UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Desarrollo de un Sistema de Corrección Automática de Programas Haskell

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Meritxell Sáez Povedano

Directores: Josep Silva Galiana
David Insa Cabrera

Curso 2015-2016

A mis padres y a mi hermana, por su apoyo incondicional y por depositar en mí toda su confianza.

A mi profesor, Josep, por haberme acompañado en mi última etapa de la carrera y por transmitirme sus valores y conocimientos.

Gracias de todo corazón.

Resumen

Este proyecto aborda el diseño e implementación de un sistema de corrección automática de ejercicios realizados en el lenguaje de programación Haskell. Concretamente, el sistema a desarrollar es una extensión del ya existente sistema ASys, el cual permite la corrección automática de ejercicios realizados en el lenguaje de programación Java. La extensión de ASys al lenguaje Haskell implica el uso de herramientas y compiladores totalmente diferentes a los usados en la versión actual de ASys. Implica a su vez un cambio tecnológico que ha sido definido como una generalización del sistema, de tal forma que en el futuro, ASys pueda ser adaptado a otros lenguajes con facilidad y soporte.

El objetivo inicial del nuevo sistema ASys es ser utilizado en la asignatura "*Lenguajes, Tecnologías y Paradigmas de la Programación*" (LTP) del Grado en Ingeniería Informática de la Universitat Politècnica de València. En dicha asignatura se pretende que los alumnos utilicen ASys como herramienta docente, la cual les permitirá autocorregir baterías de ejercicios que ellos mismos resolverán durante el curso.

Para la autocorrección de ejercicios, el sistema contará con un sistema de testing automático que asignará a cada test una puntuación, de tal forma que el conjunto de tests superados por un ejercicio resuelto por un alumno determinará su puntuación. Se ha optado por usar el compilador más extendido para Haskell en la actualidad, GHC (*Glasgow Haskell Compiler*), de entre muchos otros que dispone este lenguaje.

Palabras clave: lenguaje funcional, lenguaje máquina, Haskell, Java, compilador, LTP.

Resum

Aquest projecte abasta el disseny i implementació d'un sistema de correcció automàtica d'exercicis realitzats amb el llenguatge de programació Haskell. Concretament el sistema a desenvolupar és una extensió del sistema ja existent ASys, que permet la correcció automàtica d'exercicis realitzats en el llenguatge de programació Java. L'extensió de ASys al llenguatge Haskell implica l'ús d'eines i compiladors totalment diferents als utilitzats en la versió actual de ASys. Al mateix temps suposa un canvi tecnològic definit com una generalització del sistema de manera que ASys puga ser fàcilment adaptat amb el suport corresponent a altres llenguatges en el futur.

L'objectiu inicial del nou sistema ASys és la seua utilització a l'assignatura "*Llenguatges, Tecnologies i Paradigmes de la Programació*" (LTP) del Grau d'Enginyeria Informàtica. de la Universitat Politècnica de València. En aquesta assignatura es pretén que els alumnes utilitzen ASys com eina d'ensenyament amb la qual autocorregir bateries d'exercicis que ells mateixos resoldran durant el curs.

Per a l'autocorrecció d'exercicis, el sistema comptarà amb un sistema de testing automàtic, el qual assignarà a cada test una puntuació. Així, el conjunt de tests superats per un exercici resolt per un alumne determinarà la seua puntuació. S'ha optat per l'ús del compilador més estès

per a Haskell en l'actualitat, GHG (*Glasgow Haskell Compiler*), d'entre molts altres de què disposa aquest llenguatge.

Paraules clau: llenguatge funcional, llenguatge màquina, Haskell, Java, compilador, LTP.

Abstract

This project is about the design and implementation of an automatic correction system of exercises related to the Haskell programming language. In particular, such a system development is an extension of the existing ASys System that allows the automatic correction of the exercises for the Java programming language. The ASys extension for the Haskell language implies using tools and compilers totally different to those used in the current version of ASys. It also involves a technological change that has been defined as a generalization of the ASys system so that it may be easily adapted to other programming languages in the future.

The main aim of this new ASys extension is to be applied in the subject "*Lenguajes, Tecnologías y Paradigmas de Programación*" (LTP) that is part of the Grado en Ingeniería Informática (Computer Engineering Degree) at the Universitat Politècnica de València. ASys is a helpful tool for students, since it is able to correct automatically a large number of exercises set out in the aforementioned subject.

The system includes an automatic testing system for the automatic correction of exercises that will evaluate and give a mark to every test. Furthermore, the compiler used for this project is the most known one for Haskell today, which is called GHC (Glasgow Haskell Compiler), even though there exist other different compilers for Haskell as well.

Keywords : functional language, machine language, Haskell, Java, compiler, LTP.

Índice general

Índice de figuras

Índice de tablas

1. Introducción

En este primer punto se explican cada uno de los motivos que me han llevado a la realización de este trabajo, así como cada uno de los objetivos que se pretenden alcanzar junto con una breve descripción acerca de cada uno de los puntos que conforman la memoria de dicho trabajo.

1.1. Motivación

La asignatura "*Lenguajes, Tecnologías y Paradigmas de Programación*" (LTP) perteneciente al primer cuatrimestre de segundo curso de Grado en Ingeniería Informática fue el principal motivo que me llevó a generar la idea a partir de la cual se ha elaborado este trabajo.

Además, cabe destacar que mi interés hacia dicha asignatura se debe, en concreto, a un profesor, Josep Silva Galiana, quien también es mi tutor de mi Trabajo de Fin de Grado (TFG). No solo porque llevaba a cabo explicaciones que conseguían atraer la atención de los alumnos sino porque se aseguraba de que lo explicado era comprendido a la perfección.

Con esta asignatura, LTP, lo que se pretende es lo siguiente: facilitar el aprendizaje de un nuevo lenguaje, simular características en lenguajes que carecen de ellas, mejorar la habilidad de desarrollar algoritmos eficientes, mejorar el uso del lenguaje de programación disponible, aumentar el vocabulario del programador, facilitar el diseño de un nuevo lenguaje y, en general, dotar a los alumnos de una visión multiparadigma de la programación.

Por otro lado, el objetivo de la misma es introducir los conceptos fundamentales de los lenguajes de programación, presentar sus características y las principales aplicaciones de los paradigmas clave en los que se sitúan los lenguajes de programación, y las tecnologías de soporte. Entre uno de estos lenguajes se encuentra el lenguaje de programación puramente funcional, Haskell [1, 2, 3, 4], y el cual es el lenguaje aplicado para el desarrollo de mi trabajo.

1.2. Objetivos

Uno de los problemas a los que se enfrentan la mayoría de los programadores acostumbrados a trabajar con lenguajes imperativos es que en estos lenguajes el programador, explícitamente, le dice a la computadora cómo realizar una tarea paso a paso. Sin embargo, en un lenguaje de programación funcional como Haskell, en el que se parte de la solución a un problema, y no se parte del problema en sí, es decir, el programador solo se preocupa por lo que el programa calcula, no por cuándo ni cómo se calcula. Esto hace que Haskell sea un lenguaje más flexible y fácil de usar, aunque los inicios de su aprendizaje sea más complejo.

En este trabajo lo que se pretende es diseñar e implementar una aplicación destinada a los alumnos de la asignatura "*Lenguajes, Tecnologías y Paradigmas de Programación*" (LTP) para practicar el lenguaje de programación funcional Haskell, mediante una batería de ejercicios auto-evaluables, todo ello con el fin de adquirir los nuevos conceptos de este lenguaje de una forma sencilla, cómoda y atractiva.

La batería de ejercicios ha sido cuidadosamente diseñada para cubrir todos los aspectos estudiados en la asignatura, de tal forma que todos los temas tienen asociados ejercicios. Además, los ejercicios han sido revisados por un profesor de dicha asignatura para garantizar que la dificultad se ajusta a los estándares de los exámenes oficiales. Con todo ello, el sistema desarrollado junto con la batería de ejercicios auto-corregibles constituye un recurso docente de gran valor.

1.3. Estructura de la memoria

Este trabajo consta de doce puntos principales. A continuación, se procede a realizar una breve descripción de cada una de las tareas que se han llevado a cabo para el desarrollo de todos y cada uno de ellos.

- **Capítulo 1**, Introducción: este punto explica la motivación que ha llevado a la realización de este trabajo, así como los objetivos que se pretenden alcanzar.

- **Capítulo 2**, Introducción a la Programación Funcional: esta parte expone una breve explicación acerca de los orígenes del lenguaje de programación funcional Haskell, indicando además las características que caracterizan a este tipo de programación.

- **Capítulo 3**, Haskell: en este punto se pretende exponer el origen en la historia de Haskell así como su definición. Además, se explican las características que mejor definen a este lenguaje así como algunas de las extensiones de las que consta.

- **Capítulo 4**, Conceptos básicos: en este apartado se describen cada uno de los conceptos fundamentales que son de vital importancia para facilitar la comprensión al lector.

- **Capítulo 5**, Análisis previo: esta parte presenta el sistema y contexto inicial del que parte el trabajo realizado.

- **Capítulo 6**, Selección de tecnologías: aquí se procede a la definición de cada una de las tecnologías que se han empleado para el desarrollo de la aplicación.

- **Capítulo 7**, Metodologías empleadas: en este capítulo se explican cada una de las fases que componen el modelo que se ha utilizado como metodología a seguir.

- **Capítulo 8**, Especificación de requisitos: en este apartado se lleva a cabo una especificación completa de los requisitos que ha de cumplir el sistema.

- **Capítulo 9**, Análisis y diseño de la aplicación: este punto se centra en explicar el análisis y el diseño llevado a cabo con el fin de que la aplicación sea sencilla y de fácil comprensión por parte de usuarios inexpertos en el lenguaje Haskell.

- **Capítulo 10**, Implementación: aquí se muestran las herramientas que se han empleado para la elaboración del sistema desarrollado.

- **Capítulo 11**, Conclusiones: este penúltimo punto, expone las conclusiones obtenidas, así como una visión general del conjunto de funcionalidades implementadas para esta aplicación.

- **Capítulo 12**, Ampliaciones futuras: en esta última parte se proponen algunas de las funcionalidades que se pueden crear para extender y mejorar esta aplicación en un futuro. Estas implican un esfuerzo adicional de investigación y desarrollo.

Para finalizar, cabe destacar la existencia de un anexo al final de la memoria, en el cual se presenta un breve y sencillo manual de usuario para el manejo de la aplicación desarrollada.

1.4. Notas bibliográficas

En esta sección se va a realizar una breve descripción de los materiales bibliográficos que se han utilizado, así como su uso y relación en cada uno de los apartados de este documento.

Inicialmente, con el objetivo de situarnos dentro del contexto histórico de la programación declarativa y de los lenguajes de programación relacionados con la misma, hacemos uso de las referencias bibliográficas **[1, 9, 3, 27]**.

Para situar en contexto al lector de este documento, se explican cada una de las características del lenguaje Haskell así como las diferencias con otro tipo de programación y las aplicaciones que soportan los archivos pertenecientes a dicho lenguaje, para ello se han empleado las siguientes referencias bibliográficas **[2, 4, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 22, 45]**.

Por otra parte, precisamos de información acerca del lenguaje de programación Java en el cual está basado la implementación del sistema ASys [5]. Para ello se han empleado las referencias **[23, 24, 25, 26, 38, 43]**.

Además, para la explicación de cada una de las tecnologías y/o herramientas empleadas para la implementación se han seguido las siguientes referencias bibliográficas **[18, 19, 20, 21, 31, 32, 41, 44, 46]**.

Para la explicación de ciertos conceptos con las que se pueda estar poco familiarizado y que son de gran importancia, se han empleado las siguientes referencias **[28, 29, 30, 33, 39, 40, 42, 47, 49]**.

Asimismo, para explicar en qué modelo se basa este proyecto y bajo que requerimientos se ha desarrollado, se han empleado las siguientes referencias bibliográficas **[5, 34, 35, 36, 37]**.

Por último, para situar al lector en la actualidad de la programación en el capítulo de conclusiones, se ha requerido de la referencia bibliográfica **[48]**.

2. Introducción a la Programación Funcional

La programación funcional [6] pertenece al paradigma de programación declarativa, el cual hace uso de funciones matemáticas para la creación de programas.

Para la programación funcional, la computadora evalúa una expresión reduciéndola a su forma más simple, la cual es ejecutada mediante la evaluación de la expresión centrándose en la cuestión QUÉ se va a computar y no en el CÓMO se va a llevar a cabo.

Las raíces de la programación funcional se remontan al matemático Alonzo Church (Universidad de Princeton), el cual se interesó por la matemática abstracta y, con ello, desarrolló un lenguaje abstracto denominado Cálculo Lambda en los años 1930. Dicho lenguaje llevaba a cabo la evaluación de expresiones haciendo uso de funciones como mecanismo de cómputo. De hecho, en algunos de los lenguajes de programación funcionales, se puede observar cierta continuidad y/o evolución del Cálculo Lambda.

La gran mayoría de los lenguajes de programación funcionales, en concreto los puramente funcionales, son empleados en el ámbito escolar y aprendizaje del lenguaje. Sin embargo, existen muchos otros que se utilizan en el desarrollo comercial o industrial, como pueden ser Scheme, Erlang, Rust, Objective Caml, Scala, F# y Haskell.

Las características de los lenguajes de programación funcionales modernos se van a mostrar a través del lenguaje Haskell, lenguaje funcional puro que consta de las últimas innovaciones en este ámbito. Estas innovaciones son, por ejemplo, las funciones de orden superior, evaluación perezosa (*lazy evaluation*), tipos polimórficos estáticos, etc.

2.1. Modelo Funcional

El modelo de programación funcional especifica el QUÉ va a suceder mediante el uso de funciones matemáticas puras.

Aplicando dicho modelo, presuponiendo que existe una interacción entre usuario y máquina, el usuario introduce una expresión la cual es evaluada por la máquina (o sistema) evaluándola mediante un método de reducción hasta que esta sea irreducible. Por lo tanto, se concluye lo siguiente: el usuario es el que actúa como programador y la máquina (o sistema) actúa como intérprete y/o compilador, el cual se encarga de evaluar la expresión proporcionada por el programador (usuario).

El resultado de la evaluación está condicionado única y exclusivamente por los valores de los argumentos de la función, consiguiendo así que siempre se obtenga el mismo valor para una misma expresión.

Este modelo será la base de funciones y sistemas empleados para el lenguaje Haskell como son las funciones de orden superior, el polimorfismo, la inferencia de tipos, la evaluación perezosa, etc.

2.2. Funciones de Orden Superior

Una función de orden superior es aquella que permite tener funciones como parámetros así como devolver funciones como resultado.

El uso de este tipo de funciones proporciona ciertas ventajas como es su propia definición en sí, así como la flexibilidad que se aporta al programador.

2.3. Sistemas de Inferencia de Tipos y Polimorfismo

El Sistema de Inferencia de Tipos [7] es una característica del paradigma funcional, siendo esta más sencilla para lenguajes que no permiten realizar asignaciones, lo cual hace que sea más fácil de implementar en Haskell, lenguaje declarativo puro.

Este permite la posibilidad de no tener que declarar el tipo de las expresiones y, en el caso de declarar algún tipo de alguna función, el sistema se encarga de que el tipo declarado coincida con el tipo inferido.

No solo consta Haskell de Inferencia de Tipos, también otros lenguajes como Ada, Boo, C# 3.0, Cayenne, Clean, Cobra, D, Delphi, Epigrama, F#, haXe, JavaFX Script, ML Mythryl, Nemerle, OCaml, Oxygene, Scala.

Gracias a la existencia y uso del polimorfismo, estos sistemas, es decir, los Sistemas de Inferencia de Tipos, constan de más flexibilidad.

El polimorfismo permite a las funciones recibir para un mismo parámetro valores de tipos diferentes. Si se habla de polimorfismo paramétrico solo se consta de una función, sin embargo, en el polimorfismo ad-hoc se tienen varias definiciones para una misma función lo cual hace que esta permita distintos tipos.

La mayoría de las funciones de polimorfismo paramétrico son las que se emplean para operaciones sobre listas como pueden ser filter, foldl, foldr, map, all, etc. A continuación, se muestra un ejemplo en el cual se quiere escoger de entre varias secuencias, la de mayor longitud:

```
maxLongitud :: [[a]] → [a]
maxLongitud [xs] = xs
maxLongitud (xs:ys:xss) =    if length xs > length ys
                             then maxLongitud (xs:xss)
                             else maxLongitud (ys:xss)
```

El Sistema de inferencia de tipos infiere el tipo maxLongitud :: [a] el cual indica que la función maxLongitud tiene como argumento una lista de elementos de tipo a (donde a representa a cualquier tipo) y que devuelve un valor entero.

Si no se empleara el polimorfismo, en el ejemplo anterior se tendría que haber definido una función maxLongitud para cada tipo de lista. Con esto se concluye que el polimorfismo

proporciona la reutilización de código sin tener que definir múltiples funciones para una misma tarea.

2.4. Evaluación Perezosa

La evaluación perezosa [8] (del inglés *lazy evaluation*) es una estrategia de evaluación que permite retrasar el cálculo de una expresión hasta que sea necesario su valor, así como, evita repetir la evaluación si esta se requiere en ocasiones posteriores.

Algunos de los beneficios de esta evaluación son el aumento del rendimiento evitando posibles cálculos innecesarios, la capacidad de hacer estructuras de datos potencialmente infinitas y la capacidad de definir estructuras de control como abstracciones.

Permite, además, una ejecución más eficiente y un código claro y sencillo. Por ejemplo, el siguiente código en Haskell:

```
take 3 [1..]
```

lo cual significa coger los tres primeros números de una lista infinita de números empezando por el número uno. Esto finalizaría en un bucle infinito mientras que se crea la lista si se tratara de un lenguaje con evaluación voraz (en lugar de perezosa).

Por otro lado, la evaluación perezosa también puede reducir el consumo de memoria ya que, como he comentado anteriormente, las expresiones solo se evalúan cuando es necesario y, de igual modo, los valores solo se crean cuando son necesarios.

2.5. Diferencias entre Lenguaje Funcional y Lenguaje Imperativo

Algunas de las ventajas que ofrecen los lenguajes funcionales son la transparencia referencial, es decir, que no hay efectos laterales, ya que al ejecutar la función no varía nada fuera del entorno de la misma. Una función tiene transparencia referencial si para un valor de entrada se obtiene siempre el mismo valor de salida.

Otra ventaja es la evaluación perezosa definida anteriormente, únicamente se evalúa lo que se requiere en cada momento.

Por otro lado, la alta abstracción, lo cual incluye las funciones de orden superior y el polimorfismo comentado en apartados anteriores. La flexibilidad y la facilidad de pruebas y depuración. Esto es, se hace referencia a flexibilidad ya que los programas funcionales normalmente son más claros, concisos y limpios que los programas imperativos. En cuanto a la facilidad de pruebas y depuración, se obtienen programas más seguros gracias a la transparencia referencial lo cual hace que sea más complicado que se oculten los errores que puedan surgir.

Sin embargo, la programación funcional tiene algunos inconvenientes con respecto a los lenguajes imperativos. En principio, puede resultar más difícil de aprender aunque el entendimiento y el mantenimiento sí resulta más sencillo. Se han de tener conocimientos de una gran variedad de conceptos para poder dominar el lenguaje.

La ausencia de variables de estado es otro punto desfavorable a priori en el momento de desarrollar código ya que dificulta mucho el comienzo del mismo porque es más complejo

y se han de asentar bien todos los conceptos, lo cual no es lo habitual que sucede con los lenguajes imperativos.

Para finalizar, como último inconveniente es la falta de recursos (librerías, *frameworks*, etc.). Su disponibilidad no es comparable a la de los lenguajes imperativos, los cuales constan con gran cantidad de recursos.

