Aplicación multiplataforma para la visualización y resolución numérica de potenciales cuánticos

Proyecto de Fin de Carrera

Ingeniería Técnica en Informática de Sistemas

VNiVERSiDAD D SALAMANCA

Febrero de 2011

Autor

Javier Montejo Berlingen

Tutor

José Rafael García-Bermejo Giner

Proyecto de Informática

Javier Montejo Berlingen

Febrero de 2011

A mis padres

porque en vez de traerme hasta aquí

me enseñaron a recorrer el camino por mi mismo

Agradecimientos

Quiero agradecer a todas las personas que me han apoyado durante la realización de este proyecto, y sin las cuales estas líneas serían lo único que podría haber escrito.

Este trabajo no hubiera sido posible sin la inestimable ayuda y guía de D. José Rafael García-Bermejo Giner. Gracias a su buen hacer y disponibilidad nunca hubo un solo momento de bloqueo en la realización del proyecto.

Quisiera agradecer también a D. Alfredo Valcarce Mejía, que en todo momento me ha ofrecido su ayuda desinteresada, y cuya biblioteca particular fue desapareciendo progresivamente para acabar en mis estanterías.

Por último a todos aquellos que pese a no haber tenido una participación activa en el proyecto me han apoyado en todo momento, dándome ánimos y fuerzas en los momentos de flaqueza, y que sé que siempre seguirán ahí. A mis padres y a mi hermano, a todos mis amigos, cerca y lejos, que han estado siempre a mi lado. En especial a Cayetano, Molina, Darío, Martín y Victoria que durante tantos años han compartido risas y penas conmigo, haciendo que cualquier esfuerzo sea la mitad de duro y llenando de momentos buenos mi vida.

A todos, gracias.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Este documento reúne la documentación elaborada en el proceso de construcción del sistema software a desarrollar como *Proyecto Fin de Carrera* de la Ingeniería Técnica en Informática de Sistemas.

1.1. Presentación

La aplicación *jSchroedinger* se plantea como una herramienta docente y didáctica que pueda ser empleada tanto por los distintos departamentos de Física de la Universidad de Salamanca, como por los alumnos de dicha disciplina que deseen utilizarla. Permite resolver de forma numérica la ecuación de Schrödinger para cualquier potencial, tarea que resulta necesaria en multitud de ámbitos de la Física y que sólo en casos muy puntuales puede realizarse sin recurrir al cálculo numérico por ordenador.

El movimiento de una partícula en un potencial independiente del tiempo viene gobernado por la ecuación de Schrödinger independiente del tiempo:

$$\left(-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + V(x)\right)\psi(x) = E\psi(x)$$

Dado un determinado potencial, deseamos conocer las soluciones de onda $\psi(x)$ y las energías para las cuales existe una solución. Si la forma funcional del potencial es sencilla, la ecuación se puede resolver de forma analítica. Sin embargo para potenciales más complejos es necesario recurrir a una solución numérica de la ecuación.

Para resolver numéricamente esta ecuación recurriremos al algoritmo de Numerov, el cual está especialmente indicado para resolver ecuaciones diferenciales de segundo grado sin términos de primer grado. Para más información sobre la ecuación de Schrödinger y el algoritmo de Numerov puede

consultar el capítulo 3, conceptos teóricos.

Las herramientas y aplicaciones existentes en este campo concentran sus esfuerzos en obtener la máxima precisión y velocidad en los cálculos, prescindiendo de toda presentación gráfica ya que están dirigidas al ámbito de la investigación. El desarrollo de este proyecto pretende ofrecer una herramienta didáctica que permita no solo la resolución de la ecuación de Schrödinger, sino que ofrezca una visualización gráfica de todos los componentes físicos involucrados, desde el planteamiento del potencial hasta las funciones de onda obtenidas y el espectro de energías.

Con estos objetivos surge el proyecto *jSchroedinger*. Con la ayuda de D. Alfredo Valcarce Mejía, profesor del departamento de física fundamental y bajo la tutela del profesor del departamento de informática y automática, D. José Rafael García-Bermejo Giner.

1.2. Contenido y estructura de la memoria

En el desarrollo de la memoria se ha seguido la guía "Proyectos de final de carrera en la Ingeniería Técnica en Informática: Guía de realización y documentación" del departamento de informática y automática de la USAL, versión 1.52, Marzo 2000.

En dicha guía se distingue entre dos partes claramente diferenciadas:

1.2.1. Parte 1 - Descripción del proyecto

En esta parte se plantea el problema al que se ha dado solución, indicándose los aspectos más interesantes que se han utilizado para solucionarlo.

Consta de los siguientes apartados:

Introducción: contiene una breve descripción del tema que se aborda en el proyecto, junto con la descripción del contenido y estructura de la memoria.

Objetivos del proyecto: contiene de forma precisa y concisa los objetivos que se persiguen con la realización del proyecto, distinguiendo entre objetivos del sistema a desarrollar y objetivos personales del desarrollador.

Conceptos teóricos: contiene las aclaraciones o fundamentos de carácter teórico necesarios para la comprensión de determinados aspectos del desarrollo del sistema.

Técnicas y herramientas: contiene una descripción de las técnicas metodológicas y herramientas de desarrollo utilizadas para llevar a cabo el proyecto. En el caso de que haya habido varias opciones se comentarán los aspectos más destacados de la opción escogida.

Aspectos relevantes del desarrollo del proyecto: contiene los aspectos mas interesantes del proceso de desarrollo. Abarca desde la exposición del ciclo de vida utilizado, hasta los detalles de mayor relevancia de las fases de análisis, diseño, implementación y pruebas.

Trabajos relacionados: contiene una relación de los proyectos y trabajos realizados en el campo en el que se desarrolla este proyecto.

Conclusiones y líneas de trabajo futuras: contiene las conclusiones derivadas del desarrollo del proyecto, así como un informe crítico que incluye posibles mejoras al proyecto y vías futuras de desarrollo.

1.2.2. Parte 2 - Documentación técnica

Los contenidos de esta parte son un conjunto de anexos que recogen el trabajo realizado en las distintas etapas de desarrollo.

Anexo A – Plan del proyecto software: el objetivo de este anexo es obtener un documento que mostrará al gestor un marco temporal en el que se llevará a cabo el desarrollo del sistema, junto con una estimación de costes y recursos necesarios.

Anexo B – Documentación de requisitos: se tratará de presentar el catálogo de requisitos del sistema y el análisis de los mismos. Se pretende adquirir un amplio conocimiento y comprensión sobre el dominio del problema para así poder abordar la búsqueda de la solución más óptima.

Anexo C – Especificación de diseño: el objetivo de este anexo es modelar el sistema y encontrar su forma, incluida la arquitectura, para que soporte todos los requisitos que se le suponen. Recoge información sobre los datos manejados así como su tratamiento, las interfaces con el usuario y los detalles procedimentales de la aplicación.

Anexo D – Documentación técnica de programación: en esta sección se encuentra la información necesaria referida a la compilación del código fuente y la estructura de ficheros generada por la aplicación.

Anexo E – Manual de usuario: el objetivo de este anexo es guiar al usuario en el proceso de instalación y uso de la aplicación. Así como informar al mismo sobre los distintos requisitos hardware y software necesario para su correcto funcionamiento.

Capítulo 2

Objetivos del proyecto

Pasamos ahora a concretar los objetivos de este proyecto. Distinguiremos dos tipos de objetivos, por una parte requisitos indispensables de la aplicación, objetivos que han de alcanzarse porque son el fin al que va dirigido el proyecto. Por otra parte objetivos cualitativos deseados de acuerdo a la orientación que se le pretende dar al proyecto.

Finalmente tenemos que tener en cuenta también los objetivos y retos que presenta este proyecto a nivel personal. La realización del proyecto de fin de carrera tiene como fin último no solo la realización del proyecto propiamente dicho, sino el aprendizaje adquirido en el proceso de realización.

Para más información sobre objetivos y requisitos puede consultar el apéndice B.

2.1. Objetivos fundamentales

La funcionalidad básica del proyecto es resolver la ecuación de Schrödinger, que es la ecuación que rige el movimiento de una partícula en un potencial a nivel cuántico. El usuario podrá introducir cualquier potencial en la aplicación y el programa deberá ser capaz de resolver la ecuación de Schrödinger con este potencial y mostrar los resultados.

Además del potencial, el usuario debe tener libre acceso a los parámetros del cálculo. De esta forma puede ajustar la precisión y la duración de los cálculos a sus necesidades.

El proyecto está destinado principalmente a su uso académico y por tanto didáctico. Por lo tanto se considera otro requisito imprescindible la visualización gráfica de todos los conceptos físicos involucrados. Éste es probablemente el aspecto más importante del proyecto y el que lo distingue de otras aplicaciones similares existentes en el mundo de la física. La resolución de potenciales es un elemento cotidiano e imprescindible en muchos ámbitos de la física y por tanto ya existen programas destinados a este fin. La di-

ferencia es que estos programas buscan únicamente la eficiencia y precisión en los cálculos, siendo generalmente programas de línea de comandos sin posibilidades de visualización, utilizados en el ámbito de la investigación. Por tanto el enfoque didáctico de este proyecto, centrado en que el usuario sea capaz en todo momento de visualizar los elementos físicos involucrados, distingue a este proyecto del resto de programas existentes. Para más información sobre trabajos relacionados puede consultar el capítulo 6.

Los resultados generados por el programa han de poder ser guardados para recuperarlos más adelante. Igualmente los elementos gráficos deben poder ser exportados en formato de imagen. De esta forma se permite el uso de los resultados obtenidos fuera de la aplicación.

Estos tres son los objetivos fundamentales e ineludibles del proyecto.

2.2. Objetivos cualitativos

La solución numérica de la ecuación de Schrödinger conlleva una gran carga de cálculo y procesamiento que puede llevar a tiempos de respuesta y de espera elevados para obtener las soluciones. De igual manera, una mayor precisión en el cálculo implica un tiempo de ejecución mayor. Dada la orientación didáctica del proyecto se considera prioritaria la visualización de los resultados frente al tiempo de ejecución y la precisión. A pesar de ello consideramos necesario establecer una cota mínima para ambos que se ha de respetar. Por una parte el tiempo de ejecución, para unos valores razonables de los parámetros, no debe superar los pocos segundos. Por otra parte, establecemos unos valores iniciales de precisión tales que el valor de los resultados para los potenciales más frecuentes sea exacto con al menos seis decimales.

Para completar la orientación académica consideramos que es necesario que la interfaz de la aplicación sea intuitiva y fácil de manejar. Los elementos de la interfaz deben ser autoexplicativos y evidentes y la ayuda integrada en la aplicación debe ser suficiente para realizar cualquier operación. Por todo ello la usabilidad de la aplicación debe ser igualmente un elemento a tener en cuenta durante el desarrollo del proyecto.

Por último, dada la gran variedad de usuarios potenciales que pueden usar esta aplicación, se debe intentar maximizar la portabilidad de la aplicación para su uso en el mayor número de plataformas posibles.

2.3. Objetivos personales

La realización de este proyecto tiene como principal tarea afrontar un problema del mundo real buscando una solución viable y apropiada para el mismo. Así como desarrollar la capacidad propia de investigación con ayuda de los conocimientos adquiridos durante la carrera.

Los objetivos que afronto en el presente proyecto y espero cumplir son los siguientes:

- Aplicar los conocimientos adquiridos durante la carrera, profundizando en las áreas de ingeniería del software y programación orientada a objetos entre otras.
- Alcanzar el éxito en la difícil tarea de planificación y gestión de proyectos, cumpliendo los plazos establecidos.
- Investigar y adquirir experiencia en el uso de gramáticas formales para desarrollar analizadores sintácticos complejos.
- Profundizar en la aplicación de métodos numéricos para llevar a cabo cálculos funcionales con ordenador.
- Adquirir destreza y conocimientos en la plataforma "Eclipse" sobre la que se desarrolla el proyecto, ya que se trata de una herramienta ampliamente extendida y usada por las empresas para desarrollar sus productos.
- Adquirir experiencia en el desarrollo de sistemas software complejos, considerándolo una pequeña incursión en el futuro campo laboral.

Capítulo 3

Conceptos teóricos

Trataremos en este capítulo los conceptos teóricos subyacentes necesarios para abordar la resolución numérica de la ecuación de Schrödinger. Empezaremos por los conceptos físicos representados por la ecuación de Schrödinger y continuaremos con el proceso matemático necesario para resolver dicha ecuación mediante métodos numéricos.

3.1. Ecuación de Schrödinger

Al nivel de física cuántica, el movimiento de una partícula en un potencial independiente del tiempo viene gobernado por la ecuación de Schrödinger independiente del tiempo:

$$\hat{H}\psi(x) = E\psi(x)$$

$$\hat{H} = -\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + V(x)$$

$$\left(-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + V(x)\right)\psi(x) = E\psi(x) \tag{3.1}$$

Donde por simplicidad hemos considerado una única dimensión.

Dado un determinado potencial, deseamos conocer las soluciones de onda $\psi(x)$ y las energías para las cuales existe una solución. Si la forma funcional del potencial es sencilla, la ecuación se puede resolver de forma analítica. Sin embargo para potenciales más complejos es necesario recurrir a una solución numérica de la ecuación.

La ecuación (3.1) es extensible a tres dimensiones si consideramos que nuestra variable puede variar en todo el espacio.

$$\left(-\frac{\hbar^2}{2m}\nabla^2 + V(x,y,z)\right)\psi(x,y,z) = E\psi(x,y,z) \tag{3.2}$$

Si el potencial que estamos considerando tiene simetría radial, $V(r)$, podemos reducir la ecuación de Schrödinger a una ecuación en una única variable haciendo el paso a coordenadas polares.

$$\begin{aligned}\psi(r,\theta,\phi) &= R(r)Y_{lm}(\theta,\phi)\\ \nabla^2 &= \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial}{\partial r}\right) - \hat{L}^2\\ \hat{L}^2 &= -\left(\frac{1}{r^2\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial}{\partial\theta}\right) + \frac{1}{r^2\sin^2\theta}\frac{\partial^2}{\partial\phi^2}\right)\end{aligned}$$

Haciendo uso de las propiedades de los armónicos esféricos, $Y_{lm}(\theta,\phi)$, podemos resolver la dependencia angular.

$$\hat{L}^2 Y_{lm}(\theta,\phi) = l(l+1)Y_{lm}(\theta,\phi)$$

Y sustituyendo en (3.2) obtenemos:

$$\left(-\frac{\hbar^2}{2mr^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial}{\partial r}\right) + \frac{\hbar^2}{2m}\frac{l(l+1)}{r^2} + V(r)\right)R(r) = ER(r) \tag{3.3}$$

Aún podemos simplificar más la ecuación y convertirla en una análoga al caso unidimensional. Hacemos el cambio de variable:

$$\tilde{R}(r) \stackrel{def}{=} r\cdot R(r) \tag{3.4}$$

$$\left(-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial r^2} + \frac{\hbar^2}{2m}\frac{l(l+1)}{r^2} + V(r)\right)\tilde{R}(r) = E\tilde{R}(r) \tag{3.5}$$

$$\left(-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial r^2} + V_{eff}(r)\right)\tilde{R}(r) = E\tilde{R}(r) \tag{3.6}$$

$$V_{eff}(r) = V(r) + \frac{\hbar^2}{2m}\frac{l(l+1)}{r^2} \tag{3.7}$$

Como se puede comprobar las ecuaciones (3.1) y (3.6) tienen la misma forma funcional. Veremos en las próximas secciones como resolver este tipo de ecuación.

3.2. Algoritmo de Numerov

Los algoritmos para la resolución numérica de ecuaciones se basan en discretizar el espacio sobre el que tomamos las funciones. Dividimos el espacio en intervalos de longitud constante h, de esta forma el valor de nuestra variable en la abscisa siempre será un múltiplo entero de h.

