UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

U.A.C.P. y P. I.I.M.A.S.

DISEÑO DE UN EDITOR MATEMATICO USANDO EL PROCESO UNIFICADO

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS DE LA COMPUTACION

P R E S E N T A :

ALEJANDRO AGUILAR SIERRA

DIRECTOR: DRA. HANNA OKTABA OKTABA

MEXICO, D. F. JUNIO 1999

Reconocimientos

- A Hanna Oktaba, directora de esta tesis e impulsora de toda una generación de maestros en ciencias de la computación.
- A Manuel Romero, por su dedicada revisión de la tesis.
- A Guadalupe Ibargüengoitia, María Garza y Ernesto Bribiesca, por aceptar ser sinodales del examen de grado.
- A mi ex compañero de la maestría, Alfonso Martínez, por revisar una versión preliminar de la tesis.
- Muy especialmente a la comunidad mundial de programadores y usuarios de *software* libre. A ellos debo y dedico este trabajo.

Alejandro Aguilar Sierra
México DF, junio de 1999

Índice General

1 Introducción 1
- 1.1 Antecedentes . . . 1
- 1.2 Objetivos . . . 2
- 1.3 Organización de la tesis . . . 2

2 Estado del arte de los editores de ecuaciones 3
- 2.1 Características . . . 3
- 2.2 Requisitos . . . 4
- 2.3 Lenguajes de comunicación matemática . . . 4
- 2.4 Editores de ecuaciones actuales . . . 6
  - 2.4.1 MathType . . . 6
  - 2.4.2 ScientificWord . . . 6
  - 2.4.3 Publicon . . . 7
  - 2.4.4 Amaya . . . 7
  - 2.4.5 WebEQ . . . 8
- 2.5 Resultados . . . 8
- 2.6 Conclusión . . . 9

3 Metodología 11
- 3.1 Características clave del Proceso Unificado . . . 11
- 3.2 Ciclo de vida del Proceso Unificado . . . 12
- 3.3 Herramientas . . . 13
  - 3.3.1 UML . . . 14
  - 3.3.2 Patrones de diseño . . . 14
  - 3.3.3 Programación genérica . . . 14

4 Captura de requerimientos 17
- 4.1 Requerimientos . . . 17
- 4.2 Modelo del dominio . . . 18
  - 4.2.1 Objetos del dominio y sus atributos . . . 19

4.3 Casos de Uso 20
4.3.1 Identificación de actores 20
4.3.2 Identificación de casos de uso 21
4.4 Requerimientos suplementarios 22
4.5 Interfase con el usuario 23

5 Análisis **25**
5.1 Identificación de clases 25
5.1.1 Clases frontera 26
5.1.2 Entidades 26
5.1.3 Clases control 27
5.1.4 Diagrama de clases 27
5.2 Realización de casos de uso 28
5.2.1 Caso Crear 29
5.2.2 Caso Navegar 29
5.2.3 Caso Editar 30
5.2.4 Caso Leer 33
5.2.5 Caso Escribir 34
5.3 Paquetes 34
5.3.1 Identificación de paquetes 35
5.3.2 Definición de dependencias entre paquetes 35

6 Diseño **37**
6.1 Identificación de subsistemas y sus interfases 37
6.1.1 Identificación de subsistemas 37
6.1.2 Interfases 38
6.2 Identificación de clases 38
6.2.1 Interfase gráfica 38
6.2.2 Clases relativas al Almacenamiento Interno 40
6.2.3 Clases relativas al subsistema edición 41
6.2.4 Subsistema de lectura 42
6.2.5 Subsistema de escritura 43
6.3 Diseño de los casos de uso 43
6.3.1 Crear 43
6.3.2 Insertar elemento 44
6.3.3 Navegar 44
6.3.4 Seleccionar 45
6.3.5 Pegar 47
6.3.6 Leer 48
6.3.7 Escribir 48

6.4 Diagramas de Clases 49
6.5 Patrones de diseño utilizados 53

7 Resultados **55**
7.1 Implementación 55
7.2 Interfase gráfica 56
7.3 Integración a un sistema 56

8 Conclusión y perspectivas **61**

A Glosario **63**

B Símbolos matemáticos comunes **65**

C Licencia Pública General **69**

Índice de Figuras

3.1 Relación entre etapas y fases del ciclo de vida (tomada de [JBR99]). . . 13

4.1 Propiedades métricas de un caracter 19
4.2 Casos de uso 21

5.1 Clases del análisis y sus relaciones. 28
5.2 Caso de uso Crear. 28
5.3 Caso Navegar. 29
5.4 Caso Insertar o Borrar. 30
5.5 Caso Seleccionar. 31
5.6 Caso Pegar. 32
5.7 Caso Leer. 33
5.8 Caso Escribir. 34
5.9 Capas y dependencias de los paquetes. 36

6.1 Subsistemas: sus dependencias e interfaces. 39
6.2 Caso Crear 43
6.3 Caso Insertar elemento 44
6.4 Caso Navegar. 45
6.5 Caso Seleccionar 46
6.6 Caso Pegar 47
6.7 Caso Leer 48
6.8 Caso Escribir 49
6.9 Estructura interna y clases de acceso. 50
6.10 Administración de inserciones. 51
6.11 Arquitectura del sistema. 52

7.1 Interfase gráfica en LyX. 57
7.2 Interfase gráfica experimental en GTK+. 58
7.3 Pantalla de LyX 59
7.4 Versión impresa del mismo documento. 60

Capítulo 1

Introducción

Un editor de ecuaciones es una herramienta interactiva que permite al usuario editar expresiones matemáticas por medio de una interfase gráfica. Permite realizar las mismas operaciones básicas de edición que un editor de texto pero, debido a la estructura compleja que puede tener una expresión matemática, su diseño implica algunos problemas que no se encuentran en el diseño de un editor de texto convencional.

La aplicación más común de un editor de ecuaciones es la inserción de fórmulas en documentos técnicos. También puede utilizarse como interfaz para sistemas de cálculo numérico, de matemática simbólica, de graficación, etc.

1.1 Antecedentes

A mediados de los años 90, el sistema operativo Linux[1] empezó a cobrar amplia popularidad como un sistema poderoso y estable. Sin embargo, no se tenían algunas de las aplicaciones típicas de una computadora personal, tales como un editor de palabras que permitiera escribir documentos técnicos y editar ecuaciones amigablemente. A finales de 1995 salió a la luz en Internet el procesador de documentos LyX[2], desarrollado por el estudiante alemán Matthias Ettrich, bajo los principios del software libre [Ray97]. Al conocer la existencia de dicho proyecto, el autor de esta tesis decidió desarrollar e integrar un editor de ecuaciones, al que llamó Mathed. La primera versión, empírica, de Mathed fue integrada a LyX a principios de 1996. Desde entonces forma parte de ese programa y ha tenido una gran aceptación por parte de cientos de usuarios en todo el mundo; lo cual ha brindado la oportunidad de recibir una considerable e invaluable retroalimentación.

[1] Sistema operativo compatible con Unix, de alta calidad y de distribución libre, que inicialmente se usaba solo en plataformas intel 386.

[2] `http://www.lyx.org`

1.2 Objetivos

A pesar de que actualmente existen varios editores de ecuaciones, ninguno de ellos satisface todos los requisitos siguientes: que corra bajo plataformas Linux (o Unix en general), que tenga *disponible el código fuente* y que pueda *usarse desde diferentes* aplicaciones.

La versión empírica de Mathed ha sufrido varias correcciones y adiciones durante los últimos 3 años, exponiendo debilidades y necesidades que no fueron consideradas en el diseño original. Al seguir desde el principio una metodología, avalada por varias décadas de experiencia en ingeniería de software orientada a objetos [JBR99], se podrán anticipar esas debilidades y en general se obtendrá un producto de mayor calidad.

Los objetivos principales de este trabajo son:

1. Diseñar e implementar un editor de ecuaciones.
2. Aplicar los métodos más robustos de la ingeniería de software orientada a objetos: El Proceso Unificado y los Patrones de Diseño (ver capítulo 3).

Los beneficiarios inmediatos serán los usuarios del procesador de documentos LyX, pero se espera expandir la base de usuarios, por medio de la integración del editor a programas tales como hojas de cálculo, aplicaciones de geometría y matemática simbólica, etc.

1.3 Organización de la tesis

En el capítulo 2 se da un panorama general del estado del arte de los editores matemáticos actuales y sus necesidades. Se definen las características que distinguen a una expresión matemática. Incluye una breve descripción de algunos lenguajes matemáticos que permiten el almacenamiento e intercambio de expresiones matemáticas a través de medios electrónicos.

En el capítulo 3 se describe la metodología a seguir en el desarrollo de este trabajo. Los capítulos del 4 al 7 corresponden a la aplicación de dicha metodología orientada a objetos. Finalmente, en el capítulo 8 se dan las conclusiones sobre los aspectos estudiados, las perspectivas así como algunos tópicos aún objeto de investigación.

En los apéndices se incluyen un glosario con la terminología usada en este trabajo, tablas de los símbolos matemáticos más comunes y una traducción al español de la General Public License, que define los principios que rigen al software libre.

Capítulo 2

Estado del arte de los editores de ecuaciones

En este capítulo se listan los requisitos básicos que debe cumplir un editor de ecuaciones. Con base en esta lista se evalúan algunos de los editores de ecuaciones actuales. Además de editar una fórmula, debe ser posible insertarla en un documento técnico o transmitirla por medios electrónicos. Por ese motivo se hace una revisión de los lenguajes actuales para la comunicación de expresiones matemáticas.

2.1 Características

Una expresión matemática se caracteriza por contener elementos que no pueden introducirse por medio de un editor de texto plano. Estos elementos son:

1. *Símbolos no convencionales:* caracteres griegos (α, β), operadores binarios ($\pm, \uplus$), relaciones ($\leq, \simeq$), flechas ($\rightarrow$, $\Rightarrow$), operadores mayores ($\sum$, $\int$), etc.

2. Estructuras compuestas o ajustables: la raíz cuadrada $\surd$ contiene una expresión matemática en su interior y deben ajustarse al tamaño de la misma. Casos similares son la fracción $\frac{1-x}{1+x}$, los paréntesis $\left(\frac{1+\sqrt{5}}{1-\sqrt{5}}\right)$, etc.

