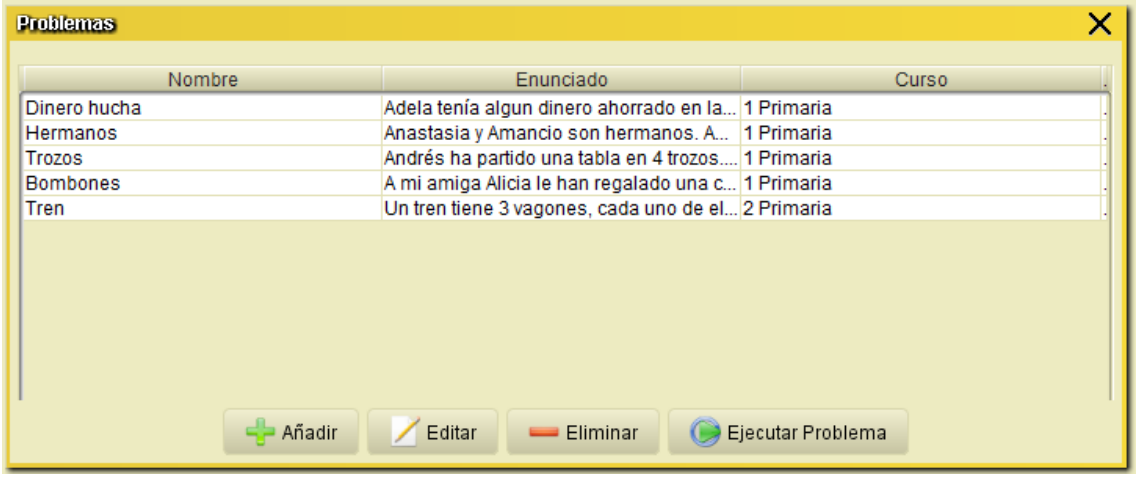


*Figura 17 – Pantalla problemas*

La Figura 18 muestra la pantalla con la información del problema seleccionado. Se pueden realizar cambios, validar los cambios, añadir, editar y eliminar parámetros, añadir, editar y eliminar instrucciones.