Denotamos:

$$\begin{aligned}x_n &= n * h\\ f_n &= f(x_n)\end{aligned}$$

Mediante el desarrollo en serie de Taylor podemos evaluar el valor de las derivadas de una función en un punto, en función de valores vecinos. La precisión de la aproximación vendrá determinada por el número de puntos vecinos que utilicemos.

$$
\begin{aligned}
f_{n+1} &= f(x_n + h) = f_n + hf'_n + \frac{h^2}{2!}f''_n + \frac{h^3}{3!}f'''_n + \mathcal{O}(h^4) \\
f_{n-1} &= f(x_n - h) = f_n - hf'_n + \frac{h^2}{2!}f''_n - \frac{h^3}{3!}f'''_n + \mathcal{O}(h^4) \\
f'_n &= \frac{f_{n+1} - f_{n-1}}{2h} + \mathcal{O}(h^2) \\
f'_n &= \frac{1}{12h}\left(f_{n-2} - 8f_{n-1} + 8f_{n+1} - f_{n+2}\right) + \mathcal{O}(h^4) \\
f''_n &= \frac{f_{n+1} - 2f_n + f_{n-1}}{h^2} + \mathcal{O}(h^2)
\end{aligned}
$$

3.2.1. Solución a la ecuación de onda

Pasamos ahora a definir el algoritmo de Numerov. Este algoritmo está especialmente indicado para resolver ecuaciones diferenciales de la forma:

$$\frac{\partial^2 f}{\partial x^2} + k^2(x)f = 0 \tag{3.8}$$

Que es la forma funcional de las ecuaciones (3.1) y (3.6).

Empezamos calculando la derivada segunda a partir de los desarrollos de Taylor anteriores escribiendo explícitamente el término de error en $\mathcal{O}(h^2)$.

$$\frac{f_{n+1} - 2f_n + f_{n-1}}{h^2} = f''_n + \frac{h^2}{12}f''''_n + \mathcal{O}(h^4) \tag{3.9}$$

Y aprovechando la expresión de la ecuación a resolver (3.8) podemos escribir:

$$
\begin{aligned}
f''''_n &= \frac{\partial^2}{\partial x^2}\left(f''_n\right) \\
&= \frac{\partial^2}{\partial x^2}\left(-k^2(x)f\right) \\
&= -\frac{(k^2 f)_{n+1} - 2(k^2 f)_n + (k^2 f)_{n-1}}{h^2} + \mathcal{O}(h^2)
\end{aligned}
$$

Sustituyendo en (3.9) y reagrupando términos obtenemos:

$$\left(1 + \frac{h^2}{12}k^2_{n+1}\right)f_{n+1} - 2\left(1 - \frac{5h^2}{12}k^2_n\right)f_n + \left(1 + \frac{h^2}{12}k^2_{n-1}\right)f_{n-1} = 0 + \mathcal{O}(h^6) \tag{3.10}$$

Mediante esta ecuación podemos calcular f_{n+1} a partir de f_n y f_{n-1}, obteniendo así una relación de recurrencia que nos permite obtener todos los valores de la función a partir de dos puntos iniciales. Igualmente podemos resolver la ecuación para f_{n-1} y calcular a partir de los dos puntos finales.

3.2.2. Solución a la energía

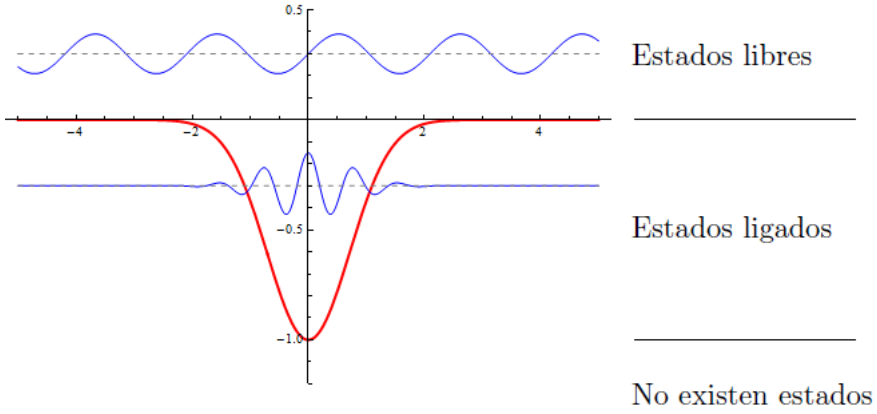

Figura 3.1: Potencial con estados ligados

En un potencial cuántico, el espectro de energías es discreto. No nos hemos preocupado hasta ahora de obtener los valores de la energía sino simplemente de resolver la ecuación diferencial para cualquier valor posible. Sin embargo, la ecuación que deseamos resolver solo tiene soluciones para ciertos valores de la energía. Para poder hallar la solución de onda necesitamos saber para que valores de la energía existen soluciones.

Dado un potencial para el cual existen estados ligados (restricción que imponemos a todos nuestros potenciales, véase figura 3.1) nuestra solución de onda oscilará en la zona permitida clásicamente, $E > V(x)$, y decaerá exponencialmente en la zona prohibida clásicamente, $E < V(x)$, hasta anularse en el infinito.

Una forma de obtener la solución a la energía es usar un valor de prueba E_o, resolver la ecuación y comprobar si nuestra solución de onda cumple las condiciones de contorno $\Psi(\infty) = \Psi(-\infty) = 0$.

Sin embargo, al calcular la solución desde una zona prohibida, llamémosla $\Psi_<$, obtendríamos una solución que crece exponencialmente y oscila al pasar a la zona clásica. Si proseguimos el cálculo al entrar en la nueva zona prohibida, la solución se volverá inestable si sigue creciendo exponencialmente. Por tanto es mejor calcular una nueva solución partiendo de la segunda condición de contorno, $\Psi_>$, y comprobar si las dos soluciones empalman en un punto de corte arbitrario que elijamos x_m.

Puesto que ambas soluciones, $\Psi_<$ y $\Psi_>$, satisfacen una ecuación homogénea, se pueden normalizar mediante una constante multiplicativa para que adopten el mismo valor en el punto de corte $\Psi_<(x_m) = \Psi_>(x_m)$, sin embargo a menos que el valor de la energía sea correcto, la onda resultante no será solución, véase figura 3.2. Las soluciones de energía vendrán dadas

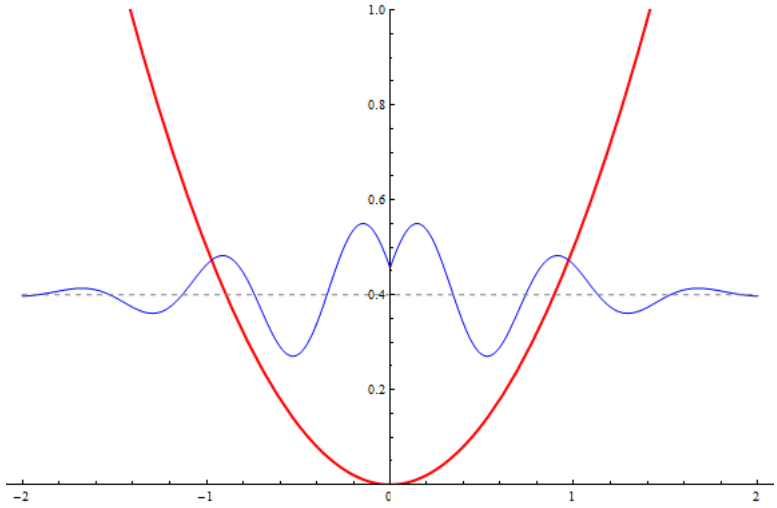

Figura 3.2: Solución no válida

por una igualdad en las derivadas, de forma que la solución completa sea continua y derivable, véase figura 3.3.

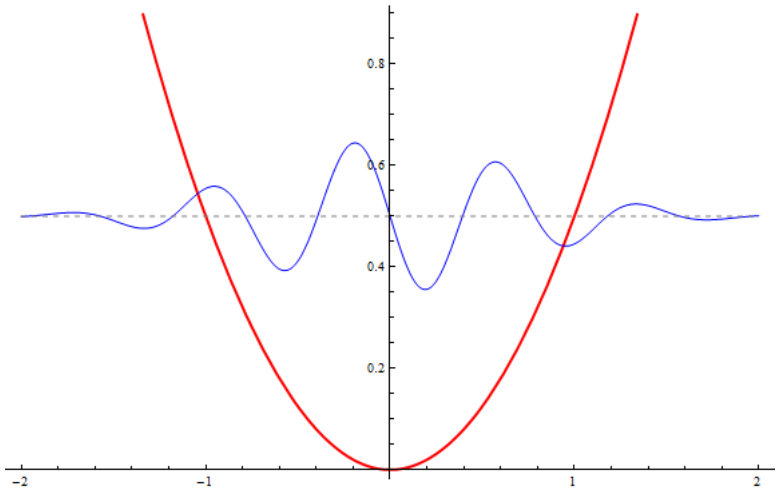

Figura 3.3: Solución válida

3.3. Cambio de variables

Hemos visto como el algoritmo de Numerov nos permite la resolución de ecuaciones diferenciales de segundo orden sin término en derivada primera. Nos enfrentamos ahora al problema de calcular una solución que se extiende hasta el infinito. Para solucionar esto podemos optar por dos planteamientos:

- Imponer la condición de contorno en un punto lejano pero distinto de infinito, $\Psi(x_{max}) = 0$.
- Hacer un cambio de variable que recorriendo un intervalo finito cubra el espacio completo en nuestro sistema de coordenadas. De esta forma podemos imponer, $\Psi(\infty) = 0$.

En nuestro caso hemos optado por implementar ambas soluciones y ofrecer al usuario la elección del método a aplicar. Exponemos ahora la forma más general del cambio de variable, siendo ambas implementaciones únicamente una elección particular del cambio de variable a realizar.

Sea x la coordenada en la que deseamos calcular nuestra función de onda. Sea $v \in [-1, 1]$ la variable, con dominio finito, sobre la cual iteraremos para hallar la solución. Sea $g(v)$, la función de cambio de variable.

$$x = g(v) \tag{3.11}$$

Efectuamos paso a paso el cambio de variable en la ecuación a resolver:

$$\begin{aligned}
\frac{\partial^2 \Psi}{\partial x^2} + k^2(x)\Psi &= 0 \\
\frac{\partial}{\partial x}\left(\frac{\partial \Psi}{\partial x}\right) + k^2(x)\Psi &= 0 \\
\left(\frac{\partial x}{\partial v}\right)^{-1} \frac{\partial}{\partial v}\left(\left(\frac{\partial x}{\partial v}\right)^{-1} \frac{\partial \Psi}{\partial v}\right) + k^2(g(v))\Psi &= 0 \\
\left(\frac{\partial x}{\partial v}\right)^{-1} &= a(v) \\
a(v)\left(a'(v)\cdot\Psi'(v) + a(v)\cdot\Psi''(v)\right) + k^2(v)\Psi(v) &= 0
\end{aligned}$$

De esta forma hemos obtenido la ecuación de Schrödinger en nuestra nueva variable v. Sin embargo esta ecuación ya no es resoluble mediante el algoritmo de Numerov ya que tenemos un término en derivada primera. Para solucionar este problema hacemos un nuevo cambio, esta vez en la función Ψ. Definimos:

$$\Psi(v) = u(v)\Phi(v) \tag{3.12}$$

Introducimos este cambio en la ecuación que obtuvimos anteriormente y reorganizamos los términos. Omitimos a partir de ahora la dependencia en la variable v.

$$\begin{aligned}
a\cdot\left(a'\cdot\left(u'\cdot\Phi + u\cdot\Phi'\right) + a\cdot\left(u''\cdot\Phi + 2\cdot u'\cdot\Phi' + u\cdot\Phi''\right)\right) + k^2\cdot u\cdot\Phi &= 0 \\
a^2\cdot u\cdot\Phi'' + a\cdot\left(a'\cdot u + 2\cdot a\cdot u'\right)\Phi' + \left(a\cdot a'\cdot u' + a^2\cdot u'' + k^2\cdot u\right)\Phi &= 0 \\
\Phi'' + \frac{a'\cdot u + 2\cdot a\cdot u'}{a\cdot u}\Phi' + \left(\frac{a'\cdot u' + a\cdot u''}{a\cdot u} + \frac{k^2}{a^2}\right)\cdot\Phi &= 0
\end{aligned} \tag{3.13}$$

De esta forma, dado cualquier cambio de variable $g(v)$, podemos elegir $u(v)$ de forma que el término en derivada primera se anule:

$$\frac{a' \cdot u + 2 \cdot a \cdot u'}{a \cdot u} = 0$$

$$u'(v) = \frac{a'(v) \cdot u(v)}{2 \cdot (a)} \tag{3.14}$$

Y de esta forma obtenemos una ecuación resoluble mediante el algoritmo de Numerov.

$$\Phi'' + \left(k_{add}^2(v) + \frac{k^2(v)}{a^2(v)} \right) \cdot \Phi = 0 \tag{3.15}$$

$$k_{add}^2(v) = \frac{a' \cdot u' + a \cdot u''}{a \cdot u} \tag{3.16}$$

3.3.1. Aproximación WKB

Según el método WKB [5], para puntos en los que la energía es menor que el potencial, y por tanto no están permitidos clásicamente, el valor de la función de onda se puede aproximar por:

$$\Phi(x) \approx exp\left(-x\sqrt{V(x) - E} \right) \tag{3.17}$$

Si el valor de x es suficientemente grande, se puede aproximar el valor de la función por cero. En la práctica esto significa que calculamos la función de onda hasta un punto x_{max} en el cual imponemos que la función se anule.

WKB en una dimensión

Definimos nuestro cambio de variable $g(v)$ como:

$$x = v \cdot x_{max} \tag{3.18}$$

De esta forma, para $v \in [-1, 1]$ tenemos que $x \in [-x_{max}, x_{max}]$. Al hallar la función de onda mediante este método, el programa puede advertir al usuario si x_{max} no se halla fuera de la zona permitida clásicamente, o si no es aceptable aproximar la función de onda por cero según el valor del método WKB.

Con este cambio de variable podemos ver que:

$$a(v) = \left(\frac{\partial x}{\partial v} \right)^{-1} = \frac{1}{x_{max}}$$
$$a'(v) = 0$$
$$u'(v) = 0$$

Y por tanto podemos tomar:

$$u(v) = 1$$

Y la ecuación a resolver queda:

$$k^2_{add}(v) = 0$$
$$\Phi'' + \frac{k^2(v)}{a^2(v)} \cdot \Phi = 0$$

WKB radial

El cambio de variable apropiado en la coordenada radial es:

$$r = \frac{(v+1)}{2} \cdot r_{max} \tag{3.19}$$

De esta forma, para $v \in [-1, 1]$ tenemos que $r \in [0, r_{max}]$. Igualmente, el programa puede advertir al usuario de que la solución puede ser incorrecta si r_{max} se encuentra en la zona clásicamente permitida o no es suficientemente grande para que la aproximación WKB pueda considerarse aceptable.

Con este cambio de variable:

$$a(v) = \frac{2}{x_{max}}$$
$$a'(v) = 0$$
$$u'(v) = 0$$

Y por tanto de nuevo podemos tomar:

$$u(v) = 1$$

Y la ecuación a resolver queda, al igual que en el caso unidimensional:

$$k^2_{add}(v) = 0$$
$$\Phi'' + \frac{k^2(v)}{a^2(v)} \cdot \Phi = 0$$

3.3.2. Función de onda extendida al infinito

Otra aproximación al problema es prescindir de la aproximación a cero en un punto lejano. En su lugar podemos utilizar un cambio de coordenadas que nos permita hallar la solución extendida al infinito. Obviamente este cambio ya no es lineal y tendremos una mayor densidad de puntos en el entorno de cero. Esto puede ser una ventaja ya que ciertas características como la anchura de desintegración leptónica depende del valor de la función en el origen y una mayor densidad de puntos implican una mayor precisión del valor.