2.6. La Crisis del Software

Fue en 1968 cuando Friedrich L. Bauer habló por primera vez de las dificultades y/o errores sucedidos durante la planificación, estimación de costes, productividad y calidad de un software en una conferencia elaborada por la OTAN (Organización del Tratado del Atlántico Norte). A esto se le conoce como la crisis del software [9], término que ya había sido empleado por Edsger Dijkstra en su libro *The Humble Programmer*.

Los programadores se dieron cuenta de que el producto que ofrecían no era fiable al cien por cien, por lo tanto, esto generó incertidumbre en los usuarios lo cual ponía en peligro su confianza en cuanto al sistema.

El origen del problema fue el cumplimiento de ciertos requisitos por parte del sistema ya que, en aquella época, la verificación formal de programas era de alto coste.

Por otro lado, la tecnología iba avanzando, lo cual conllevó un aumento en la complejidad del software, y esto provocó en los usuarios una exigencia mayor de prestaciones.

Algunas de las posibles soluciones al problema fueron plantear nuevos modelos de la Ingeniería del Software (como fue el caso de las Metodologías Orientadas a Objetos), proporcionar una verificación formal de programas menos costosa, elaborar técnicas de Síntesis de Programas, crear nuevas Arquitecturas de Computadores, y por último y más destacable para el tema que se trata, proponer un modelo de computación distinto al modelo actual en ese momento, el modelo imperativo.

Dentro de estos modelos de computación distintos al imperativo se encuentran la programación funcional (definida con anterioridad) y la programación lógica, la cual resuelve problemas mediante predicados y relaciones.

3. Haskell

3.1. Introducción

Haskell es un lenguaje de programación. Se trata de un lenguaje puramente funcional, polimórfico, de evaluación perezosa, y con características como el orden superior que lo hacen muy diferente a muchos otros lenguajes de programación.

Debe su nombre al matemático estadounidense Haskell Brooks Curry [10].

El logo que caracteriza a dicho lenguaje es el símbolo lambda ya que Haskell está basado en el cálculo lambda.

3.2. ¿Qué es Haskell?

Haskell es un lenguaje estándar de programación puramente funcional (no-estricto). Dicho lenguaje es breve, de fácil comprensión, no tiene errores de memoria, reutiliza código, tiene evaluación perezosa, ofrece otras formas para encapsular abstracciones de mucho poder así como dispone de funciones de primera clase, administra la memoria y dispone de un colector de basura siendo mínimo el coste de ejecución.

Además, este lenguaje dispone de una sintaxis expresiva y gran variedad de tipos primitivos (tipos enteros, punto flotante, booleanos, etc.).

Por otro lado, consta de una gama de compiladores e intérpretes los cuales son gratuitos. Entraré en detalle más adelante para explicar cada uno de ellos ya que gran parte de este proyecto se centra en el uso y aplicación de los mismos.

3.3. Utilidad de Haskell

Una de las principales ventajas de Haskell es que resulta un lenguaje barato y fácil de aprender, siendo esto favorecedor mayormente en el caso de tener que trabajar con sistemas software de gran tamaño.

Entre otras ventajas, Haskell [11] ofrece:

- Un importante incremento de productividad para el programador.
- Código más breve, sencillo y claro.
- Mayor seguridad, menos errores.
- Menor diferencia semántica entre el lenguaje y el programador.
- Desarrollo de aplicaciones en menos tiempo que con otros lenguajes.

3.4. Extensiones de Haskell

Algunas de las extensiones, librerías e implementaciones propuestas para Haskell son las siguientes:

FFI (*Foreign Function Interface*) [12]

Esta extensión permite que programas de Haskell puedan llamar a las funciones de otros idiomas permitiendo así la cooperación con programas escritos en otros lenguajes.

Es muy sencillo de usar, el caso más común es convertir el prototipo de una función en otro idioma al prototipo equivalente de Haskell.

Gofer (*Good For Equational Reasoning*) [13] — Adecuado para razonar ecuaciones

Se trata de una implementación de Haskell la cual fue desarrollada por Mark Jones en Oxford y con fines educativos.

Gofer carece de algunas características propias del lenguaje como es la cláusula que se deriva de las definiciones de tipos de datos, pero sí proporciona una característica no adoptada por Haskell la cual es la generalización de la lista por comprensión.

Este fue sustituido por el intérprete Hugs que será explicado con detalle en puntos más avanzados de este documento.

GPH (*Glasgow Parallel Haskell*)

Es una extensión de Haskell que admite la ejecución de programas paralelos en las mismas máquinas y en multiprocesadores individuales.

GPH es distribuido y desarrollado de forma paralela a GHC (*Glasgow Haskell Compiler*) [14, 15] el cual es un compilador nativo de código libre para el lenguaje Haskell. Sin embargo, la versión actual de GPH se basa en una versión muy antigua de GHC).

El método más simple para extraer el paralelismo de código puro es el uso de dos primitivas, *par* y *seq*. Siendo *par* para composiciones paralelas y *seq* para composiciones secuenciales. Si ambas primitivas se emplean de forma adecuada, un programa es capaz de ser evaluado de forma paralela.

PH (*Parallel Haskell*)

Es un lenguaje paralelo, variante del lenguaje Haskell. El modelo de evaluación se asemeja al empleado por Haskell, con evaluación impaciente y estructuras sintácticas para barreras, bucles y estructuras de almacenamiento.

Goffin

Se trata de una extensión de Haskell dirigida a la programación paralela y distribuida.

O'Haskell

Es una extensión orientada a objetos para Haskell el cual fue desarrollado en la Universidad Tecnológica Chalmers.

Incorpora tres características de gran importancia, una mónada de concurrencia, objetos reactivos con encapsulado y sistemas de tipos con subtipos los cuales funcionan como tipos de datos.

Existe una implementación disponible la cual es un derivado del intérprete Hugs denominada O'Hugs.

Eden

Eden es un lenguaje funcional paralelo que aporta una nueva perspectiva a la programación paralela. Ofrece el control suficiente para implementar de forma paralela sus algoritmos y además de forma eficiente liberándolos al mismo tiempo de los detalles de bajo nivel de la gestión de procesos.

Este lenguaje, a pesar de extender de Haskell, elimina la evaluación perezosa siempre que sea necesario para llevar a cabo el paralelismo.

3.5. Archivos Haskell

En este punto se va a explicar brevemente en qué extensión se guardan los archivos de código fuente Haskell así como las posibles opciones para la apertura y visualización de los mismos.

Los archivos con la extensión *.hs* [16] son aquellos archivos cuyo código fuente está escrito en lenguaje de programación Haskell. Comparando con el lenguaje Java, en el cual los archivos tienen la extensión *.java*, entre otras.

Existen distintas posibilidades para proceder a la visualización de este tipo de archivos. Se va a mostrar a continuación, una explicación breve acerca de cada uno de los programas que lo permiten:

Emacs

Emacs [17] es un editor de texto con gran variedad de funciones. Es uno de los más conocidos y más empleados en el mundo de los programadores. EMACS significa ***E**ditor **MAC**ro**S*** el cual fue desarrollado en el año 1975. Además, este editor es uno de los más potentes que existen en el mercado.

Emacs dispone de un resaltado de sintaxis. Algunas de sus funcionalidades son:

- Manipular palabras y párrafos mediante comandos.
- Ejecutar macros de teclado que contienen conjuntos de comandos de edición establecidos por el usuario.

Geany

Geany [18, 19] es un editor de texto que emplea las herramientas GTK[1] [20], siendo este un entorno de desarrollo integrado (IDE). Está basado en Scintilla[2] [21] y está disponible para varios sistemas operativos, como GNU/Linux, Mac OS X, Solaris y Microsoft Windows. Es software libre.

[1] GIMP Tool Kit (GTK) es una biblioteca del equipo GTK+ que contiene objetos y funciones para crear la interfaz gráfica de usuario. Más información aquí: *https://es.wikipedia.org/wiki/GTK*.
[2] Es un componente libre de edición de código fuente. Más información aquí: *https://es.wikipedia.org/wiki/Scintilla*.

gedit

Se trata de un editor de textos programado en C y Python, ambos lenguajes de programación. Es compatible con los siguientes sistemas operativos: GNU/Linux, Mac OS X y Microsoft Windows. Este dispone de herramientas para proceder a la edición del código fuente y, además, textos estructurados como es el lenguaje de marcado.

Este editor incorpora también un coloreado de sintaxis para algunos lenguajes de programación, además permite editar múltiples archivos al mismo tiempo.

Por otro lado, gedit [22] dispone de un corrector ortográfico multilenguaje.

Pueden darse algunos problemas para abrir este tipo de archivos a pesar de tener el software instalado en el ordenador. Las posibles causas pueden ser las siguientes:

Una asociación incorrecta del archivo HS en el registro.

El archivo HS que se intenta abrir está dañado.

El archivo HS puede contener un virus.

Las prestaciones del hardware del ordenador pueden ser insuficientes.

Los controladores no están actualizados.

4. Conceptos básicos

En esta parte se va a proceder a definir cada uno de los conceptos más relevantes empleados para la explicación de este proyecto y que no han sido definidos en la memoria más adelante. El objetivo de este apartado es facilitar la comprensión del texto por parte del lector, independientemente de su nivel de familiarización con las tecnologías empleadas o con el mundo de la Ingeniería Informática en general.

Java [23, 24, 25]: Es un lenguaje de programación concurrente[3] [26], orientado a objetos cuyo propósito principal para su diseño fue para tener las menos dependencias de implementación posibles. Apareció en 1995, pertenece al paradigma imperativo y, a día de hoy, es uno de los lenguajes de programación más usado, concretamente para aplicaciones "cliente-servidor" de web. Su última versión es *Java Standard Edition 8* y sus extensiones más comunes son .java, .class, .jar, .jad.

Programación imperativa [27]: en esta programación, el programa está compuesto de uno o más subprocesos denominados procedimientos, subrutinas o funciones formados por enunciados o instrucciones cuyo orden de ejecución depende de las estructuras de control que se empleen.

Lenguaje máquina [28]: es un código que puede ser interpretado directamente por el microprocesador. Dicho lenguaje es distinto para cada arquitectura de computadora, está compuesto de una serie de instrucciones que se ejecutan secuencialmente y que representan las acciones que podría llevar a cabo la máquina.

URL (*Uniform Resource Locator*): las siglas en español significan Localizador Uniforme de Recursos [29]. Están formados por una secuencia de caracteres, en base a un formato estándar empleado para designar recursos, como documentos u otros archivos en Internet por su localización.

WWW (*World Wide Web*) [30]: La Web o WWW cuyas siglas significan "gran telaraña mundial", es un sistema que sirve como medio de comunicación de información de archivos y otros objetos multimedia por medio de Internet. La Web está formada por páginas o sitios a los que se tiene acceso con el uso de un navegador.

API (*Application Programming Interface*): Es el conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser empleado por otro software como una capa de abstracción

DSL (*Domain specific language*) [31]: Se trata de un lenguaje específico de dominio, es decir, un lenguaje de programación cuyo fin es resolver un problema concreto y a su vez, proporciona una técnica para dar solución a una situación específica.

Testeo: Este procedimiento, es una investigación técnica de un producto el cual está sometido a una serie de pruebas para verificar su correcto funcionamiento y que además

[3] Un lenguaje de programación será concurrente si define y maneja diferentes tareas o hilos de ejecución dentro de un programa. Más información aquí:
http://informatica.uv.es/iiguia/LP/teoria/apuntes/cuatr1/tema3_1_concurrencia.pdf.

cumpla con unos estándares de calidad. A través del testeo, se pueden llevar a cabo muchos cambios los cuales son beneficiosos para el producto. Lo importante en esta fase es interpretar de una forma adecuada la información obtenida para así, sacarle el máximo partido al producto gracias a cada una de las pruebas a las que se somete.

Pruebas de caja blanca [32]: Se trata de un método de testeo gracias al cual se llevan a cabo diseños de casos de prueba sobre las funciones internas de un módulo, es decir, dichas pruebas están dirigidas a las funciones internas. Algunas de las técnicas empleadas para las pruebas de caja blanca son: la cobertura de caminos, pruebas sobre expresiones lógico-aritméticas, pruebas de camino de datos, comprobación de bucles, etc. Los ejemplos típicos de este tipo de pruebas son las pruebas unitarias (comprobar el correcto funcionamiento de los métodos, comprobar si existen variables inutilizables, etc.).

Pruebas de caja negra [32]: Se trata de un método de testeo que se emplea sobre la interfaz del software dejando a un lado el comportamiento interno y la estructura del producto.

Este tipo de pruebas tienen como objetivo que las funciones sean operativas, que la entrada haya sido aceptada de forma adecuada y, a su vez, que la salida producida sea correcta, y por último, que la integridad de la información externa se mantenga.

A diferencia de las pruebas de caja blanca, las pruebas de caja negra pretenden encontrar errores como funciones ausentes o mal empleadas, errores en la interfaz, errores en la estructura de datos, o bien, en el acceso a posibles bases de datos externas, errores de rendimiento y errores tanto de inicio como de fin.

Como valor añadido, cabe decir que existen dos tipos de prueba de caja negra: la prueba de partición equivalente y la prueba de análisis de valores límites.

JUnit: Es un conjunto de bibliotecas utilizadas en programación para hacer pruebas unitarias de aplicaciones Java. Permite realizar la ejecución de clases en Java de forma controlada.

CVS (*Concurrent Versions System*): Es una aplicación que implementa un sistema de control de versiones, conserva el registro del trabajo realizado, los cambios en los ficheros y permite que colaboren distintos desarrolladores.

Ant (*Apache Ant*): Es una herramienta usada para la realización de tareas mecánicas y repetitivas. Es un software para procesos de automatización de compilación.

Subversion: SVN (*Apache Subversion*) es una herramienta de control de versiones de código abierto basada en un repositorio [33] cuya función se asemeja a la de un sistema de archivos.

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 tradicional y el modelo de objetos de la aplicación, mediante archivos o anotaciones. Es software libre.

5. Análisis previo

El sistema a desarrollar debe extender un sistema ya desarrollado anteriormente denominado ASys [5, 34]. Dicho sistema proporciona una herramienta para la evaluación y corrección semiautomática de ejercicios y exámenes escritos en Java.

ASys permite cargar un ejercicio y la solución al mismo. A continuación, mediante un sofisticado proceso de análisis, informa de todos los errores detectados en tiempo de compilación, y en tiempo de ejecución, así como de errores de diseño (incumplimiento de las especificaciones del ejercicio).

Las principales contribuciones de ASys son las siguientes:

1. Una librería Java con métodos de extracción para la verificación de las propiedades en código Java.
2. Un DSL (*Domain Specific Language*), construido a partir de la librería, que permite la especificación de los exámenes y sus correspondientes plantillas de evaluación.
3. Una herramienta de evaluación semiautomática para los exámenes y ejercicios de Java. Esta herramienta es capaz de comparar la salida de las funciones desarrolladas por el alumno con las desarrolladas por el profesor, y también es capaz de verificar propiedades usando introspección e intercesión Java. El principal objetivo de este trabajo es extender la funcionalidad de ASys con el fin de que sea aplicado para otro de los lenguajes que han de usar los alumnos para la asignatura a la que hace referencia el proyecto.

Para llevar a cabo dicha extensión, se estudió detalladamente qué partes de la aplicación anterior se podían modificar o enlazar con el nuevo lenguaje y así, poder ampliar su funcionalidad.

No solo se va a extender su funcionalidad, sino que también se va a proceder a la modificación de la interfaz en base a las nuevas necesidades que se han generado para este trabajo.

Las áreas que van a ser modificadas son las siguientes:

- La aplicación únicamente funciona para la realización de ejercicios y exámenes en Java, por lo cual se va a modificar este aspecto para que funcione para evaluar ejercicios y exámenes en Haskell.

- En cuanto a la interfaz de ASys, esta será modificada en base a las nuevas necesidades del nuevo programa para Haskell. A continuación, se visualizan cada una de las pantallas pertenecientes al sistema para así, tener conocimiento sobre ello y poder comprender cada uno de los cambios realizados que serán visibles en el capítulo 8, Análisis y Diseño de la aplicación.

6. Selección de Tecnologías

Se ha de tener en cuenta que el proyecto que se ha desarrollado parte de una aplicación anterior (ASys).

Han sido evaluadas cada una de las tecnologías que fueron empleadas para la realización de ASys, siendo la más importante de ellas el propio lenguaje Java. Las extensiones que deben implementarse para ASys se realizarán usando las mismas tecnologías que ASys utiliza actualmente. De esta manera la interferencia con el funcionamiento de ASys será mínima, y la extensión será conservativa (ASys seguirá funcionando de igual manera en todos los dispositivos que ahora lo hace). Así pues, el proyecto seguirá dando uso de los mismos lenguajes de programación que ahora utiliza ASys, exceptuando el lenguaje Haskell que será una nueva incorporación al desarrollo.

Por tanto, los dos lenguajes principales que se utilizarán en la implementación del nuevo sistema son Java y Haskell:

Java: Es el lenguaje de programación más empleado actualmente y, además, era el idóneo ya que lo que se pretendía con el sistema anterior era la evaluación y corrección de ejercicios en código Java. Por otro lado, se mantiene dicho lenguaje para el desarrollo del nuevo proyecto ya que, se desarrolló una biblioteca de evaluación que utiliza la abstracción y meta-programación con funciones de Java para permitir la verificación de propiedades. Dicha biblioteca así como cada una de las funcionalidades desarrolladas en Java son necesarias para la nueva alternativa propuesta con Haskell.

Haskell: El sistema anterior es empleado, entre otras cosas, para la práctica de una batería de ejercicios en código Java. Sin embargo, en esta nueva versión, se ha llevado a cabo la elaboración de una nueva batería de ejercicios para la práctica del lenguaje de programación Haskell ya que el sistema va a ser usado por alumnos para el aprendizaje de dicho lenguaje.

7. Metodologías empleadas

En este capítulo, se va a explicar brevemente la metodología empleada para la realización de la aplicación, explicando las fases del modelo que se han seguido.

7.1. Modelo en cascada

El modelo en cascada [35, 36] es el más empleado en el desarrollo de software cuando no existe una presión inmediata por desarrollar un prototipo.

Dicho modelo se basa en cinco fases las cuales se han de llevar a cabo de forma secuencial, es decir, para comenzar una fase se ha de haber completado la anterior. Cada una de las etapas tiene una serie de metas y actividades. Uno de los inconvenientes es el siguiente: si se comete un error en alguna de las fases iniciales, este se arrastra hasta la fase final suponiendo un gran coste corregir dicho error. La secuencia de fases en la cual se basa el modelo en cascada es la siguiente:

1. **Especificación de requisitos**: se han de establecer los requisitos de cada uno de los elementos que compone el sistema, con ello se obtiene un análisis previo sobre las necesidades de dicho sistema, los objetivos a cumplir así como una visión de los resultados que se han de obtener.

2. **Análisis de los requisitos**: esta fase está relacionada directamente con la implementación, es decir, con decisiones técnicas como son la función, el rendimiento y cada una de las interfaces requeridas.

3. **Implementación**: en esta fase se lleva a cabo la codificación del software a través de un lenguaje de programación. En este caso, el diseño del software ha de traducirse de forma que este sea legible para la máquina.

4. **Pruebas**: generado el código en la fase anterior, se procede a realizar una serie de pruebas exhaustivas sobre el software para asegurar que se producen los resultados requeridos y que no se haya cometido ningún error.

5. **Implantación**: en esta fase se procede a la implantación del software en el sistema en el cual se va a utilizar.

Este modelo ha sido empleado para el desarrollo de este proyecto porque se trata de un modelo conocido y fácil de implementar y comprender, además de que requiere de menos capital y herramientas para que funcione correctamente y de forma óptima. Por otro lado, los requerimientos de dicho proyecto no iban a variar durante su desarrollo lo cual hace más ventajoso dicho modelo al especificar desde el inicio cada uno de los requisitos.

8. Especificación de Requisitos

8.1. Introducción

Para la especificación de los requerimientos software se va a llevar a cabo una especificación siguiendo el estándar IEEE 830 [IEEE 98]. El enfoque de dicha especificación será para el desarrollo de una aplicación en Java para la práctica del lenguaje de programación Haskell.