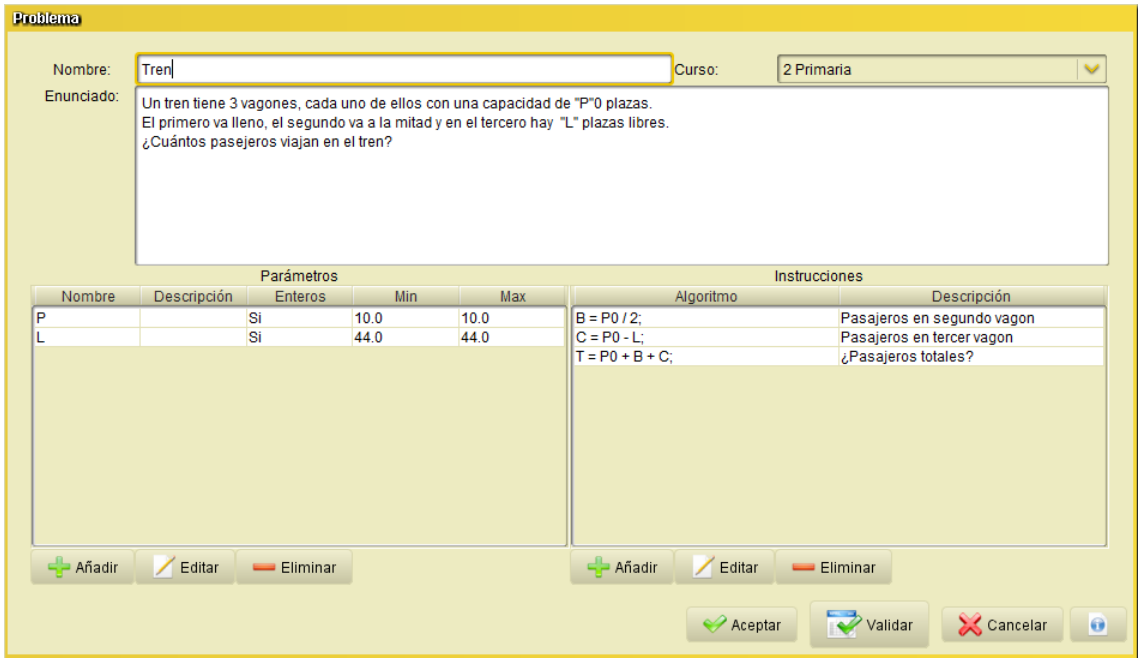


*Figura 18 – Pantalla problema*

Existe la posibilidad de mostrar un pequeño manual explicativo de cada una de las opciones disponibles en la pantalla. La Figura 19 muestra el icono correspondiente al manual.

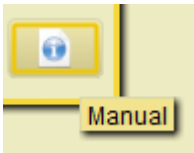


*Figura 19 – Icono manual*

La Figura 20 muestra el manual explicativo de cada uno de los campos existentes en la definición de un problema.

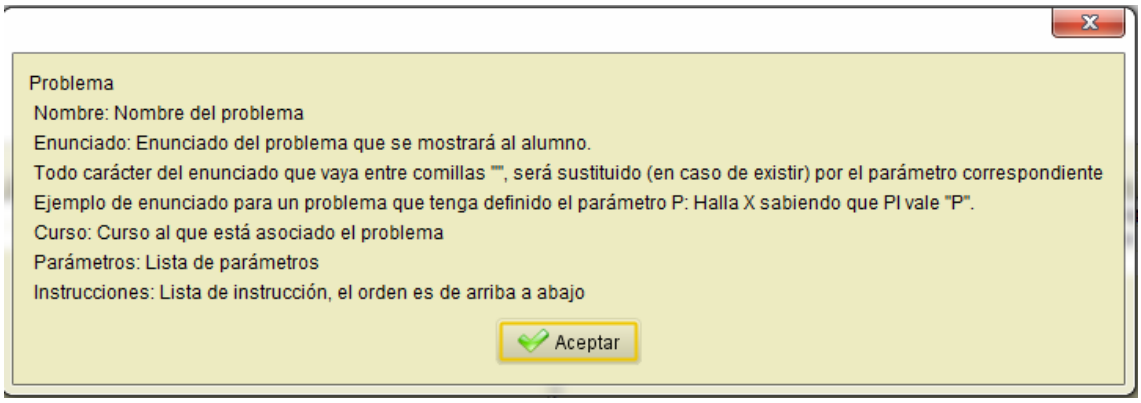


*Figura 20 – Manual pantalla problema*

La Figura 21 muestra la pantalla con la información de las ecuaciones disponibles. Se pueden añadir, editar, eliminar y ejecutar los problemas que se deseen oportunos.