3. Subíndices x_2 y superíndices o exponenciales e^x.

4. Arreglos bidimensionales de expresiones matemáticas: $\begin{matrix} 1 & 4x^2 \\ 0 & 1 \end{matrix}$.

2.2 Requisitos

Para evaluar objetivamente los editores de ecuaciones actuales, con base en la experiencia adquirida en el desarrollo y mantenimiento del Mathed empírico, se listan a continuación los requisitos básicos que deben satisfacer:

1. Debe ser capaz de realizar todas las funciones básicas de un editor, tales como insertar, borrar, copiar, cortar y pegar.

2. Debe ser suficientemente intuitivo para que un novato con alguna experiencia en procesadores de palabras pueda usarlo en el menor tiempo posible.

3. Debe ser posible introducir la mayoría de los símbolos usados en matemáticas (apéndice B).

4. Debe soportar estructuras como raíces, fracciones, matrices, índices y exponentes.

5. Debe ser posible insertar los símbolos y estructuras, tanto por medio del ratón como por medio del teclado, para que los usuarios experimentados puedan editar sus ecuaciones con mayor rapidez.

6. Debe ser posible exportar las fórmulas para que puedan usarse en otro sistema, o que puedan intercambiarse y transmitirse por correo electrónico. Para ello conviene que se utilice un lenguaje de comunicación matemática estándar.

7. Debe poder integrarse a otros sistemas, tales como un editor de palabras de uso general, o un programa de graficación científica.

8. El código fuente del editor debe estar disponible públicamente bajo alguna licencia de código abierto (ver apéndice C y `http://www.opensource.org`).

En el punto 6 se exige que el editor sea capaz de exportar e importar fórmulas por medio de un lenguaje matemático estándar. Antes de pasar a evaluar los editores de ecuaciones actuales, revisaremos el estado del arte en lenguajes para comunicar matemáticas.

2.3 Lenguajes de comunicación matemática

Desde que las computadoras se convirtieron en una herramienta práctica para el almacenamiento e intercambio de información, se han desarrollado varios lenguajes para comunicar expresiones matemáticas.

Estos lenguajes se dividen en lenguajes de *presentación* y lenguajes de *contenido*. Como su nombre lo indica, los lenguajes de presentación comunican la representación

de una expresión, mientras que los lenguajes de contenido comunican su significado. Para ilustrar la diferencia entre ambos, usaremos como ejemplo una conversación telefónica entre dos estudiantes. Uno de ellos quiere comunicar al otro la expresión $a + b$. En un lenguaje de presentación, el estudiante dirá: "escribe los símbolos a, $+$ y b consecutivamente". En un lenguaje de contenido, diría: "escribe la suma de a y b". Los primeros lenguajes matemáticos se aproximaban más a la categoría de lenguajes de presentación y solamente recientemente se han introducido lenguajes de contenido.

TEX No es un lenguaje matemático propiamente dicho, sino un lenguaje de tipografía electrónica. Fue desarrollado por el matemático Donald Knuth [Knu94], de la Universidad de Stanford, con el propósito de facilitar la impresión de alta calidad de documentos con expresiones matemáticas. Es actualmente uno de los sistemas de tipografía electrónica de mayor prestigio mundial, al menos en el ambiente académico.

LATEX Es un conjunto de macros de TEX especializado en documentos técnicos [Lam94]. Se caracteriza por tener un conjunto de comandos que describen la manera en que debe ser formado un documento. Se considera un lenguaje tipo *markup* debido a que contiene comandos que indican los componentes lógicos de la estructura de un documento. En modo matemático incluye expresiones de alto nivel, lo que lo convierte en un antecesor de un lenguaje matemático de tipo presentación. Actualmente se utiliza en el ambiente académico y técnico, en casi todas las áreas del conocimiento.

MathML Es un lenguaje *markup* específicamente creado para codificar expresiones matemáticas y poder incluirlas en páginas Web[1]. Considera en su especificación los dos tipos de lenguaje: presentación y contenido. Es un estándar internacional, que forma parte del estándar XML (*Extensible Markup Language*) propuesto por el W^{3}C (*World Wide Web Consortium*) como sucesor del formato HTML (*Hypertext Markup Language*).

OpenMath Es un estándar para la comunicación de representaciones semánticamente ricas, de objetos matemáticos entre todo tipo de programas de computadora[2](lo cual implica un lenguaje de contenido). Utiliza diccionarios para asignar un significado específico a objetos y símbolos usados en determinadas áreas de las matemáticas. De esa manera las distintas aplicaciones pueden reconocer la manera de transformar dichos objetos a su representación interna.
No profundizaremos en este tema debido a que no se espera que un editor de ecuaciones utilice el significado de las expresiones; solamente nos interesa la presentación gráfica de las mismas.

Los lenguajes más aceptados en la actualidad como estándares para transmitir ecuaciones por medios electrónicos son LaTeX y MathML.

2.4 Editores de ecuaciones actuales

En esta sección se analizan y evalúan algunos de los editores de ecuaciones más usados en la actualidad. El criterio de evaluación consiste en verificar que cumplan satisfactoriamente con los puntos listados en la sección 2.2.

2.4.1 MathType[3]

Diseñado por la compañia Design Science, es el editor de ecuaciones usado por los procesadores de palabras más populares bajo Windows de Microsoft. Se usa integrado a otros sistemas, como el procesador de palabras Word, por lo que confirmamos que cumple con el punto 7. La única manera de introducir símbolos es por medio del ratón, además de que solo tiene un número limitado de símbolos matemáticos. Es muy difícil armar expresiones muy complejas debido a que no tiene el concepto de anidación de expresiones (como operadores dentro de otros operadores). Todo esto lo hace tedioso y definitivamente limitado para un usuario que necesita escribir matemáticas frecuentemente. Sin embargo es sin lugar a dudas el editor de ecuaciones más usado en plataformas Windows.

Design Science ha anunciado que la próxima versión del MathType ha sido rediseñada para traducir su representación interna a varios lenguajes, incluyendo LaTeX y MathML. El proceso de traducción será controlado por un archivo de definición, por lo que se podrán crear traductores para cualquier lenguaje de notación matemática, según sus anunciantes.

2.4.2 ScientificWord[4]

El Scientific Workplace de TCISoft incorpora un editor de ecuaciones excelente, intuitivo y bastante poderoso. Los menús de símbolos están divididos y clasificados estratégicamente para que sea posible seleccionar los símbolos de manera lógica y ágil. Por ejemplo, hay un menú para símbolos griegos, otro para operadores binarios y otro para relaciones binarias. Además es posible introducir todos los símbolos y operadores por medio de secuencias de teclas, lo que lo hace más rápido para usuarios experimentados (punto 5). Además, las fórmulas escritas con este editor pueden usarse para

[1] http://www.w3.org/TR/REC-MathML
[2] http://www.openmath.org
[3] http://www.mathtype.com/
[4] http://www.tcisoft.com

realizar cálculos o gráficas con el sistema de algebra por computadora Maple sin salirse del editor.

Este editor cumple satisfactoriamente con casi todos los puntos listados en la sección anterior. De hecho, se tomó como modelo para el desarrollo del Mathed original. Sin embargo no cumple con el punto 8 y no existe una implementación que corra bajo Linux.

2.4.3 Publicon[5]

Difícilmente podríamos soslayar un producto de la compañía Wolfram, los creadores de Mathematica, el famoso sistema de matemática simbólica. Publicon es un procesador de documentos técnicos que incluye un editor matemático. Tiene la misma apariencia y funcionalidad que Mathematica para manipular expresiones, pero no realiza cálculos con ellas, sólo permite editarlas. Se pueden introducir símbolos tanto por medio del teclado como con el ratón. Es capaz de exportar documentos en distintos formatos. Su editor matemático no se puede separar para usarse en otras aplicaciones.

2.4.4 Amaya[6]

Amaya es a la vez un navegador de la Web y una herramienta de edición de documentos. Se diseñó originalmente con el propósito de probar y demostrar nuevas tecnologías para la Web en un ambiente WYSIWYG (*What You See Is What You Get*). La versión actual soporta las características más avanzadas de la especificación HTML 4.0. Adicionalmente permite crear y editar expresiones matemáticas complejas dentro de páginas Web y codificarlas en MathML.

Aunque el código fuente está disponible al público, para hacer esta evaluación utilizamos una versión precompilada para plataformas Linux. Es posible entrar a modo matemático de dos maneras: por medio de una entrada del menú de inserción y oprimiendo un botón de la barra de herramientas.

Al soportar la especificación MathML se da por hecho que soporta la mayoría de los elementos matemáticos como símbolos, estructuras, etc. El despliegue en pantalla está limitado a las fuentes disponibles en el sistema, no incluye fuentes de símbolos matemáticos. Otra limitación importante es que únicamente genera código MathML insertado en documentos HTML por lo que no se puede considerar como un programa de edición matemática de propósito general.

[5] `http://www.wolfram.com`
[6] `http://www.w3.org/Amaya`

2.4.5 WebEQ[7]

Este no es propiamente un editor sino un conjunto de herramientas para desplegar y manipular expresiones matemáticas en navegadores de la WWW que soporten Java y Javascript. Lo interesante de este sistema es que es uno de los primeros intentos de visualizar expresiones matemáticas escritas en MathML en el Web sin recurrir a imágenes.

Contiene un editor sencillo escrito en Java, por lo que puede ejecutarse en prácticamente cualquier plataforma que soporte este lenguaje. Para introducir símbolos se usa una tabla que contiene cuatro secciones:

1. Símbolos griegos.
2. Operadores, relaciones binarias y símbolos especiales como infinito ∞ y h barra $\hbar$.
3. Delimitadores.
4. Flechas en todas direcciones así como puntos suspensivos.

El código fuente no está disponible ni puede integrarse libremente a otras aplicaciones.

2.5 Resultados

La mayoría de estos programas se ejecutan en la plataforma Windows de Microsoft y no son de distribución libre. No se cuenta con el código fuente ni con información técnica de su diseño. En otras palabras no cumplen con el punto 8. En caso de qué soporten el lenguaje MathML, solo soportan la parte de presentación (ver sección 2.3).

El siguiente cuadro sinóptico muestra una comparación entre los programas evaluados:

	Edición	**Símbolos**	**Estruct.**	**Exporta**	**Integ.**	**Licencia**
MathType	s	s	s	-	s	p
ScientificWord	e	e	e	s	-	p
Publicon	s	e	e	s	-	p
Amaya	e	s	e	s	-	W^3
WebEQ	s	s	s	s	s	p

Evaluación: e = Excelente, s = Mínimo esperado, - = insatisfactorio.
Licencia: p = propietaria, W^3 = licencia del consorcio W^3.