8.1.1. Propósito

El propósito de este documento es la realización de la especificación completa de los requisitos que se han de cumplir por la aplicación. Se va a llevar a cabo el desarrollo de cada uno de ellos con la correspondiente explicación de los requerimientos que ha de contener dicha aplicación así como el abordaje de las posibles limitaciones que esta pueda tener.

8.1.2. Ámbito

La aplicación a desarrollar está orientada para ser utilizada en un ámbito docente. Se accederá a la aplicación mediante su correspondiente descarga en la plataforma ofrecida a los alumnos por la Universidad Politécnica de Valencia, en este caso, PoliformaT. Por lo tanto, dicha aplicación ha de ser sencilla de instalar y de fácil comprensión.

La aplicación será de uso exclusivo para los alumnos y profesores de la asignatura "Lenguajes, Tecnologías y Paradigmas de Programación", teniendo disponible todas las opciones de la aplicación así como las instrucciones para su uso cada uno de los usuarios que van a emplear la herramienta.

8.1.3. Definiciones, acrónimos y abreviaturas

Usabilidad [37]: La Organización Internacional para la Estandarización (ISO) proporciona dos definiciones de usabilidad:

- Norma ISO/IEC 9126: la cual dice lo siguiente, "La usabilidad se refiere a la capacidad de un software de ser comprendido, aprendido, usado y ser atractivo para el usuario, en condiciones específicas de uso". Esta norma mide el grado de utilidad, la facilidad de uso así como la de aprendizaje y la apreciación del producto en un contexto determinado.

Accesibilidad: esto significa el grado de facilidad con lo que algo puede ser usado o accedido por los usuarios, haciendo hincapié sobre las personas que poseen alguna discapacidad.

Plantilla de ejercicios: En ASys, una plantilla de ejercicios es un fichero llamado "Template.java" el cual se encuentra en la carpeta del examen, y que especifica cómo corregir el ejercicio automáticamente con funciones Java.

8.2. Descripción general

En este apartado, se van a explicar los factores que afectan a la realización de este sistema y sus requerimientos.

8.2.1. Perspectiva del producto

El sistema a desarrollar debe proporcionar la información suficiente para asegurar el aprendizaje de los usuarios así como una guía para su correcta utilización.

El proyecto consta de dos partes:

1. Implementar una extensión de ASys para el lenguaje de programación Haskell.
2. Crear un repositorio de ejercicios en Haskell: dichos ejercicios auto-corregibles deben cubrir todos los aspectos básicos de la asignatura LTP.

El alumno dispondrá de una carpeta en la cual se encuentran las plantillas de los ejercicios a desarrollar con la información necesaria para que estos se puedan resolver. Así mismo, el sistema recibirá la información aportada por el alumno, es decir, su código de la solución del ejercicio realizado y, a partir de ello, el sistema llevará a cabo una evaluación del código verificando las propiedades de dicho código, es decir, una evaluación semiautomática del código del alumno. Un ejemplo de ello podría ser la comprobación de los tipos de variables, el resultado obtenido de las funciones, si se han contemplado todos los casos necesarios para las funciones, etc. Dichas propiedades de los ejercicios son establecidas por el desarrollador bajo la revisión de los profesores de la asignatura.

Por otro lado, cada uno de los ejercicios tienen su correspondiente enunciado en un archivo con formato PDF, una plantilla de ejercicios ASys que corresponde al archivo "Template.java" así como una serie de tests que deberán ser pasados por la solución del ejercicio aportada por el alumno y así obtener la máxima puntuación.

8.2.2. Funciones del producto

El sistema a implementar debe satisfacer las siguientes funcionalidades:

- **Permitir cargar un ejercicio**: la herramienta ofrecerá al usuario una carpeta en la cual este deberá aportar su documento con la solución desarrollada.

- **Realizar la corrección de un ejercicio**: gracias a esta funcionalidad, se podrá llevar a cabo la corrección de la solución del ejercicio aportada por el usuario.

- **Mostrar errores cometidos**: en caso de que la solución no sea la correcta, se mostrarán una serie de errores que permitirán al usuario guiar su solución. Algunos de estos errores son procedentes del compilador encargado de compilar el código que ha proporcionado dicho usuario. Otros errores serán detectados por la herramienta utilizando un proceso de testeo.

- **Mostrar consejos de ayuda**: junto con cada uno de los errores posibles que se puedan mostrar durante la corrección del ejercicio, aparecerán una serie de consejos que sirvan de ayuda al usuario para así obtener la solución correcta.

- **Establecer una puntuación**: para cada uno de los apartados en los que se divide el ejercicio, se aportará una puntuación al usuario por cada uno de ellos. El usuario tendrá acceso a cada una de las puntuaciones que irá obteniendo de cada uno de los ejercicios que haya realizado.

- **Generar un informe**: el usuario tendrá acceso a un informe generado por el sistema en referencia a las calificaciones obtenidas.

8.2.3. Características del usuario

Los usuarios que utilicen la herramienta serán estudiantes relacionados con la asignatura "Lenguajes, Tecnologías y Paradigmas de Programación" así como con el entorno de programación en el cual se desarrolla dicha asignatura.

Cada uno de los usuarios que accedan al sistema podrán disponer de todas las funciones que este ofrece (las cuales han sido descritas en el apartado anterior).

8.2.4. Obligaciones generales

Para el correcto funcionamiento de la herramienta es necesario que el usuario tenga el compilador de Haskell "GHC[4]" instalado para poder recibir la información que proviene del compilador, así como una máquina virtual Java [38] correspondiente al sistema operativo que se vaya a usar.

8.3. Requerimientos específicos

8.3.1. Requerimientos funcionales

El sistema que se va a implementar estará formado por varios módulos los cuales aportarán una serie de funcionalidades diferentes.

La herramienta mostrará únicamente un menú en la parte superior de la ventana y este estará presente en todo momento con los siguientes campos: ASys, Crear ejercicio y Puntuar ejercicios. Para este desarrollo, la funcionalidad de la pestaña Crear ejercicio no será empleada ya que este proyecto está orientado únicamente a la puntuación de ejercicios y no a la creación de los mismos.

Si el usuario carga el ejercicio correctamente, accederá a una pantalla donde se visualizarán seis módulos: Abrir instrucciones, Puntuar ejercicios, Abrir recursos, Ver informe y Salir del ejercicio.

Estos seis módulos nombrados con anterioridad son los siguientes:

- **ASys**: dentro de esta pestaña el usuario encontrará cuatro subpestañas:

  - *Sobre*: si el usuario accede a ella, se mostrará una ventana emergente en la cual se mostrará la información acerca del lugar de desarrollo así como datos personales de las personas responsables del sistema (Nombre, correo electrónico, número de teléfono, dirección, etc.).

  - *Ayuda*: a través de esta subpestaña, el usuario tendrá acceso a una guía de iniciación de ASys en los idiomas inglés y español.

[4] GHC (*Glasgow Haskell Compiler*) es un compilador nativo de código libre para el lenguaje de programación funcional Haskell. Más información aquí: *https://wiki.haskell.org/GHC*.

- *Idioma*: el usuario podrá escoger el idioma en que quiere visualizar el sistema (inglés, español o francés).

- *Salir*: esta opción permitirá salir al usuario del sistema.

- **Puntuar ejercicios**: dentro de esta pestaña el usuario podrá cargar el ejercicio a puntuar para llevar a cabo su carga en el sistema y así, poder acceder a los siguientes módulos:

  - *Abrir instrucciones*: al acceder el usuario sobre este módulo, se abrirá un documento en otra ventana en el cual se aportarán las instrucciones del ejercicio a puntuar.

  - *Puntuar ejercicios*: gracias a este módulo, el usuario tendrá acceso a la pantalla en la cual podrá visualizar su propio código así como cada uno de los posibles errores cometidos con sus correspondientes consejos de ayuda.

  - *Abrir recursos*: a través de esta opción, el usuario dispondrá de los recursos que sean necesarios para el desarrollo del ejercicio a puntuar.

  - *Ver informe*: en este módulo, el usuario accederá a una pantalla en la cual aparecerán cada una de las puntuaciones de los ejercicios que haya realizado en el sistema, así como por cada uno de los ejercicios aparecerán notas por apartados según en los que este se haya dividido por el creador de los mismos.

  - *Salir del ejercicio*: con esta opción el usuario accederá a la pantalla de inicio en la cual únicamente se muestra el menú superior con las tres pestañas nombradas al principio de este punto.

8.3.2. Requerimientos de interfaz externos

8.3.2.1. Interfaz de usuario

La interfaz gráfica debe estar relacionada con el concepto de usabilidad de forma que el usuario pueda interactuar con el sistema de manera fácil, rápida y con fluidez.

Los enlaces de acceso a cada una de las funcionalidades de la herramienta deberán de ser bien visibles, es decir, que resulten fáciles de encontrar.

8.3.2.2. Interfaz de comunicaciones

La herramienta a implementar se centra en la corrección de ejercicios en Haskell, por tanto es de vital importancia la comunicación entre el compilador de dicho lenguaje y el sistema.

Es una necesidad fundamental para el uso de la herramienta tener instalado el compilador “GHC” ya que el sistema lo que hace automáticamente es buscar la ruta donde se encuentra localizado el compilador instalado en la computadora para así poder llevar a cabo tanto la compilación como la ejecución de los ejercicios, por lo tanto, únicamente es

necesario que dicho compilador esté instalado en el ordenador donde se vaya a utilizar el sistema ASys.

8.3.2.3. Interfaces software

Es un requisito indispensable tener la máquina virtual de Java (JVM[5]) para poder ejecutar ASys, ya que esta se distribuye como un fichero ZIP[6] [39] que contiene un fichero JAR[7] [40] directamente ejecutable con JVM.

8.3.3. Requerimientos de eficiencia

Para este desarrollo, no existen requerimientos mínimos de eficiencia estipulados aunque la herramienta sí debe asegurar una correcta gestión de las funciones proporcionadas.

8.3.4. Obligaciones de diseño

El diseño de la herramienta es clave para que esta sea lo más clara y sencilla para los usuarios. Se ha de tener en cuenta que los usuarios que empiecen a utilizar este sistema estarán poco familiarizados con el lenguaje de programación Haskell en el cual se van a desarrollar los ejercicios, por lo tanto, el sistema ha de ser lo suficientemente claro para no generar frustración al usuario ya que este se encuentra en pleno aprendizaje de un nuevo lenguaje de programación.

8.3.4.1. Estándares cumplidos

No se ha definido un estándar en concreto pero sí se ha de cuidar mucho el diseño para que este garantice la satisfacción del usuario durante su uso.

[5] *Java Virtual Machine:* máquina virtual de proceso nativo ejecutable en una plataforma específica. Más información aquí: *https://es.wikipedia.org/wiki/M%C3%A1quina_virtual_Java.*
[6] *Zone Improvement Plan:* es un formato de archivo que soporta la compresión de datos sin pérdidas. Más información aquí: *https://en.wikipedia.org/wiki/Zip_(file_format).*
[7] *Java Archive:* es un tipo de archivo que permite ejecutar aplicaciones en Java. Más información aquí: *https://en.wikipedia.org/wiki/JAR_(file_format).*

9. Análisis y Diseño de la Aplicación

Teniendo en cuenta que esta herramienta puede ser usada por usuarios sin conocimientos avanzados de informática, se ha intentado desarrollar un sistema sencillo y de fácil comprensión.

El diseño de la herramienta se ha llevado a cabo con una usabilidad muy alta para que la comprensión de los usuarios no se vea afectada.

9.1. Diseño del programa

Las pantallas serán la parte central del sistema, ya que toda la información que nos pueda proporcionar se reflejará a través de las mismas así como en las cuales el usuario tendrá que interactuar para cualquier tarea que desee realizar dentro de las disponibles.

Se mantendrá una estructura de pantalla uniforme para evitar posibles confusiones al usuario y, manteniendo así un orden y una claridad.

En los siguientes puntos se van a explicar los aspectos que se han de tener en cuenta.

9.1.1. Espacio de pantalla

Para el diseño de las páginas, se va a intentar mostrar la cantidad de información necesaria para la pantalla en concreto siempre de forma ordenada y concisa.

Al ser una herramienta destinada al ámbito docente, cada uno de los módulos proporcionados por la misma estarán relacionadas con un dibujo que hará referencia al nombre de dicho módulo, de tal forma que sea más atractivo por el usuario. Cada una de las cajas que conforman los módulos son de tamaño suficientemente grande como para su fácil acceso y visualización.

A continuación, se va a explicar la distribución de cada uno de los objetos de las pantallas.

En la parte superior izquierda de las pantallas se encontrará un menú con tres pestañas distintas (se observa en la figura 1 de este documento). Este menú se encontrará en la misma ubicación en todas las pantallas facilitando así la navegación al usuario. Es la posición más esperada por los usuarios porque la lectura siempre se realiza de izquierda a derecha.

El título de las pantallas siempre se incluirá al principio de las mismas ubicado en la parte superior central.

Esta distribución ha sido estudiada y elaborada con el fin de facilitar al usuario su acceso y manejo del sistema.

9.1.2. Navegación

La herramienta permitirá el acceso a cualquier pestaña, subpestaña o módulo por parte del usuario únicamente pulsando sobre los mismos.

Todas las pantallas contendrán botones de regreso a pantallas anteriores para facilitar la navegación. Concretamente, existirán botones para regresar a la pantalla de menú (se

observa en la figura 2 del documento) la cual permita salir del ejercicio, o bien a la pantalla de menú (véase la figura 1 del documento) para salir de la herramienta.

Las pestañas contenidas en el menú superior izquierdo estarán dispuestas de manera vertical, ofreciendo diferentes opciones en cada una de ellas.

9.1.3. Tiempos de respuesta

Como tiempo de respuesta se entiende el tiempo que transcurre desde que se accede a la pantalla hasta que se realice la acción que tenga que suceder.

Lo ideal es que los tiempos de respuesta de los sistemas sean lo más corto posible, en este caso el tiempo de respuesta únicamente dependerá del compilador, ya que en el caso de que se tenga que compilar un código muy extenso, habría que esperar a que se compilara por completo y esto conllevaría un tiempo de espera por parte del usuario fuera de lo habitual.

Factores que van a influir en el tiempo de respuesta del sistema:

- **El rendimiento del compilador**: En principio, en base a cómo estén desarrollados los ejercicios para los usuarios, el código no será de gran extensión y, por lo tanto, el compilador tendrá un rendimiento muy eficiente y apenas apreciable por el usuario.

- **La velocidad del ordenador del usuario**: No debe suponer ningún problema porque cualquier ordenador que cumpla los requisitos mínimos explicados en el punto 7.3.3 del documento tendrá suficiente potencia para que el sistema se ejecute de forma correcta.

9.1.4. Conclusiones del diseño de programa

Como bien se ha comentado en los puntos anteriores, el diseño de las pantallas se ha realizado con el objetivo de que resulten sencillas y agradables para los usuarios, de tal forma que el tiempo de aprendizaje sea muy reducido.

En cuanto al tamaño de la pantalla, se ha escogido un tamaño acorde con las necesidades de la herramienta, es decir, el tamaño ideal para que se muestre la cantidad de información necesaria y para que el usuario lo vea con facilidad y no le resulte difícil la navegación.

9.2. Diseño del contenido

El contenido es una de las partes esenciales del diseño de una herramienta, es decir, los resultados que los usuarios esperan obtener forman parte del contenido de la propia herramienta.

Dicho contenido ha de ser conciso, es decir, no tener distracciones que puedan afectar a la atención del usuario. Si esto no se cumpliera, se podría generar una situación de frustración para el usuario y así, no sentirse cómodo con el uso de la herramienta.

9.2.1. Lenguaje claro

Toda aplicación y/o herramienta ha de contar con textos escritos de forma clara y entendible para los usuarios a los cuales va dirigida.

En el caso de este sistema, se va a emplear bastante texto escrito, no solo en las explicaciones de cada uno de los ejercicios sino en el código que el usuario desarrolle así como la guía de iniciación que aporta la herramienta.

Uso del texto en cada una de las zonas del contenido:

- La cabecera de cada una de las pantallas contendrá un texto aportando la información del título de la misma, es decir, de en qué pantalla se encuentra el usuario.

- Los botones, ya sean de menú o de módulos, contendrán un texto claro de la función que realizan (además, en el caso de los módulos irán acompañados de una imagen representativa como se ha nombrado en puntos anteriores).

- En cuanto a la fuente utilizada para cada uno de los textos que aparecen en la herramienta cabe destacar:

    - El tamaño de la fuente es variable según en qué tipo de pantalla o contexto se encuentre, es decir, el tamaño correspondiente a documentos es diferente al que aparece en los botones.

    - El color empleado es mayormente color negro, excepto en alguna información que se aporta al usuario en la que el color de la fuente es azul para destacar de la tonalidad gris de la herramienta así como del color negro generalmente empleado.

9.2.2. Imágenes

Esta aplicación consta de una serie de imágenes que hacen referencia a cada uno de los módulos visualizados por el usuario. Estas imágenes estarán en formato PNG[8] [41].

Algunas de las características del formato PNG (*Portable Network Graphics*) son:

- Admite imágenes indexadas con transparencia de un bit o "binaria", además, no requiere de un canal adicional.

- Admite formatos con profundidad de color de millones de colores aportando una amplia gama de rangos de color más precisos y canal alfa[9] [42].

- No soportan animación.

[8] Gráficos de Red Portátiles es un formato gráfico basado en un algoritmo de compresión sin pérdida para bitmaps no sujeto a patentes. Más información aquí: *https://es.wikipedia.org/wiki/Portable_Network_Graphics*

[9] La opacidad de un píxel en una imagen. Más información aquí: *https://es.wikipedia.org/wiki/Composici%C3%B3n_alfa*.

- Las imágenes PNG pueden ser transparentes.

La herramienta dispondrá de imágenes que generalmente no superan los 8 Kb a excepción de alguna imagen cuyo tamaño alcanza los 586 Kb, a pesar de ello, esto no supondrá ningún esfuerzo grande de carga de la pantalla.

9.2.3. Conclusiones sobre el contenido

En esta herramienta la parte más importante del contenido para el usuario serán las imágenes representativas de cada uno de los módulos, de forma que sean representativas y claras para los mismos.

Las imágenes serán de un tamaño acorde a la pantalla y al texto que las representa. Dicho texto servirá como orientación al usuario y, por tanto, van a ser muy cortos para que sean lo suficientemente claros y sencillos.

9.3. Diseño del sitio

Este diseño sí es el más influyente en cuanto a la usabilidad de la herramienta se refiere.

Un buen diseño y estructura harán que el usuario no pierda el hilo de las acciones que está realizando además de sentirse cómodo durante su uso.

En todo momento de su navegación, se ha de poder volver atrás en cualquier pantalla en la que se ubique el usuario.

9.3.1. La pantalla principal

Como pantalla principal, es decir, pantalla en la cual estará situado el usuario cuando entre a la herramienta, le aparecerá el logo de la aplicación así como un menú superior izquierdo que le permita empezar a usar la herramienta (véase figura 1 de este documento).