Cambio de variable en una dimensión

El cambio de variable elegido es:

$$x = \frac{v}{1 - v^2} \tag{3.20}$$

Que cumple que para $v \in [-1, 1]$ tenemos que $x \in (-\infty, +\infty)$.

$$\begin{aligned} a(v) &= \frac{\left(1 - v^2\right)^2}{1 + v^2} \\ a'(v) &= \frac{2v \cdot \left(v^4 + 2v^2 - 3\right)}{\left(1 + v^2\right)^2} \\ u'(v) &= \frac{a' \cdot u}{2 \cdot a} \end{aligned}$$

Resolvemos la ecuación diferencial y obtenemos como solución:

$$u(v) = \frac{\sqrt{1 + v^2}}{1 - v^2} \tag{3.21}$$

Por tanto los términos de la ecuación quedan de la forma:

$$\begin{aligned} k^2_{add}(v) &= \frac{3}{1 + v^2} \\ \frac{k^2(v)}{a^2(v)} &= \frac{\left(1 + v^2\right)^2}{\left(1 - v^2\right)^4} k^2(v) \\ \Phi'' &+ \left(\frac{3}{1 + v^2} + \frac{\left(1 + v^2\right)^2}{\left(1 - v^2\right)^4} k^2(v) \right) \cdot \Phi = 0 \end{aligned}$$

Que a pesar de ser compleja analíticamente, es perfectamente resoluble mediante el algoritmo de Numerov.

Cambio de variable radial

El cambio de variable elegido es:

$$r = \frac{1 + v}{1 - v} \tag{3.22}$$

Mediante el cual proyectamos $v \in [-1, 1]$ a $r \in [0, +\infty)$.

$$\begin{aligned} a(v) &= \frac{(v - 1)^2}{2} \\ a'(v) &= (v - 1) \\ u'(v) &= \frac{u}{v - 1} \end{aligned}$$

Resolvemos la ecuación diferencial y obtenemos:

$$u = \frac{1}{1-v} \tag{3.23}$$

En este caso tenemos que el término adicional se anula:

$$k_{add}^2(v) = \frac{a' \cdot u' + a \cdot u''}{a \cdot u} = 0$$

Por tanto la ecuación a resolver queda de la forma:

$$\Phi'' + \left(\frac{4}{(1-v)^2} \cdot k^2(v) \right) \cdot \Phi = 0 \tag{3.24}$$

3.3.3. Recolocación del origen

En el caso de una dimensión, es recomendable centrar el origen de coordenadas en el mínimo de potencial para una mayor precisión en el cálculo. Podemos sustituir los cambios de variable 3.18 y 3.20 por:

$$\overline{x} = x + x_0$$

Donde x_0 representa el mínimo del potencial. Al haber añadido únicamente una constante, el resto de términos no cambia ya que dependen de la derivada de x, que no se ve afectada. Con esto conseguimos que los límites del cálculo esten distribuidos uniformemente a ambos lados del mínimo, lo cual es muy deseable para potenciales no centrados. En el caso de que extendamos el cálculo de la onda a infinito estaremos concentrando la mayor densidad de puntos en el entorno del mínimo.

Capítulo 4

Técnicas y herramientas

4.1. Programación orientada a objetos o estructurada

Una de las elecciones más determinantes es la del lenguaje de programación que se va a usar. Por una parte se va a resolver un problema mediante cálculo numérico, en el que deseamos la mayor precisión posible y un tiempo de ejecución bajo. Por lo tanto una opción obvia sería tomar un lenguaje estructurado altamente eficiente en cálculos en punto flotante como pueden ser Fortran o C. Por otra parte deseamos hacer especial hincapié en la parte gráfica y visual de la aplicación y mantener un buen diseño modular para que el proyecto pueda ser fácilmente ampliable, lo cual requeriría un lenguaje orientado a objetos como puede ser Java.

De los objetivos que definimos en el capítulo 2 podríamos concluir que dado que anteponemos el aspecto visual a la eficiencia de los cálculos, la elección natural sería la programación orientada a objetos. Sin embargo los lenguajes orientados a objetos tienen un rendimiento mucho menor en cálculos numéricos que los lenguajes estructurados. Por tanto antes de decantarnos por uno u otro hemos de comprobar si somos capaces de cumplir las restricciones que nos impusimos respecto a precisión y tiempo de ejecución con un lenguaje orientado a objetos.

Para ello vamos a implementar el algoritmo y hallar las soluciones usando como referencia el estado fundamental y el cuarto estado excitado del oscilador armónico tridimensional:

$$V(r) = \frac{1}{2}mr^2$$
$$E_0 = 1{,}5MeV$$
$$E_4 = 9{,}5MeV$$

Donde estamos usando unidades naturales: $\hbar = c = 1$

Puntos de red	E_0	E_4	C (segundos)	Java (segundos)
100	1.4999996	9.4993	0.010	0.013
200	1.49999997	9.49996	0.011	0.023
400	1.499999998	9.499997	0.018	0.041
800	1.49999999991	9.4999998	0.034	0.082
1600	1.499999999994	9.499999990	0.066	0.170
3200	1.500000000004	9.4999999994	0.107	0.289
6400	1.499999999992	9.49999999996	0.240	0.671
12800	1.50000000002	9.500000000005	0.414	1.186
25600	1.49999999997	9.49999999997	0.951	2.708
51200	1.5000000001	9.49999999997	1.669	4.781

Cuadro 4.1: Comparativa de tiempos de ejecución entre C y Java

Conociendo la solución de referencia podemos comparar los tiempos de ejecución y la precisión del algoritmo de Numerov implementado en un lenguaje orientado a objetos como es Java, frente a uno estructurado, en este caso C. Los resultados obtenidos[1] aparecen en la tabla 4.1 y en la figura 4.1. En rojo los tiempos de ejecución de C, en azul de Java. El tiempo de ejecución es la media tomada sobre tres ejecuciones del algoritmo y los resultados se han tomado con los decimales necesarios hasta incluir la primera cifra inexacta. Los archivos necesarios para reproducir estos resultados se encuentran en el CD del proyecto: "/jSchroedinger/codigo fuente/comparativas/comparativa C vs Java"

Podemos ver que a partir de 3200 y 6400 puntos de red hemos rebasado con creces la precisión mínima que deseábamos y la diferencia en el tiempo de ejecución es inapreciable para el usuario. Hay que tener en cuenta que estos tiempos de ejecución son únicamente del algoritmo y evaluando un potencial escrito en el código fuente. La duración total será mayor debido a la interfaz y a la necesidad de evaluar una expresión dada por el usuario. En cualquier caso este tiempo añadido afectará a la aplicación en ambos casos y por lo tanto nos centramos aquí en el cálculo numérico.

Por tanto, una vez que hemos visto que la repercusión para el usuario es mínima, y dado que el tiempo de ejecución no es un factor crítico para nuestro programa, creemos justificada la realización del proyecto en el lenguaje de programación Java.

[1] Todos los tiempos de ejecución han sido medidos en un procesador Core Duo @2.00Ghz de 64bits. Se ha comparado Java 6.19 y C con gcc 4.4.1

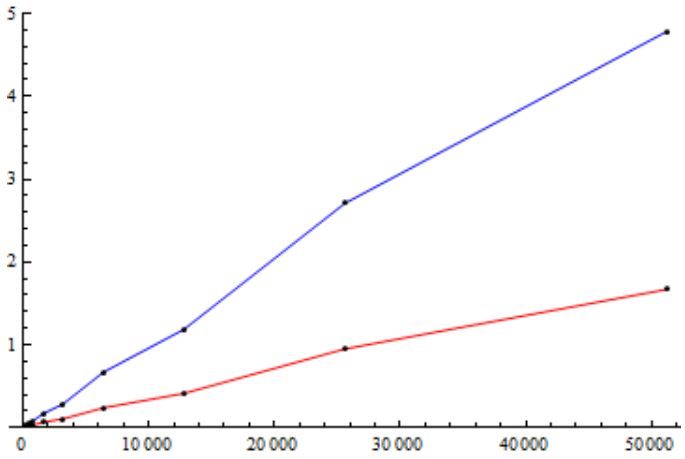

Figura 4.1: Comparativa gráfica de tiempos de ejecución entre C y Java

4.2. Evaluación del potencial

Durante el uso de la aplicación, el usuario introducirá la expresión del potencial a través de la interfaz correspondiente. Esta expresión es evaluada un número muy elevada de veces, en el ejemplo anterior se evalua $225 * puntos\ de\ red$ veces antes de obtener la solución. Más de 11 millones de veces en el caso de 51200 puntos de red. Por lo tanto es imprescindible que la evaluación sea eficiente, sin embargo partimos de una cadena de caracteres con la expresión introducida por el usuario que no es directamente evaluable.

Una opción es construir un parser que analice la cadena de caracteres y construya un objeto ecuación a partir de ella. Dado que una ecuación puede tener varias expresiones anidadas en paréntesis, cada objeto ecuación estaría compuesto de forma recursiva por dos ecuaciones y un operador que los una. Para evaluar un objeto ecuación evaluaríamos las dos ecuaciones que la componen recursivamente y compondríamos los resultados con el operador correspondiente. A primera vista este planteamiento no parece muy eficaz ya que implica la utilización de recursividad y la creación de un número elevado de objetos.

Otra opción es conseguir llevar la expresión introducida por el usuario al código fuente. Java permite la carga dinámica de clases, por tanto es posible escribir la expresión introducida por el usuario en el punto correspondiente del código fuente, compilarlo y cargarlo, todo ello en tiempo de ejecución. Una pega a este planteamiento es que sería necesario que el usuario tuviera instalado el JDK para poder compilar la nueva clase.

Comparamos de nuevo el rendimiento de ambos enfoques antes de tomar

Puntos de red	Parser (segundos)	Compilador (segundos)
100	0.227	1.216
200	0.235	1.189
400	0.239	1.180
800	0.270	1.282
1600	0.345	1.311
3200	0.457	1.405
6400	0.807	1.616
12800	1.378	2.069
25600	3.112	2.964
51200	5.964	4.531

Cuadro 4.2: Comparativa de tiempos de ejecución entre parser y compilador

una decisión. La comparativa entre los tiempos de ejecución se puede ver en la tabla 4.2 y en la figura 4.2. En rojo la implementación del parser, en azul la compilación en tiempo de ejecución. Los archivos necesarios para reproducir estos resultados se encuentran en el CD del proyecto: "/jSchroedinger/codigo fuente/comparativas/comparativa parser vs compilador"

Dado que el grado de recursividad puede afectar al rendimiento comparamos la misma expresión escrita en formas distintas.

- $0{,}5 * (r * r)$
  Profundidad de la recursividad: 2
  Nodos: 3
  Línea contínua en la gráfica

- $0{,}5 * (r * r + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 - 10)$
  Profundidad de la recursividad: 12
  Nodos: 14
  Línea discontínua en la gráfica

Como se puede ver, el uso de una expresión compilada merece la pena para un número de puntos de red superior a 12800. Por debajo de esta cantidad, la penalización causada por el tiempo de compilación y carga de la clase hace que sea más rápido el uso del parser. A pesar de lo que cabría esperar, la profundidad de recursión penaliza de forma prácticamente constante ambas implementaciones. Dado que vimos que el intervalo óptimo en cuanto a precisión y tiempo de ejecución se encuentra entre 1600 y 3200 puntos de red, es preferible decantarse por la opción de un parser.

En el siguiente apartado trataremos en profundidad la implementación del parser. Para más detalles sobre la implementación de la compilación y carga de clases en tiempo de ejecución puede consultar la sección 5.5.4.

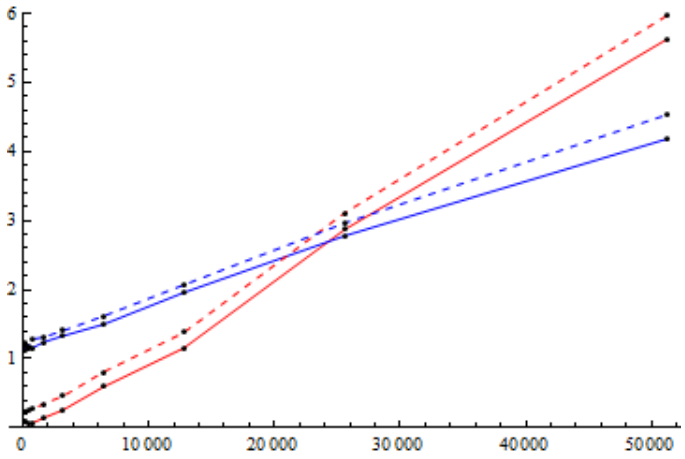

Figura 4.2: Comparativa gráfica de tiempos de ejecución entre el uso del parser y del compilador

4.3. Parser

La implementación manual de un parser es una opción arriesgada, ya que es imposible comprobar su correcto funcionamiento para todas las cadenas de caracteres posibles y muchas de las operaciones posteriores se basan en la asunción de que la cadena con la que se trabaja está correctamente formada. Para asegurar la fiabilidad del parser hemos basado todas sus operaciones sobre la estructura de una gramática formal que ahora pasamos a definir.

4.3.1. Gramática formal

Definición

Definimos la gramática que genera expresiones válidas para el potencial como:

$$
\begin{aligned}
&G = \{N,\ T,\ S,\ R\}\\
&N = \{S,\ T_1,\ {T_1}^{-s},\ T_2,\ {T_2}^{s},\ T_3,\ O_1,\ O_2,\ O_3,\ O_U,\ F\}\\
&T = \{+,\ -,\ *,\ /,\ \hat{},\ (,\ ),\ Exp,\ Log,\ Sqrt,\ Cos,\ Sin,\ n \in \mathbb{R},\ a \in L\}\\
&R = \{S \to T_1,\ T_1 \to {T_2}^{s},\ {T_2}^{s} \to T_2,\ {T_1}^{-s} \to T_2,\ T_2 \to T_3,\ T_3 \to (S)|F,\\
&\quad T_1 \to T_1 O_1 {T_1}^{-s},\ {T_1}^{-s} \to {T_1}^{-s} O_1 {T_1}^{-s},\\
&\quad T_2 \to T_2 O_2 T_2,\ {T_2}^{s} \to O_1 T_2,\ T_3 \to T_3 O_3 T_3 | O_U(S),\\
&\quad O_1 \to +|-,\ O_2 \to *|/,\ O_3 \to \hat{},\ O_U \to Exp|Log|Sqrt|Cos|Sin,\ F \to a|n\}
\end{aligned}
$$

donde $n \in \mathbb{R}$ implica que n puede ser cualquier número real y $a \in L$ significa que a puede ser cualquier cadena alfabética válida.

Explicación

Es necesaria una breve explicación de los diversos símbolos no terminales en la gramática para una mejor comprensión:

S: es el símbolo inicial a partir del cual comienza la construcción de la expresión.

F: representa cualquier elemento final e indivisible de la expresión, como valores numéricos o parámetros.

T_1: representa un término de la expresión que contiene únicamente términos con mayor o igual prioridad aritmética que la *suma* o la *resta*, y que por tanto pueden considerarse indivisibles frente a estas operaciones.

${T_1}^{-s}$: es el equivalente a T_1 pero con la restricción de que no puede ir precedido por un signo.

T_2: representa un término de la expresión que contiene únicamente términos con mayor o igual prioridad aritmética que la *multiplicación* o la *división*, y que por tanto pueden considerarse indivisibles frente a estas operaciones.

${T_2}^{s}$: es el equivalente a T_2 pero permite la opción de que el término vaya precedido por un signo.

T_3: representa un término con la mayor prioridad aritmética posible y por tanto indivisible ante las operaciones de prioridad inferior.

O_1: representa los operadores *suma* y *resta*.

O_2: representa los operadores *multiplicación* y *división*.

O_3: representa el operador *exponenciación*.

O_U: representa operadores unitarios como la *exponenciación en base e* y el *logaritmo natural*.