[7] `http://www.webeq.com`

A continuación se explican las características destacadas en el cuadro y los criterios de calificación.

Edición Todos tienen las funciones mínimas de un editor (capacidad de insertar, borrar, copiar, cortar y pegar.) Solo alcanzan la calificación de excelente si cumplen con los puntos 1, 2 y 5.

Símbolos Conjunto de símbolos matemáticos (punto 3). Exhaustivo (e) o que al menos tenga los caracteres griegos y los símbolos más comunes (s).

Estructuras Conjunto de estructuras como raíces y fracciones (punto 4). Al igual que con los símbolos, la excelencia se alcanza si soportan más del conjunto mínimo esperado de estructuras.

Exporta Capacidad de exportar a algún lenguage matemático estándar (punto 6). Para alcanzar la calificación de excelente debe soportar ambos lenguajes LaTeX y MathML. Si solo soporta uno de ellos se calificará como mínimo esperado.

Integración Integración en otros sistemas (punto 7). Ninguno alcanza la calificación de excelente en este campo debido a que todavía no existe un estándar universal para este tipo de integración. Bajo el sistema Windows existe el protocolo OLE2[8], que es el que usa Mathtype. Bajo Unix, existen actualmente algunos proyectos que involucran una implementación del protocolo CORBA[9], como el proyecto GNOME[10].

Licencia Tipo de licencia (punto 8). Todos los programas evaluados tienen una licencia propietaria, con excepcion de Amaya. La licencia del Consorcio W3(organismo dedicado a regular los protocolos de la WWW) permite obtener el código fuente.

2.6 Conclusión

En esta evaluación destacó el editor de ecuaciones de ScientificWord, pues resultó excelente en casi todos los puntos, con excepción de la licencia y la incapacidad de ser integrado a sistemas externos. El editor de ecuaciones de Amaya es el único cuyo código fuente está disponible públicamente, pero solo contiene lo mínimo esperado en la mayoría de los puntos y no esta diseñado para integrarse a otros sistemas.

Se concluye que ninguno de los editores evaluados cumple satisfactoriamente con todos los requisitos listados en la sección 2.2.

[8] http://www.microsoft.com
[9] http://www.omg.org
[10] http://www.gnome.org

Capítulo 3

Metodología

El desarrollo de este trabajo se rigió por el *Proceso de Desarrollo de Software Unificado* [JBR99], o *PU* (*Proceso Unificado*), para abreviar. Este proceso es resultado de la unificación de varias metodologías de ingeniería de software orientado a objetos desarrolladas durante los últimos 20 años, realizada principalmente por los autores de las 3 metodologías más exitosas de la última década: Grady Booch [Boo94], James Rumbaugh [R+91] e Ivar Jacobson [J+92].

3.1 Características clave del Proceso Unificado

El Proceso Unificado es multifacético, toma en consideración ciclos, fases, mitigación de riesgos, control de calidad, administración de proyecto y control de configuración. El Proceso Unificado se basa en componentes.

Las tres características clave del PU son: 1) se conduce por los casos de uso, 2) se centra en la arquitectura y 3) es iterativo e incremental.

Todo sistema de software es creado para servir a sus usuarios, sean estos humanos u otros sistemas. Un *usuario* es alguien o algo que interacciona con el sistema. Cada interacción es un *caso de uso*. Todos los casos de uso reunidos conforman el modelo de casos de uso, que describe la funcionalidad completa del sistema. Este modelo reemplaza a la especificación funcional tradicional y responde a la pregunta: ¿qué es lo que el sistema debe hacer para cada usuario? Sin embargo, los casos de uso no solo ayudan a especificar los requerimientos el sistema, también conducen el proceso de desarrollo. Todas las demás etapas del proceso se hacen con el fin de realizar los casos de uso.

Así como el modelo de casos de uso describe la funcionalidad del sistema, la arquitectura define su *forma*. Es una visión del diseño del sistema en conjunto, enfatizando las características más importantes y desdeñando los detalles. Da la estructura en la

cual se guiará el desarrollo del sistema. La arquitectura se construye para implementar los casos de uso y a su vez influye en la selección de los casos de uso.

En cada iteración se identifican y especifican los casos de uso relevantes, se crea un diseño usando la arquitectura escogida como guía, se implementa el diseño en componentes y se verifica que los componentes satisfacen los casos de uso. Si es el caso se pasa a la siguiente iteración, resultando en un incremento del producto final.

Un proceso iterativo controlado tiene varias ventajas: reduce el esfuerzo de desarrollo, ya que los equipos de desarrollo trabajan mejor obteniendo resultados a corto plazo. Al identificarse los riesgos en las etapas inicialès del desarrollo, se reduce el riesgo de retardar el desarrollo del producto, como ocurre cuando se hace hasta la etapa de pruebas. Por último, permite refinar los requerimientos en iteraciones sucesivas y se adapta fácilmente a requerimientos cambiantes.

3.2 Ciclo de vida del Proceso Unificado

El Proceso Unificado se repite en una serie de ciclos que conforman la vida de un sistema. Cada ciclo concluye con una liberación del producto al cliente y consiste en 4 fases[1]:

Incepción En esta fase se fundamenta la idea del desarrollo del sistema.

Elaboración Se define la arquitectura que servirá de guía, no solo en el presente ciclo sino como un estándar para ser seguido en los ciclos futuros.

Construcción Se implementa y se prueba el sistema al punto de que está listo para ponerse en manos del cliente.

Transición El sistema es puesto en manos de los usuarios.

Dentro de cada fase pueden ocurrir una o varias iteraciones. Cada iteración incluye las siguientes etapas, llamadas *core workflows*, las cuales no necesariamente se realizan secuencialmente:

Requerimientos Se realiza la captura de requerimientos de manera que se llega a un acuerdo con el cliente respecto a qué debe y qué no debe hacer el sistema.

Análisis Se analizan y refinan los requerimientos y se define el dominio del problema. Su enfoque es el *qué hacer*.

[1] *Inception* viene del latín inceptus y significa iniciar algo. Usaremos el barbarismo *incepción* para denotar la primera fase, especifica del Proceso Unificado.

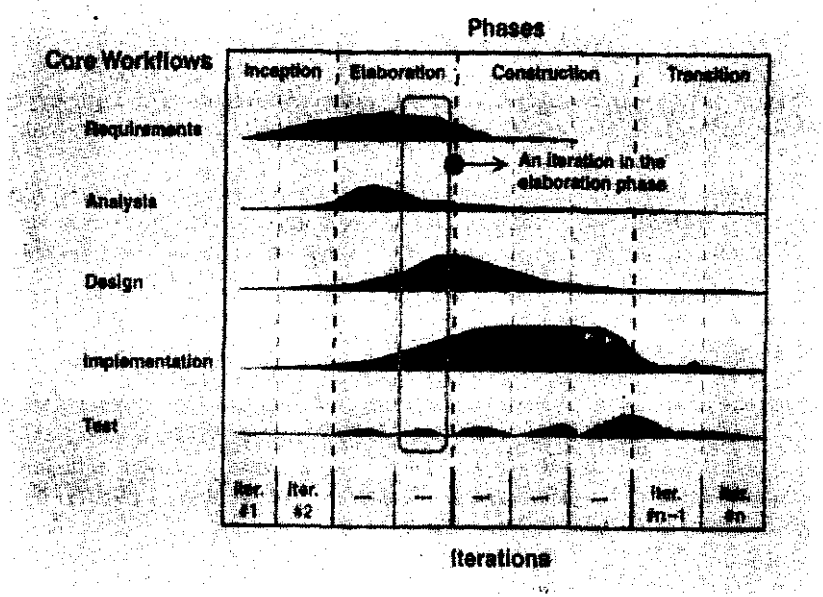

Figura 3.1: Relación entre etapas y fases del ciclo de vida (tomada de [JBR99]).

Diseño Se enfoca en el *cómo hacerlo*. En esta etapa se toman decisiones estratégicas para cumplir con los requerimientos (casos de uso), a nivel lógico.

Implementación Su propósito esencial es implementar el sistema en componentes, código fuente, binarios, ejecutables, etc.

Pruebas Se verifica el resultado de la implementación, probando cada versión ejecutable del sistema.

En la figura 3.1 se muestra la relación de estas etapas con las fases del ciclo de vida del proceso.

3.3 Herramientas

Existen otras herramientas conceptuales que complementan la metodología usada durante algunas de las diversas etapas del desarrollo del Proceso Unificado. Entre ellas están los Patrones de Diseño (etapa de diseño) y la Programación Genérica (etapa de implementación).

3.3.1 UML

La herramienta de modelación visual usada por el PU es el UML (*Unified Modeling Language*), elaborado por los mismos autores. El propósito del UML es visualizar, especificar, contruir y documentar sistemas orientados a objetos. Los autores del PU también son autores de una guía del usuario sobre el UML [BRJ99], y el manual de referencia del UML [RJB99].

El UML permite modelar a distintos niveles de abstracción, tanto desde el punto de vista estructural como de comportamiento. Su lenguaje incluye diversos tipos de objetos, relaciones y diagramas que se usarán en todas las etapas del desarrollo de este trabajo.

3.3.2 Patrones de diseño

Los patrones de diseño se originan de la teoría arquitectónica de Christopher Alexander [AIS77]. Cada patrón describe la solución a un problema que ocurre una y otra vez, por lo que dicha solución puede ser usada millones de veces. Aunque esta teoría se refiere a la arquitectura, se aplica igual al diseño orientado a objetos [GHJV94].

En general un patrón tiene 4 elementos:

1. El nombre del patrón. Este nombre pasa a formar parte del vocabulario de diseño, lo que permite diseñar a un nivel mayor de abstracción.

2. El problema describe cuando aplicar el patrón. A veces incluye una lista de condiciones que deben cumplirse para que tenga sentido aplicar el patrón.

3. La solución describe los elementos que dan forma al diseño, sus relaciones, responsabilidades y colaboraciones.

4. Las consecuencias describen el resultado y los riesgos de aplicar el patrón. Son importantes para evaluar las alternativas de diseño y para entender los costos y beneficios de aplicar el patrón.