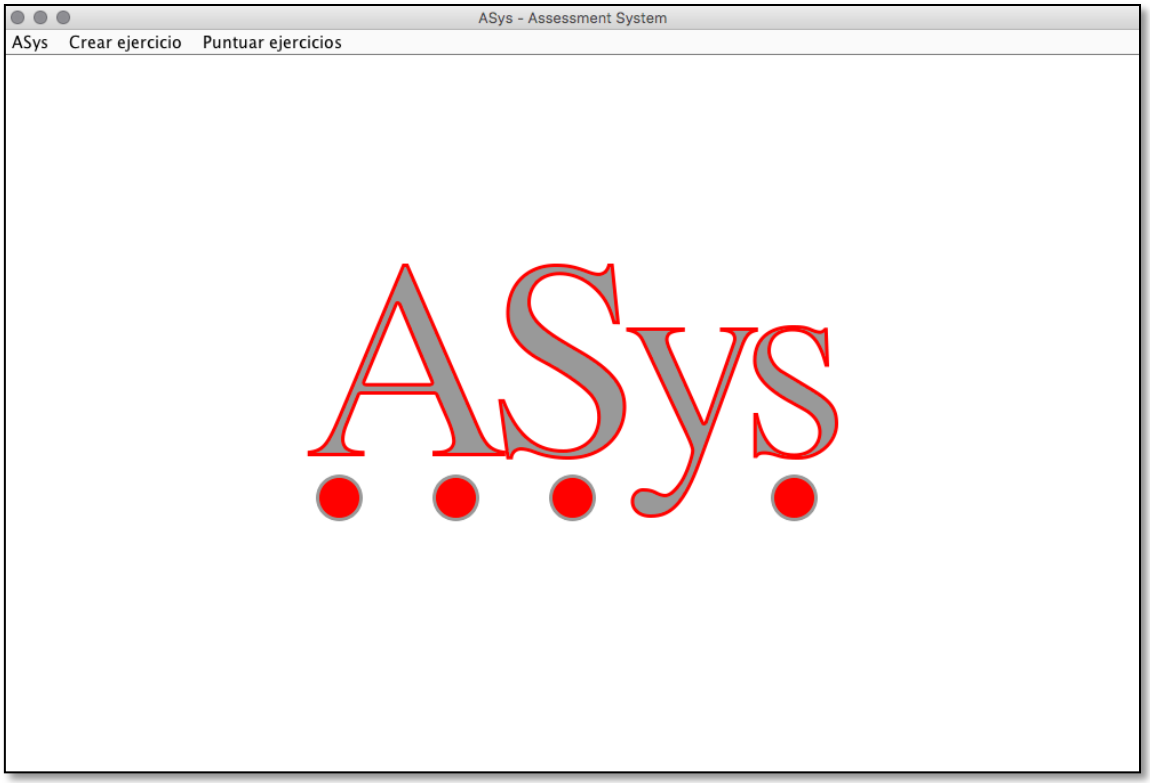


Figura 1: Pantalla principal

En esta pantalla será en la única en la que se visualice el logo de la herramienta, aparecerá en la parte central. Por otro lado, para interactuar con la misma solo se dispone del menú superior izquierdo.

9.3.2. La pantalla de bienvenida

Para esta aplicación no sería necesaria una pantalla de bienvenida ya que está dirigida a un número no muy elevado de usuarios y, además éstos ya conocen a qué está destinada dicha herramienta y no necesitarían una pantalla de bienvenida.

9.3.3. La pantalla de menú

A continuación se muestra la pantalla a la cual accederá el usuario al cargar un ejercicio por medio del menú superior izquierdo aportado desde la pantalla de inicio (véase figura 2).

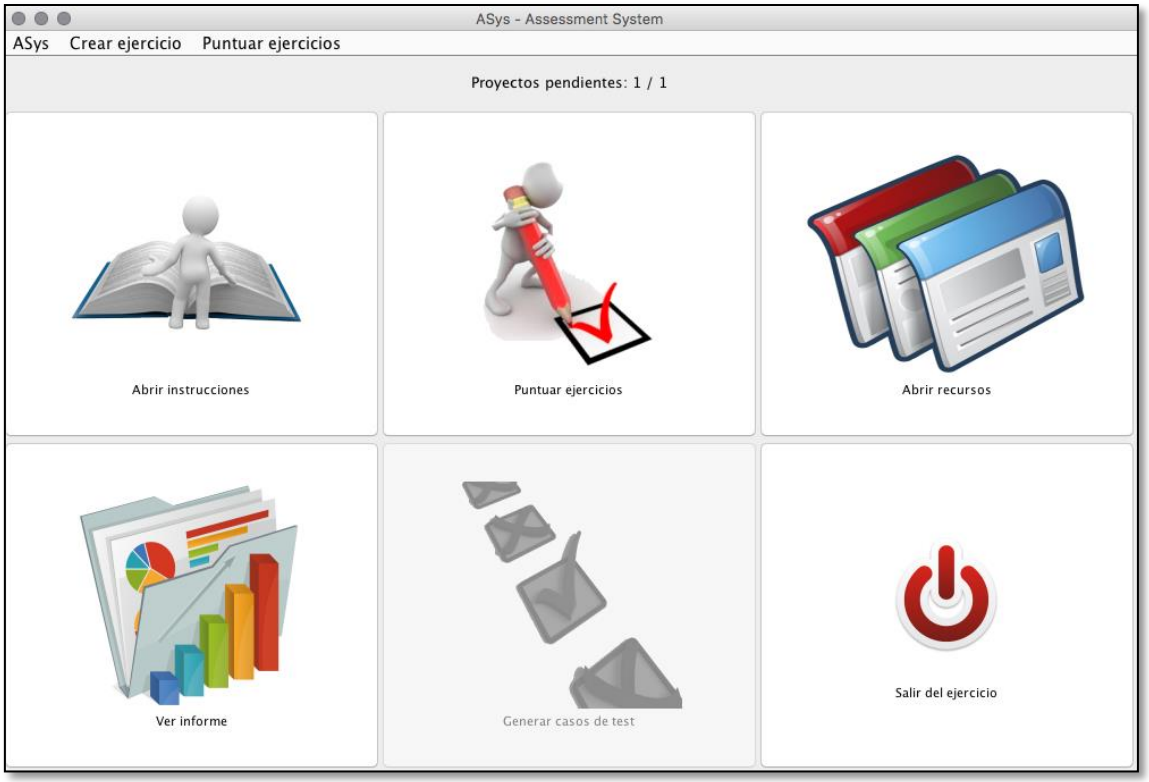


Figura 2: Pantalla de menú

Como se ha nombrado con anterioridad, cada uno de los módulos van asociados a una imagen representativa del mismo. En esta pantalla se sigue conservando el menú superior izquierdo a través del cual, entre otras acciones, el usuario podrá salir de la herramienta.

Por medio de esta pantalla no solo el usuario ya tendrá cargado el ejercicio de forma correcta sino que, si lo desea, puede salir del ejercicio y volver a cargarlo de nuevo, o bien cargar otro diferente.

9.3.4. La pantalla de corrección de ejercicio

La pantalla que se va a mostrar en este punto es la correspondiente a la puntuación de ejercicios, es decir, donde el usuario podrá visualizar su ejercicio así como la solución aportada por el profesor de dicho ejercicio. Además, el usuario podrá visualizar los posibles errores cometidos y los consejos correspondientes.

Una vez cargado un ejercicio se abre la siguiente ventana (véase figura 3).

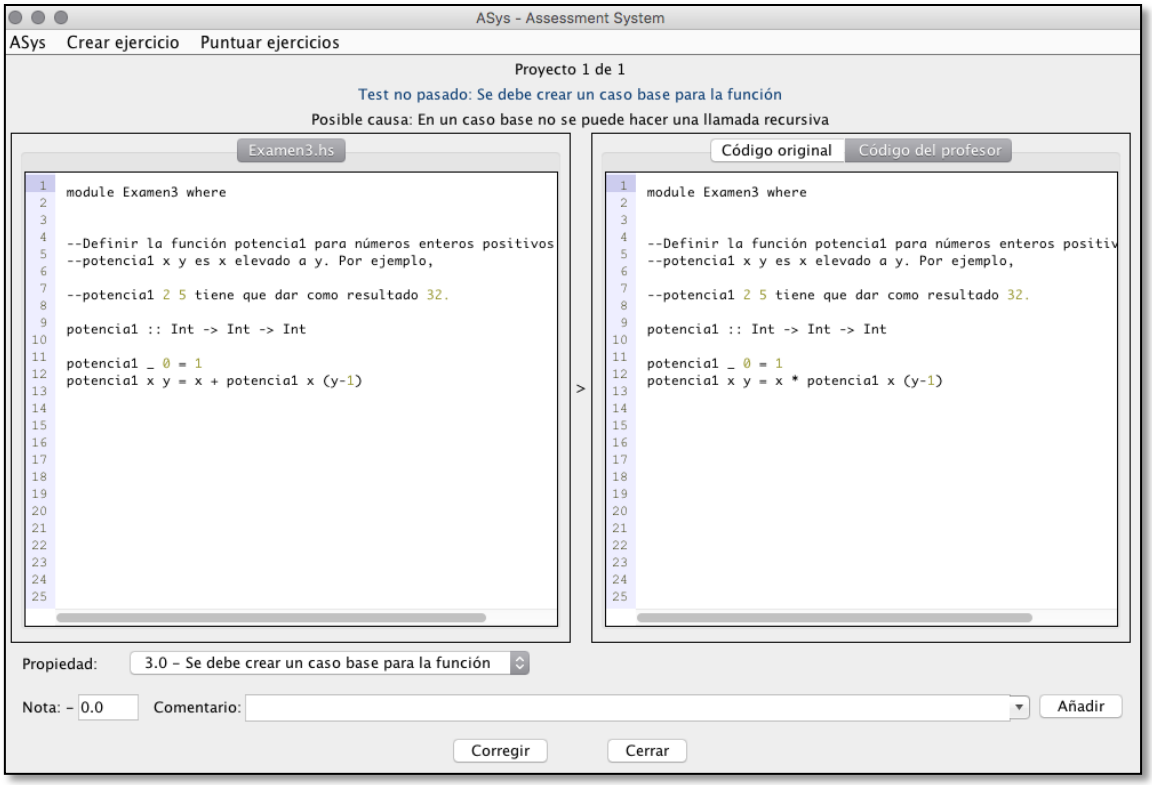


Figura 3: Pantalla de corrección

Funciones que se llevan a cabo en esta pantalla:

- **Mensaje de error**: Los errores que detecte ASys en el código se mostrarán en esta ventana. Hay tres tipos de errores:
    - *Errores de compilación*: Se muestra el mensaje de error devuelto por el compilador. Si se deja el ratón sobre el error se muestra más información.
    - *Errores de testeo*: El ejercicio no supera algún caso de test.
- **Ventana de corrección** (figura 3 del documento): Las clases con la solución propuesta por el alumno se muestran aquí. Los cambios realizados por el usuario para solucionar los problemas detectados se hacen aquí.
- **Ventana del código original** (figura 3 del documento): Tanto el examen original del alumno, como la solución del profesor se muestran aquí. Las clases con la solución al ejercicio se muestran aquí en la pestaña (*Código del profesor*). Las clases con la solución inicial del alumno se muestran en la pestaña (*Project code*).
- **Clases**: En cualquier momento se puede hacer control más clic (comando más clic en Mac) sobre el nombre de alguna clase para abrirla.
- **Coloreado del error**: La parte detectada por el compilador como error se colorea automáticamente para resaltarla.

- **Propiedad**: Cada error detectado es asociado a una propiedad que no se cumple. Algunas propiedades son especificadas por el profesor al crear el ejercicio, por ejemplo, que el parámetro de entrada de una función sea de un tipo concreto y el alumno ha empleado otro tipo distinto. Otras propiedades son predefinidas, como puede ser "Error de compilación".

- **Nota y Comentario**: En estas cajas de texto se puede asociar una nota (negativa) al error que se ha detectado. Con el botón "Añadir" se pueden asociar tantas notas y anotaciones (con una explicación de esas notas) como se desee a la propiedad que hay seleccionada (obviamente, no se puede restar más puntuación de la que tiene asignada esa propiedad).

- **Corregir**: El botón "Corregir" debe pulsarse cada vez que hayamos solucionado el problema detectado (por ejemplo, mirando la solución en el panel de la derecha) y hayamos (opcionalmente) asignado una nota a ese error.

La imagen correspondiente a la figura 3 muestra un claro ejemplo de solución del alumno y solución del profesor, así como cada uno de los apartados que componen la pantalla.

Como se puede observar, en la parte inferior existen dos botones: Corregir y Cerrar. El botón "Cerrar" permitirá al usuario volver a la pantalla de menú correspondiente a la figura 2 del documento.

9.3.5. La pantalla de calificaciones

La pantalla que se muestra en este punto (figura 4) es la correspondiente a la puntuación de ejercicios, es decir, donde el usuario podrá visualizar su ejercicio así como la solución aportada por el profesor.

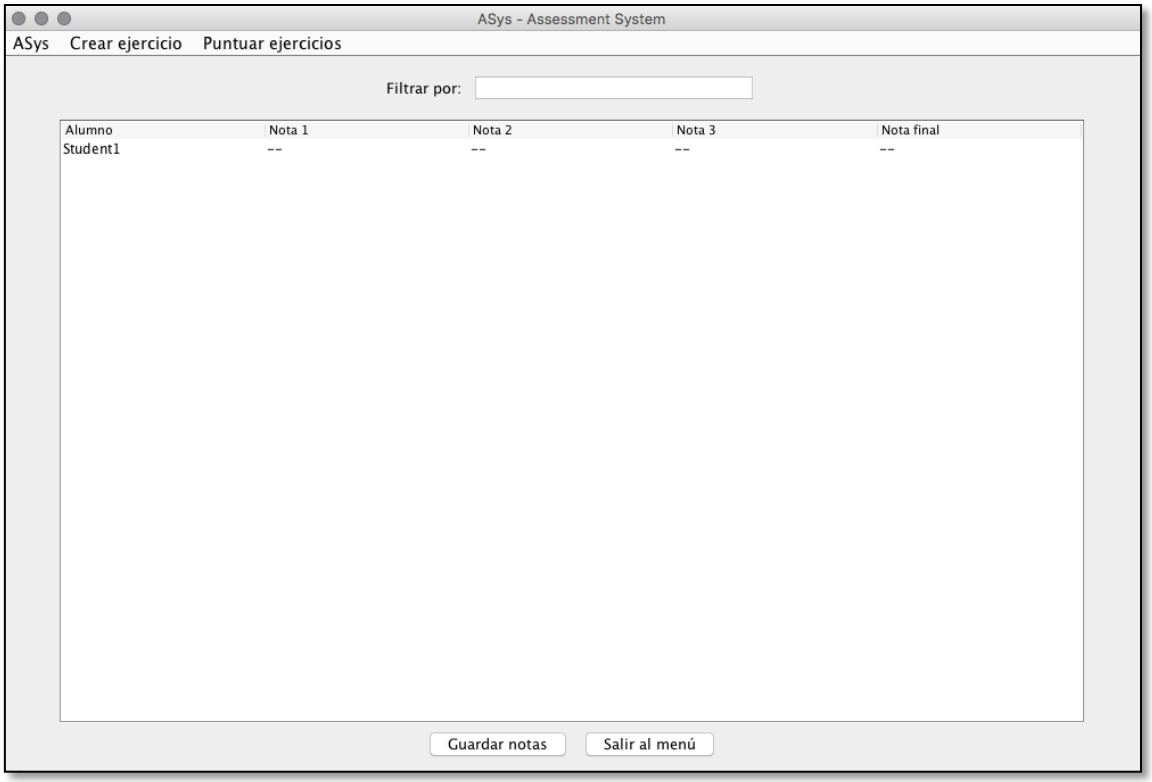


Figura 4: Pantalla de calificaciones

Se mostrarán las notas de cada uno de los apartados por los que esté compuesto el ejercicio, para este ejemplo, el usuario Student1 va a puntuar un ejercicio el cual está compuesto por tres partes y de las cuales obtendrá una calificación por cada una. En la última columna se le mostrará la nota final obtenida en el ejercicio evaluado.

Se observa en la parte inferior central dos botones, “Guardar notas” y “Salir al menú”. Se sigue permitiendo al usuario el retroceso a la pantalla de menú (véase figura 2 del documento).

10. Implementación

En este capítulo se pretende aportar cada una de las herramientas que se han empleado para el desarrollo de la herramienta. Además, se hará una explicación de cada uno de los módulos que proporciona el sistema dando detalle de su realización y de los problemas que han surgido durante su desarrollo.

10.1. Tecnologías de soporte a la aplicación

En este apartado se van a definir los elementos que ofrecen soporte a la herramienta para que así esta pueda funcionar correctamente, así como los distintos lenguajes empleados en la programación del sistema.

10.1.1. Java

Java es un lenguaje de programación así como una plataforma informática la cual fue comercializada por primera vez por Sun Microsystems en el año 1995.

Es un lenguaje de programación de propósito general[10], concurrente y orientado a objetos[11] cuyo objetivo es reducir el máximo número posible de dependencias de implementación. Fue desarrollado por James Gosling[12] [43].

Los principales objetivos de James Gosling eran implementar una máquina virtual y, por otro lado, un lenguaje similar a C++ [44].

La sintaxis de este lenguaje deriva principalmente de los lenguajes C y C++[13]. Las aplicaciones de Java son generalmente compiladas a *bytecode*[14] las cuales se pueden ejecutar en cualquier máquina virtual Java (JVM) sin tener importancia la arquitectura de la computadora.

En cuanto al nombre de JAVA existe una gran polémica en cuanto a su origen se refiere. Existen distintas teorías pero ninguna de ellas se puede considerar oficial.

El lenguaje Java fue creado con cinco objetivos principales:

1. El uso del paradigma de la programación orientada a objetos.
2. Permitir la ejecución de un mismo programa en distintos sitios.
3. Incluir por defecto soporte para trabajar en red.
4. Ser diseñado para ejecutar código de forma segura.

[10] Dícese de los lenguajes de programación que pueden ser empleados para varios propósitos (acceso a bases de datos, comunicación entre computadoras, comunicación entre dispositivos, captura de datos, cálculos matemáticos, diseño de imágenes o páginas…). Más información aquí: *https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n).*

[11] La programación orientada a objetos está basada en la herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento. Los objetos manejan los datos de entrada para obtener datos de salida concretos, donde cada objeto ofrece una función especial. Más información aquí: *https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n).*

[12] (19 de mayo de 1955, Calgary, Alberta, Canadá) Científico de la computación licenciado en ciencias de la computación por la Universidad de Calgary y doctorado por la Universidad Carnegie Mellon. Más información aquí: *http://www.ecured.cu/James_Gosling.*

[13] Lenguaje de programación diseñado a mediados de los años 1980 cuyo propósito fue permitir la manipulación de objetos. Más información aquí: *https://es.wikipedia.org/wiki/C%2B%2B.*

[14] Es el tipo de instrucciones que recibe la máquina virtual Java (JVM) y que posteriormente serán compiladas a lenguaje máquina por un compilador. Más información aquí: *https://es.wikipedia.org/wiki/M%C3%A1quina_virtual_Java*.

5. Sencillo de usar y coger lo más ventajoso de otros lenguajes orientados a objetos como es el caso de C++.

Actualmente, Java es uno de los lenguajes más usados del mundo. Con una gran comunidad y más de cuatro millones de desarrolladores, además de existir millones de dispositivos que lo usan.

10.1.2. Haskell

Haskell es un lenguaje de programación funcional[15] [45], puro[16], de evaluación perezosa[17], de tipado estático[18] y con inferencia de tipos[19].

A continuación se van a proceder a aportar una explicación detallada de cada una de las características de este lenguaje:

- **Tipado estático**: cada expresión tiene un tipo el cual se determina en tiempo de compilación. Todos los tipos compuestos juntos para una función tienen que coincidir, si no lo hacen, el programa será rechazado por el compilador. Los tipos no son solo una forma de garantía, sino un lenguaje para expresar la construcción de programas.

  Se va a exponer un ejemplo sobre esta característica:

  Todos los valores tienen un tipo en Haskell, véase el siguiente programa, el cual verifica si el valor entero que se pasa como parámetro es positivo.

```
esPositivo :: Int → Bool

esPositivo x = x > 0
```

  Se ha de pasar el tipo correcto de los parámetros de la función o el compilador rechazará el programa, por ejemplo, si se introduce lo siguiente:

```
esPositivo 5.5                                    Error
```

- **Puramente funcional**: cada función perteneciente a Haskell es una función en el sentido matemático. Únicamente se tienen expresiones que no pueden mutar variables (locales o globales), es decir, no existen ni declaraciones ni instrucciones.