Con estas definiciones el significado de las reglas es bastante claro:

$T_1 \to T_1 O_1 {T_1}^{-s},\ T_2 \to T_2 O_2 T_2,\ {T_2}^{s} \to O_1 T_2,\ T_3 \to T_3 O_3 T_3 | O_U(S)$:
Define la construcción de todas las operaciones. Como se puede ver, solo se permite la composición de términos que tienen la misma prioridad o superior que la operación que los une. De esta forma se respeta la prioridad de las operaciones aritméticas.

$S \to T_1,\ T_1 \to {T_2}^{s},\ {T_2}^{s} \to T_2,\ {T_1}^{s} \to T_2,\ T_2 \to T_3,\ T_3 \to (S) | F$:
Recordamos la definición que dimos de los diversos términos, según la

cual se podían considerar como indivisibles frente a las correspondientes operaciones. Podemos convertir cualquier término en uno de orden superior ya que seguirán siendo indivisibles ante las operaciones que les afectaban. Cualquier expresión dentro de un paréntesis debe estar igualmente bien formada y desde el exterior es considerada como un término de orden máximo.

$O_1 \to +|-,\ O_2 \to *|/,\ O_3 \to \wedge,\ O_U \to Exp|Log|Sqrt|Cos|Sin,\ F \to a|n$: Por último, las reglas que convierten los símbolos no terminales con los que estamos trabajando en terminales.

Estructura

Dado un lenguaje, la gramática que lo genera no es única. Queremos hacer énfasis aquí en porqué hemos elegido esta gramática. La respuesta es su estructura, en la cual los únicos bucles posibles en las reglas es produciendo el mismo elemento del que partimos. Partiendo del elemento inicial vamos subiendo niveles en una estructura de forma que es imposible volver a bajar. Al llegar al final nos encontramos con un símbolo terminal o el símbolo inicial, de forma que éste puede analizarse de forma recursiva. Se puede ver de forma gráfica en la figura 4.3.

4.3.2. Validar expresiones

Ahora que tenemos una base formal para discernir qué expresiones son correctas, construimos un algoritmo para tal fin.

Nuestro algoritmo detectará los elementos finales, los cuales obligatoriamente proceden de T_3 e irá retrocediendo niveles en la estructura de la gramática. Si al final encontramos que únicamente tenemos el símbolo S la expresión era correcta.

Algoritmo

Nivel T_3: Localizamos primeramente todos los elementos que provienen de T_3. Los términos en el interior de paréntesis y los argumentos de logaritmos y exponenciales han de ser a su vez expresiones correctas que se validan recursivamente. Una vez validados los sustituimos por un término T_3 del cual tienen que provenir. Igualmente para todos los términos numéricos y parámetros.

Una vez hecho esto buscamos apariciones de términos de la forma $T_3{}^\wedge T_3$ y los sustituimos por T_3. Llegados a este punto, todos los términos que provenían de T_3 han sido reducidos a su origen.

Nivel T_2: Convertimos todos los términos T_3 a T_2. Posteriormente reducimos todos los términos de la forma $T_2 * T_2$ o T_2/T_2 a su origen, T_2.

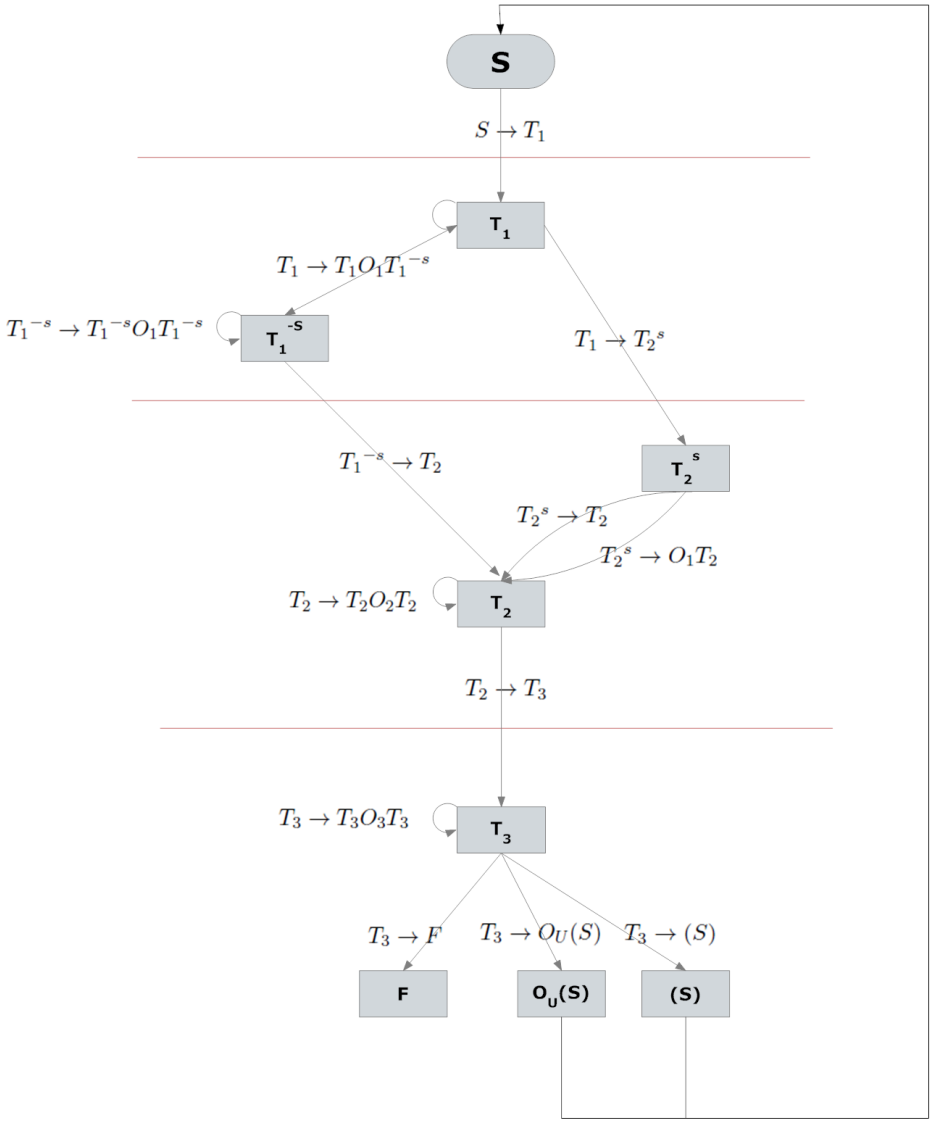

Figura 4.3: Estructura de la gramática

La estructura de las reglas respecto a ${T_1}^{-s}$ y ${T_2}^s$ lo que consigue es asegurar que solo existe la opción de introducir un signo adicional delante del primer término. Si nuestra expresión comienza por $\pm T_2$ lo sustituiremos por T_2.

Nivel T_1: Convertimos todos los términos T_2 a T_1. Reducimos todos los términos de la forma $T_1 \pm T_1$ a T_1. El término T_1 solo puede provenir de S. Si llegados a este punto nuestra expresión es única y exclusivamente S la expresión era correcta. Si es cualquier otra cosa la expresión

original no estaba formada correctamente.

Ejemplo

Procedemos con un ejemplo para visualizar mejor el funcionamiento del algoritmo:

Inicial:	$-(r\hat{}2-3)$	$+$	$Exp(-2+r)$	$*$	4	$+$	$3*2$
T_3:	$-T_3$	$+$	T_3	$*$	T_3	$+$	T_3*T_3
T_2:	$-T_2$	$+$	T_2	$*$	T_2	$+$	T_2*T_2
	$-T_2$	$+$	T_2			$+$	T_2
Signo:	T_2	$+$	T_2			$+$	T_2
T_1:	T_1	$+$	T_1			$+$	T_1
			T_1				
S:			S				

Las expresiones dentro de los paréntesis y los argumentos de logaritmos y exponenciales se analizan recursivamente:

Inicial:	$r\hat{}2$	$-$	3	-2	$+$	r
T_3:	$T_3\hat{}T_3$	$-$	T_3	$-T_3$	$+$	T_3
	T_3	$-$	T_3	$-T_3$	$+$	T_3
T_2:	T_2	$-$	T_2	$-T_2$	$+$	T_2
Signo:	T_2	$-$	T_2	T_2	$+$	T_2
T_1:	T_1	$-$	T_1	T_1	$+$	T_1
		T_1			T_1	
S:		S			S	

4.3.3. Generar ecuación

De nuevo nos apoyamos en la gramática definida para asegurar que el proceso seguido no deja cabida a imprevistos. Nuestra implementación de la ecuación es recursiva, de forma que una ecuación es un término final y tiene un valor, o bien es un término no final y se compone de un operador actuando sobre una o dos ecuaciones subordinadas. Por tanto construimos la ecuación siguiendo los mismos pasos mediante los que la gramática construye sus expresiones.

Algoritmo

Nivel S: Todas las expresiones han de partir de S. Nosotros generamos una ecuación vacía que irá alojando jerárquicamente los distintos términos.

Nivel T_1: Buscamos operadores suma y resta que no se encuentren en el interior de paréntesis. Construimos dos nuevas ecuaciones con las subcadenas a la izquierda y a la derecha del operador y las unimos mediante el operador correspondiente. Dado que ya hemos comprobado que la cadena está bien formada, sabemos que si la expresión comienza con un signo + o -, éste ha de provenir del siguiente paso, y por tanto no se considerará como proveniente de la regla $T_1 O_1 {T_1}^{-s}$.

Nivel ${T_2}^s, {T_1}^{-s}$: Si nuestra expresión va precedida de un signo negativo construimos una nueva ecuación con el resto de la expresión afectado por el operador *signo*.

Nivel T_2: Al igual que en el nivel T_1, buscamos operadores multiplicación y división y construimos ecuaciones con los factores a ambos lados.

Nivel T_3: Buscamos el operador exponenciación y operamos al igual que en los casos anteriores. Posteriormente buscamos los operadores unitarios logaritmo y exponenciación natural y generamos una nueva ecuación con el contenido de sus argumentos. Por último todos los términos contenidos en un paréntesis se consideran como un único término T_3. Todo el contenido del paréntesis se considera como una nueva ecuación y se genera de forma recursiva.

4.3.4. Conclusión

Hemos definido una gramática formal sobre la cual sustentar nuestro parser. A partir de ella hemos construido sendos algoritmos para analizar expresiones y para construir ecuaciones. De esta forma podemos garantizar la corrección de nuestro parser ante cualquier tipo de expresión.

4.4. Técnicas Metodológicas

Resulta necesario seguir una metodología que preste apoyo a todo el ciclo de vida para poder establecer un enfoque sistemático y disciplinado a la hora de llevar a cabo el desarrollo de un producto software.

Desde el punto de vista de la ingeniería del software, una metodología es un proceso para producir software de forma organizada, empleando una colección de técnicas y convenciones de notación predefinidas.

De esta manera, con una metodología se intentan cubrir las siguientes necesidades:

- Mejores aplicaciones.
- Mejor proceso de desarrollo.
- Establecer un proceso estándar en una organización.

Existen varios tipos de metodologías, y de entre todas ellas se decidió seguir una orientada a objetos, concretamente el proceso unificado.

Las metodologías orientadas a objetos acortan la distancia existente entre el espacio de conceptos, lo que los expertos o usuarios conocen, y el espacio de diseño e implementación. Esto supone que el sistema tendrá mucha más relación con los conceptos semánticos sobre los que trata que si se hubiese utilizado una metodología de otra índole.

Además estas metodologías pueden manejar sistemas de gran complejidad, permiten arquitecturas software estructuradas en niveles o capas para permitir un acceso flexible, versátil e independiente a cada una de ellas, y hoy día la mayoría de todo sistema software con una calidad notable ha sido desarrollado bajo el paradigma objetual.

En cuanto al proceso unificado, podemos mencionar que es más que un simple proceso [3], es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyectos.

Como características generales se pueden destacar dos:

- Está basado en componentes.
- Utiliza UML.

El producto que se obtiene como resultado de seguir esta metodología es un sistema software, compuesto por todos los artefactos (cualquier tipo de información creada, producida, cambiada o utilizada por los trabajadores en el desarrollo del sistema) necesarios para representarlo de forma comprensible.

El artefacto más importante del Proceso Unificado es el modelo, y así un sistema posee una colección de modelos y las relaciones entre ellos.

Los modelos obtenidos durante el desarrollo del sistema son los siguientes:

Modelo de casos de uso: Compuesto por los diagramas de casos de uso, donde se plasman los requisitos de la aplicación.

Modelo de análisis: Formado por los diagramas de clases y de secuencia de análisis, obtenidos como refinamiento de los diagramas anteriores y que esbozan la vista estática y dinámica del sistema.

Modelo de diseño: Formado por los diagramas de clases y de secuencia de diseño, obtenidos como refinamiento de los diagramas anteriores y que presentan ya de manera firme la vista estática y dinámica del sistema.

Las características principales del proceso unificado son:

Es un proceso conducido por casos de uso

- Los casos de uso guían el desarrollo del sistema.
- Como los casos de uso contienen las descripciones de las funciones, afectan a todas las fases y vistas.
- Esto lo hemos podido comprobar en la obtención de los modelos del sistema donde se tenía presente las especificaciones de los casos de uso.

Está centrado en la arquitectura

- La arquitectura se presenta mediante vistas del modelo que capturan diferentes aspectos del sistema.
- Se puede tomar como arquitectura de referencia el modelo de arquitectura 4+1 vistas propuesto por Philippe Kruchten (1995). Estas vistas son: lógica, vista de implementación, vista de procesos, vista de despliegue y vista de casos de uso.

Es iterativo e incremental

- En cada iteración se identifican y especifican los casos de uso relevantes, se crea un diseño basado en la arquitectura seleccionada, se implementa el diseño mediante componentes y se verifica que los componentes satisfacen los casos de uso.
- Si una iteración cumple con sus objetivos se pasa a la siguiente.
- En cada iteración se va desarrollando el sistema de forma incremental.

En la figura 4.4 se presenta un gráfico donde puede observarse la vida del proceso unificado y donde puede observarse claramente esa idea de iteratividad que le caracteriza.

El proceso unificado se repite a lo largo de una serie de ciclos de desarrollo que constituyen la vida de un sistema, finalizando cada uno de los ciclos con una versión entregable del producto.

Cada ciclo de desarrollo consta de dos grandes etapas, cada una de las cuales formada por dos fases:

Etapa de ingeniería

- Fase de inicio: Se define el alcance del proyecto y se desarrollan los casos de negocio.
- Fase de elaboración: Se planifica el proyecto, se especifican en detalle la mayoría de los casos de uso y se diseña la arquitectura del sistema.

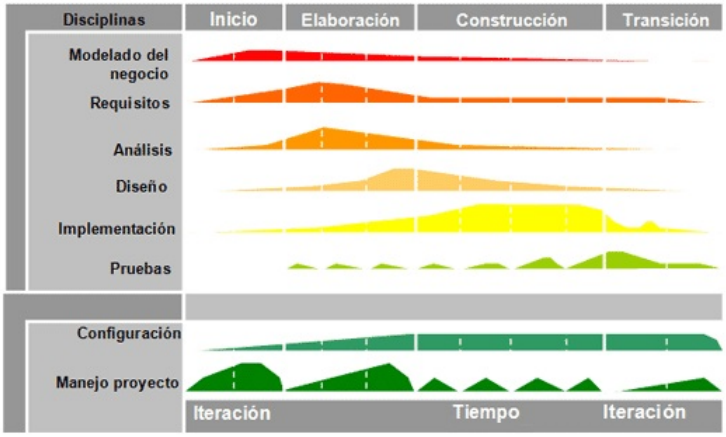

Figura 4.4: Fases del proceso unificado

Etapa de producción

- Fase de construcción: Se construye el producto.
- Fase de transición: El producto se convierte en versión beta y se corrigen problemas y se incorporan mejoras sugeridas en la revisión.

Cada fase está formada por un conjunto de iteraciones, y al final de cada fase se termina con un hito principal. A su vez en cada iteración se pasa por todas las disciplinas o flujos de trabajo, obteniendo al final lo que se conoce como hito secundario. Los hitos son puntos de control en los cuales los participantes en el proyecto revisan el progreso del mismo.