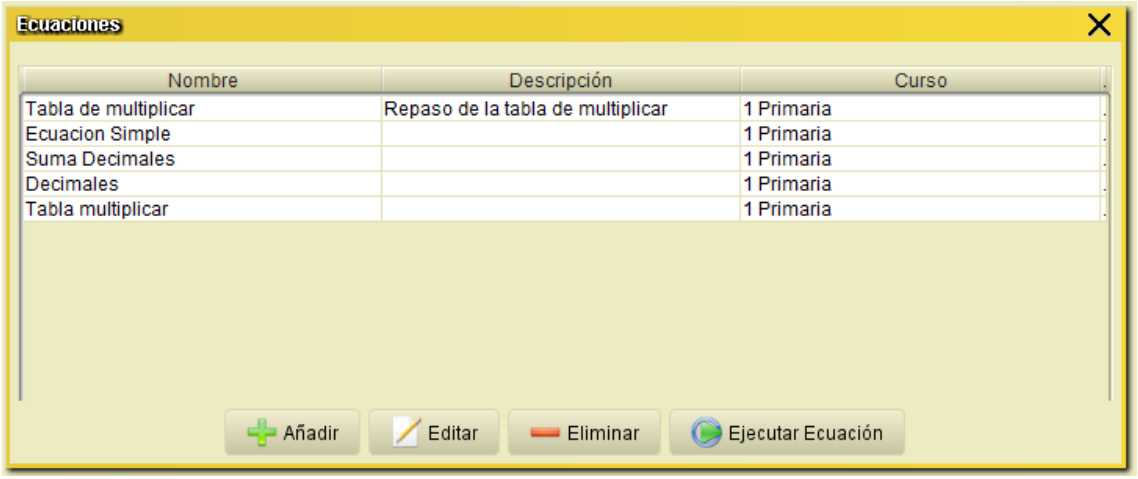


*Figura 21 – Pantalla ecuaciones*

La Figura 22 muestra la pantalla con la información de la ecuación seleccionado. Se pueden realizar cambios, validar los cambios, añadir, editar y eliminar parámetros, añadir, editar y eliminar instrucciones. Dispone también de un pequeño manual explicativo del significado de cada campo.

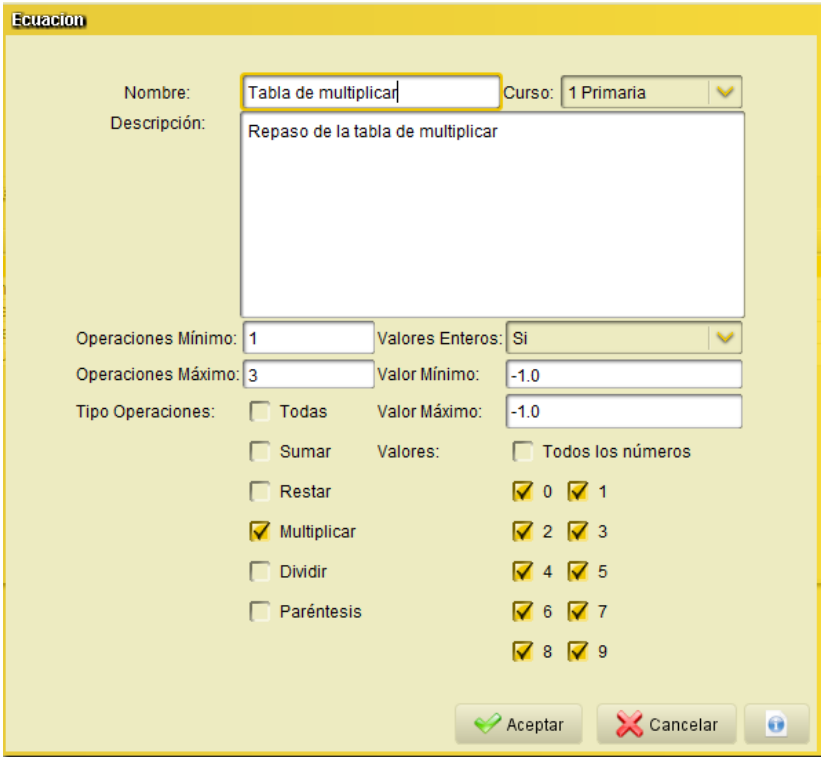


*Figura 22 – Pantalla ecuación*

# Anexo C: Manual de usuario alumno

La Figura 23 muestra la pantalla de acceso a la aplicación.

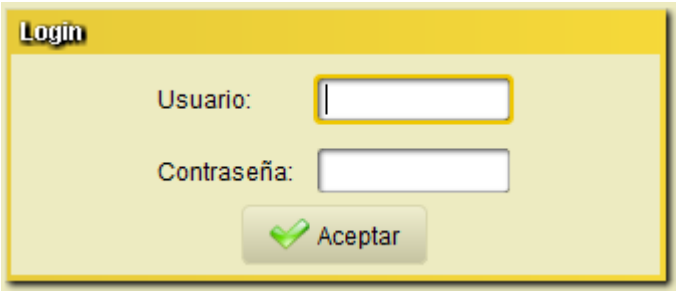


*Figura 23 – Pantalla acceso alumno*

En la Figura 24 se muestra la pantalla de inicio de la aplicación para los usuarios con perfil alumno. Desde esta pantalla se puede acceder a distintas pantallas como son: Datos Personales, Histórico y Resolver Problema.