[15] Dícese de los lenguajes de programación que están constituidos únicamente por definiciones de funciones. Más información aquí: *https://wiki.haskell.org/Es/Introduccion#.C2.BFQu.C3.A9_es_Haskell.3F*
[16] Dícese de los lenguajes de programación que contienen funciones que no tienen efectos colaterales. Más información aquí: *https://wiki.haskell.org/Es/Introduccion#.C2.BFQu.C3.A9_es_Haskell.3F*
[17] Del inglés (*lazy evaluation*) es un mecanismo de evaluación que retrasa el cálculo de una expresión hasta que se requiera su valor. Más información aquí: *https://es.wikipedia.org/wiki/Evaluaci%C3%B3n_perezosa*.
[18] Se determina el tipo de todas las expresiones antes de la ejecución del programa. Más información aquí: *https://wiki.haskell.org/Es/Introduccion#.C2.BFQu.C3.A9_es_Haskell.3F*.
[19] La inferencia de tipos hace referencia a algoritmos que deducen en tiempo de compilación y de forma automática el tipo asociado de un objeto en uso del programa. Más información aquí: *http://uqbar-wiki.org/index.php?title=Inferencia_de_tipos.*

- **Inferencia de tipos**: no se han de escribir de forma explícita todos los tipos en un programa. Aun así, si se desea, se pueden escribir tipos, o bien, pedírselos al compilador.

- **Concurrente**: Haskell es un lenguaje de programación concurrente debido a su manipulación explícita de los efectos. Su compilador más empleado actualmente, GHC, contiene una biblioteca de alto rendimiento y concurrencia de peso ligero que contiene un conjunto de primitivas de concurrencia útiles y abstracciones.

- **Perezoso**: esto es, las funciones no evalúan sus argumentos. Estos se evalúan donde y cuando se necesite su valor. La pureza de este lenguaje de programación hace que resulte más sencillo fusionar cadenas de funciones lo cual aporta beneficios al rendimiento.

- **Paquetes**: Haskell contiene una gran variedad de paquetes que se encuentran disponibles en los servidores públicos de paquetes. Véase tabla 1 de este documento.

Nombre	Explicación	Nombre	Explicación
bytestring	Datos binarios	base	Prelude, Entrada/Salida, hilos
network	Redes	text	Texto Unicode
parsec	Biblioteca analizador	directory	Directorio de archivos
hspec	Pruebas RSpec-like	attoparsec	Analizador rápido
monad-logger	Registros	persistent	ORM base de datos
template-haskell	Meta programación	tar	Archivos tar
snap	Framework web	time	Fecha, hora, etc.
happstack	Framework web	yesod	Framework web
containers	Mapas, gráficos, juegos	fsnotify	Reloj de sistema de archivos
hint	Interpretar Haskell	unix	Asignaciones de UNIX
SDL	SDL vinculante	OpenGL	Sistema de gráficos
criterion	Evaluación comparativa	pango	Representación de texto
cario	Gráficos	statistics	Análisis estadístico
gtk	Librería GTK+	glib	Biblioteca Glib
test-framework	Marco de pruebas	resource-pool	Puesta en común de recursos
conduit	Streaming[20] de Entrada/Salida	mwc-random	Azar de alta calidad
QuickCheck	Ensayo de propiedades	stm	Enhilado atómico
blaze-html	Generación de marcado	cereal	Análisis sintáctico binario/impresión
xml	XML analizador/impresora	http-client	Motor de cliente HTTP
zlib	zlib / gzip / prima	yaml	YAML analizador/impresora
pandoc	Conversión de marcado	binary	Publicación por entregas
tls	TSL / SSL	zip-archive	Compresión zip
warp	Servidor web	text-icu	Codificaciones de texto
vector	Vectores	async	Concurrencia Asyn
pipes	Transmisión de Entrada/Salida	scientific	Tipos numéricos científicos

[20] Transmisión de datos o descarga continua.

process	Procesos de lanzamiento	aeson	JSON analizador/impresora
dlist	Difflists	syb	Sistema de los genéricos

Tabla 1: Paquetes de Haskell

10.1.3. Lenguaje de Dominio Específico (DSL) para la autocorrección

En la implementación de ASys se ha utilizado un DSL desarrollado específicamente para la autocorrección de ejercicios. El DSL está construido sobre una biblioteca de funciones Java que puede ser reutilizado en cualquier otro software.

La biblioteca de funciones Java desarrollada utiliza la abstracción y la meta-programación avanzada de Java para permitir la verificación de propiedades. Esta biblioteca proporciona un API compuesto de setentaisiete métodos que podrán ser usados para inspeccionar y analizar el código fuente. Asimismo, el DSL permite la especificación y evaluación automática de plantillas de examen auto-corregibles.

10.1.4. Aplicaciones utilizadas para la programación

En este apartado se explican brevemente las aplicaciones empleadas para el desarrollo de la herramienta.

10.1.4.1. Eclipse

Se trata de una plataforma software formada por una serie de herramientas de programación de código abierto multiplataforma para implementar aplicaciones. Dicha plataforma ha sido empleada para desarrollar entornos de desarrollo integrados como JDT (*Java Develpment Toolkit*) y el compilador (ECJ) que se encuentra como parte de Eclipse [46]. Además, es también una comunidad de usuarios.

Eclipse fue originalmente desarrollado por IBM aunque actualmente es mantenido y desarrollado por la Fundación Eclipse, la cual es una organización independiente sin ánimo de lucro que aglutina una comunidad de código abierto y una serie de complementos, capacidades y servicios. Comenzó como un proyecto de IBM Canadá, desarrollado por OTI (*Object Technology International*) y, en 2003, fue creada la fundación independiente de IBM.

En cuanto a las características de esta plataforma, Eclipse consta de un editor de texto con analizador sintáctico. La compilación se realiza en tiempo real. Además, contiene pruebas unitarias con JUnit, control de versiones con CVS, integración con Ant, asistentes para creación de proyectos, clases, refactorización, tests, etc. También es posible mediante "plugins" añadir control de versiones con Subversion e integración con Hibernate.

10.1.4.2. Geany

Este editor de textos ha sido explicado brevemente en el punto 3.5 de este documento.

10.2. Descripción de la aplicación. Modo usuario

En este punto se va a detallar cada una de las funcionalidades que contiene la herramienta. Se va a llevar a cabo una secuencia de pantallas para su correcta visualización.

Además, se comentarán algunos de los problemas que han surgido durante el desarrollo del sistema, así como las soluciones alternativas empleadas para su correcta resolución.

10.2.1. Visión general de la aplicación

Se ha desarrollado un sistema denominado ASys para la evaluación semi-automática de ejercicios en Haskell. Asimismo, el sistema permite que los usuarios puedan acceder al mismo de una forma sencilla y agradable.

La arquitectura en la cual se basa este sistema es la siguiente:

La entrada al sistema es una ruta al directorio con los exámenes para proceder a su correspondiente evaluación, y una ruta al ejercicio auto-evaluable, el cual contiene la solución del examen y una plantilla con código de evaluación que define cómo se ha de evaluar cada propiedad del examen.

La salida del sistema es un informe que detalla la nota final junto con una lista detallada de los problemas que se han ido encontrando. Dicha salida es útil tanto para los profesores como para los alumnos (usuarios a los cuales está destinada la herramienta). Cada informe se presentará de dos formas distintas: uno para el profesor con información para la publicación de las notas, y otro para el estudiante, el cual contendrá retroalimentación explicando los problemas que se han encontrado. Este informe se compone de información del sistema, errores de compilación, análisis de propiedades y errores de tiempo de ejecución.

Puesto que la motivación principal del presente proyecto ha sido la extensión de la funcionalidad que presentaba ASys originalmente, la nueva funcionalidad añadida es la de poder evaluar ejercicios en otro lenguaje de programación el cual no es Java sino Haskell. Para ello, se han adaptado cada una de las funciones a este lenguaje con el fin de que ASys se convierta en una plataforma multilenguaje y que resulte beneficiosa para los alumnos ya que, en una misma herramienta, se podrán practicar los dos lenguajes que se cursan en la asignatura LTP[21].

Por último, cabe nombrar que en un origen lo que se pretendía proporcionar también a este sistema, es la compilación por varios compiladores de Haskell. Por falta de actualizaciones y sucesivos problemas de instalación en el sistema operativo Mac OS X, se decidió emplear el más empleado de los compiladores de este lenguaje, GHC.

10.2.2. Entrada en la aplicación

El proceso de entrada en la aplicación es muy sencillo para el usuario. Simplemente, una vez descargada la herramienta, se hace doble clic en el fichero ASys.jar y se abrirá el programa.

[21] Lenguajes, Tecnologías y Paradigmas de Programación.

10.2.3. Menú principal de la aplicación

En la parte superior izquierda de cada una de las pantallas está situado el menú de la aplicación. Este menú estará disponible en todas las pantallas principales de la herramienta aportando buena usabilidad de la misma (véase figura 5).

Figura 5: Menú de usuario

Todos los usuarios que accedan al sistema dispondrán de dicho menú a través del cual podrán moverse con facilidad por cualquier funcionalidad del sistema siempre y cuando esta esté disponible.

El menú de la aplicación está contenido en una clase llamada "ASys.java". En este archivo están contenidas cada una de las opciones que ofrece dicho menú.

A continuación, en la figura 6 de este documento, se muestra parte del código fuente del archivo "ASys.java" en el cual se añaden cada uno de los componentes al menú .

```
private void initMenu()
{
    final Version version = Config.getVersion();

    final JMenuBar menuBar = new JMenuBar();
    final JMenu ASysMenu = new JMenu("ASys");
    final JMenuItem about = new JMenuItem("About", KeyEvent.VK_B);
    final JMenu help = new JMenu("Help");
    final JMenuItem englishHelp = new JMenuItem("English", KeyEvent.VK_E);
    final JMenuItem spanishHelp = new JMenuItem("Español", KeyEvent.VK_S);
    final JMenu language = new JMenu("Language");
    final JMenuItem english = new JMenuItem("English", KeyEvent.VK_S);
    final JMenuItem spanish = new JMenuItem("Español", KeyEvent.VK_E);
    final JMenuItem french = new JMenuItem("Français", KeyEvent.VK_F);
    final JMenuItem exit = new JMenuItem("Exit", KeyEvent.VK_E);
    final JMenu createExercise = new JMenu("Create exercise");
    final JMenu recentCreatedExercises = new JMenu("Recent exercises");
    final JMenuItem createExercise0 = new JMenuItem("Create exercise...", KeyEvent.VK_R);
    final JMenu markMenu = new JMenu(version == Version.Student ? "Open exercise" : "Mark exercises");
    final JMenu recentExercises = new JMenu("Recent exercises");
    final JMenuItem loadExercise = new JMenuItem("Load exercise...", KeyEvent.VK_N);

    this.textComponents.add(ASysMenu, "@0001", "ASys");
    this.textComponents.add(about, "@0002", "About");
    this.textComponents.add(help, "@0003", "Help");
    this.textComponents.add(englishHelp, "@0004", "English");
    this.textComponents.add(spanishHelp, "@0005", "Español");
    this.textComponents.add(language, "@0013", "Language");
    this.textComponents.add(english, "@0014", "English");
    this.textComponents.add(spanish, "@0015", "Español");
    this.textComponents.add(french, "@0016", "Français");
    this.textComponents.add(exit, "@0006", "Exit");
    this.textComponents.add(createExercise, "@0007", "Create exercise");
    this.textComponents.add(createExercise0, "@0008", "Create exercise...");
    if (version == Version.Student)
        this.textComponents.add(markMenu, "@0009", "Open exercise");
    else
        this.textComponents.add(markMenu, "@0010", "Mark exercises");
    this.textComponents.add(recentExercises, "@0011", "Recent exercises");
    this.textComponents.add(loadExercise, "@0012", "Load exercise...");

    // ASys
    menuBar.add(ASysMenu);
    ASysMenu.setMnemonic(KeyEvent.VK_A);
```

Figura 6: Código fuente de ASys.java para menú principal

10.2.4. Menú de la aplicación

El usuario podrá acceder a este menú al realizar una correcta carga de un ejercicio. Para saber si se ha hecho de forma correcta, se mostrará en verde la ruta del ejercicio a evaluar (véase figura 7).

```
private void openExercise(File exerciseFile, String exerciseName)
{
    if (exerciseFile == null)
        return;

    this.exercise = new Exercise(this, exerciseFile, exerciseName);
    this.currentView = this.exercise;
    this.reload();
    this.exercise.openMenu();
}
```

Figura 7: Validación de ruta del ejercicio

Una vez realizado esto de forma correcta, se abrirá la pantalla de menú. Ello se encuentra en el código fuente en la clase "ASys.java". La parte que hace referencia a dicha funcionalidad se observa en la figura 8 de este documento.

Figura 8: Código fuente de ASys.java para menú

El menú que se abrirá a continuación será el que contenga cada uno de los módulos con sus respectivas funcionalidades. Se observa en la figura 9 de este documento.

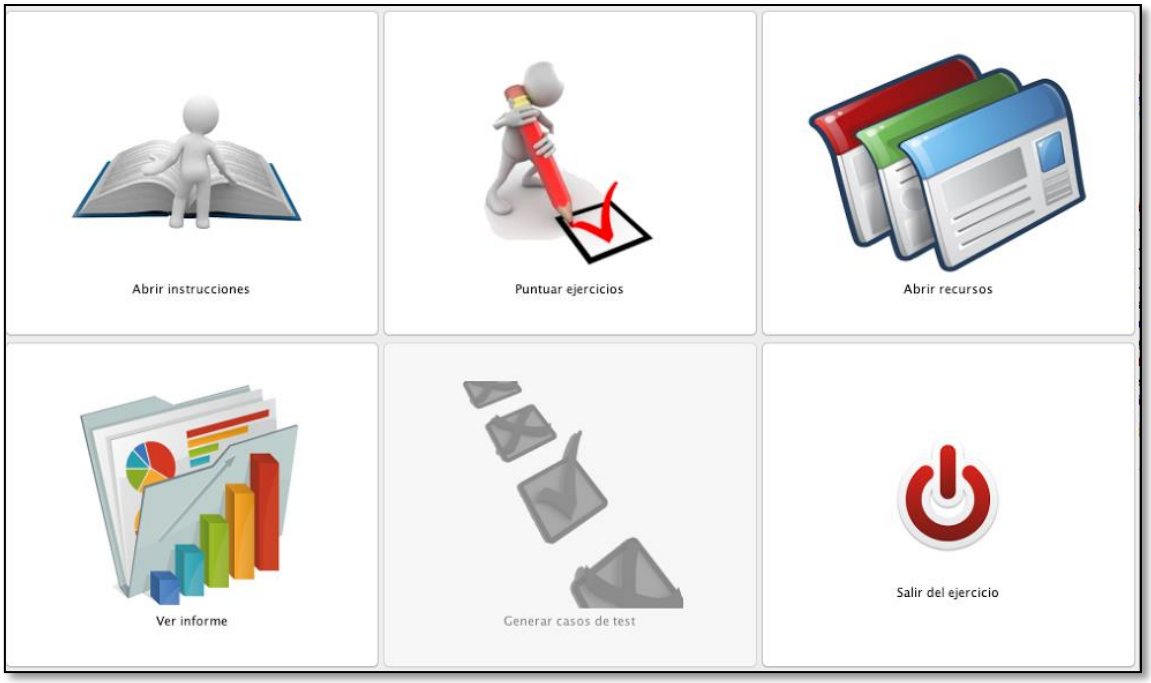


Figura 9: Menú de la aplicación

El código fuente perteneciente a la figura 8, se encuentra en la clase "Menu.java" que se observa en la figura 10, en la cual se implementan sus correspondientes funcionalidades.

```
public class Menu extends JPanel implements Localizable
{
    private static final long serialVersionUID = 1L;

    private final Exercise exercise;
    private final int assessmentIndex;
    private final Logic logic = Logic.getLogic();
    private final JLabel projectsInfo = new JLabel();
    private final JButton openInstructions = new JButton("Open instructions");
    private final JButton markExercises = new JButton("Mark exercises");
    private final JButton openResources = new JButton("Open resources");
    private final JButton viewReport = new JButton("View report");
    private final JButton generateTestCases = new JButton("Generate test cases");
    private final JButton exitExercise = new JButton("Exit exercise");

    public Menu(Exercise exercise, int assessmentIndex)
    {
        this.exercise = exercise;
        this.assessmentIndex = assessmentIndex;

        this.initComponents();
        this.addComponents();
    }
    private void initComponents()
    {
        this.setLayout(new GridBagLayout());

        this.textComponents.add(this.openInstructions, "@0050", "Open instructions");
        this.textComponents.add(this.markExercises, "@0051", "Mark exercises");
        this.textComponents.add(this.openResources, "@0052", "Open resources");
        this.textComponents.add(this.viewReport, "@0053", "View report");
        this.textComponents.add(this.generateTestCases, "@0054", "Generate test cases");
        this.textComponents.add(this.exitExercise, "@0055", "Exit exercise");

        this.openInstructions.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e)
            {
                Menu.this.exercise.openInstructions();
            }
        });
```

Figura 10: Código fuente de la clase Menu.java

En esta parte del código, se aprecian cada uno de los módulos o componentes implementados: “Abrir instrucciones”, “Puntuar ejercicios”, “Abrir recursos”, “Ver informe”, “Generar casos de test” y “Salir del ejercicio”. Cada uno de ellos lleva asociado una imagen representativa y un texto, lo cual se encuentra implementado en la misma clase. A continuación se muestra tal implementación en las figuras 11, 12, 13, 14, 15 y 16.

```
final ImageIcon openInstructionsIcon = new ImageIcon(miscPath + "Icons" + File.separator + "Instructions.png");
final Image openInstructionsImage = openInstructionsIcon.getImage();
final Image newOpenInstructionsImage = openInstructionsImage.getScaledInstance(250, 200, java.awt.Image.SCALE_SMOOTH);
final Icon newOpenInstructionsIcon = new ImageIcon(newOpenInstructionsImage);
this.openInstructions.setMargin(new Insets(0, 0, 0, 0));
this.openInstructions.setVerticalAlignment(SwingConstants.CENTER);
this.openInstructions.setAlignmentX(JComponent.CENTER_ALIGNMENT);
this.openInstructions.setAlignmentY(JComponent.CENTER_ALIGNMENT);
this.openInstructions.setVerticalTextPosition(SwingConstants.BOTTOM);
this.openInstructions.setHorizontalTextPosition(SwingConstants.CENTER);
this.openInstructions.setIcon(newOpenInstructionsIcon);
```

Figura 11: Instructions

```
final ImageIcon markExercisesIcon = new ImageIcon(miscPath + "Icons" + File.separator + "Mark.png");
final Image markExercisesImage = markExercisesIcon.getImage();
final Image newMarkExercisesImage = markExercisesImage.getScaledInstance(200, 200, java.awt.Image.SCALE_SMOOTH);
final Icon newMarkExercisesIcon = new ImageIcon(newMarkExercisesImage);
this.markExercises.setMargin(new Insets(0, 0, 0, 0));
this.markExercises.setVerticalAlignment(SwingConstants.CENTER);
this.markExercises.setAlignmentX(JComponent.CENTER_ALIGNMENT);
this.markExercises.setAlignmentY(JComponent.CENTER_ALIGNMENT);
this.markExercises.setVerticalTextPosition(SwingConstants.BOTTOM);
this.markExercises.setHorizontalTextPosition(SwingConstants.CENTER);
this.markExercises.setIcon(newMarkExercisesIcon);
```

Figura 12: Mark

```
final ImageIcon openResourcesIcon = new ImageIcon(miscPath + "Icons" + File.separator + "Resources.png");
final Image openResourcesImage = openResourcesIcon.getImage();
final Image newOpenResourcesImage = openResourcesImage.getScaledInstance(250, 200, java.awt.Image.SCALE_SMOOTH);
final Icon newOpenResourcesIcon = new ImageIcon(newOpenResourcesImage);
this.openResources.setMargin(new Insets(0, 0, 0, 0));
this.openResources.setVerticalAlignment(SwingConstants.CENTER);
this.openResources.setAlignmentX(JComponent.CENTER_ALIGNMENT);
this.openResources.setAlignmentY(JComponent.CENTER_ALIGNMENT);
this.openResources.setVerticalTextPosition(SwingConstants.BOTTOM);
this.openResources.setHorizontalTextPosition(SwingConstants.CENTER);
this.openResources.setIcon(newOpenResourcesIcon);
```