En el caso del sistema actual, un hito principal era el momento en el que el desarrollador tras haber realizado las pruebas oportunas, se reunía con el cliente para que éste validara el trabajo realizado y junto con el desarrollador se establecieran los objetivos para la siguiente fase.

A su vez, los hitos secundarios, se caracterizaban por ser el momento en que el desarrollador realizaba las pruebas oportunas y organizaba el trabajo para la siguiente iteración.

Respecto a las disciplinas o flujos de trabajo, decir que sirven para organizar las actividades fundamentales de gestión y desarrollo del proyecto y que al contrario que las fases, las disciplinas pueden solaparse en el tiempo. Durante el desarrollo del presente proyecto se han seguido las siguientes disciplinas:

- Requisitos y Análisis.
- Diseño.
- Implementación.

- Pruebas.

Dentro de las metodologías orientadas a objetos, el desarrollador se decantó por Proceso Unificado pues se disponía de suficiente documentación y conocimiento sobre éste adquirido en la asignatura de Ingeniería del Software.

4.5. Recursos y herramientas

4.5.1. Plataforma Java

La plataforma Java es el nombre de un entorno o plataforma de computación originaria de Sun Microsystems, capaz de ejecutar aplicaciones desarrolladas usando el lenguaje de programación Java u otros lenguajes que compilen a 'bytecode' y un conjunto de herramientas de desarrollo.

En este caso, la plataforma no es un hardware específico o un sistema operativo, sino una máquina virtual encargada de la ejecución de aplicaciones y un conjunto de librerías estándar que ofrecen funcionalidad común.

Más información en: `http://java.sun.com`

Arquitectura Plataforma Java

La plataforma Java se encuentra por encima de otras plataformas. El código que generan sus compiladores no es específico de una maquina física en particular, sino de una máquina virtual. Aún cuando existen múltiples implantaciones de la máquina virtual Java (JVM), cada una específica de la plataforma sobre la cual subyace, existe una única especificación de la máquina virtual, que proporciona una vista independiente del hardware y del sistema operativo sobre el que se esté trabajando. De esta manera un programador en Java escribe su programa una vez y lo ejecuta en cualquier máquina que posea una JVM.

Es precisamente la máquina virtual Java la clave de la independencia de los programas Java sobre el sistema operativo y el hardware en que se ejecutan. Es la encargada de proporcionar la vista de un nivel de abstracción superior, donde además de la independencia de la plataforma antes mencionada, presenta un lenguaje de programación simple, orientado a objetos, con verificación estricta de tipos de datos, múltiples hilos y con recolección automática de basura.

Biblioteca gráfica Swing

Swing es una biblioteca gráfica para Java que forma parte de las Java Foundation Classes (JFC). Incluye widgets para la interfaz gráfica de usuario tales como cajas de texto, botones, desplegables y tablas. Posee las siguientes características principales:

Independencia: se trata de una plataforma independiente.

Extensibilidad: es una arquitectura altamente particionada, los usuarios pueden proveer sus propias implementaciones modificadas para sobrescribir las implementaciones por defecto. Se puede extender clases existentes proveyendo alternativas de implementación para elementos esenciales.

Personalizable: dado el modelo de representación programático del 'framework' de Swing, el control permite representar diferentes 'Look and Feel' (desde MacOS look and feel hasta Windows XP look and feel).

JavaHelp System

JavaHelp es una expansión de Java que facilita la programación de las ventanas de ayuda en las aplicaciones Java. Las ventanas de ayuda de JavaHelp se configuran por medio de varios ficheros en formato XML. Los textos de ayuda que se quieran mostrar se escribirán en ficheros con formato HTML, con lo cual la ayuda queda igualmente disponible para su integración en otros formatos de publicación. JavaHelp no se incluye en la JDK, ni en la JRE, sino que debe conseguirse como un paquete externo.

Más información en: `https://javahelp.dev.java.net/`

4.5.2. Eclipse IDE

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de cliente enriquecido". Esto implica que se usan módulos para proporcionar funcionalidades nuevas, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas.

Esta plataforma ha sido usada típicamente para desarrollar entornos de desarrollo integrados, como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse y que son usados también para desarrollar el mismo Eclipse.

Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Más información en: `http://www.eclipse.org/`

4.5.3. Control de versiones Subversion (SVN)

Para un mantenimiento óptimo de las versiones de la aplicación ha sido necesario el uso de un control de versiones mediante el sistema Subversion.

Subversion es un software de control de versiones diseñado específicamente para reemplazar al popular CVS, el cual posee varias deficiencias.

Como ventajas de SVN cabe destacar:

- Se sigue la historia de los archivos y directorios a través de copias y renombrados.
- Las modificaciones, incluyendo cambios a varios archivos, son atómicas.
- La creación de ramas y etiquetas es una operación más eficiente. Tiene costo de complejidad constante $\mathcal{O}(1)$ y no lineal $\mathcal{O}(n)$ como en CVS.
- Maneja eficientemente archivos binarios, a diferencia de CVS que los trata internamente como si fueran de texto.
- Permite selectivamente el bloqueo de archivos. Se usa en archivos binarios que, al no poder fusionarse fácilmente, conviene que no sean editados por más de una persona a la vez.

Como cliente se ha utilizado la interfaz visual TortoiseSVN. Se trata de un cliente Subversion implementado como una extensión al 'shell' de Windows.

Más información en `http://tortoisesvn.tigris.org/`

Adicionalmente se ha usado el plugin para Eclipse IDE llamado Subversive, que integra en nuestra herramienta de trabajo la posibilidad de conectarse con el repositorio para confirmar, actualizar o revertir los cambios del proyecto.

Más información en `http://www.eclipse.org/subversive/`

4.5.4. Requisite Management (REM)

REM es una herramienta experimental gratuita de gestión de requisitos diseñada para ayudar en la fase de Ingeniería de Requisitos. Recurre a la metodología definida en la tesis doctoral "Un entorno metodológico de ingeniería de requisitos para sistemas de información", presentada por Amador Durán en septiembre de 2000.

Esta herramienta se ha utilizado durante todo el proceso de análisis y mediante ella se ha obtenido la mayoría de la documentación que figura en el apéndice B, "Documentación de requisitos del software".

Más información en:
`http://www.lsi.us.es/descargas/descarga_programas.php?id=3`

Html to Latex

La documentación generada por la herramienta REM se obtiene en formato Html. Para su conversión a formato LaTeX, en el cual ha sido escrito este documento, ha sido utilizada la herramienta *HtmlToLatex*.

Más información en: `http://htmltolatex.sourceforge.net`

4.5.5. Visual Paradigm for UML

Visual Paradigm es una herramienta de diseño y análisis UML que cubre el desarrollo de software desde el paso de los requerimientos a través de las etapas del análisis, modelos de diseño, pruebas y mantenimiento.

Algunas de sus características principales son:

- Diseño y construcción de UML.
- Casos de uso, modelos lógico, dinámico y físico.
- Intuitivo y simple de usar.

Más información: `http://www.visual-paradigm.com/product/vpuml/`

Capítulo 5

Desarrollo

5.1. Patrón Modelo-Vista-Controlador

Dada la naturaleza del proyecto a desarrollar, el uso del patrón Modelo-Vista-Controlador es una elección obligada. Podemos ver que se dan todos los requisitos y problemas típicos para los cuales es indicado el uso de este patrón.

- La misma información es presentada de maneras diferentes en distintas ventanas.
- La presentación y el comportamiento de una aplicación deben representar los cambios en los datos inmediatamente.
- Los cambios en la interfaz de usuario deben ser sencillos e incluso factibles en tiempo de ejecución.
- Evoluciones en la interfaz de usuario no deben afectar al código del núcleo de la aplicación.

Por todo ello necesitamos separar el núcleo funcional de la aplicación de su parte gráfica. Esto facilitará el desarrollo y mantenimiento del proyecto.

5.2. Ciclo de vida y metodología

Dado que la aplicación desarrollada hace uso de la orientación a objetos, debe utilizarse una metodología que refleje tal hecho, por lo que se opta por escoger la metodología del Proceso Unificado. Entre sus características principales [Jacobson et al.,1999] cabe destacar:

- Es un proceso conducido por casos de uso.
- Centrado en la arquitectura.

- Es iterativo e incremental.

Atendiendo al aspecto incremental se subdividió inicialmente el sistema global en tres subsistemas, cada uno de ellos con funcionalidades bien diferenciadas:

Entrada de datos: El usuario introduce los datos necesarios para el cálculo junto con la expresión del potencial. El sistema deberá recoger y almacenar los datos de entrada y presentar una interfaz a través de la cual el usuario interactúe. Además en el caso del potencial deberá realizar el preprocesado necesario de la expresión para construir los objetos correspondientes.

Cálculo numérico: El sistema obtiene las soluciones al potencial introducido. El usuario no participa ni interactúa durante esta fase con la aplicación.

Presentación de resultados: El sistema presenta de forma gráfica los resultados al usuario. El usuario puede guardar los resultados o exportarlos en los formatos correspondientes. También puede decidir refinar los resultados para obtener una mayor precisión.

Dado que estos subsistemas no actúan simultáneamente sino secuencialmente es perfectamente posible desarrollar cada uno de ellos por separado. Únicamente hay que tener en cuenta los datos u objetos que se precisan del subsistema anterior y cómo se van a transferir de un subsistema a otro.

En este caso optamos por el desarrollo del núcleo numérico de la aplicación en primer lugar. Lo hacemos así ya que es la pieza más importante de la aplicación y en caso de necesidad se puede ajustar la entrada de datos o la presentación de resultados a los requerimientos del núcleo numérico. Los datos necesarios del subsistema anterior se incluirán directamente en el código y la presentación de resultados se realizará de forma temporal escribiendo en archivos.

Posteriormente se realizará el desarrollo del subsistema de entrada de datos. En esta etapa se dotará a la aplicación de una interfaz para el usuario y se realizarán las operaciones necesarias para convertir los datos introducidos por el usuario al formato necesario para un posterior procesamiento por parte del núcleo numérico.

Por último se dotará al proyecto de una interfaz para la visualización de los resultados. Dado que es probable que parte del código de la interfaz pueda ser compartido por el subsistema de entrada y el de presentación de resultados, se prestará especial atención a la posible reutilización de elementos entre ambos subsistemas.

Cada uno de estos subsistemas se desarrollará en diversas iteraciones. Al acabar la iteración, es necesaria una reflexión sobre los objetivos y los progresos cumplidos, validando los resultados y decidiendo los objetivos de la siguiente iteración.

5.3. Análisis

Dadas las características del proyecto, la fase de análisis tuvo una importancia vital. Establecer los requisitos de una forma concreta, clara y concisa ayudó a la comprensión del sistema a desarrollar y a la búsqueda de la mejor solución al mismo. Para realizar esta tarea de manera uniforme se ha hecho uso de la "Metodología para la elicitación de requisitos del sistema software versión 2.3" descrita en [1].

5.3.1. Casos de uso

En un primer momento, los objetivos preliminares del sistema estaban orientados en los siguientes puntos:

Hallar soluciones: Naturalmente el objetivo principal del proyecto es su fin último, queremos hallar las soluciones a la ecuación de Schrödinger para cualquier potencial.

Control de los parámetros: El usuario ha de tener libertad para controlar no solo los parámetros físicos relacionados con el problema, sino también los parámetros relacionados con el cálculo numérico. De esta forma puede adaptar a sus necesidades la precisión y la duración del cálculo.

Gestión de los experimentos: Dado que ciertos cálculos pueden llegar a tener una duración considerable, se considera necesario que el usuario pueda guardar y cargar los resultados si lo desea, e igualmente pueda guardar experimentos para los que aún no se han realizado cálculos

Exportar resultados: Los resultados obtenidos por el programa ganan en utilidad si pueden ser usados fuera del propio programa. Por tanto se considera positivo que se puedan exportar los resultados en diversos formatos para posibles futuros usos.

Para estos objetivos se generaron los Casos de Uso mostrados en la figura 5.1.

5.3.2. Arquitectura

El desarrollo de la arquitectura inicial viene determinado por el uso del patrón MVC. En la figura 5.2 podemos observar como el sistema se construye

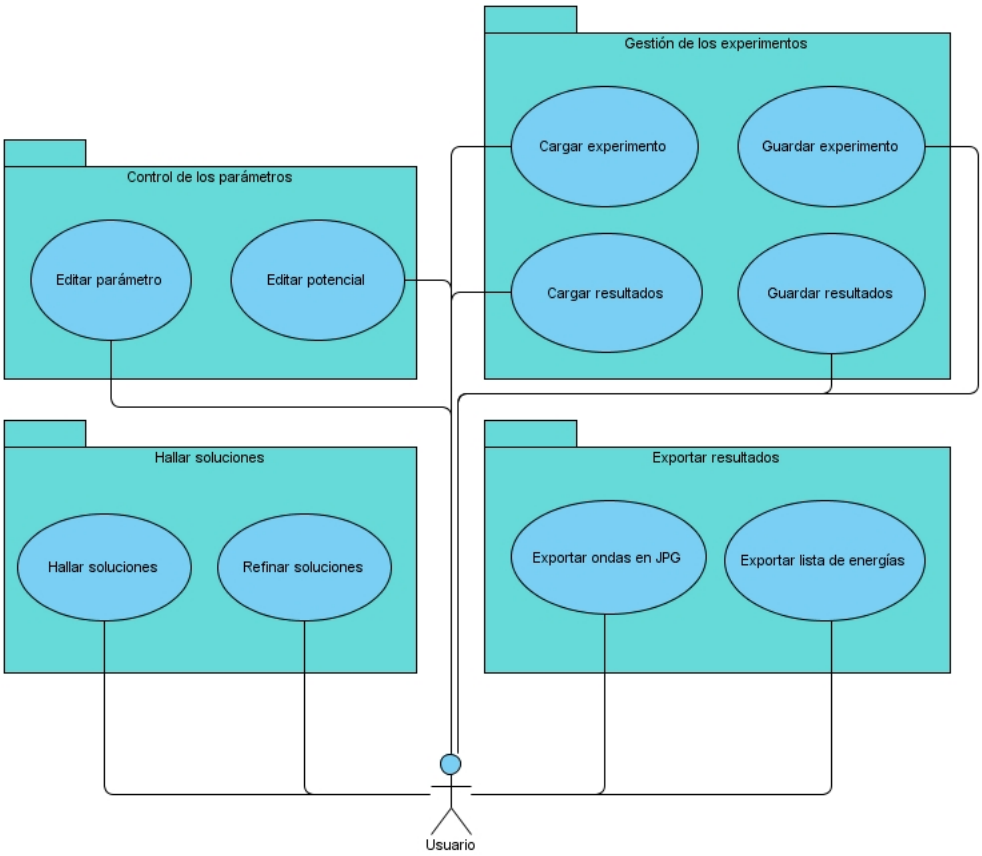

Figura 5.1: Casos de uso de la aplicación

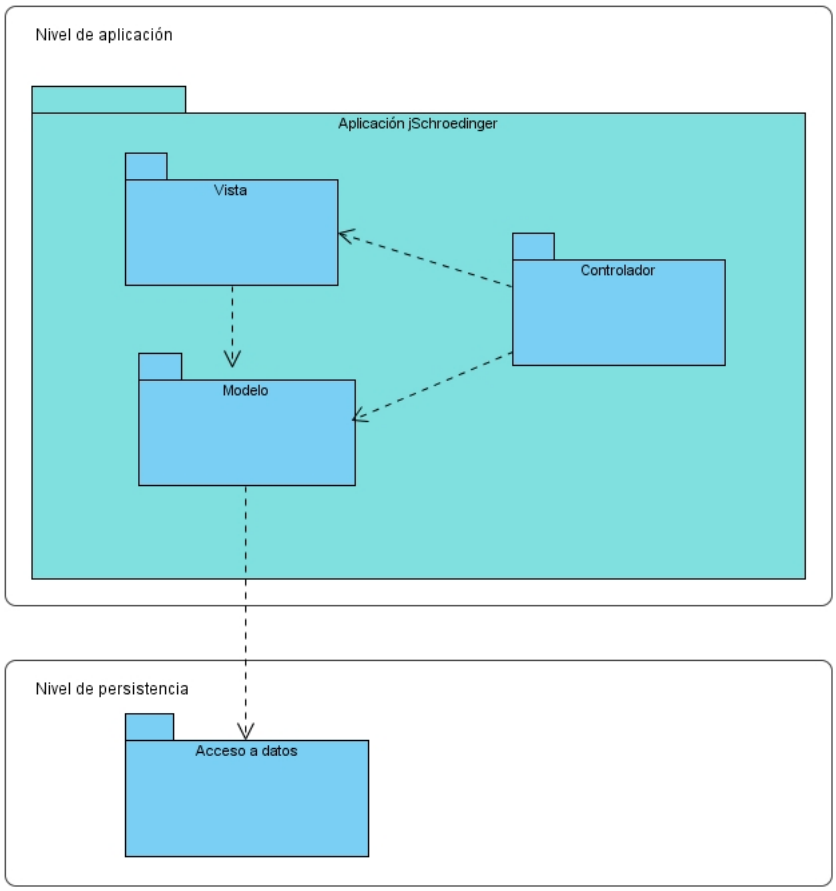

Figura 5.2: Arquitectura inicial

en torno al Modelo Vista Controlador. Bajo este modelo vamos a estructurar la administración de datos en diferentes módulos o paquetes, separando la interfaz, los eventos del sistema y la lógica de la aplicación en tres paquetes diferentes. De este modo se hace posible que una ampliación del proyecto sea más sencilla.