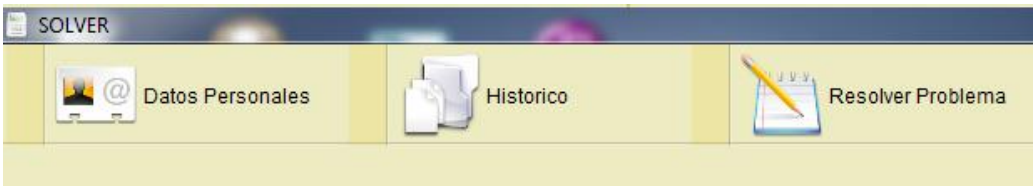


*Figura 24 – Pantalla inicio alumno*

La Figura 25 muestra la pantalla con la información de los datos personales, pudiéndose modificar los datos que se deseen oportunos.

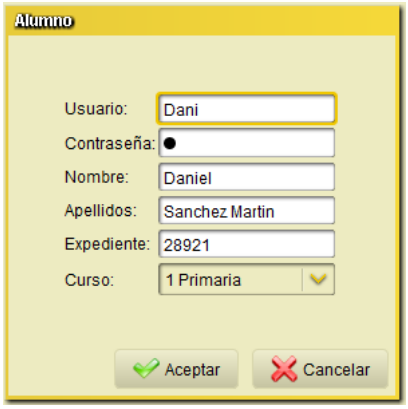


*Figura 25 – Pantalla datos personales*

La Figura 26 muestra una pantalla informativa sobre las acciones realizadas por el alumno en la aplicación.

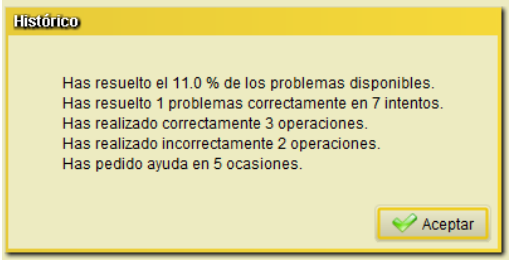


*Figura 26 – Pantalla histórico alumno*

La Figura 27 muestra la pantalla del usuario para resolver problemas.

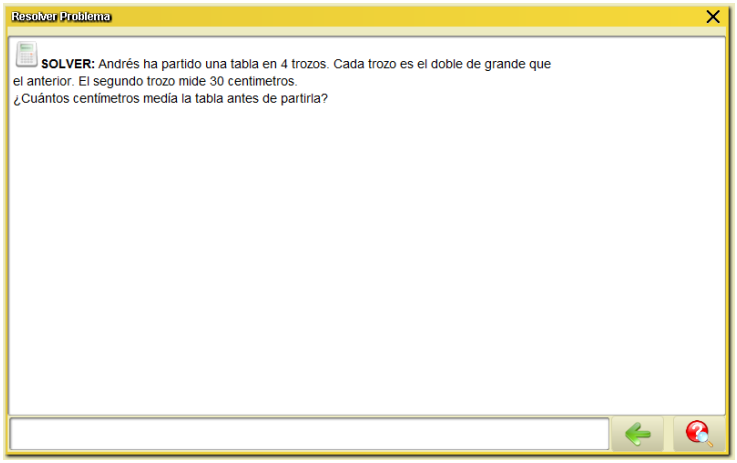


*Figura 27 – Pantalla resolver problema*

# Anexo D: Cuestionario de satisfacción

**Nombre:**

**Edad:**

**Curso:**

**Puntúe, de 0 a 10, las siguientes cuestiones:**

| | **Valoración** |
|---|---|
| ¿Le ha parecido fácil de utilizar la aplicación? | |
| ¿Le ha parecido agradable la aplicación? | |
| ¿Le ha parecido útil para practicar ejercicios? | |
| ¿Le ha parecido útil para mejorar vuestras habilidades? | |
| ¿Cree usted que este tipo de aplicaciones pueden tener futuro? | |
| ¿Le gustaría utilizar esta aplicación en vuestras clases escolares? | |

**Introduzca cualquier comentario que desee oportuno sobre la aplicación:**