Figura 13: Resources

```
final ImageIcon reportIcon = new ImageIcon(miscPath + "Icons" + File.separator + "Report.png");
final Image reportImage = reportIcon.getImage();
final Image newReportImage = reportImage.getScaledInstance(200, 200, java.awt.Image.SCALE_SMOOTH);
final Icon newReportIcon = new ImageIcon(newReportImage);
this.viewReport.setMargin(new Insets(0, 0, 0, 0));
this.viewReport.setVerticalAlignment(SwingConstants.CENTER);
this.viewReport.setAlignmentX(JComponent.CENTER_ALIGNMENT);
this.viewReport.setAlignmentY(JComponent.CENTER_ALIGNMENT);
this.viewReport.setVerticalTextPosition(SwingConstants.BOTTOM);
this.viewReport.setHorizontalTextPosition(SwingConstants.CENTER);
this.viewReport.setIcon(newReportIcon);
```

Figura 14: Report

```
final ImageIcon testCasesIcon = new ImageIcon(miscPath + "Icons" + File.separator + "Tests.png");
final Image testCasesImage = testCasesIcon.getImage();
final Image newTestCasesImage = testCasesImage.getScaledInstance(200, 200, java.awt.Image.SCALE_SMOOTH);
final Icon newTestCasesIcon = new ImageIcon(newTestCasesImage);
this.generateTestCases.setMargin(new Insets(0, 0, 0, 0));
this.generateTestCases.setVerticalAlignment(SwingConstants.CENTER);
this.generateTestCases.setAlignmentX(JComponent.CENTER_ALIGNMENT);
this.generateTestCases.setAlignmentY(JComponent.CENTER_ALIGNMENT);
this.generateTestCases.setVerticalTextPosition(SwingConstants.BOTTOM);
this.generateTestCases.setHorizontalTextPosition(SwingConstants.CENTER);
this.generateTestCases.setIcon(newTestCasesIcon);
```

Figura 15: Tests

```
final ImageIcon exitIcon = new ImageIcon(miscPath + "Icons" + File.separator + "Exit.png");
final Image exitImage = exitIcon.getImage();
final Image newExitImage = exitImage.getScaledInstance(150, 150, java.awt.Image.SCALE_SMOOTH);
final Icon newExitIcon = new ImageIcon(newExitImage);
this.exitExercise.setMargin(new Insets(0, 0, 0, 0));
this.exitExercise.setVerticalAlignment(SwingConstants.CENTER);
this.exitExercise.setAlignmentX(JComponent.CENTER_ALIGNMENT);
this.exitExercise.setAlignmentY(JComponent.CENTER_ALIGNMENT);
this.exitExercise.setVerticalTextPosition(SwingConstants.BOTTOM);
this.exitExercise.setHorizontalTextPosition(SwingConstants.CENTER);
this.exitExercise.setIcon(newExitIcon);
```

Figura 16: Exit

El resto de funcionalidades serán explicadas detalladamente en el Anexo I perteneciente a este documento siendo este el manual de usuario.

10.3. Conexión entre la aplicación y GHCi

Este punto se puede considerar el más importante de todos los existentes en este documento, ya que era todo un reto poder enlazar un compilador no perteneciente a Java con código Java. Recordemos que ASys está implementada en Java.

La dificultad principal fue conseguir de alguna forma que lo que el usuario tecleara por terminal en el compilador de Haskell fuera interpretado por Java. Esto es, la nomenclatura que se emplea para la programación en Haskell es totalmente diferente a la que se pueda usar en Java.

Para situar al lector en contexto, cada vez que se ejecuta un ejercicio en el compilador de Haskell GHCi, ya nombrado en puntos anteriores, se genera un archivo "Main.hs", el cual existe de forma temporal. Si únicamente se usa dicho compilador, es decir, no se hiciera uso de ASys, el archivo "Main.hs" se encontraría en alguna localización del ordenador en el que se encontrara el usuario. Sin embargo, para este proyecto, lo que se pretende es generar dicho archivo dentro del sistema de tal forma que pueda existir una conexión entre el sistema ASys y el compilador GHCi. Para ello se creó el método createMain() del cual se muestra su código fuente en la figura 17.

```
private static boolean createMain(String module, String rule, String...args)
{
        try
        {
                // Generar el texto que se guardara en el archivo
                String text = "";
                String arguments = Tester.createArguments(' ', args);

                text += "module Main where" + "\n";
                text += "import " + module + "\n";
                text += "\n";
                text += "main = putStrLn $ show ( " + rule + " " + arguments + " )";

                Tester.createFile("Main.hs", text);
        }
        catch (Exception e)
        {
                return false;
        }

        return true;
}
```

Figura 17: Método createMain()

Por otro lado, se creó el método compile(), a través del cual se llamará al compilador de Haskell y se compilará el ejercicio. El código fuente perteneciente a dicho método se observa en la figura 18.

```
public static String compile(String module)
{
        String error = "";

        try
        {
                final String command = "/usr/local/bin/ghc -o " + module + " " + module + ".hs";
                final Process process = Runtime.getRuntime().exec(command, null, new File(url));

                final InputStream es = process.getErrorStream();
                final InputStreamReader isr = new InputStreamReader(es);
                final BufferedReader br = new BufferedReader(isr);

                String line;
                while ((line = br.readLine()) != null)
                        error += line + "\n";
                process.waitFor();
        }
        catch (Exception e)
        {
                throw new RuntimeException("");
        }

        return error;
}
```

Figura 18: Método compile()

Para llevar a cabo la ejecución del método se ha generado el método executeMethod() del cual se muestra en la figura 19.

```
public static String executeMethod()
{
    String result = "";

    try
    {
        final String command = url + "Main";
        final Process process = Runtime.getRuntime().exec(command);

        final InputStream is = process.getInputStream();
        final InputStreamReader isr = new InputStreamReader(is);
        final BufferedReader br = new BufferedReader(isr);

        String line;
        while ((line = br.readLine()) != null)
            result += line;
    }
    catch (IOException e)
    {
        throw new RuntimeException("");
    }

    return result;
}
```

Figura 19: Método executeMethod()

10.4. Descripción de los ejercicios

En este punto se va a detallar la estructura que siguen los ejercicios y/o exámenes a cargar en el sistema, así como la explicación de su código fuente.

Además, se comentarán algunos de los problemas que han surgido durante el desarrollo del sistema, así como las soluciones alternativas empleadas para su correcta resolución.

10.4.1. Exámenes

El usuario va a disponer de un conjunto de exámenes para evaluarlos a través de este sistema.

En el ámbito de implementación, una carpeta de examen contendrá tres carpetas principales:

- **Exercise**: Dentro de esta carpeta se encuentra el archivo “Template.java” a través del cual se definirán los casos de prueba para evaluar cada uno de los ejercicios. Se muestra detalladamente el código fuente de esta clase:

```java
import japa.parser.ast.body.ModifierSet;

import java.lang.reflect.Field;
import java.util.Map;
import java.util.Hashtable;
import java.util.List;
import java.util.LinkedList;

import javassess.Assessor;
import javassess.misc.Misc;

public class Template extends Assessor
{
	public Map<String, String> getConfig()
	{
		final Map<String, String> config = new Hashtable<String, String>();

		config.put("AssessmentMode", "Manual");

		return config;
	}
	public List<Object[]> getProperties()
	{
		final List<Object[]> properties = new LinkedList<Object[]>();

		properties.add(new Object[]{ "CasoBase", "Functionality", "Examen1.hs",
					     "En un caso base no se puede hacer una llamada recursiva",
					     "Se debe crear un caso base para la función", 3.0 });
		properties.add(new Object[]{ "CasoRecursivo", "Functionality", "Examen1.hs",
					     "Tener en cuenta el valor anterior al actual para la llamada recusriva",
					     "Se debe realizar una llamada recursiva", 7.0 });

		return properties;
	}
	public List<Object[]> getTestCasesToGenerate()
	{
		return new LinkedList<Object[]>();
	}

	public boolean testCasoBase()
	{
		final String testCaseName = "Test01";
		final String testMethodName = "test";
		final Boolean expectedResult = true;

		final Object[] args = { 0, 1 };
		final Object testCaseResult = super.tester.executeTestCase(testCaseName, testMethodName, args);

		return expectedResult.equals(testCaseResult);
	}
	public boolean testCasoRecursivo()
	{
		final String testCaseName = "Test01";
		final String testMethodName = "test";
		final Boolean expectedResult = true;

		final Object[] args = { 4, 24 };
		final Object testCaseResult = super.tester.executeTestCase(testCaseName, testMethodName, args);

		return expectedResult.equals(testCaseResult);
	}
}
```

Figura 20: Template.java

Como se puede observar, el ejercicio está dividido en dos partes: caso base y caso recursivo. Ambos tendrán su propio(s) caso(s) de prueba. La puntuación que se asigna a cada uno de ellos se asigna en el método getProperties() y, además, en dicho método se definirá el error que le aparecerá al usuario si no supera el caso de prueba así como una recomendación para hallar su solución.

- **Extra**: Esta carpeta se ha creado con el fin de introducir en ella algún código fuente adicional que se quiera aportar u otro documento que sirva de ayuda.

- **Solution**: En esta carpeta se encontrará la solución oficial del profesor del ejercicio a evaluar.

- **Tests**: Por último, en esta carpeta se encuentran los archivos correspondientes a los diferentes tests a los que se va a someter el ejercicio. Se muestra a continuación un ejemplo:

```java
import java.lang.reflect.Field;

public class Test01
{
        public static boolean test(String lista, int expectedResult) throws Throwable
        {
                String result = tester.Tester.executeTest("Examen2",
                                                          "suma_de_cuadrados",
                                                          new String[] { lista });

                return result != null && result.equals("" + expectedResult);
        }
}
```

Figura 21: Test01.java

En esta clase, se encuentra el método test() al que se le pasan como parámetros aquellos parámetros que requiera como entrada la función Haskell del ejercicio a evaluar. Finalmente se devolverá el resultado esperado.

- **Instructions**: En esta carpeta, se encontrará un archivo PDF[22] [47] el cual aportará al usuario las instrucciones del ejercicio que se va a evaluar.

- **Projects**: Esta carpeta contiene dos subcarpetas:

    - *Marked*: en esta carpeta se almacenarán las calificaciones obtenidas durante la evaluación del ejercicio en el sistema.

    - *Unmarked*: en esta carpeta el usuario deberá dejar la(s) solución(es) al ejercicio que quieren corregir. Cada solución irá en una carpeta. De esta forma, el sistema corregirá los ejercicios e irá moviendo cada solución desde la carpeta *Unmarked* a la carpeta *Marked*.

10.4.2. Batería de ejercicios desarrollados

En total, se han desarrollado 20 ejercicios auto-corregibles para Haskell. Cada uno de ellos cubre un aspecto diferente del lenguaje Haskell, de tal forma que entre ellos son complementarios. En la tabla 2 se puede observar el aspecto abordado por cada ejercicio. A continuación se van a detallar un subconjunto de los ejercicios creados a modo de ejemplo para contextualizar al lector.

Ejercicios básicos	Objetivo docente
Función factorial	Obtener el factorial de un número mediante recursividad
Función suma_de_cuadrados	Obtener la suma de los cuadrados de los elementos de una lista mediante recursividad
Función potencia	Obtener la potencia de un número mediante recursividad
Función reverso	Obtener la inversa de una lista mediante recursividad
Función tamaño	Obtener el tamaño de una lista mediante recursividad
Función elemento	Determinar si un elemento pertenece a una lista mediante recursividad
Función máximo	Obtener el mayor de dos números

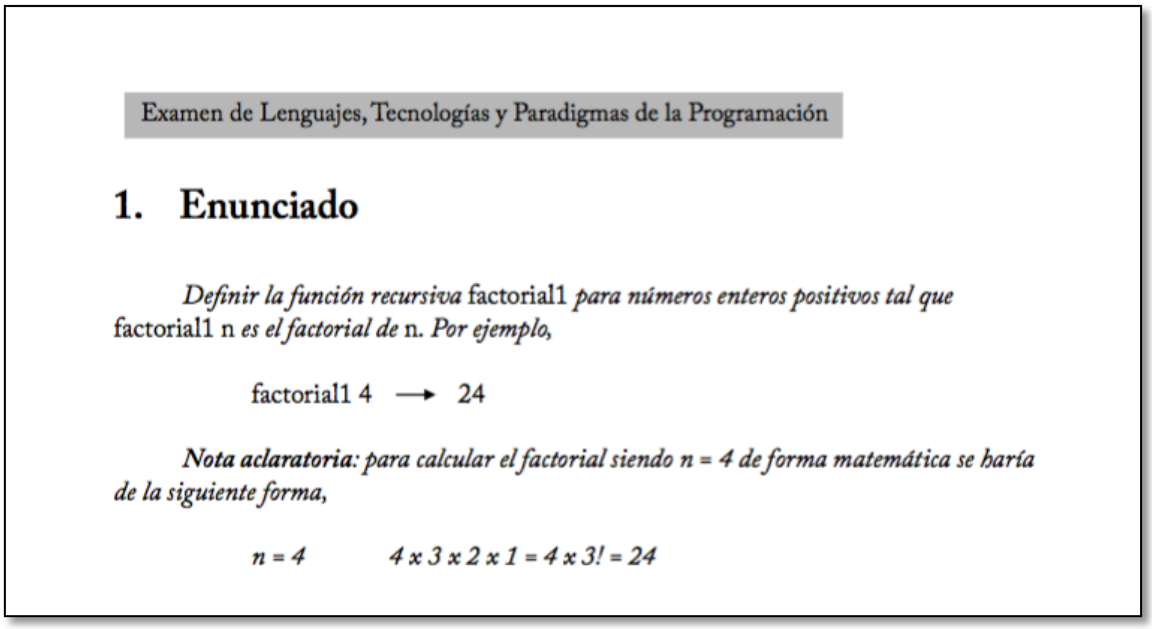
[22] PDF (*Portable Document Format*) es un formato de almacenamiento para documentos digitales independientemente del software o hardware. Más información aquí: *https://es.wikipedia.org/wiki/PDF*

Función listaArbol	Convertir una lista no vacía en un árbol mediante recursividad
Función resto	Obtener el resto de la división de dos números mediante recursividad
Función noElemento	Determinar si un elemento no pertenece a una lista mediante recursividad
Ejercicios temáticos	
Recursividad	**Objetivo docente**
Función producto	Obtener el producto de una lista de enteros
Función inserta	Insertar un número en una lista en base a un orden
Función listaArbol	Convertir una lista en un árbol balanceado
Función arbolLista	Convertir un árbol en una lista
Función fibonacci	Obtener los primeros x números de Fibonacci
Listas intensionales	**Objetivo docente**
Función primos	Obtener los primeros x números primos
Función divisores	Obtener los divisores de un número
Función prestarLibro	Prestar un libro en una base de datos
Función devolverLibro	Devolver un libro en una base de datos
Función fibonacci	Obtener los primeros x números de Fibonacci

Tabla 2: Batería de ejercicios Haskell

10.4.2.1. Ejercicio 1

El enunciado es el siguiente:



Examen de Lenguajes, Tecnologías y Paradigmas de la Programación

1. Enunciado

Definir la función recursiva factorial1 *para números enteros positivos tal que* factorial1 n *es el factorial de* n. *Por ejemplo,*

factorial1 4 ⟶ 24

Nota aclaratoria: *para calcular el factorial siendo n = 4 de forma matemática se haría de la siguiente forma,*

n = 4 *4 x 3 x 2 x 1 = 4 x 3! = 24*

Figura 22: Enunciado examen 1

La clase “Template.java” asociada a este ejercicio es la siguiente:

```
import japa.parser.ast.body.ModifierSet;

import java.lang.reflect.Field;
import java.util.Map;
import java.util.Hashtable;
import java.util.List;
import java.util.LinkedList;

import javassess.Assessor;
import javassess.misc.Misc;

public class Template extends Assessor
{
        public Map<String, String> getConfig()
        {
                final Map<String, String> config = new Hashtable<String, String>();

                config.put("AssessmentMode", "Manual");

                return config;
        }
        public List<Object[]> getProperties()
        {
                final List<Object[]> properties = new LinkedList<Object[]>();

                properties.add(new Object[]{ "CasoBase", "Functionality", "Examen1.hs",
                                             "En un caso base no se puede hacer una llamada recursiva",
                                             "Se debe crear un caso base para la función", 3.0 });
                properties.add(new Object[]{ "CasoRecursivo", "Functionality", "Examen1.hs",
                                             "Tener en cuenta el valor anterior al actual para la llamada recusriva",
                                             "Se debe realizar una llamada recursiva", 7.0 });

                return properties;
        }
        public List<Object[]> getTestCasesToGenerate()
        {
                return new LinkedList<Object[]>();
        }

        public boolean testCasoBase()
        {
                final String testCaseName = "Test01";
                final String testMethodName = "test";
                final Boolean expectedResult = true;

                final Object[] args = { 0, 1 };
                final Object testCaseResult = super.tester.executeTestCase(testCaseName, testMethodName, args);

                return expectedResult.equals(testCaseResult);
        }
        public boolean testCasoRecursivo()
        {
                final String testCaseName = "Test01";
                final String testMethodName = "test";
                final Boolean expectedResult = true;

                final Object[] args = { 4, 24 };
                final Object testCaseResult = super.tester.executeTestCase(testCaseName, testMethodName, args);

                return expectedResult.equals(testCaseResult);
        }
}
```

Figura 23: Template del ejercicio 1

Para este ejercicio, se llevan a cabo dos comprobaciones: una para el caso base y otra para el caso recursivo. En el hipotético caso de que el alumno cometiera un error en el caso base, aparecería lo siguiente, lo cual está reflejado en el código fuente de la Figura 25,

"En un caso base no se puede hacer una llamada recursiva",

acompañado de un consejo, en este caso,

"Se debe crear un caso base para la función"

Ambas comprobaciones contienen una calificación: el valor de 3.0 puntos para el caso base y, el valor de 7.0 puntos para el caso recursivo. Lo más habitual es que la puntuación

del caso recursivo sea la más alta debido a que suele resultar más complicado de solucionar que el caso base.

La clase "Test01.java" asociada a este examen sería la siguiente:

```
import java.lang.reflect.Field;

public class Test01
{
	public static boolean test(int n, int expectedResult) throws Throwable
	{
		String result = tester.Tester.executeTest("Examen1", "factorial1", new String[] { n + "" });

		return result != null && result.equals("" + expectedResult);
	}
}
```

Figura 24: Test examen 1

Como se puede comprobar en la Figura 26, se pasa como parámetro al método un valor entero llamado "n", ya que para esta función los parámetros han de ser valores enteros positivos lo cual quiere decir que serán de tipo "int" (o enteros).