Los patrones de diseño identifican las clases e instancias, sus roles y colaboraciones y la distribución de responsabilidades para la solución de algún problema específico de diseño orientado a objetos.

3.3.3 Programación genérica

Con una filosofía similar a la de los patrones de diseño, el objetivo de la programación genérica es escribir una vez los mejores algoritmos para resolver un problema, de manera que puedan ser reusados en cualquier aplicación específica. En C++, el

lenguaje elegido para implementar este proyecto, la programación genérica se realiza por medio de la STL (Standard Template Library) [Aus98]. Esta es una biblioteca genérica, sus componentes están parametrizados; casi todos sus componentes son un patrón (*template*) en C++.

La STL se compone principalmente de:

Contenedores El propósito de estos componentes es el de contener otros objetos. La STL incluye arreglos, listas ligadas, mapas y muchos otros. Todo el manejo de memoria se hace en forma transparente para el usuario, no es necesario apartar memoria manualmente.

Algoritmos Estos algoritmos pueden usarse para manipular datos contenidos en los contenedores. Son funciones globales y actúan en un rango de elementos del contenedor. Son resultado de varias décadas de investigación, por lo que se consideran algoritmos muy solidos y eficientes.

Iteradores Los argumentos de los algoritmos son iteradores, los cuales pueden accesar cualquier elemento del contenedor en cualquier orden. Son el mecanismo mediante el cual se puede desacoplar los algoritmos de los contenedores, de modo que los algoritmos resultan funciones genéricas.

Todos estos elementos han sido construidos con la experiencia de varios años para asegurar la mayor calidad, y actualmente se encuentran en la mayoría de los compiladores de C++ modernos, ya que es un estándar del lenguaje.

Capítulo 4

Captura de requerimientos

Una lista de requisitos básicos para un editor de ecuaciones ya ha sido dada en la sección 2.2. En esta sección se extenderá y formalizará dicha lista en la forma del Modelo de Casos de Uso [JBR99].

4.1 Requerimientos

El Proceso Unificado recomienda empezar esta etapa con una lista de candidatos a requerimientos. La lista de requisitos del capítulo 2 ya contiene los requisitos más importantes. En esta sección se amplía y detalla dicha lista.

1. Debe ser capaz de realizar todas las funciones básicas de un editor, tales como insertar, borrar, copiar, cortar y pegar.

2. Debe ser suficientemente intuitivo para que un novato con alguna experiencia en procesadores de palabras pueda usarlo en el menor tiempo posible. Para conseguir este propósito, los mecanismos de edición deben corresponder, en lo posible, a los mecanismos de edición usados comúnmente en modo texto. Secuencias similares de teclazos, etc.

3. Debe ser posible introducir la mayoría de los símbolos usados comúnmente en matemáticas. Debe soportar por lo menos el conjunto de símbolos matemáticos de LaTeX (ver apéndice B y [Lam94] sección 3.3.2):

   (a) Símbolos griegos

   (b) Operadores binarios

   (c) Relaciones binarias

   (d) Flechas

(e) Operadores grandes: integral, sumatoria, etc.

(f) Miscelánea

4. Debe soportar como mínimo las siguientes estructuras:

   (a) Raíces y raíces cuadradas en particular.

   (b) Fracciones y coeficientes binomiales.

   (c) Subíndices y superíndices (exponentes).

   (d) Espacios variables (ver [Lam94] sección 3.3.7).

   (e) Matrices.

   (f) Delimitadores (ver [Lam94] sección 3.3.4).

   (g) Decoraciones y acentos matemáticos (ver [Lam94] sección 3.3.6).

5. Debe ser posible insertar los símbolos y estructuras, tanto por medio del ratón como por medio del teclado, para que los usuarios experimentados puedan editar sus ecuaciones con mayor rapidez. Esta es una condición para el diseño de la interfase con el usuario.

6. Debe ser posible exportar las fórmulas para que puedan usarse por algún sistema externo, o simplemente que el usuario pueda intercambiarlas con otros colegas por medio del correo electrónico; para lo cual conviene que se utilice un lenguaje de comunicación matemática estándar. Debe leer y escribir por lo menos LaTeX.

7. Debe poder integrarse a otros sistemas, tales como un editor de palabras de uso general, o un programa de graficación científica.

8. Por último, el código fuente del editor debe estar disponible públicamente bajo alguna licencia de código abierto (`http://www.opensource.org`).

4.2 Modelo del dominio

El propósito del modelado del dominio es contribuir al entendimiento del problema que el sistema debe resolver y el contexto en el que se desenvolverá el sistema. Se trata de entender y describir los objetos más importantes dentro del contexto del dominio. A veces no es necesario desarrollar un modelo de objetos; basta con un glosario de términos. En el apéndice A se ha incluído un glosario con el lenguaje del dominio del problema.

Antes de implementar un sistema enteramente nuevo, es prudente estudiar los sistemas existentes. De este modo se puede sacar provecho de la experiencia de otros

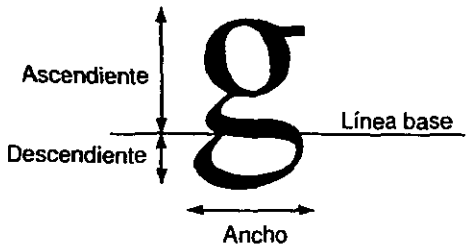

Figura 4.1: Propiedades métricas de un caracter

proyectos que tuvieron que tomar decisiones similares. Esto es lo que se llama *análisis vertical del dominio* ([Boo94], capítulo 4). En el capítulo 2 se hizo una evaluación de los editores de ecuaciones actuales. Así se identificaron los siguientes objetos.

4.2.1 Objetos del dominio y sus atributos

Dentro del contexto de un editor de ecuaciones, lo primero es definir una ecuación. En la sección 2.1 se definieron las diferencias entre un texto y una ecuación.

Los elementos más importantes que verá un usuario al momento de operar con él son:

Caracter

Cualquier símbolo, incluyendo caracteres griegos, latinos, números, operadores, etc.

Un caracter ocupa una posición y un espacio que conviene conocer, ya sea para imprimir el caracter siguiente, para calcular la posición del cursor y para colocar en el lugar correcto un índice o un exponente.

En una ecuación se consideran dos líneas sobre las cuales se alinearán todos los símbolos y caracteres: una *línea base* (la línea sobre la que se escribiran todos los caracteres) y un *eje matemático* (la altura a la que se coloca el signo menos y otros símbolos de operadores).

Las principales propiedades métricas de un caracter son:

Ascendiente Altura sobre la línea base.

Descendiente Altura bajo la línea base.

Ancho Extensión horizontal del objeto.

El significado de estas dimensiones se ilustra claramente en la figura 4.1.

Estructura

Raíz cuadrada, fracción, etc. En general todo elemento que contiene uno o más *párrafos* (ver definición siguiente) y que adapta su tamaño al tamaño de su contenido. *Los atributos específicos de la estructura* son un número definido de párrafos. Sus propiedades métricas son las mismas que las propiedades métricas de un caracter (ver sección 4.2.1).

Una estructura es equivalente a las cajas manejadas por TEX ([Knu94] y [Lam94]), las cuales son manejadas como si se tratara de un caracter, pero pueden contener párrafos enteros.

Párrafo

En este contexto entenderemos por párrafo un arreglo secuencial que contiene caracteres y estructuras. En la mayoría de los casos, un párrafo no contiene más de una línea. Pero es posible que en algunos casos contenga varias líneas e inclusive varias columnas. Es básicamente un contenedor de elementos.

Ventana

Interfaz gráfica con el usuario. Sus responsabilidades son presentar una imagen gráfica de una ecuación y permitir al usuario modificarla por medio de eventos de interacción por medio de dispositivos externos, tales como el teclado y el ratón.

Cursor

Da al usuario la información visual de la posición donde se efectuará cualquier cambio de un paso que realice. No forma parte del párrafo pero tiene acceso a cualquier parte del mismo y permite al usuario hacer las operaciones de edición básicas en cualquier posición dentro del párrafo.

La única propiedad del cursor que nos interesa por el momento es la posición del mismo dentro del párrafo.

4.3 Casos de Uso

4.3.1 Identificación de actores

El principal agente externo al sistema es el usuario. Independientemente del hecho de que el sistema tenga una interfase con el usuario propia o que esté integrado a un

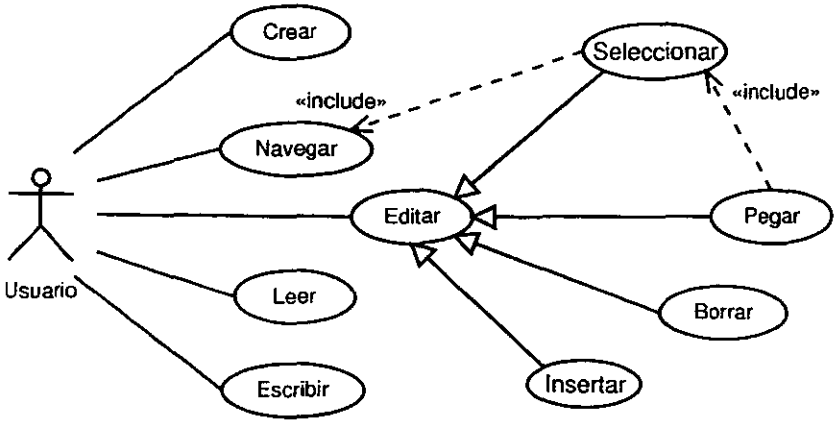

Figura 4.2: Casos de uso

sistema externo (como un editor de palabras), siempre habrá un ser humano operando el sistema. Por el momento identificaremos como único actor al **Usuario**. Sus responsabilidades son crear y modificar ecuaciones.

4.3.2 Identificación de casos de uso

Cada manera en que un actor usa el sistema se representa como un caso de uso. Una forma de identificar los casos de uso es por medio de la siguiente definición ([J+92] y [JBR99]): *un caso de uso es una secuencia de acciones que dan un resultado observable de valor para un actor particular.*

De este modo identificamos los siguientes casos de uso:

Crear Sucede cuando se crea una ecuación totalmente nueva. El resultado observable es una ecuación nueva sobre la que se puede trabajar.