Este modelo de arquitectura es crucial ya que nos sirve de entrada al diseño, donde se refinará añadiéndole nuevos paquetes, dependencias y capas más específicas.

En el siguiente apartado se observa como el refinamiento de la arquitectura de clases queda englobado en este marco. Sin embargo hay que señalar que debido al funcionamento de Java, las vistas y los controladores están implementados dentro de la misma clase. El paquete gráfico Swing provee de interfaz a los elementos y la adición de Listeners a estas clases hace que actúen como controladores, recogiendo la actividad del usuario.

5.3.3. Almacenamiento y recuperación de información

Como respuesta a uno de los requisitos de la aplicación, se propuso poder almacenar los progresos hechos por el usuario en un fichero binario externo y con ello dotar de la posibilidad de poder continuar el trabajo en cualquier momento desde el punto donde lo dejó.

Es por ello, que tanto en la fase de análisis como en diseño se refinaron las siguientes pautas:

- Se optó por implementar una clase central que engloba el estado del experimento en cada momento, de esta forma el volcado de esta clase a disco almacena toda la información relativa al experimento y a los parámetros definidos por el usuario. De igual manera podemos cargar de nuevo esta clase desde un archivo y recuperar todo el trabajo realizado por el usuario.

- Igualmente debe haber una clase responsable de toda la información y resultados generados por el cálculo. De esta forma se evita repetir cálculos que ya se han realizado previamente, ahorrando tiempo al usuario.

- Por otra parte, se buscó la manera de exportar todos los contenidos gráficos generados, de forma que se puedan usar fuera del marco de la aplicación.

5.4. Diseño

El diseño se centra en la obtención de un modelo físico. Se pretende realizar un refinamiento de los artefactos obtenidos durante el análisis de forma que se pueda perfilar la solución final. El diseño, a diferencia del análisis está muy ligado a la implementación.

5.4.1. Diseño arquitectónico

En esta fase se produce la identificación y documentación de los subsistemas que conforman el sistema completo, es decir se describe la arquitectura de alto nivel. Para ello se identifican y documentan las relaciones entre los subsistemas con el fin de obtener una estructura modular y así poder representar el control entre módulos. Antes de entrar en detalle a identificar las clases específicas que conformarán el proyecto podemos hacer un primer refinamiento en paquetes para ayudarnos a englobar y situar las futuras clases. Podemos ver en la figura 5.3 el desglose de los paquetes.

A partir de esta estructura comenzamos una identificación precisa de los objetos del problema. Este proceso de refinamiento e identificación de

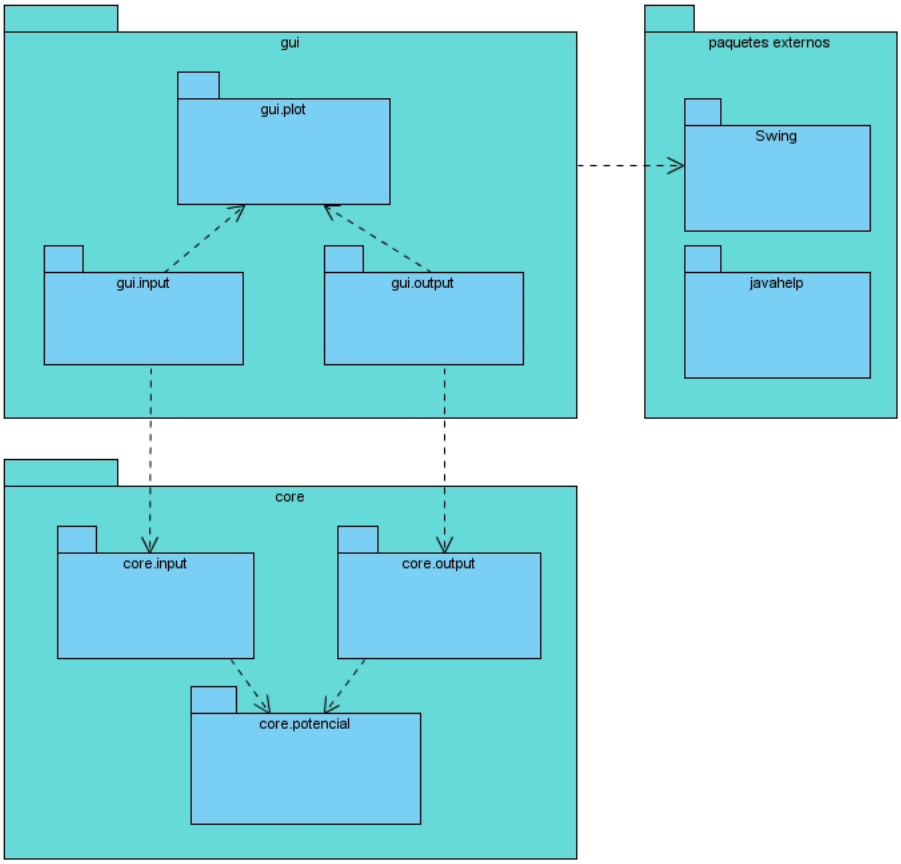

Figura 5.3: Estructura de paquetes

clases nos lleva a un primer diagrama de clases en el que se pretende capturar la estructura conceptual del sistema. El refinamiento de estas clases en las posteriores etapas de desarrollo conducirá a un modelo de clases más avanzado.

Diagrama de clases

El diagrama de clases recoge todo el proceso de análisis y diseño que lleva a establecer las tareas de cada clase y su relación con el resto de clases para completar la funcionalidad esperada del sistema. El diagrama de clases de la figura 5.4 nos da una visión global del sistema sin especificar el funcionamiento último de cada clase. Este diagrama de clases no pretende ser una especificación final del sistema sino un paso más en el proceso de diseño refinando la estructura del sistema. Para un mayor detalle de cada clase y de la interacción entre ellas se recomienda consultar el apéndice C, "Especificación de diseño".

5.4.2. Diseño de la interfaz

Interfaz de datos

La figura 5.5 muestra la interfaz de datos. Con esta interfaz interactuará la mayor parte del tiempo el usuario, definiendo el experimento que desea resolver. La interfaz consta de cuatro zonas, cada una de ellas etiquetada en función a su contenido.

Potencial: en el recuadro superior izquierdo tenemos la entrada de datos para la expresión del potencial. En ella podremos definir nuestro potencial y los parámetros que deseemos. Para cada parámetro nuevo que definamos se creará un nuevo parámetro de forma que podamos editar su valor.

Vista previa: en el recuadro superior derecho tendremos en todo momento la visualización del potencial que estamos definiendo, dentro del intervalo en el que se desean buscar soluciones.

Parámetros: en el recuadro inferior izquierdo se muestran los parámetros del cálculo. Con ellos podemos definir los límites a la energía y otros parámetros numéricos.

Notificaciones: en el recuadro inferior derecho tendremos un cuadro de texo en el cual se irán mostrando en todo momento las alertas debido a valores incorrectos o no recomendables de los parámetros.

Por último el botón de cálculo, estará activado únicamente si todos los valores de los parámetros son correctos. Presionándolo iniciaremos el cálculo

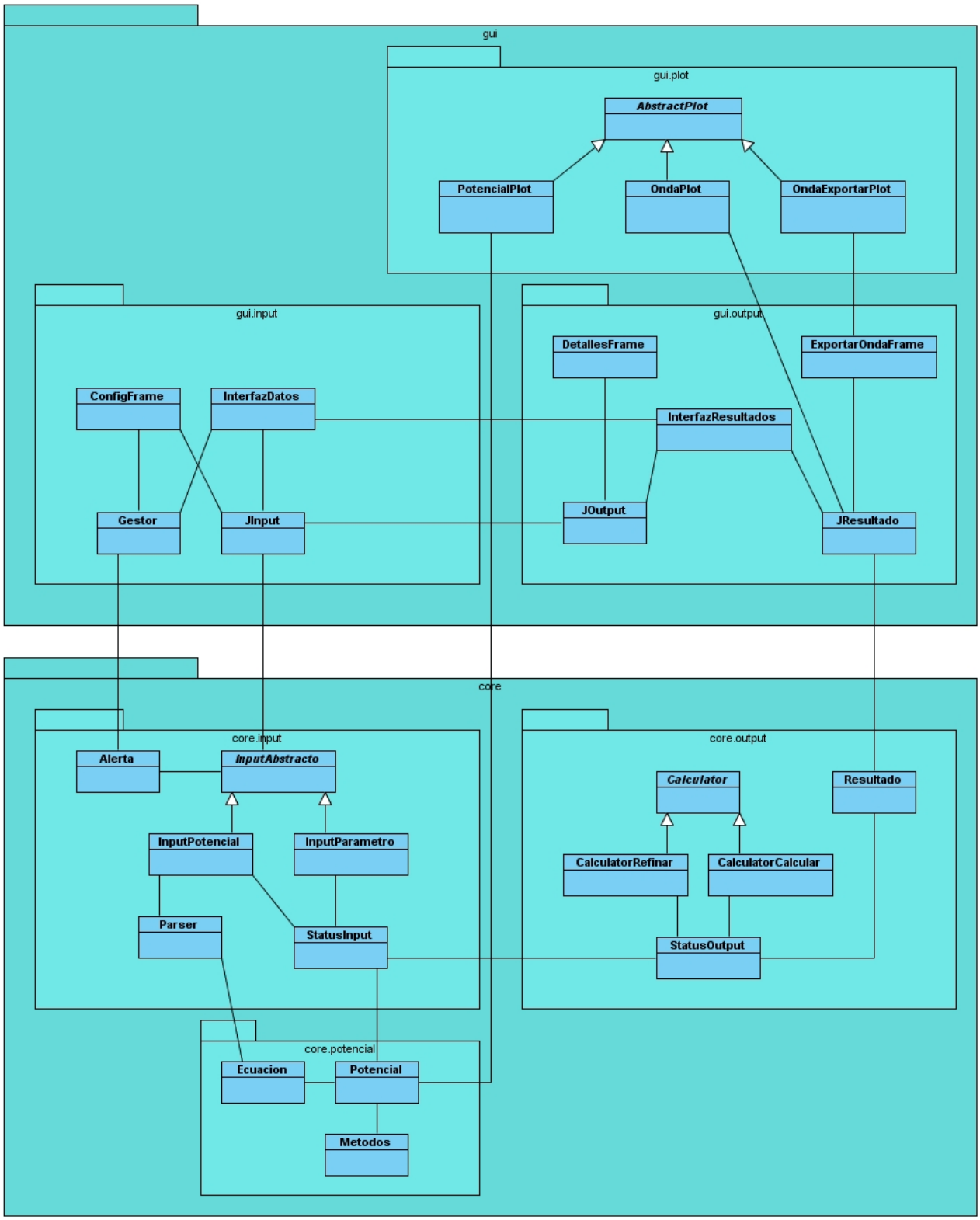

Figura 5.4: Diagrama de clases

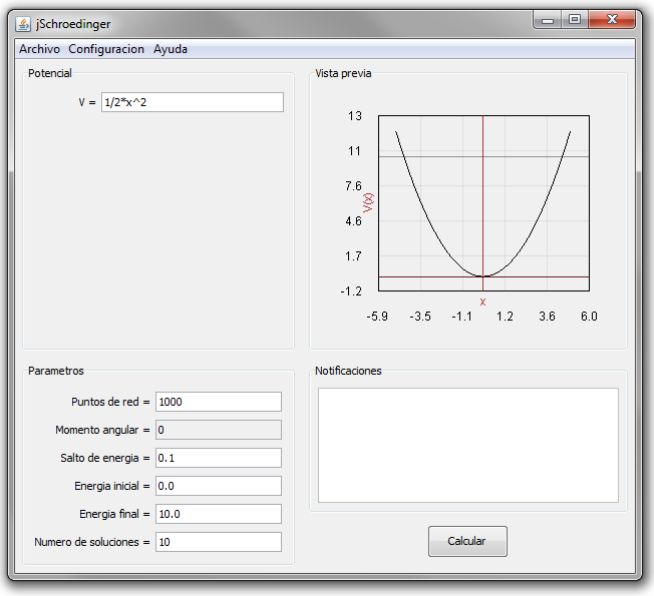

Figura 5.5: Interfaz de datos

y pasaremos a la pantalla de resultados. Para más detalles sobre la interfaz puede consultar la sección 5.4.2, diseño de la interfaz.

Interfaz de resultados

La figura 5.6 muestra la interfaz de resultados. En esta interfaz visualizará el usuario los resultados que se van obteniendo durante el cálculo, la evolución del mismo y la energía en la cual se están buscando soluciones en el momento. La interfaz consta de cuatro zonas, cada una de ellas etiquetada en función a su contenido.

Potencial: en el recuadro superior izquierdo se muestra la expresión del potencial y los parámetros para los que se está calculando o se ha calculado la solución.

Vista previa: en el recuadro superior derecho tendremos en todo momento la visualización del potencial que estamos definiendo, dentro del intervalo en el que se desean buscar soluciones. Durante el cálculo veremos una linea barrer la franja de energías, sirviendo de indicativo para la evolución del cálculo. Cuando se obtenga una solución quedará marcada la energía de la solución mediante una linea discontínua.

Energías: en el lateral izquierdo se muestra un área de texto con los valores de las energías solución que se han hallado hasta el momento.

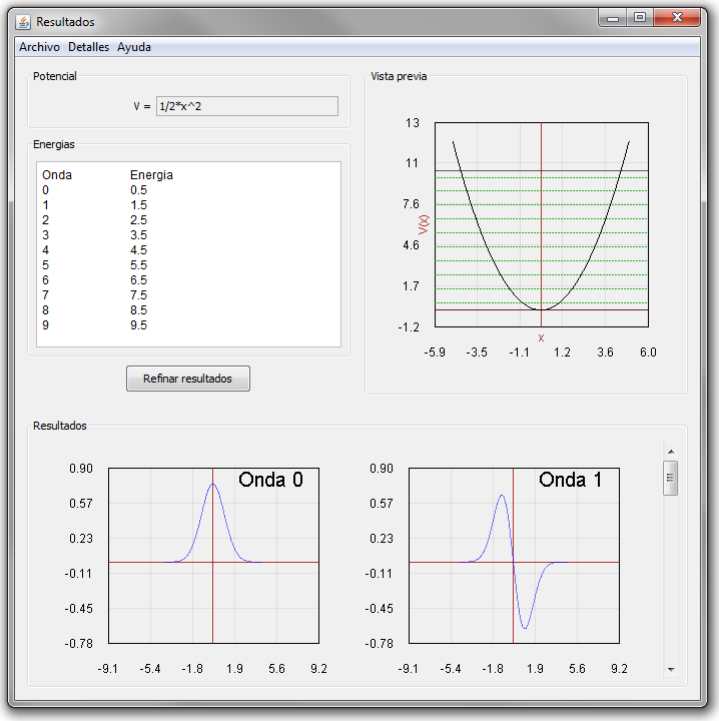

Figura 5.6: Interfaz de resultados

Resultados: en el recuadro inferior tendremos las representaciones de las soluciones de onda halladas, a través de objetos `JResultado`. Haciendo click en ellos podremos ver las mismas soluciones de onda a mayor tamaño

Por último el botón de 'Refinar resultados', estará activado únicamente si el cálculo ha finalizado y se ha obtenido al menos un resultado. Presionándolo obtendremos de nuevo las soluciones pero calculadas con un mayor número de puntos de red y por tanto con una mayor precisión.