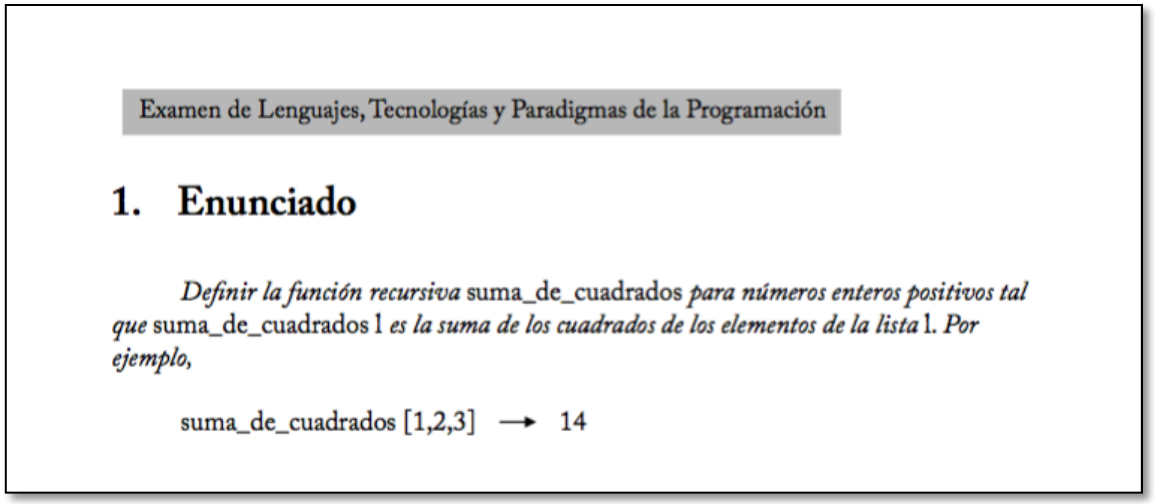
Una solución correcta para este examen (por ejemplo, podría ser la que aportara el profesor como solución oficial) sería la siguiente:

```
module Examen1 where


--Definir la función recursiva factorial1 para números enteros positivos
--tal que factorial1 n es el factorial de n. Por ejemplo,

--factorial1 4 tiene que dar como resultado 24

factorial1 :: Int -> Int

factorial1 0 = 1
factorial1 n = n * (factorial1 (n-1))
```

Figura 25: Solución examen 1

10.4.2.2. Ejercicio 2

El enunciado es el siguiente:



Examen de Lenguajes, Tecnologías y Paradigmas de la Programación

1. Enunciado

Definir la función recursiva suma_de_cuadrados *para números enteros positivos tal que* suma_de_cuadrados l *es la suma de los cuadrados de los elementos de la lista* l. *Por ejemplo,*

suma_de_cuadrados [1,2,3] ⟶ 14

Figura 26: Enunciado 2

La clase “Template.java” asociada a este examen es la siguiente:

```
import japa.parser.ast.body.ModifierSet;

import java.lang.reflect.Field;
import java.util.Map;
import java.util.Hashtable;
import java.util.List;
import java.util.LinkedList;

import javassess.Assessor;
import javassess.misc.Misc;

public class Template extends Assessor
{
        public Map<String, String> getConfig()
        {
                final Map<String, String> config = new Hashtable<String, String>();

                config.put("AssessmentMode", "Manual");

                return config;
        }
        public List<Object[]> getProperties()
        {
                final List<Object[]> properties = new LinkedList<Object[]>();

                properties.add(new Object[]{ "CasoBase", "Functionality", "Examen2.hs",
                                             "En un caso base no se puede hacer una llamada recursiva",
                                             "Se debe crear un caso base para la función", 3.0 });
                properties.add(new Object[]{ "CasoRecursivo", "Functionality", "Examen2.hs",
                                             "Sólo se admiten número enteros positivos",
                                             "Se debe hacer una llamada recursiva", 7.0 });

                return properties;
        }
        public List<Object[]> getTestCasesToGenerate()
        {
                return new LinkedList<Object[]>();
        }

        public boolean testCasoBase()
        {
                final String testCaseName = "Test01";
                final String testMethodName = "test";
                final Boolean expectedResult = true;

                final String lista = "[]";
                final Object[] args = { lista, 0 };
                final Object testCaseResult = super.tester.executeTestCase(testCaseName, testMethodName, args);

                return expectedResult.equals(testCaseResult);
        }
        public boolean testCasoRecursivo()
                {
                        final String testCaseName = "Test01";
                        final String testMethodName = "test";
                        final Boolean expectedResult = true;

                        final String lista = "[1,2,3]";
                        final Object[] args = { lista, 14 };
                        final Object testCaseResult = super.tester.executeTestCase(testCaseName, testMethodName, args);

                        return expectedResult.equals(testCaseResult);
                }
}
```

Figura 27: Template del ejercicio 2

Para este ejercicio, se llevan a cabo dos comprobaciones: una para el caso base y otra para el caso recursivo. En el hipotético caso de que el alumno cometiera un error en el caso base, aparecería lo siguiente, lo cual está reflejado en el código fuente de la Figura 25,

"Solo se admiten números enteros positivos",

acompañado de un consejo, en este caso,

"Se debe hacer una llamada recursiva"

Ambas comprobaciones contienen una calificación: el valor de 3.0 puntos para el caso base y, el valor de 7.0 puntos para el caso recursivo. Lo más habitual es que la puntuación del caso recursivo sea la más alta debido a que suele resultar más complicado de solucionar que el caso base.

La clase “Test.java” asociada a este examen es la siguiente:

```java
import java.lang.reflect.Field;

public class Test01
{
	public static boolean test(String lista, int expectedResult) throws Throwable
	{
		String result = tester.Tester.executeTest("Examen2", "suma_de_cuadrados", new String[] { lista });

		return result != null && result.equals("" + expectedResult);
	}
}
```

Figura 28: Test examen 2

Como se puede comprobar en la Figura 30, se pasa como parámetro al método una cadena de enteros llamada “lista”, ya que para esta función los parámetros han de ser de tipo “String” (o cadena).

Una solución correcta para este examen (por ejemplo, podría ser la que aportara el profesor como solución oficial) sería la siguiente:

```haskell
module Examen2 where


--Definir una función recursiva suma_de_cuadrados para números enteros
--positivos tal que suma_de_cuadrados l es la suma de los cuadradros de
--los elementos de la lista l. Por ejemplo,

--suma_de_cuadrados[1,2,3] tiene que dar como resultado 14.

suma_de_cuadrados :: [Int] -> Int

suma_de_cuadrados []     = 0
suma_de_cuadrados (x:xs) = x*x + suma_de_cuadrados xs
```

Figura 29: Solución examen 2

11. Conclusiones

La programación informática [48] es de vital importancia y afecta a una gran cantidad de tareas y procesos industriales y de la vida cotidiana. En los últimos sesenta años ha sido clave para comprender la evolución de la sistematización de tareas y del manejo de la información. Su finalidad es conseguir que tareas que en un pasado se realizaban manualmente y con un alto costo sean ejecutadas por un ordenador, ahorrando así un tiempo significativo.

Con el sucesivo desarrollo de las primeras computadoras, el trabajo físico fue paulatinamente reemplazado por máquinas e incluso también gran cantidad de trabajos intelectuales. Conforme han ido pasando los años, se ha conseguido que las máquinas realicen cálculos de mucha complejidad abriendo la posibilidad de procesar y generar gran cantidad de información en muy poco tiempo.

Por todo ello, el aprendizaje y la práctica de los lenguajes de programación son imprescindibles para los alumnos a los cuales está dirigido el desarrollo de este proyecto. Es necesario tener un mínimo conocimiento sobre distintos lenguajes para así tener distintas visiones a la hora de programar.

En el caso concreto del lenguaje Haskell, existe una gran variedad de compiladores que permiten compilar cualquier ejercicio que se desarrolle. En este proyecto se ha pretendido llegar más allá de la detección de errores de un compilador, y se ha proporcionado a los usuarios una evaluación automática del ejercicio a realizar en este lenguaje.

Lo que se ha conseguido con este proyecto es una extensión de ASys la cual permite extender su uso para otros lenguajes de programación como es en este caso el lenguaje Haskell. Esta aplicación va a ser usada este año por los alumnos en las clases reales de la asignatura LTP. El sistema consta de una batería de ejercicios los cuales tendrán los alumnos disponibles para su manipulación correspondiente, además de haber sido validados por los profesores de dicha asignatura. ASys lleva a cabo una evaluación semiautomática del código solución aportado por el alumno, basada en testeo, es decir, cada uno de los ejercicios serán sometidos a una serie de casos de prueba (tests) los cuales han de superar para considerarse una solución válida para los profesores. Si el ejercicio supera cada una de estas validaciones, el alumno obtendrá la máxima calificación.

Por otro lado, los ejercicios estarán divididos por partes para asignarles una calificación, esto es, un ejercicio puede estar formado por dos partes (por ejemplo, un caso base y un caso recursivo) y cada una de ellas tendrá una calificación diferente en base a la dificultad de las mismas.

Por último, se espera que esta extensión de ASys sirva de gran utilidad para los alumnos que cursen LTP y puedan aprender Haskell de una forma sencilla y agradable gracias a las facilidades aportadas por este sistema.

12. Ampliaciones futuras

1. Una de las ampliaciones que se podrían llevar a cabo en un futuro, es conseguir que la evaluación de los ejercicios sea por medio de varios compiladores de Haskell, obteniendo así información distinta de cada uno de ellos y, consiguiendo que el usuario disponga de toda la información posible para potenciar su aprendizaje. En el caso de que se cometiera algún error en el ejercicio, sería muy ventajoso para el usuario obtener los distintos puntos de vista de cada uno de los compiladores que se puedan enlazar con este sistema.

2. Otra posible ampliación sería crear una extensión de ASys para el lenguaje de programación Prolog[23] [49], lenguaje que también se aprende, no a tanta profundidad como Haskell, en la asignatura LTP. Lo que se exige en LTP de Prolog no es de mucha dificultad, por lo cual, si se pudiera usar este sistema serviría de gran ayuda a los alumnos ya que aprenderían de una forma más ágil y sencilla.

3. Por otro lado, uno de los aspectos que se podrían ampliar de la extensión de ASys desarrollada en este proyecto es ampliar la batería de ejercicios de forma que se pudieran incorporar ejercicios de mayor dificultad, o ejercicios grupales (que requieran el desarrollo coordinado de varios alumnos) para sacar el máximo provecho de los conocimientos aportados por los profesores a los alumnos que cursan LTP.

4. Sería de gran utilidad tener disponible la batería de ejercicios a resolver de forma online, sin la necesidad de tener que descargarse cada uno de los ejercicios y almacenarlos en una carpeta. De esta forma, ASys debería conectarse al repositorio[24] a través del cual el alumno tendría la opción de elegir el ejercicio que quisiera practicar.

5. Existe la posibilidad de que ASys contenga un entorno de desarrollo que también permita programar, esto es, que los alumnos tengan la opción de poder realizar los ejercicios directamente sobre la aplicación, en el panel en el cual visualizan tanto su solución como la solución del profesor. De esta forma, podría ir corrigiendo los posibles fallos cometidos en el mismo sistema sin tener que volver a modificar su código de forma externa a través del programa que se use para abrir los ficheros *.hs* y tener que introducirlo modificado a otra carpeta externa para que ASys lo pueda evaluar. Sería una funcionalidad más eficiente y rápida para los alumnos.

[23] (*PROgrammation en LOGique*) es un lenguaje perteneciente a la programación lógica basado en el lenguaje de la Lógica de Primer Orden. Más información aquí: *http://www.dccia.ua.es/logica/prolog/docs/prolog.pdf*.
[24] es un sitio centralizado en el cual se almacena y se mantiene la información digital por medio de bases de datos o archivos informáticos. Más información aquí: *https://es.wikipedia.org/wiki/Repositorio*.

Bibliografía

[1] Apuntes. *Historia de la Programación Declarativa*. *Departamento de Lenguajes y Ciencias de la Computación*. Universidad de Málaga. Consultado en http: http://www.lcc.uma.es/~blas/apuntes/PDAv/p2000-2001/historia.pdf.

[2] Alvivi, *¡Aprende Haskell por el bien de todos!*. Consultado en http: http://aprendehaskell.es/.

[3] Wiki Haskell. *Introducción*. Consultado en https://wiki.haskell.org/Es/Introduccion#.C2.BFQu.C3.A9_es_Haskell.3F. Diciembre, 2007.

[4] Wiki Haskell. *Future of Haskell*. Consultado en https://wiki.haskell.org/Future_of_Haskell. Diciembre, 2012.

[5] David Insa, Josep Silva. *Semi-Automatic Assessment of Unrestrained Java Code: A Library, a DSL, and a Workbench to Assess Exams and Exercises*, ITiCSE 2015: 39-44. Disponible en http://dl.acm.org/citation.cfm?doid=2729094.2742615. 2015.

[6] Toni Cárdenas. *Programación funcional, un enfoque diferente a los problemas de siempre*. Consultado en http://www.genbetadev.com/paradigmas-de-programacion/programacion-funcional-un-enfoque-diferente-a-los-problemas-de-siempre. Abril, 2011.

[7] Uqbar-Project. *Inferencia de tipos*. Consultado en http://uqbar-wiki.org/index.php?title=Inferencia_de_tipos. Mayo, 2015.

[8] Wikipedia. *Evaluación perezosa*. Consultado en https://es.wikipedia.org/wiki/Evaluaci%C3%B3n_perezosa. Abril, 2014.

[9] Wikipedia. *Crisis del software*. Consultado en https://es.wikipedia.org/wiki/Crisis_del_software. Mayo, 2016.

[10] Wikipedia. *Haskell Curry*. Consultado en https://es.wikipedia.org/wiki/Haskell_Curry. Enero, 2016.

[11] Haskell. *Features*. Consultado en https://www.haskell.org/.

[12] Wiki Haskell. *Foreign Function Interface*. Consultado en https://wiki.haskell.org/Foreign_Function_Interface. Agosto, 2015.

[13] Wikipedia. *Gofer* (programming language). Consultado en https://en.wikipedia.org/wiki/Gofer_(programming_language). Agosto, 2016.

[14] Haskell. *Concurrent and Parallel Haskell* . Consultado en https://downloads.haskell.org/~ghc/7.8.4/docs/html/users_guide/lang-parallel.html.

[15] Haskell. *GHC*. Consultado en https://wiki.haskell.org/GHC. Marzo, 2015.

[16] FileExtensions. *Programas que abren el archivo HS*. Consultado en http://www.fileextensions.info/es/file-extension/hs.

[17] Wikipedia. *Emacs*. Consultado en https://es.wikipedia.org/wiki/Emacs. Junio, 2016.

[18] Geany. *Welcome!*. Consultado en https://www.geany.org/. Enero, 2016.

[19] Wikipedia. *Geany*. Consultado en https://es.wikipedia.org/wiki/Geany. Diciembre, 2015.

[20] Wikipedia. *GTK*. Consultado en https://es.wikipedia.org/wiki/GTK. Octubre, 2015.

[21] Wikipedia. *Scintilla*. Consultado en https://es.wikipedia.org/wiki/Scintilla. Octubre, 2015.

[22] Wikipedia. *gedit*. Consultado en https://es.wikipedia.org/wiki/Gedit. Abril, 2014.

[23] Wikipedia. *Java (lenguaje de programación)*. Consultado en https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n). Agosto, 2016.

[24] Oracle. *¿Qué es la tecnología Java y para qué la necesito?*. Consultado en https://www.java.com/es/download/faq/whatis_java.xml.
[25] tuProgramación. *Historia de Java*. Consultado en http://www.tuprogramacion.com/programacion/historia-de-java/.
[26] Universitat de València. *Programación Concurrente*. Consultado en http://informatica.uv.es/iiguia/LP/teoria/apuntes/cuatr1/tema3_1_concurrencia.pdf.
[27] Buenas Tareas. *Paradigma imperativo*. Consultado en http://www.buenastareas.com/ensayos/Paradigma-Imperativo/2375475.html. Junio, 2011.
[28] Leandro Alegsa. *Definición de Lenguaje máquina*. Consultado en http://www.alegsa.com.ar/Dic/lenguaje%20maquina.php. Noviembre, 2008.
[29] Wikipedia. *Localizador de recursos uniforme*. Consultado en https://es.wikipedia.org/wiki/Localizador_de_recursos_uniforme. Agosto, 2016.
[30] Masadelante. *Definición de World Wide Web, web o www*. Consultado en http://www.masadelante.com/faqs/www.
[31] Wikipedia. *Lenguaje específico del dominio*. Consultado en https://es.wikipedia.org/wiki/Lenguaje_espec%C3%ADfico_del_dominio. Abril, 2016.
[32] Indalog. *Técnicas de prueba*. Consultado en http://indalog.ual.es/mtorres/LP/Prueba.pdf.
[33] Wikipedia. *Repositorio*. Consultado en https://es.wikipedia.org/wiki/Repositorio. Agosto, 2016.
[34] Josep Silva, David Insa. *ASys: Assessment System*. Consultado en http://users.dsic.upv.es/~jsilva/ASys/. 2015.
[35] Carlos Ble. *Modelo en cascada*. Consultado en http://librosweb.es/libro/tdd/capitulo_1/modelo_en_cascada.html.
[36] Wikipedia. *Desarrollo en cascada*. Consultado en https://es.wikipedia.org/wiki/Desarrollo_en_cascada. Agosto, 2016.
[37] Wikipedia. *Usabilidad*. Consultado en https://es.wikipedia.org/wiki/Usabilidad. Mayo, 2016.
[38] Wikipedia. *Máquina Virtual Java*. Consultado en https://es.wikipedia.org/wiki/M%C3%A1quina_virtual_Java. Julio, 2016.
[39] Wikipedia. *Zip (file format)*. Consultado en https://en.wikipedia.org/wiki/Zip_(file_format). Agosto, 2016.
[40] Wikipedia. *JAR (file format)*. Consultado en https://en.wikipedia.org/wiki/JAR_(file_format). Julio, 2016.
[41] Wikipedia. *Portable Network Graphics*. Consultado en https://es.wikipedia.org/wiki/Portable_Network_Graphics. Julio, 2016.
[42] Wikipedia. *Composición alfa*. Consultado en https://es.wikipedia.org/wiki/Composici%C3%B3n_alfa. Diciembre, 2014.
[43] Ecured. *James Gosling*. Consultado en http://www.ecured.cu/James_Gosling.
[44] Wikipedia. *C++*. Consultado en https://es.wikipedia.org/wiki/C%2B%2B. Agosto, 2016.
[45] Agustín Ramos Fonseca. *Programación Funcional en Haskell*. Consultado en http://es.slideshare.net/zentimental.software/programacion-funcional-con-haskell. Octubre, 2013.
[46] Wikipedia. *Eclipse (software)*. Consultado en https://es.wikipedia.org/wiki/Eclipse_(software)#Datos. Junio, 2016.
[47] Wikipedia. *PDF*. Consultado en https://es.wikipedia.org/wiki/PDF. Junio, 2016.

[48] Importancia. *Importancia de la Programación (informática)*. Consultado en http://www.importancia.org/programacion-informatica.php.
[49] Faraón Llorens Largo, Mª Jesús Castel de Haro. *Prolog*. Consultado en http://www.dccia.ua.es/logica/prolog/docs/prolog.pdf.

ANEXO I. Manual de usuario

1. Introducción

La presente sección tiene como objetivo ser una guía de la herramienta. A continuación se explica el manejo de la aplicación para cada uno de los usuarios que vayan a proceder a su uso.

2. Funcionalidades

El sistema ASys ofrece las siguientes funcionalidades:

- Abrir un nuevo ejercicio.
- Corregir un ejercicio.
- Generar un informe de la corrección realizada.

3. Entrada al sistema

Se va a proceder a establecer cómo conseguir ASys en el ordenador del usuario. Se trata de un sistema gratuito y se podrá descargar a través del siguiente enlace:

http://www.dsic.upv.es/~jsilva/ASys/

Una vez se ha descargado el fichero, se descomprime el fichero ZIP descargado y, hacienda doble clic sobre el archivo llamado "ASys.jar" se procederá a la apertura automática del programa.

4. Preparación previa

A continuación, para la apertura de un ejercicio es necesario que el profesor lo haya preparado previamente y que se disponga de una solución o soluciones aportadas por el usuario o usuarios (en este caso, alumnos).

Lo próximo a realizar por parte del usuario es situar la solución al ejercicio en la carpeta *Unmarked*.

5. Menú principal de la aplicación

Una vez realizada la preparación previa, es decir, el punto 4, el usuario ha de acceder al menú principal situado en la parte superior izquierda de la pantalla y pulsar la opción "Puntuar ejercicios" y la subpestaña "Cargar ejercicio". Véase la figura 30 de este documento.