Editar Cualquier modificación en un paso, hecha por el usuario a una ecuación existente, se considerará una instancia del caso de uso Editar. El resultado observable es un cambio inmediato sobre la ecuación.
En realidad este caso de uso es una abstracción de los casos de uso insertar, borrar, seleccionar y pegar. Para pegar, es preciso que el caso de uso Seleccionar haya sido realizado, aunque no necesariamente inmediatamente antes. El área copiada o cortada se pegara en la posición del cursor, al momento de realizar el caso de uso pegar.

Navegar Antes de editar una ecuación, el usuario debe poder señalar exactamente la parte del párrafo donde se realizará la modificación. El valor obtenido es un cambio de la posición del cursor, observable por medio de una representación visual del cursor sobre el párrafo en la vista (ver sección 4.5).

Seleccionar A partir de una posición inicial sobre el párrafo, se realizan movimientos del cursor (navegación) en modo selección, hasta llegar a una posición final. La señal visible de esta actividad es usualmente un cambio de color en el área marcada. El caso de uso termina cuando se copia o se corta el área seleccionada. En este último caso se obtiene otro resultado visual al desaparecer el área que había sido marcada.

Leer Recuperar una fórmula almacenada en algún medio persistente. El resultado observable es la ecuación recuperada en la Ventana.

Escribir Escribir la fórmula en el medio persistente, usando un código que pueda recuperarse por medio de lectura. El resultado observable es, por ejemplo, un campo en un archivo en disco.

Los casos de uso del sistema, y sus relaciones, se ilustran en la figura (4.2). El estereotipo `include` se usa para denotar la inclusión de un comportamiento representado en otro caso de uso.

Nótese como este modelo de casos de uso, que representa el comportamiento del sistema, podría aplicarse a cualquier editor. Las características específicas de un editor matemático (ver sección 2.1) no son visibles en este nivel de abstracción.

4.4 Requerimientos suplementarios

Los requerimientos suplementarios son requerimientos no funcionales que no pertenecen a un caso de uso específico. Usualmente son propiedades del sistema, tales como restricciones ambientales y de implementación, rendimiento y dependencias de la plataforma. Los requerimientos que se impondrán en esta etapa son:

- La plataforma donde correrá el sistema será Unix/X11.
- El lenguaje de programación que se usará será C++ por las siguientes razones:
  - C++ es un lenguaje maduro y estándar, y puede ser muy eficiente sin abandonar sus propiedades de orientación a objetos.
  - En casi cualquier plataforma se dispone de un compilador C++ actualizado.

- La versión empírica de Mathed fue escrita en C++. Es posible que, por lo menos al principio, se reúse parte de ese código.
- Se usará la biblioteca STL, que está escrita en C++.

- Debe ser suficientemente eficiente en velocidad y memoria para correr en cualquier estación de trabajo basada en arquitectura Intel x86 con memoria RAM mínima de 16MB. Esta es la plataforma mínima de los sistemas Linux.

- Por último, el código fuente del editor debe ser disponible públicamente bajo la licencia GPL (apéndice C), que es una licencia de código abierto (requisito 8, sección 4.1).

4.5 Interfase con el usuario

Una vez definidos los casos de uso, es necesario definir una interfaz con el usuario que permita al usuario ejecutar los casos de uso. Como se definió en la sección 4.2.1, el sistema contará con una interfaz gráfica que llamaremos Ventana, la cual contará con los siguientes elementos:

Vista Contiene una vista de la ecuación en proceso de edición. Debe ser capaz de recibir los eventos del teclado. Los cambios que se hagan a la ecuación deben ser observables de inmediato. La posición del cursor debe ser visible por medio de una barra vertical. Deben admitirse por lo menos dos formas de navegación:

Secuencial Por medio de las teclas de navegación. En un teclado estándar de computadora personal, las teclas de navegación son las flechas en las direcciones vertical y horizontal, las teclas de avance y retroceso de página, y las teclas de inicio y fin.

Aleatorio Colocando el apuntador del ratón en cualquier posición dentro de la vista y oprimiendo el botón izquierdo del ratón.

Menú de símbolos Puesto que la mayoría de los símbolos matemáticos no corresponden con una de las teclas de los teclados estándar de las estaciones de trabajo, se requiere de un menú de símbolos en el que aparezcan la mayoría de los símbolos matemáticos permitidos (ver apéndice B).

Barra de herramientas Otros elementos que no son símbolos, tales como fracciones o raíces, pueden insertarse usando una barra de herramientas.

Capítulo 5

Análisis

En esta etapa se analizan, refinan y reestructuran los requerimientos usando un lenguaje de desarrollador, en lugar de un lenguaje de usuario. Se persigue comprender con precisión los requerimientos y dar estructura al sistema como un todo, incluyendo su arquitectura.

5.1 Identificación de clases

Las clases en el análisis se enfocan en el manejo de requerimientos funcionales. Las clases del análisis caen en uno de tres estereotipos básicos, cada uno con su icono correspondiente. Estos estereotipos no forman parte del UML, son una extensión del Proceso Unificado [JBR99, sec. B.4] al UML y fueron introducidos originalmente en el método de Jacobson [J+92] :

Frontera  Una clase frontera se usa para modelar la interacción entre el sistema y los actores. Frecuentemente implica el intercambio de información. Modela las partes del sistema que dependen de los actores. Un cambio en una interfaz con el usuario es usualmente aislada en una o más clases de frontera. Cada clase frontera debe estar relacionada con al menos un actor, y viceversa.

Entidad  Una clase entidad se usa para modelar información de vida larga, frecuentemente persistente. Información y comportamiento asociado a algún fenómeno o concepto tal como un individuo, objeto o evento de la vida real. Muestran una estructura de datos lógica. Un objeto entidad no es necesariamente pasivo y puede tener un comportamiento complejo, relativo a la información que representa.

En la mayoría de los casos las clases entidad se derivan directamente de una clase correspondiente del dominio.

Control  Las clases control representan coordinación, secuenciación, transacciones y control sobre otros objetos. Se usan para encapsular el *control relativo* a un caso de uso específico. La dinámica del sistema se modela por clases control, puesto que coordinan las acciones principales y el flujo de control, y delegan trabajo a otros objetos.

5.1.1 Clases frontera

Ventana En el modelo del dominio se definio la clase Ventana como la que se encarga de recibir todos los eventos emitidos por el usuario por medio de la interfaz gráfica. Es por lo tanto el único objeto del dominio que tiene interacción directa con los actores, por lo que coincide con la definición de clase frontera de la sección anterior.

Los atributos y miembros de esta clase se derivarán directamente de los requerimientos para la interfaz con el usuario (sección 4.5), pero no son relevantes en esta etapa.

5.1.2 Entidades

Párrafo En el modelo del dominio se definió el objeto párrafo como el que contiene los elementos matemáticos tales como símbolos y estructuras. Dado que el párrafo contiene toda la información de una ecuación, y es persistente (requerimiento 6, sección 4.1), coincide con la definición de clases entidad.

Inserción En el modelo del dominio observamos que los atributos métricos de un caracter eran los mismos que los de una estructura, por lo que conviene que ambos tengan una clase base común. Esto simplifica la estructura del párrafo ya que solo contendrá inserciones y no necesitará duplicar código para las necesidades respectivas de caracteres y estructuras.

Estructura Aparte de los atributos métricos que comparte con los caracteres, una estructura contiene uno o más instancias de Párrafo, lo cual es un atributo exclusivo de una estructura.

5.1.3 Clases control

De las dos secciones anteriores tenemos la clase que recibe los eventos del usuario, y la clase que contiene los datos. Falta una o varias clases que interpreten los eventos emitidos por el usuario y procesados por Ventana y con base en esta interpretación coordinen la modificación de los datos. En el modelo del dominio se definió la clase Cursor, que mantiene la posición en la cual se realizarán los cambios emitidos por el actor. Esto es evidentemente la manifestación externa de la actividad de un objeto interno que controla los cambios locales del usuario.

Las clases control pueden obtenerse para encapsular control relativo a un caso de uso específico. De los casos de uso podemos derivar las siguientes clases:

Controlador Esta clase será usada para encapsular el control relativo a los casos de uso Crear y Editar. Una instancia de esta clase recibe los eventos procesados por Ventana, y modifica el Párrafo con base en esos eventos. También mantiene la posición del cursor.

Selector Las instancias copiar, cortar y pegar, del caso de uso Editar, requieren un control que permita seleccionar un área específica del párrafo, y almacenarlo en un párrafo temporal. Estas operaciones estarán bajo el control de la clase Selector. Esta clase no tiene relación directa con la clase Ventana, sino que es una clase al servicio del Controlador. Tiene a su vez una instancia de la clase Párrafo como almacenamiento temporal del área copiada.

Escritor Esta clase encapsula el control requerido para exportar un párrafo a un medio de almacenamiento persistente. Puesto que no requiere ninguna actividad de edición, es totalmente independiente de las demás clases control y solamente interacciona con la clase Ventana.

Lector Es similar a la clase Escritor, solo que se usa para importar un párrafo de un medio de almacenamiento persistente.

5.1.4 Diagrama de clases

En este punto ya han sido definidas las clases principales del análisis del sistema. La figura 5.1 muestra una vista general de la arquitectura del sistema, tomando en cuenta las clases y sus relaciones para los casos de uso discutidos en este capítulo. En este diagrama solo se han tomado en cuenta las clases que tienen relevancia en la arquitectura del sistema.

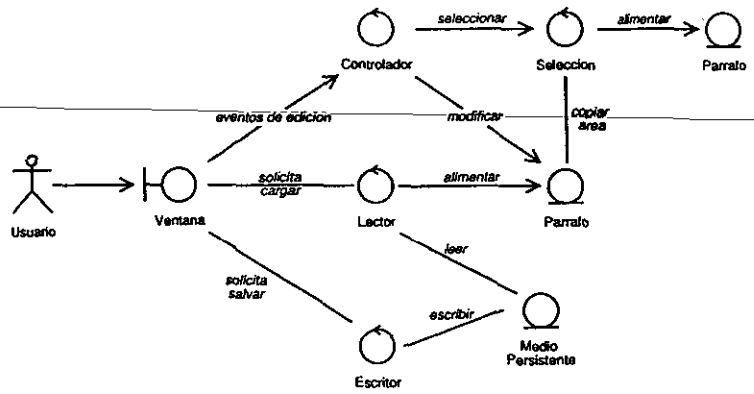

Figura 5.1: Clases del análisis y sus relaciones.

5.2 Realización de casos de uso

En esta sección se realizan los casos de uso definidos en el capítulo anterior, a un nivel de abstracción del análisis. De las clases identificadas en la sección anterior solo se toman en cuenta las clases que son arquitectónicamente relevantes para la realización de los casos de uso.