5.4.3. Modelo dinámico

Finalmente se describen las interacciones entre las diferentes clases y actores a la hora de realizar los diferentes casos de uso por medio de los diagramas de secuencia. El objetivo es comenzar a realizar un esbozo de los métodos que más tarde formarán parte de las clases del sistema.

En concreto vamos a detallar el proceso de edición de un parámetro por ser uno de los casos de uso más básicos y utilizados. En primer lugar analizamos el caso en que el valor del parámetro no es correcto, por tratarse de una cadena de caracteres o por no cumplir alguna de las restricciones

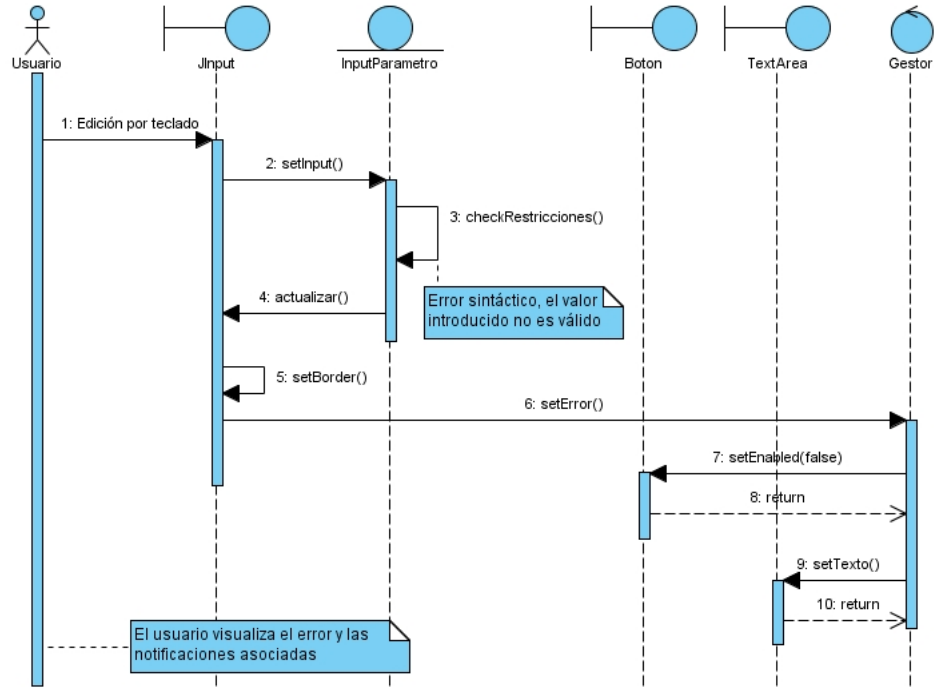

Figura 5.7: Diagrama de secuencia, error sintáctico en un parámetro

impuestas al parámetro (por ejemplo ser un número entero). En este caso el parámetro detecta que el valor no es válido y lo notifica a su recubrimiento gráfico. Éste establece un distintivo gráfico de error, un reborde rojo, y pasa la alerta al gestor el cual se encarga de mostrarla y desactivar el botón de cálculo. La figura 5.7 muestra el diagrama de secuencia correspondiente.

En segundo lugar vamos a considerar un caso más complejo. El usuario edita el potencial y en el contexto del nuevo potencial hay valores de los parámetros que no son correctos, por ejemplo la energía inicial es superior al máximo del potencial. En este escenario hay que notificar al parámetro *Energía inicial* que su valor es semánticamente incorrecto, para lo cual el estado del experimento tiene que comprobar los valores de todos sus parámetros y notificarles que su valor ya no es aceptable, a diferencia del caso anterior en el que el error se producía en el parámetro que recibía la llamada. Este mismo caso también se puede dar al editar parámetros, por ejemplo la energía inicial pasa a ser superior a la final. La figura 5.8 muestra el diagrama de secuencia correspondiente.

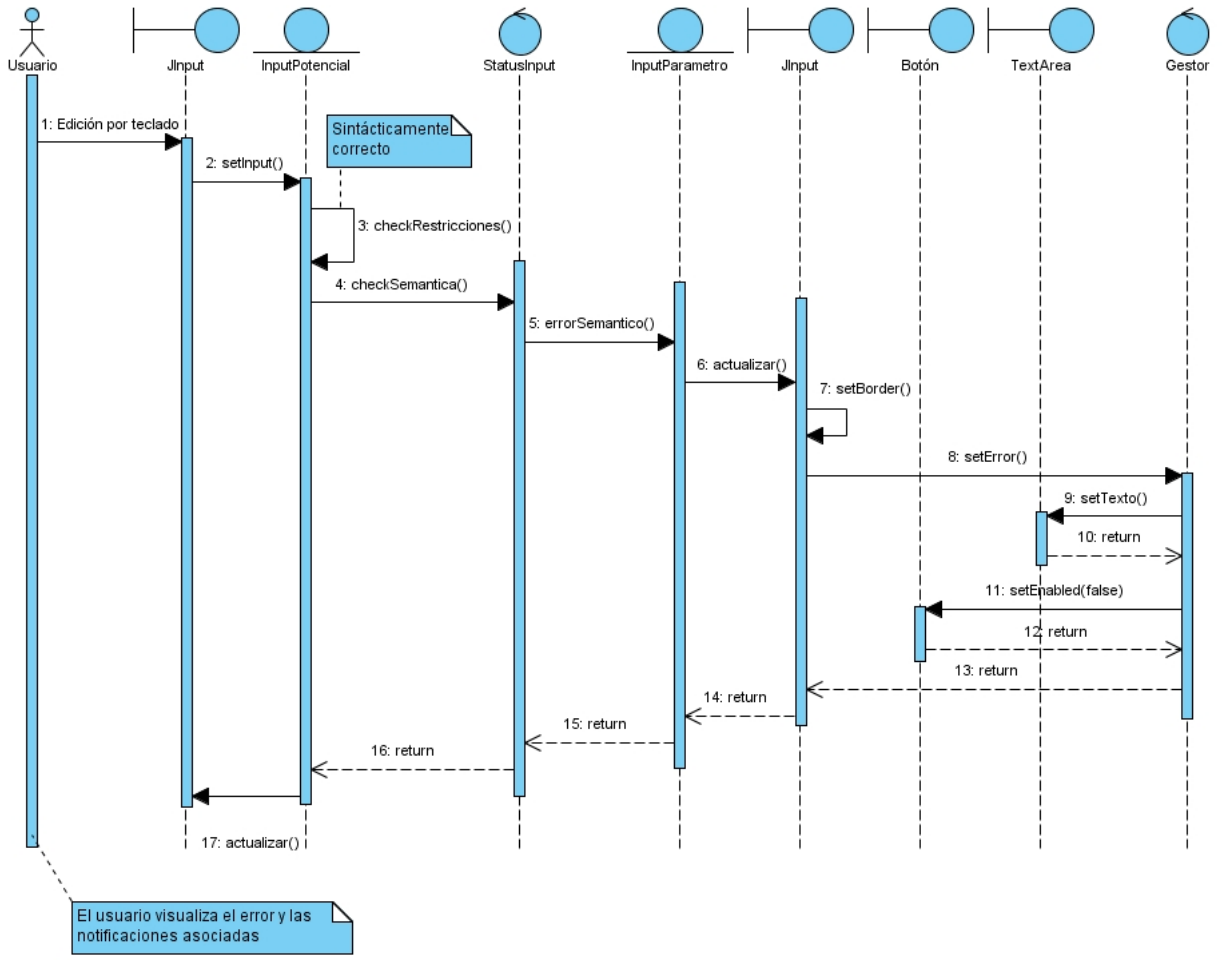

Figura 5.8: Diagrama de secuencia, error semántico en un parámetro

5.5. Implementación

5.5.1. StatusInput y StatusOutput

Las clases `StatusInput` y `StatusOutput` son las clases principales de entrada de datos y de resultados respectivamente. Por tanto creemos necesario comentar su implementación en detalle aquí.

Ambas clases centralizan la información en sus respectivos subsistemas. La serialización de estas clases permite el almacenamiento del estado del experimento o de sus resultados en archivos. Todos los objetos contenidos en estas clases que son capaces de generarse a partir del resto de datos contenidos serán definidos de tipo `transient`, es decir, no serán serializados y al cargar el archivo su valor será `null`. Por ejemplo el potencial o la interfaz gráfica ya que está separada del modelo de datos y es capaz de generar todos sus elementos a partir de una instancia `StatusInput` o `StatusOutput` respectivamente. De esta forma, almacenamos en archivo únicamente la información imprescindible y lo que es más importante, la implementación de la interfaz puede variar libremente y los archivos almacenados con versiones anteriores seguirán siendo compatibles.

StatusInput

La clase `StatusInput` es la clase central en el subsistema de entrada de datos. Almacena todos los parámetros y componentes que definen el experimento. Asimismo `StatusInput` se ha diseñado haciendo uso del patrón de diseño *Singleton* (instancia única) cuya finalidad consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella. Queremos evitar que todas las clases tengan que tener una referencia al modelo, pero al mismo tiempo que esté disponible para todas ellas. Para solucionarlo se ha implementado un método de clase `getInstance()` que devuelve la instancia única de StatusInput. Así pues cada vez que una clase necesite acceder a la información del modelo no tiene más que hacer `StatusInput.getInstance()`. La propia clase es la encargada de gestionar la única instancia existente, para lo cual el constructor se define como privado y se permite el acceso global a dicha instancia a través de un método estático.

```
//Atributo estático privado
private static StatusInput status;

//Constructor privado
private StatusInput(){...}

//Metodo estático que gestiona la única instancia de la clase
public static StatusInput getInstance(){
```

```
   if(status==null)
      status = new StatusInput();
   return status;
}
```

La clase contiene también un método que permite cargar un estado anterior. Esto lo hace copiando los valores sobre el estado actual, de esta forma todos los objetos que hayan obtenido una instancia del estado actual pasan a tener el nuevo estado sin necesidad de solicitar una nueva instancia.

StatusOutput

A diferencia del estado de entrada de datos, que solo puede tener una instancia, el usuario puede desear hacer varios cálculos y mantener abiertos los resultados a fin de compararlos. Por tanto todas las clases que requieran de una instancia de `StatusOutput` deberán recibirla como argumento en sus métodos.

5.5.2. Potencial

La evaluación del potencial en cada punto es una operación que se realiza un número muy elevado de veces durante el cálculo. Como vimos en la sección 4.2, evaluación del potencial, el método elegido implica recursividad y la evaluación de varias subecuaciones hasta llegar al valor final, por tanto la implementación debe ser lo más eficiente posible para evitar evaluaciones innecesarias.

La ecuación del potencial depende de una variable contínua como es la posición. Sin embargo, dado que hemos discretizado el espacio, el cálculo siempre se evaluará en un número finito y fijo de puntos. Al iniciar el objeto potencial iniciamos una tabla hash con clave la posición, y valor la evaluación del potencial en ese punto. De esta forma, al realizar el cálculo no será necesario volver a evaluar la ecuación sino que obtendremos el valor de la tabla hash.

Hay que tener en cuenta que esta mejora solo servirá para el proceso de cálculo, en el que se evalúa siempre en los puntos de red. Otros cálculos como hallar el mínimo del potencial o el punto de retorno para una determinada energía precisarán del valor del potencial en puntos arbitrarios del espacio.

```
public double evaluar(double pos){
   Double temp = hashtable.get(pos);
   if (temp==null){ //La posición no está almacenada en la tabla
      if (l>0 && dimension==3)
         return ecuacion.evaluar(pos)+
               l*(l+1)/(Math.pow(pos,2)*gamma);
```

```
        else
            return ecuacion.evaluar(pos);
    }
    else //La posición sí está almacenada en la tabla
        return temp.doubleValue();
}
```

5.5.3. Cambios de variable

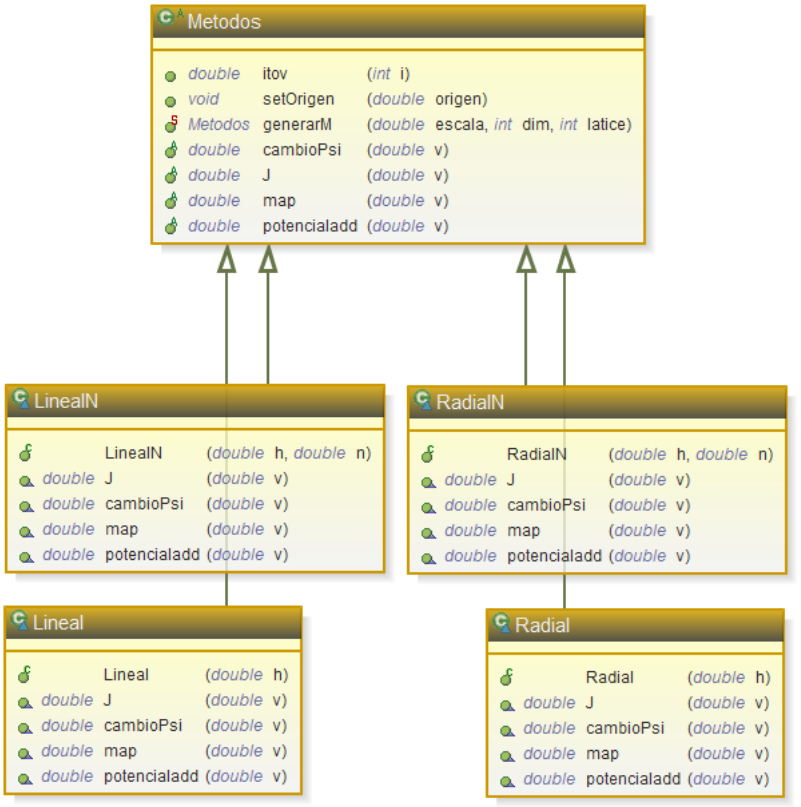

Figura 5.9: Clase Metodos

En la sección 3.3 vimos los cambios de variables necesarios para resolver la ecuación de Schrödinger. La forma de implementar estos cambios de variable ha sido crear una clase abstracta `Metodos` que define todas las operaciones necesarias. Las clases herederas serán las encargadas de implementar estos métodos abstractos. De esta forma solo es necesario implementar una vez el algoritmo numérico actuando sobre la variable espacial normalizada

y posteriormente deshacer los cambios de variable para obtener la solución en el espacio de coordenadas original.

Los métodos `map(double v)`, `J(double v)`, `cambioPsi(double v)` y `potencialAdd(double v)` son los equivalentes a $g(v)$, $a^{-1}(v)$, $u(v)$ y $k^2_{add}(v)$ que definimos en la sección 3.3, cambio de variables.

Para obtener el cambio de variables acorde a nuestro problema haremos uso del método estático `Metodos.generarM(...)`. Este método actúa como constructor llamando al constructor correspondiente de las clases herederas.

La figura 5.9 muestra el contenido de la clase.