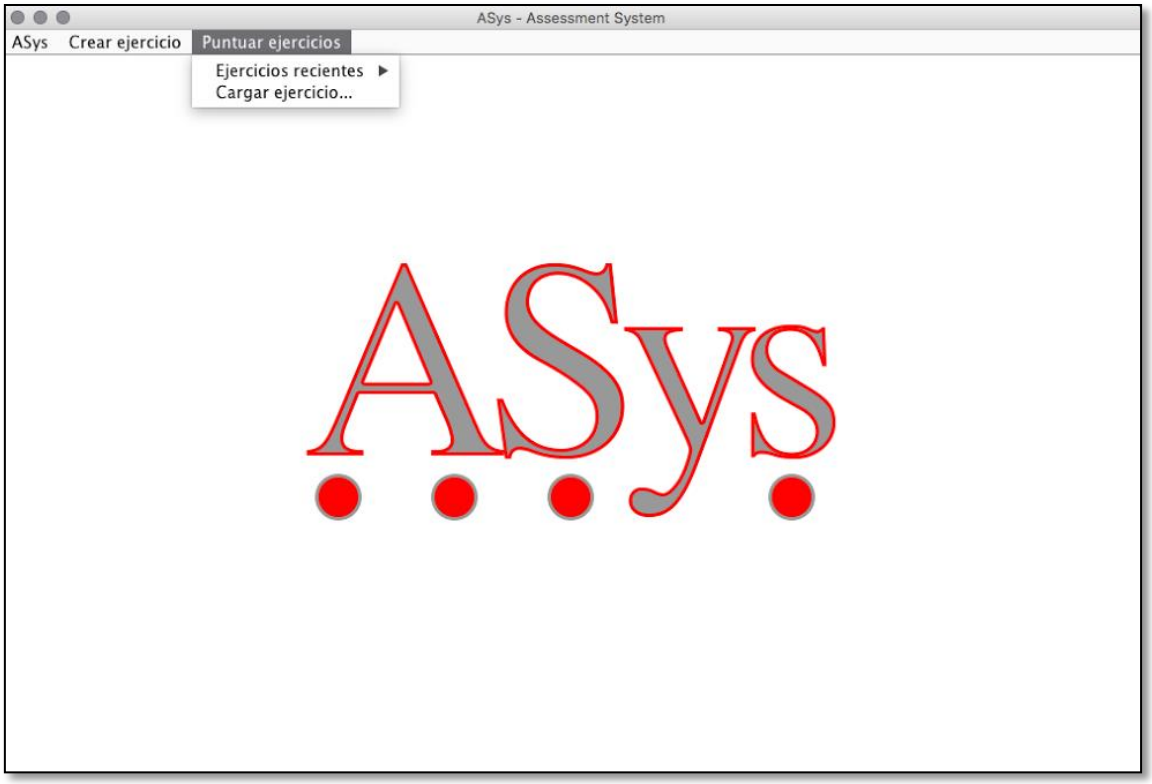


Figura 30: Menú de inicio. Manual de usuario

Si el ejercicio se ha cargado correctamente, se visualizará en letra verde y se pulsará el botón “Cargar” (Véase la Figura 31 de este documento). Una vez pulsado dicho botón, se accederá al menú de la aplicación.

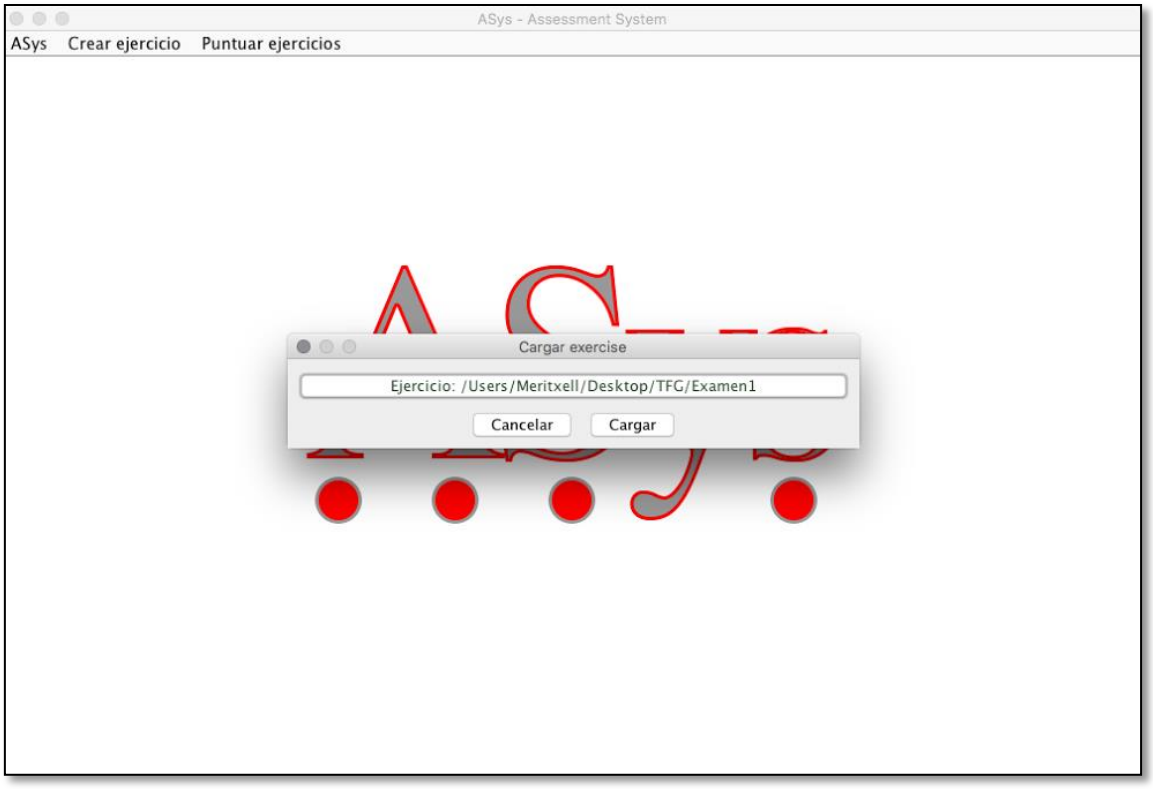


Figura 31: Cargar ejercicio. Manual de usuario

Por otro lado, se tiene la opción de poder cargar de forma más rápida un ejercicio que ya ha sido cargado con anterioridad. Ello se realiza a través de la pestaña “Puntuar ejercicios” y la subpestaña “Ejercicios recientes” (véase figura 32 del documento).

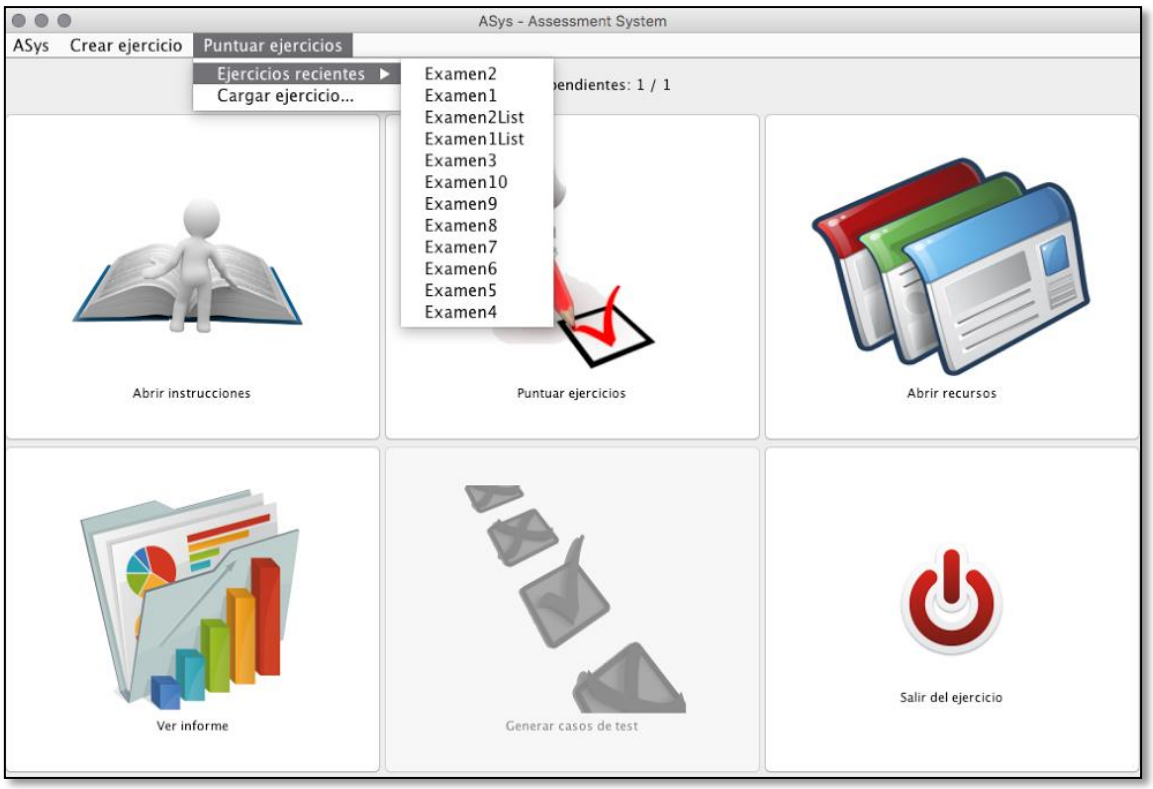


Figura 32: Cargar ejercicio reciente. Manual de usuario

6. Menú de la aplicación

Para iniciar la corrección se deberá hacer doble clic sobre la opción “Puntuar ejercicios” abriéndose la pantalla correspondiente a la figura 32 de este documento.

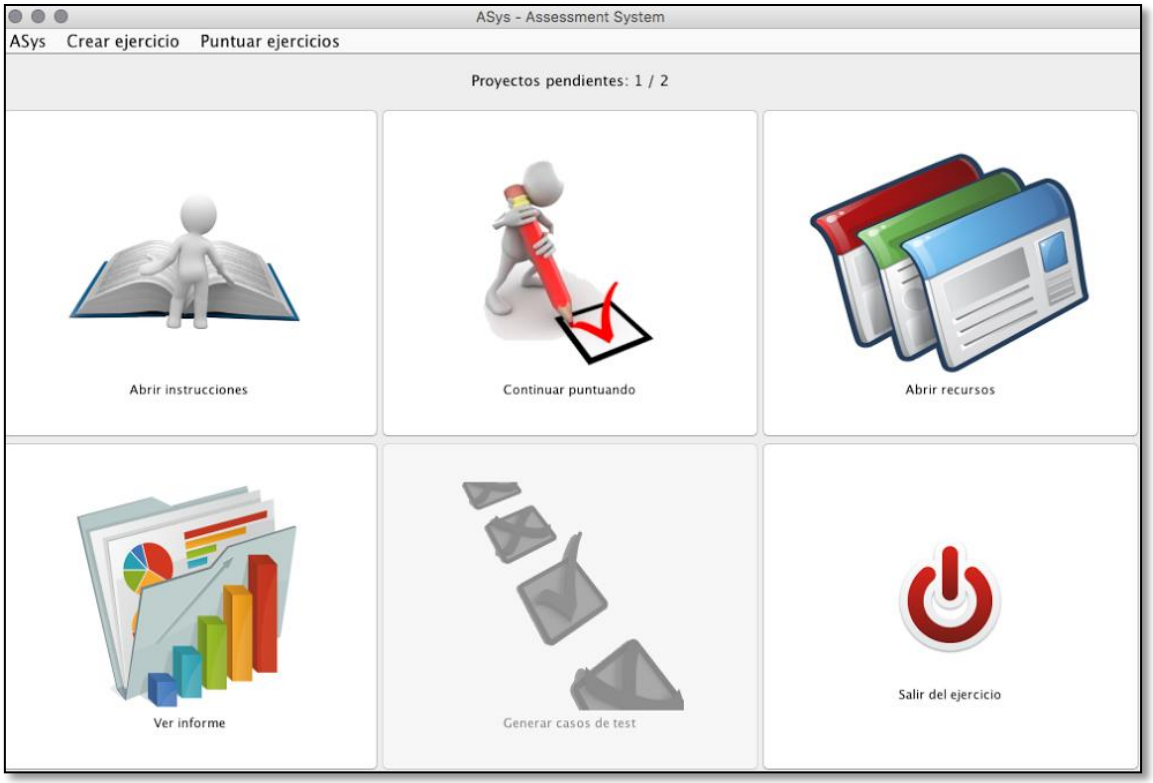


Figura 33: Menú principal. Manual de usuario

En esta pantalla (véase figura 33), el usuario podrá ver el resultado de la evaluación de su solución o soluciones que haya introducido en la carpeta *Unmarked*.

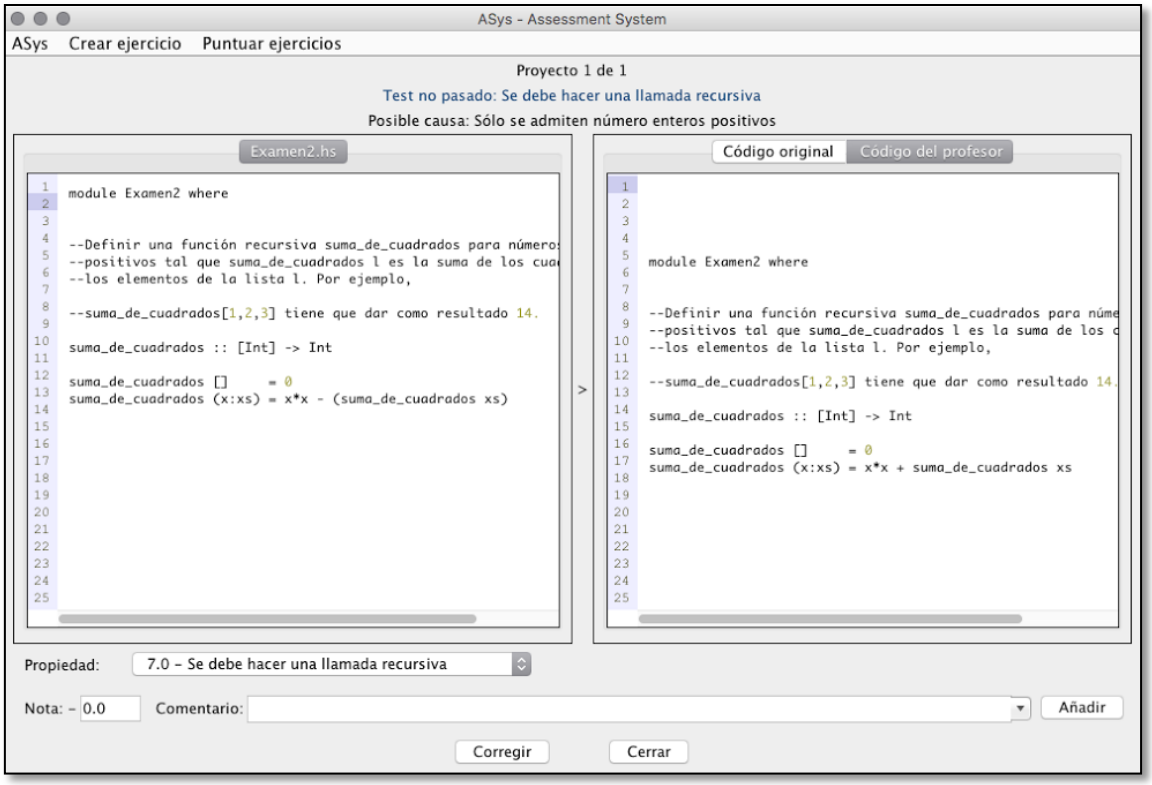


Figura 34: Pantalla de corrección. Manual de usuario

Si el usuario desea ver las instrucciones correspondientes al ejercicio solo tendrá que acceder al módulo denominado “Abrir instrucciones” (véase figura 32). Si, por otro lado, necesitara de algún recurso adicional del cual se haga nombramiento en el enunciado, este estará disponible en el módulo llamado “Abrir recursos” (véase figura 32).

Además, si el usuario quisiera visualizar las calificaciones que ha ido obteniendo mediante la corrección de su solución o soluciones, tendrá que hacerlo accediendo al módulo “Ver informe” (véase figura 32) en el cual aparecerá una tabla con calificaciones por apartados así como una calificación final puntuada sobre diez. Se muestra en la figura 34 de este documento.

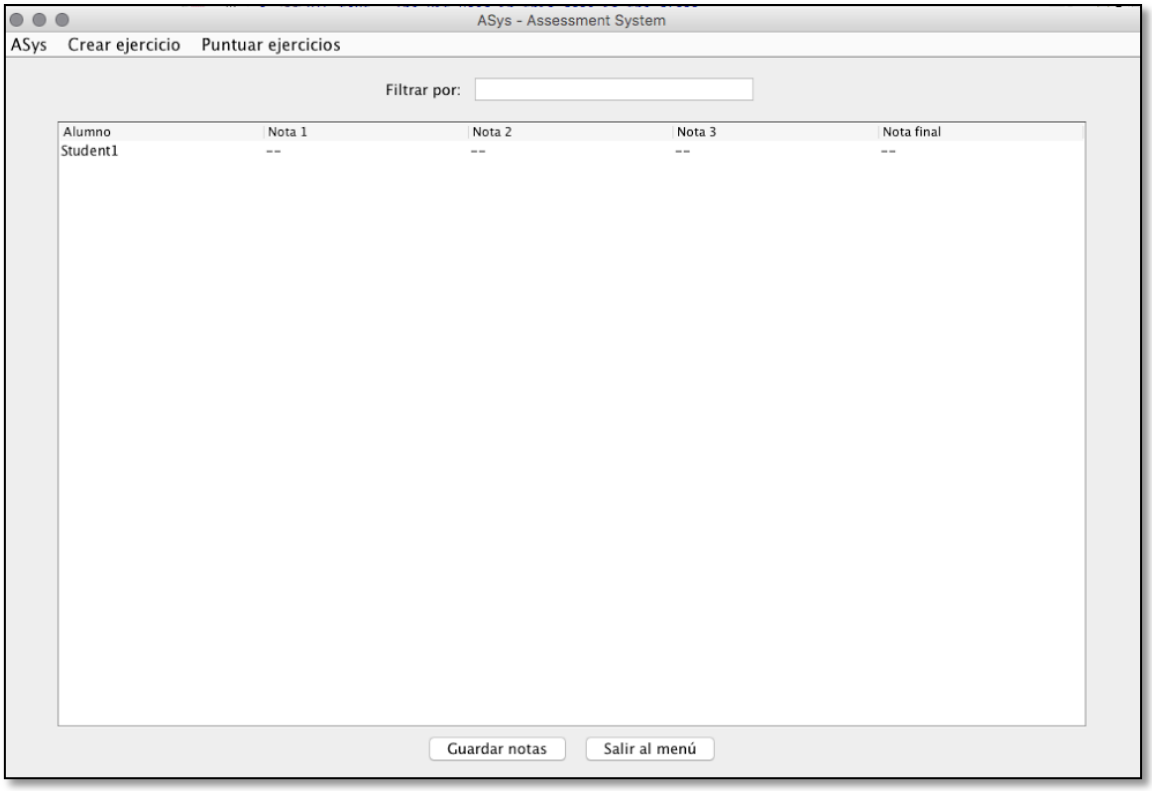


Figura 35: Pantalla de calificaciones. Manual de usuario

Por otro lado, en la carpeta *Marked* se guardará un informe con cada una de las notas obtenidas y, además, las posibles notas y anotaciones que se hayan tomado durante la corrección por parte del usuario. Además, se puede observar en la figura 35 el botón "Salir al menú" el cual nos permite retornar a la pantalla de la figura 33.

Por último, para salir del ejercicio, el usuario solo tendrá que pulsar sobre el módulo "Salir del ejercicio" (véase figura 32).