Una vez obtenido un esbozo de las clases necesarias para realizar los casos de uso, es necesario describir como interactúan sus instancias correspondientes. Esto se hace presentando su diagrama de colaboración y su flujo de eventos.

La realización de algunos casos de uso contienen requerimientos no funcionales. Estos son descritos en una sección de requerimientos especiales.

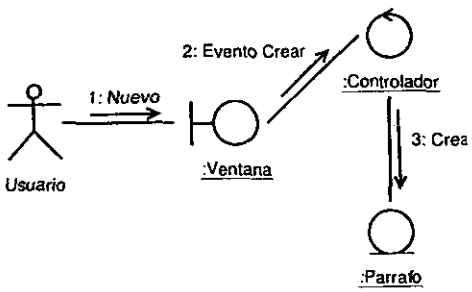

Figura 5.2: Caso de uso Crear.

5.2.1 Caso Crear

En este caso, el usuario crea una ecuación nueva (ver figura 5.2).
Flujo de eventos:

1. El usuario selecciona la opción de crear una nueva ecuación.
2. La ventana emite el evento `Crear` al controlador.
3. El controlador crea un nuevo objeto "Parrafo".

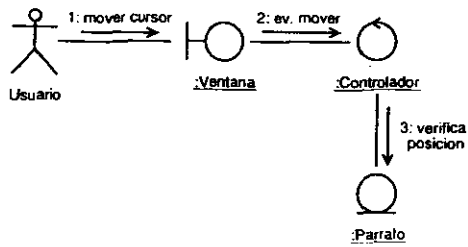

Figura 5.3: Caso Navegar.

5.2.2 Caso Navegar

El Usuario puede colocar el cursor en una posición arbitraria.
Flujo de eventos:

1. El Usuario mueve el cursor.
2. La Ventana emite el evento `Mover`.
3. El Controlador verifica que la posición del cursor coincida con una posición de un elemento del párrafo. Si la posición no es valida, simplemente no se cambia la posición del cursor.

Requerimientos especiales:

- Deben permitirse dos formas de mover el cursor: secuencialmente por medio de las teclas de navegación, o aleatoriamente, al mover el ratón sin soltar el botón izquierdo (ver sección 4.5).

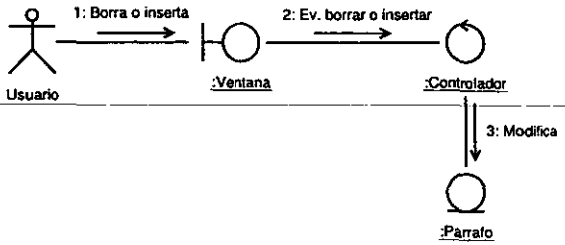

Figura 5.4: Caso Insertar o Borrar.

5.2.3 Caso Editar

Un caso de uso abstracto no tiene secuencia de comportamiento; ésta debe ser dada por los casos de uso concretos ([RJB99], pág. 495).

Caso Insertar o Borrar un elemento

El usuario inserta o borra un elemento en un solo paso, usualmente oprimiento una sola tecla del teclado (ver figura 5.4).

Flujo de eventos:

1. El Usuario inserta un elemento al oprimir una secuencia de teclas o seleccionar con el ratón algún elemento del menú de símbolos (ver sección 4.5).
2. La Ventana interpreta la secuencia de teclas y emite el evento `insertar`. En particular si la tecla oprimida fue `Del` o `Backspace`, se emite el evento `borrar`.
3. De acuerdo con el evento emitido por la ventana, el Controlador modifica el párrafo en la posición actual del cursor.

Requierimientos especiales:

- Debe ser posible insertar cualquier elemento tanto por medio de una secuencia de teclazos o por medio de la selección con el ratón en el menú de símbolos o la barra de herramientas (requisito 5, sección 4.1).
- Debe ser posible introducir un extenso conjunto de símbolos matemáticos. el problema con esta exigencia es que la codificación para los símbolos matemáticos puede ser diferente que la usada para los símbolos más usados en la cultura occidental, la cual es una codificación estándar para poder ser usados en computadoras. Entre los códigos más comunes han estado el ASCII, el ISO-8859-1.

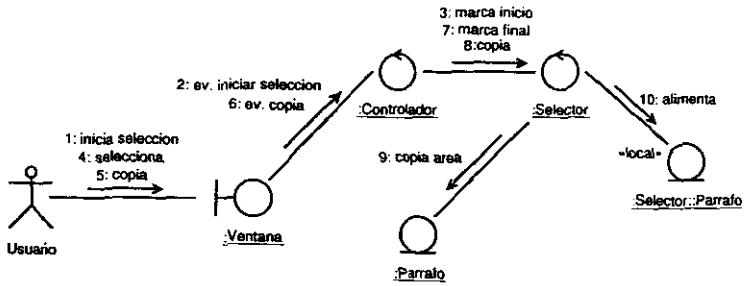

Figura 5.5: Caso Seleccionar.

Caso Seleccionar

El usuario selecciona una parte del párrafo y la copia o la corta para insertarla posteriormente en otra parte del párrafo (caso pegar, sección siguiente). Ver figura 5.5.

Flujo de eventos:

1. El Usuario inicia la selección
2. La ventana emite el evento `Iniciar_seleccion`.
3. El Controlador solicita al Selector marcar la posicion inicial de la selección.
4. El Usuario procede a navegar en modo selección (`incluye(Navegar)`), lo que llamaremos simplemente *seleccionar*.
5. El Usuario selecciona Copiar
6. La ventana emite el evento `Copiar`.
7. El Controlador solicita al Selector marcar la posición final de la selección.
8. El Controlador solicita al Selector proceder a copiar.
9. El selector copia la parte del párrafo entre las posiciones inicial y final
10. El selector almacena esos datos en su párrafo local.

Camino alternativo:

5. El usuario selecciona Cortar
6. La Ventana emite el evento `Cortar`

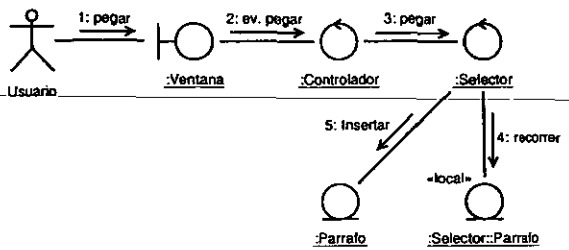

Figura 5.6: Caso Pegar.

7. El Controlador solicita al Selector cortar.

8. El Selector copia la parte del párrafo entre las posiciones inicial y final y almacena esos datos en su párrafo local.

9. El Selector elimina la parte del párrafo entre las posiciones inicial y la final.

Requerimientos especiales:

- Para cumplir con el requisito 2 (sección 4.1), la selección se inicia y continúa de dos formas: al oprimir las teclas de movimiento (flechas) manteniendo al mismo tiempo oprimida la tecla `Shift`; o seleccionar el área con el ratón manteniendo oprimido el botón izquierdo. Este es el comportamiento esperado en la mayoría de los editores de texto.

Caso Pegar

El área previamente copiada o cortada (caso Seleccionar) se inserta en una posición arbitraria del párrafo (ver figura 5.6). Suponemos que el Usuario ya colocó el cursor en cualquier posición del párrafo, por medio del ratón o de las teclas de navegación (caso navegación).

Precondiciones: Deben haberse realizado previamente los casos Seleccionar y Navegar, en ese orden.

Flujo de eventos:

1. Usuario selecciona pegar.

2. Ventana emite el evento `Pegar`.

3. Controlador solicita a Selector proceder a pegar.

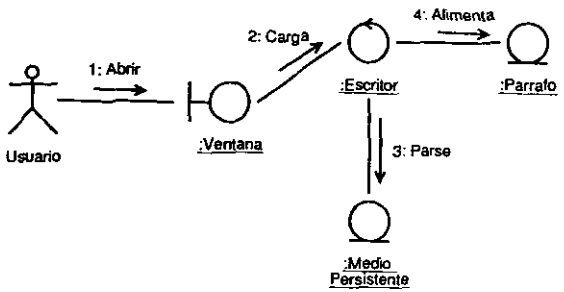

Figura 5.7: Caso Leer.

4. Selector recorre el contenido de su Párrafo local
5. Selector inserta el contenido de su Párrafo local en la posición actual del cursor en el Párrafo.

Requerimientos especiales:

- El Usuario selecciona pegar por medio de una secuencia de teclazos idéntica a la que usaria en modo texto, o bien por medio de la selección de una entrada en el menú de edición.

5.2.4 Caso Leer

Se recupera una ecuación almacenada en un medio de almacenamiento persistente. Ver figura 5.7.

Flujo de eventos:

1. El Usuario selecciona abrir
2. La Ventana emite el evento `Cargar` al Lector.
3. El lector parsea[1] los datos en el medio persistente.
4. El lector alimenta el Párrafo con los datos obtenidos en el paso anterior.

Requerimientos especiales:

- El Lector debe ser capaz de analizar un lenguaje de comunicación matemática estándar. Por lo menos LaTeX.

[1] De parser, en inglés: analiza una oración y la descompone en componentes y la describe gramaticalmente. El término se usa en teoría de compiladores [ASU86].

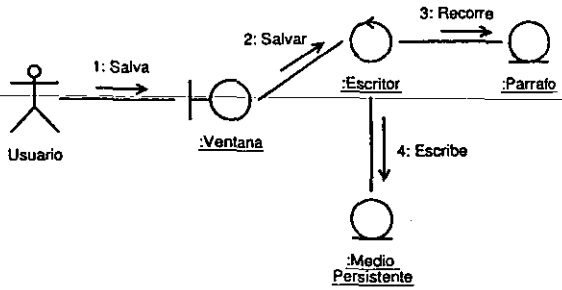

Figura 5.8: Caso Escribir.

5.2.5 Caso Escribir

Se almacena una ecuación en un medio persistente. Ver figura 5.8.
Flujo de eventos:

1. Usuario selecciona salvar
2. Ventana emite el evento `Salvar` al Escritor.
3. Escritor recorre el Párrafo.
4. Escritor escribe los datos obtenidos en el paso anterior, en el medio persistente.

Requerimientos especiales:

- El Escritor debe ser capaz de escribir en al menos un lenguaje de comunicación matemática estándar. Por lo menos LaTeX. De ser posible también MathML.