5.5.4. Compilación y carga dinámica de clases

Si bien no ha sido la elección final para la evaluación del potencial, creemos necesaria una breve explicación de como ha sido implementado el prototipo con compilación y carga de clases en tiempo de ejecución.

Compilación

La secuencia completa para usar el método de compilación y carga dinámica sería la siguiente:

- El usuario introduce la expresión deseada a través de la interfaz.
- La aplicación escribe la expresión en un archivo que contiene el código fuente del potencial a falta de la expresión que introducirá el usuario, en nuestro ejemplo `PotencialLinker.java`.
- La aplicación compila el nuevo código fuente de la clase con la expresión del potencial introducida.
- La aplicación carga la nueva clase, la cual lleva en su código la expresión introducida por el usuario.

Si la compilación es correcta habremos obtenido un nuevo archivo de clase `PotencialLinker.class` que deberemos cargar en la aplicación que ya se está ejecutando. Además de menor rendimiento en el ámbito de puntos de red que nos interesa, otra pega de esta opción es la necesidad de tener instalado el JDK para poder hacer uso del compilador de Java, `javac`.

Carga dinámica de clases

Todas las clases en Java son cargadas mediante un cargador que ha de ser subclase de `java.lang.ClassLoader`. La carga de clases en tiempo de ejecución tiene que ser realizada por tanto igualmente por una subclase de `java.lang.ClassLoader`. Cuando se carga una clase se cargan también todas las clases a las que hace referencia. Este patrón de carga se aplica recursivamente hasta que se hayan cargado todas las clases necesarias, las

cuales no tienen porque ser todas las clases de la aplicación. Las clases no referenciadas no se cargan hasta el momento en el que son necesarias.

En Java los cargadores de clases están organizados en jerarquía. Cuando se crea un nuevo `ClassLoader` hay que establecer su `ClassLoader` padre. Cuando se pide a un `ClassLoader` que cargue una clase, éste pasará la petición al `ClassLoader` padre. Si el padre no es capaz de encontrar la clase entonces el hijo tratará de cargarla él mismo.

Los pasos ejecutados por un `ClassLoader` al cargar clases son:

- Comprobar si la clase ya ha sido cargada.
- Si no ha sido cargada, pedir al `ClassLoader` padre que cargue la clase.
- Si el cargador padre no puede cargar la clase, intentar cargarla por si mismo.

Para implementar un cargador de clases capaz de recargar clases habrá que desviarse un poco de esta secuencia.

Cargar una clase de forma dinámica no es difícil. Lo único que hace falta es obtener un `ClassLoader` e invocar su método `loadClass()`. He aquí un ejemplo:

```
public class ClaseMain {

   public static void main(String[] args){

      ClassLoader classLoader = ClaseMain.class.getClassLoader();

      try {
         Class MiClase = classLoader.loadClass(``MiClase'');
         System.out.println(``Nombre = '' + MiClase.getName());
      } catch (ClassNotFoundException e) {
         e.printStackTrace();
      }
   }
}
```

Volver a cargar clases previamente cargadas requiere un poco más de esfuerzo. Los cargadores de clases internos de Java siempre comprueban si una clase ya ha sido cargada antes de cargarla. Por tanto no es posible recargar una clase usando los cargadores de clases propios de Java. Para recargar una clase deberemos implementar nuestra propia subclase de `ClassLoader`.

Incluso con nuestro propio cargador de clases tendremos problemas. Cada clase cargada necesita ser enlazada. Esto se hace usando el método `ClassLoader.resolve()`. Este método es final, y por tanto no puede ser

sobreescrito en nuestra implementación del cargador. El método `resolve()` no permite enlazar la misma clase dos veces. Por tanto, cada vez que queramos recargar una clase tendremos que usar una nueva instancia de nuestra subclase de `ClassLoader`. Es necesario tener esto en cuenta a la hora de diseñar nuestra clase para permitir recargarla.

Como hemos dicho, no se puede recargar una clase usando un `ClassLoader` que ya ha cargado esa clase una vez. Por tanto tendremos que usar una instancia distinta de `ClassLoader` cada vez que necesitemos recargar la clase. Pero esto crea nuevos problemas.

Cada clase cargada en una aplicación Java viene identificada por su nombre cualificado y la instancia de `ClassLoader` que la ha cargado. Esto implica que MiClase cargada por el `ClassLoader` A, no es la misma clase que MiClase cargada por el `ClassLoader` B. El pseudocódigo a continuación muestra los problemas provocados por esta diferencia.

```
MiObjeto objeto = (MiObjeto) ClassLoaderA.getInstance();
...el cargador B vuelve a cargar la clase ...
objeto = (MiObjeto) ClassLoaderB.getInstance();
//Surge un ClassCastException
```

Es posible evitar este problema usando como tipo de la variable una superclase, y recargar únicamente la subclase. Para que esto funcione es necesario que nuestra implementación de `ClassLoader` permita que la superclase sea cargada por su `ClassLoader` padre a fin de no incurrir de nuevo en el mismo problema.

Ejemplo

A continuación mostramos la implementación que se ha usado a partir de los conceptos explicados en los apartados anteriores. Una vez que el usuario ha introducido la expresión, se escribe en el punto apropiado del archivo `PotencialLinker.java` y se compila, creando así la clase que deseamos cargar `PotencialLinker.class` en la cual ya está codificada la expresión del potencial.

El programa trabajará con objetos de tipo `Potencial`, nuestro `ClassLoader` se encargará de recargar la clase `PotencialLinker`, la cual es subclase de la clase `Potencial`.

En primer lugar creamos una subclase de `ClassLoader`, de la cual usaremos una instancia nueva cada vez que deseemos recargar la clase `PotencialLinker` para generar un objeto `Potencial` con la expresión inmersa.

```
class MyClassLoader extends ClassLoader{

   public MyClassLoader(ClassLoader parent) {
```

```
      super(parent);
   }

   public Class loadClass(String name) {

   if(name.equals(``PotencialLinker''')){
      File file = new File("bin/PotencialLinker.class");
      ...lee la clase desde el archivo
      return defineClass("PotencialLinker",...);
   }
   else
      return super.loadClass(name);
   }
}
```

A continuación utilizamos este cargador para recargar la clase hija y crear una instancia suya.

```
ClassLoader parentClassLoader = MyClassLoader.class.getClassLoader();
MyClassLoader classLoader = new MyClassLoader(parentClassLoader);
Class clasePotencial = classLoader.loadClass("PotencialLinker");

Potencial potencial = (Potencial) clasePotencial.newInstance();
```

De esta forma hemos conseguido un objeto de la clase Potencial en el cual está inscrito la expresión que introdujo el usuario.

Capítulo 6

Trabajos relacionados

6.1. Finalidad

Las aplicaciones existentes para resolver la ecuación de Schrödinger son aplicaciones escritas en lenguajes orientados al cálculo numérico, tales como Fortran o C. Carecen completamente de interfaz gráfica, donde el potencial va escrito directamente en el código fuente de la aplicación. Estos programas comparten con este proyecto el algoritmo numérico y su finalidad última de hallar las soluciones.

En la mayoría de los casos, incluyendo el departamento de física fundamental de la Universidad de Salamanca, se trata de implementaciones propias del algoritmo de Numerov. La expresión del potencial, así como todos los parámetros físicos y del cálculo numérico, están definidos dentro del código. Esto tiene como ventaja una mayor eficiencia en el cálculo pero acarrea varios problemas:

- La definición de un problema nuevo se hace editando el archivo con el código fuente del problema.
    - Una persona no familiarizada con el código puede tener dificultades para identificar y editar los parámetros que desea.
    - La edición involuntaria de parte del código puede inutilizar el correcto funcionamiento de la aplicación.
- Es más difícil para el usuario ver en cada momento el conjunto de parámetros que se están usando.
- Un usuario que desconozca el lenguaje de programación en el que está escrita la aplicación puede ser incapaz de identificar y definir el potencial según sus necesidades.

Por tanto la reusabilidad y portabilidad de estos programas es prácticamente nula, quedando restringido su uso a la persona que escribió el progra-

ma o requiriendo unos conocimientos y esfuerzos por parte del usuario para poder hacer uso de él.

Dentro de estas restricciones hay que remarcar que, dado que el problema a resolver es un problema muy común en física, cada departamento e incluso cada profesor tiene su propia implementación con la que está familiarizado. Por tanto la cantidad de programas que existen, en su inmensa mayoría escritos por el propio usuario, son innumerables.

A pesar de las numerosas implementaciones propias sigue existiendo un vacío en el ámbito académico. El proyecto *jSchroedinger* aspira a cubrir esta vacante. Es perfectamente capaz de resolver los mismos potenciales que sus homólogos sin interfaz a costa de una ligera pérdida de eficiencia. A cambio incorpora una interfaz que facilita el uso del programa y que ofrece una visualización gráfica de los elementos físicos. Esta visualización gráfica convierte el proyecto en una herramienta docente de gran utilidad en el ámbito académico.

6.2. Orientación

Por los motivos expuestos en el apartado anterior, este proyecto no pretende ofrecer otra implementación más para un problema que ya ha sido resuelto. En su lugar se pretende ofrecer una aplicación centrada en la usabilidad y portabilidad, de forma que pueda ser usada por alumnos de forma didáctica.

En este ámbito los únicos proyectos parecidos que se han encontrado son dos applets que permiten la resolución y visualización de la ecuación de Schrödinger, si bien con bastantes limitaciones.

Se puede ver su funcionamiento en los siguientes sitios:

`http://fermi.la.asu.edu/Schroedinger/html/node2.html`

- Solo permite el cálculo de estados fundamentales.
- No permite sistemas de dos partículas.
- No hace comprobación de estados ligados para el potencial, da resultados para potenciales inválidos como x^3.
- Permite incorrectamente el uso de momento angular para potenciales unidimensionales.

`http://www.benfold.com/sse/shoot.html`

- No se permite elegir qué estado calcular, halla una de las soluciones al azar.

- No permite sistemas de dos partículas.
- No hace comprobación de estados ligados para el potencial, da resultados para potenciales inválidos como x^3.
- Solo permite resolver potenciales unidimensionales.
- No permite establecer los límites de integración, dando lugar a errores en el resultado para potenciales que crecen muy lentamente y para estados altamente excitados.

Capítulo 7

Conclusiones y líneas de trabajo futuras

7.1. Conclusiones

Analizando el trabajo realizado y teniendo en cuenta los objetivos propuestos inicialmente podemos llegar a las siguientes conclusiones:

- Se ha construido una herramienta capaz de resolver la ecuación de Schrödinger con eficiencia, cumpliendo los requisitos y objetivos de precisión establecidos.
- La aplicación se ha dotado de una interfaz gráfica visualmente atractiva e intuitiva de usar. A través de ella el usuario puede tener una representación mental en todo momento de los conceptos físicos involucrados.
- Se ha construido un analizador léxico-sintáctico que permite al usuario introducir expresiones personalizadas en la aplicación.
- Se ha dotado a la aplicación de los medios necesarios para importar y exportar los contenidos generados, permitiendo su uso más allá de la aplicación y guardando los resultados deseados para su uso posterior.
- Se ha construido una aplicación didáctica sencilla de usar, que puede ser utilizada por cualquier alumno o profesor sin necesidad de conocimientos específicos en informática.

En lo relativo a lo personal:

- Se han podido llevar a la práctica todos los conocimientos adquiridos durante la carrera de forma satisfactoria. En particular elementos teóricos, normalmente considerados como menos aplicables, como es el uso de una gramática formal.

- Se ha afrontado un proyecto real de gran envergadura, que ha permitido conocer todos los aspectos que antes sólo eran conocidos en la teoría. En lo referente a los conocimientos, el desarrollo del proyecto ha permitido la integración de los conocimientos adquiridos en distintos campos como pueden ser la ingeniería del software, interfaces gráficas, lenguajes formales o métodos numéricos.

- El resultado final del proyecto deja múltiples aspectos que podrían haber sido mejorados o ampliados, no obstante se considera que los objetivos marcados al inicio, tanto de software como personales, se han superado de manera muy satisfactoria.

7.2. Líneas de trabajo futuras

Las mejoras posibles en cualquier proyecto son sin duda innumerables. Muchas de las ideas nuevas que fueron surgiendo durante la realización del proyecto fueron incluidas en su desarrollo. Muchas otras, debido a su mayor complejidad o a la falta de tiempo, quedaron en el tintero con vistas a posibles futuras versiones de este proyecto.

Listamos a continuación algunas de las posibles líneas de trabajo a seguir para futuras mejoras del proyecto:

- Uno de los aspectos más problemáticos del proyecto fue la correcta visualización de las soluciones de onda y del potencial. Al ser funciones que se extienden hasta el infinito, y que en el caso del potencial pueden valer infinito en ciertos puntos, es muy difícil encontrar los límites de representación apropiados para que la visualización contenga toda la información relevante y a una escala de representación cómoda para el usuario. El algoritmo para establecer los límites funciona de forma correcta para la mayoría de las visualizaciones, pero en algunos casos sería preferible permitir al usuario el manejo personalizado del área de visualización.

- Otro punto que mejoraría notablemente el proyecto sería la implementación de multilenguaje para la interfaz. De esta forma se aprovecharía toda la funcionalidad de la aplicación, permitiendo su uso en otros idiomas, ampliando así la accesibilidad y usabilidad del proyecto.

- El uso y definición de parámetros propios por parte del usuario facilita enormemente la definición de la expresión del potencial y su claridad. Esto podría ampliarse permitiendo al usuario la definición de funciones propias, más allá de las funciones matemáticas elementales utilizadas por la aplicación. Un ejemplo de uso sería permitir la codificación del potencial de Yukawa por parte del usuario, $Y(x) = \frac{e^{-x}}{x}$, utilizado como término en muchos potenciales.

- La representación de potenciales tridimensionales en gráficas bidimensionales es perfectamente posible por el uso de la variable radial como abscisa. Sin embargo, una ampliación en la representación gráfica, permitiendo la visualización tridimensional de potenciales tridimensionales, mejoraría las posibilidades didácticas y visuales del proyecto.

Bibliografía

[1] Durán Toro, A. y Bernárdez Jiménez, B. *Metodología para la Elicitación de Requisitos de Sistemas Software (versión 2.3)*. Universidad de Sevilla. Abril, 2002.

[2] García Peñalvo, Francisco José, Maudes Raedo, Jesús Manuel, Piattini Velthuis, Mario Gerardo, García-Bermejo Giner, José Rafael y Moreno García, María N. *Proyecto de Final de Carrera en la Ingeniería Técnica en Informática: Guía de Realización y Documentación*. Versión 1.52. Marzo, 2000.

[3] Jacobson, I., Booch, G., Rumbaugh, J. *El proceso unificado de desarrollo de software*. Addison Wesley 2000.

[4] Koonin, Steven E., Meredith, Dawn C. *Computational Physics*. Fortran version. Perseus Books 2002.

[5] A. Galindo, P. Pascual, BUSCAR METODO WKB *Quantum Mechanics I*. Ed. Eudema Universidad.

[6] Garcia-Bermejo Giner, Jose Rafael. *Java Se 6 & Swing. El lenguaje más versátil*. Prentice-Hall 2007.

[7] Computer algorithms for solving the Schrödinger and Poisson equations `http://www.teorfys.lu.se/personal/Anders.Blom/useful/scr.pdf`

[8] Beyond Newton's method `http://research.microsoft.com/en-us/um/people/minka/papers/minka-newton.pdf`

[9] API Specification de Java SE 6 `http://java.sun.com/javase/6/docs/api/`

[10] Manuales y Tutorial Sun de Java `http://java.sun.com/docs/books/tutorial/`

[11] Web oficial del programa de elicitación de requisitos REM `http://www.lsi.us.es/descargas/descarga_programas.php?id=3`

[12] Web oficial del IDE Eclipse
`http://www.eclipse.org/`

[13] Plugin SVN para Eclipse
`http://www.eclipse.org/subversive/`

[14] Web de documentación de la herramienta "Html to Latex"
`http://htmltolatex.sourceforge.net/manual/index.html`