5.3 Paquetes

Los paquetes del análisis brindan un medio de organizar el modelo del análisis en piezas más organizadas. Los paquetes deben ser relativamente independientes, con poco acoplamiento entre sí y al mismo tiempo con alta cohesion interna. Esto fácilita el mantenimiento ya que cualquier cambio que se haga dentro de un paquete no afectará a los demás paquetes. Por lo tanto el número de relaciones entre clases de diferentes paquetes debe ser mínimo.

5.3.1 Identificación de paquetes

Los paquetes identificados para este proyecto, derivados algunos de los casos de uso identificados en el capítulo anterior (ya que los casos de uso añadidos en este capítulo son extensiones o especializaciones de los mismos) son los siguientes:

Interfaz gráfica Encapsula la funcionalidad de la clase Ventana y los requerimientos del prototipo (ver sección 4.5). Para cumplir con el requerimiento 7 (integración a otros sistemas), la interfaz gráfica debe ser lo más independiente posible, por lo que decidimos encapsularla en su propio paquete. En este nivel de abstracción reduciremos su interacción con las clases de los demás paquetes (clases control) a la emisión de eventos, por medio de una interfaz única que será definida en el diseño (siguiente capítulo).

Edición En este paquete se encapsula la funcionalidad de edición, que reúne todos los casos de uso de la sección 5.2.3. Las clases principales son los controles Controlador y Selector.

Lectura Encapsula la funcionalidad de importar una ecuación de un medio persistente.

Escritura Encapsula la funcionalidad de escribir una ecuación en un medio persistente para que pueda ser recuperada más tarde por medio de lectura, o bien pueda ser exportada para su integración a un documento externo o su comunicación por correo electrónico.

Almacenamiento interno Los últimos 3 paquetes comparten el objeto Párrafo lo que contradice el objetivo de independencia y poco acoplamiento entre paquetes. Una manera apropiada de manejar este problema es extraer esta clase y que tenga su propio paquete. Llamaremos a este paquete Almacenamiento Interno (no persistente).

5.3.2 Definición de dependencias entre paquetes

Si las clases contenidas en un paquete están relacionadas con clases de otros paquetes, las dependencias entre ellas deben ser definidas. Para hacer claras dichas dependencias puede ser útil disponer los paquetes en niveles de generalidad. Esto hace clara la distinción entre funcionalidades específica y general.

Una división preliminar en capas se muestra en la figura 5.9. La capa A es específica de la aplicación. La capa B es general, independiente de la aplicación. Las relaciones entre la capa A y la B se harán por medio de un mecanismo de emisión de eventos.

Capítulo 6

Diseño

En este capítulo se expande y detalla el modelo desarrollado en el análisis, tomando en cuenta las implicaciones técnicas y las restricciones. El modelo del diseño se aplicará en la implementación. Una consecuencia del requerimiento 8 (código fuente público internacionalmente) es que se use el idioma inglés en los nombres de clases en el código fuente.

6.1 Identificación de subsistemas y sus interfases

Al igual que los paquetes del análisis, los subsistemas brindan un medio de organizar el diseño en piezas manejables. De hecho, la descomposición en paquetes hecha durante el análisis puede tomarse como punto de partida para identificar los subsistemas del diseño.

6.1.1 Identificación de subsistemas

En general se usarán los subsistemas correspondientes a los paquetes del análisis (ver sección 5.3.1). En el capítulo anterior vimos que la interfase gráfica se encarga de comunicar las órdenes del usuario humano al sistema. Pero no se ha tomado en cuenta el despliegue de la ecuación que se está editando, en el elemento Vista de la ventana (ver sección 4.5). Si bien este despliegue se manifiesta en un elemento de la Ventana, debe estar bajo el control del subsistema de edición, ya que ningún otro objeto tiene acceso al estado interno del párrafo en edición. Esto se logra pasando como parámetro un *contexto gráfico* (ver glosario) que no es otra cosa sino una interfase para un subsistema nuevo:

Despliegue *Este subsistema proporciona un medio en el cual el subsistema de edición* puede graficar una ecuación sin saber los detalles de bajo nivel del medio en el

que se graficará la ecuación, sino solamente la interfase.

Otros subsistemas que habría que considerar son los que corresponden a los niveles bajos, no considerados en las 2 capas superiores de los paquetes del análisis. Este es el caso el sistema gráfico será el sistema X window, que se usa en las estaciones de trabajo Unix. En ningún caso se llamará ninguna rutina de bajo nivel de este sistema, sino que se utilizará el subsistema Despliegue, el cual encapsulará todas las funciones gráficas necesarias.

Si el contenido de un subsistema tiene relaciones con otro subsistema, deben definirse las dependencias entre subsistemas. La dirección de la relación debe ser la misma.

6.1.2 Interfases

Las interfases dadas por el subsistema definen operaciones accesibles externamente al subsistema. Estas interfases están dadas por clases o por otros subsistemas dentro del subsistema.

El subsistema interfase Gráfica se relacionará con los subsistemas de la capa intermedia por medio de un único subsistema, que por el momento llamaremos simplemente Fachada. Esta idea corresponde al patrón de diseño Facade (ver sección 6.5). Los subsistemas de la capa intermedia usarán como interfase la clase correspondiente del análisis.

Los subsistemas principales de la capa intermedia (lectura, escritura y edición) no se relacionan entre sí, pero se relacionan con un subsistema común: Almacenamiento interno.

6.2 Identificación de clases

En la etapa de diseño pueden esbozarse algunas clases a partir de las clases encontradas en el análisis. Las relaciones entre clases del análisis dan un punto de partida para definir las relaciones en las clases del diseño. Identificamos las clases dentro del contexto del subsistema correspondiente.

6.2.1 Interfase gráfica

En el análisis, la clase Ventana encapsulaba la interfase con el usuario. Uno de los requerimientos exigen que el sistema pueda integrarse a un sistema externo, lo que implica una restricción en las relaciones de la ventana con las clases control que realizan los casos de uso. Los controladores deben mantener una dependencia mínima con cualquier clase que no sea interna al sistema. Este mismo requerimiento implica que los detalles de diseño de la Ventana deben mantenerse a un alto nivel de abstracción

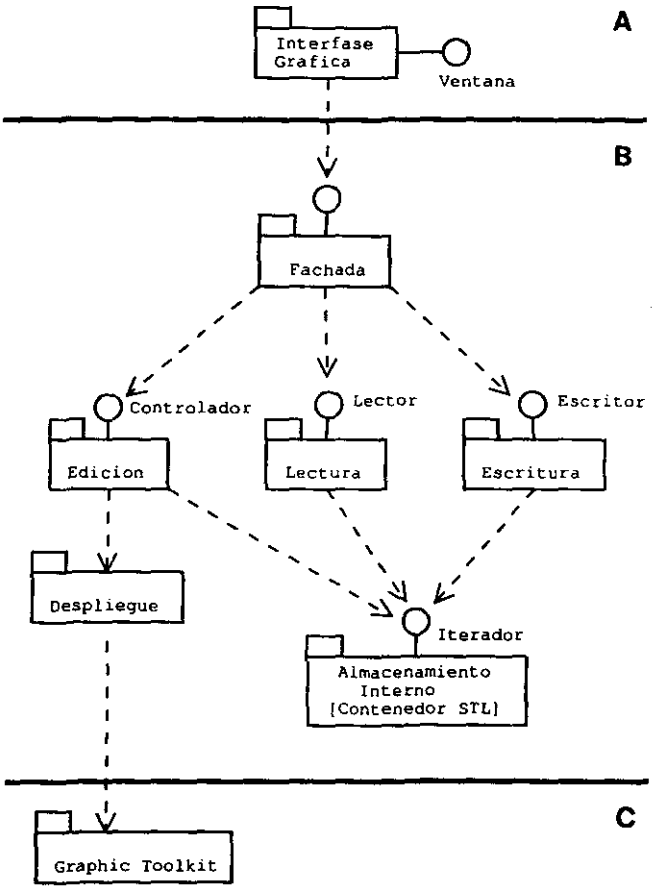

Figura 6.1: Subsistemas: sus dependencias e interfaces.

en esta etapa. Las relaciones del subsistema gráfico con el resto del sistema se realiza a través de una sola interfase.

6.2.2 Clases relativas al Almacenamiento Interno

Durante el análisis la clase Párrafo se definió simplemente como una clase que contendrá pasivamente los elementos que componen un párrafo matemático. Las relaciones de esta clase con otras clases son básicamente de acceso a los datos: consultar un elemento, insertar un elemento, recorrer el contenedor.

Párrafo

Contenedor de elementos de párrafo. Los elementos de párrafo fueron identificados en el análisis simplemente como insersiones (ver sección 5.1.2).

Iterador

Los objetos que tienen acceso a los datos del Párrafo fueron definidos en el capítulo anterior como clases control. Estas clases realizan funciones muy diferentes por lo que conviene que la estructura interna del Párrafo sea invisible para ellas y que solamente puedan tener acceso a los datos por medio de un intermediario. Este intermediario tendrá como única función el proveer un medio de acceso secuencial a los elementos del contenedor sin exponer su estructura interna. Esta definición coincide con la definición del patrón *Iterator* (ver sección 6.5).

Inserción

En el capítulo anterior se definió inserción como una entidad que encapsula las propiedades métricas comunes de los caracteres y las estructuras. Es el elemento básico del párrafo. Para abreviar, llamaremos `Inset` a esta clase en el código fuente. Esta es en realidad una clase abstracta. Llamaremos `InsetSymbol` a la subclase que representa un solo símbolo matemático.

Inserción activa

Una estructura es una inserción que tiene como elementos párrafos editables. Llamaremos a esta clase inserción activa, o `InsetActive`. Esta es en realidad una clase abstracta, ya que las estructuras concretas se manfiestan de manera específica en el momento de la realización de los casos de uso. Por ejemplo la estructura `sqrt` (una raíz cuadrada) se manifiesta con el signo $\surd$ a la hora de dibujarse sobre la vista.

Tabla de Símbolos

Los símbolos más usados en la cultura occidental tienen una codificación estándar para poder ser usados en computadoras. Uno de los requisitos exige que el sistema sea capaz de manejar la mayoría de los símbolos usados en matemáticas. Desafortunadamente estos símbolos no tienen una codificación estándar, aunque el consorcio Unicode[1] está trabajando para incluirlos en su codificación internacional.

Conviene que el editor tenga solo un número identificador del símbolo y, consultando a un registro común pueda tener información tal como la fuente (al momento de desplegarlo en pantalla), el tipo de símbolo (si es un operador binario, se dejará un poco de espacio al rededor). De esto puede ocuparse una clase que encapsule los métodos necesarios para manejar una tabla de símbolos.

InsetFactory

Aunque las propiedades específicas de las inserciones se manifestarán durante la realización de los casos de uso, para el párrafo solamente son elementos. Por lo tanto es conveniente tener una interfase para crear objetos sin especificar su clase concreta. Como se verá más adelante, esta clase es parte de una combinación de los patrones Flyweight y Abstract Factory (ver sección 6.5).

6.2.3 Clases relativas al subsistema edición

En el nivel de abstracción del análisis sólo se identificaron dos clases para realizar el caso de uso Editar: Controlador y Selector. En esta sección se identificarán las clases derivadas de los requerimientos especiales de este caso de uso en el análisis.

Controlador

El controlador recibe del exterior (usualmente de la clase frontera Ventana) solicitudes para modificar el Párrafo en la posición actual del cursor. Entre las operaciones que debe coordinar el controlador están:

- El movimiento del cursor. Una operación para cada movimiento, y en el caso del ratón, uno que lo mueva a una posición arbiraria.
- Insertar y borrar un elemento en la posición actual.
- Seleccionar un área, copiar, cortar y pegar.
- Dibujar la ecuación dentro de la vista.

[1] `http://www.unicode.org`

Cursor

No es casualidad que en la definición del patrón *Iterator* en [GHJV94] (página 257) se le de a ese patrón el sobrenombre de *Cursor*. Una de las responsabilidades del cursor es recorrer secuencialmente el Párrafo sin exponer la representación interna del mismo. Otra función es indicar visualmente dicha posición para que el usuario sepa con exactitud en donde se efectuarán sus modificaciones.

Es responsabilidad del Controlador mantener un cursor visible y que corresponda exactamente a dicha posición. Para conseguirlo, debe saber en todo momento las coordenadas físicas correspondientes a la posición en donde se encuentre el cursor. Es conveniente entonces que el cursor, además de que mantenga en todo momento una posición única dentro del párrafo, tenga las coordenadas correspondientes a dicha posición dentro de la Vista.

Pila

A lo largo de este trabajo se ha repetido varias veces que un Párrafo puede contener varios Párrafos, por medio de estructuras. Al abrir una estructura el cursor aparece dentro de un párrafo de la misma. Dentro de esa estructura puede entrar a otra estructura (anidamiento). Al salir de la estructura debe recordar su posición previa dentro del párrafo o subpárrafo, sin importar el nivel de anidamiento en el que esté.

Este comportamiento de el último que entra es el primero que sale recuerda el comportamiento de uno de los patrones más antiguos de la teoría de la computación: la pila.

Cada vez que el cursor entre en una estructura será guardado en la pila.

Selector

Toma del cursor la posición inicial de la selección. La posición final es siempre marcada por el cursor del controlador. Tambien tiene un párrafo donde almacena la información cortada o copiada del párrafo original.

6.2.4 Subsistema de lectura

El subsistema de lectura tiene todos los elementos de un compilador clásico (ver [ASU86]). Debe tener un analizador léxico para descomponer la entrada en componentes gramaticales. El analizador sintáctico (mejor conocido como *parser*) que interprete dichos componentes y genere un código que en este caso es la estructura interna del párrafo.

Puesto que los elementos del párrafo se agregan en forma secuencial, el subsistema de lectura puede utilizar un iterador para construir el párrafo.

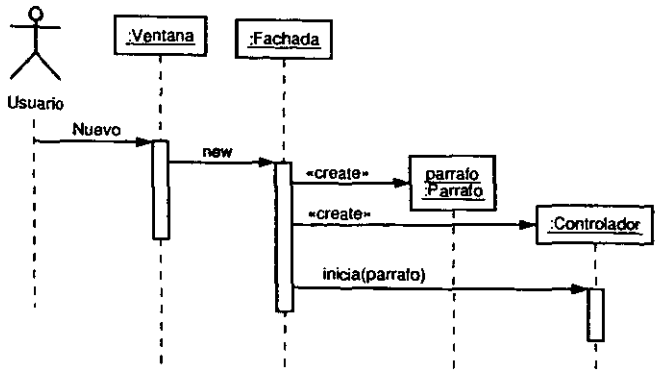

Figura 6.2: Caso Crear

6.2.5 Subsistema de escritura

El subsistema de escritura codifica el contenido del párrafo en un lenguaje de comunicación matemática (ver capítulo 2). Para acceder a todos los elementos del párrafo y generar dicho código, puede utilizar un iterador simple.

6.3 Diseño de los casos de uso

Una vez identificados los subsistemas, sus interfases y las clases que los componen, pasamos al diseño de los casos de uso. Para cada caso de uso hay que identificar los subsistemas, clases e interfases que participan. Por cada caso de uso presentamos un escenario ilustrado en el diagrama de secuencia en la figura correspondiente a cada caso.

6.3.1 Crear

El usuario solicita una nueva ecuación por medio de la ventana (entrada de menú o secuencia de caracteres). La ventana emite el evento **new** a la interfase. La interfase crea una nueva instancia de Párrafo en la variable `parrafo` y un nuevo controlador. El controlador es iniciado con `parrafo`. Esto significa que la variable `Controlador::cursor` será inicializada con `parrafo`.

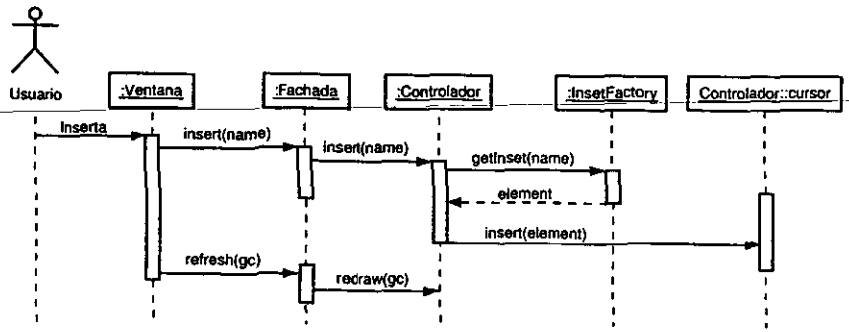

Figura 6.3: Caso Insertar elemento

6.3.2 Insertar elemento

El usuario inserta un símbolo (por medio de selección en el menú de símbolos o por secuencia de caracteres). La ventana emite el evento insertar, con el nombre del símbolo como argumento. La interfase manda el mensaje al controlador. El controlador envía un mensaje al `InsetFactory` solicitando un `inset` a cambio del nombre del símbolo y manda el `inset` al cursor. El cursor, que es un iterador, inserta en `parrafo`, en la posición actual, la instancia obtenida del `InsetFactory`. Como la ecuación ha cambiado, el controlador es redibujado en el contexto gráfico (gc) enviado por la ventana.

Si en lugar de un símbolo se inserta una estructura (por medio de una secuencia de teclazos o por presionar un botón en la barra de herramientas), la secuencia es la misma salvo que el `InsetFactory` regresará una instancia de un `InsetActive` en lugar de un `InsetSymbol` y se moverá el cursor a la derecha, causando que el cursor se introduzca en el primer párrafo editable contenido en el inset. Esto se ilustra mejor en el siguiente caso de uso.

6.3.3 Navegar

Cuando el usuario inicia el movimiento del cursor dentro de la vista, se genera una cadena de mensajes que llega al iterador miembro del Controlador, que llamamos cursor. Cuando el Controlador envía el mensaje move() correspondiente a un movimiento en una dirección cualquiera (izquierda, derecha, arriba, abajo), el cursor regresa un valor booleano. Si este valor es `true`, significa que el movimiento es válido. Lo siguiente es comprobar si existe un `InsetActive` en la nueva posición. Si ese es el caso, el cursor en la posición actual se guarda en la pila y al mismo tiempo la pila regresa un cursor nuevo con la función push. Este cursor se actualiza con el primer párrafo editable contenido

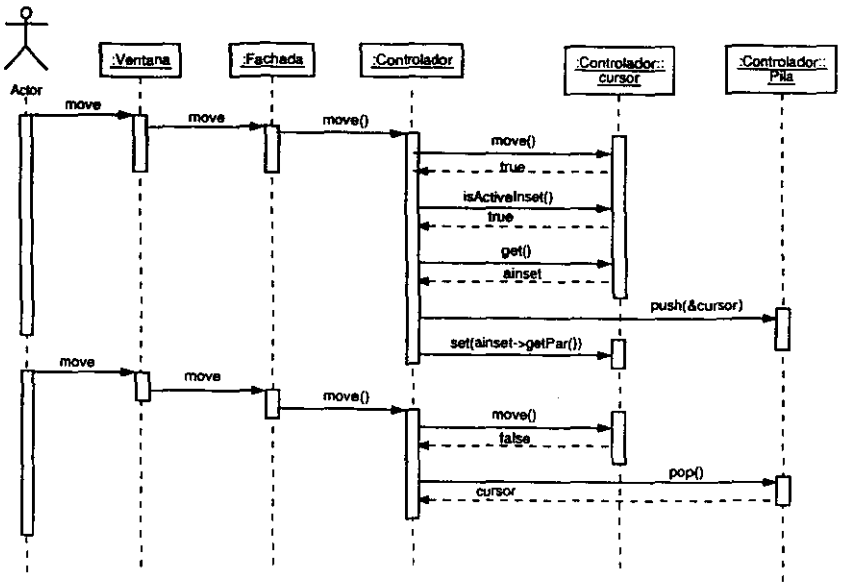

Figura 6.4: Caso Navegar.

en el `InsetActive`. De este modo el cursor aparece adentro de la estructura.

Si el usuario continúa moviendo el cursor, se repite la misma secuencia. Si cursor regresa `false`, implica que ya no es válido el movimiento en la dirección dada. Entonces se obtiene el estado anterior del cursor por medio de la función pop de la pila. De esta manera el cursor aparece afuera y junto al `InsetActive`. Este escenario, ilustrado en la figura 6.4, es una simplificación de lo que realmente puede ocurrir. Falta comprobar si la pila ha llegado a su tope. En ese caso sabremos que el cursor ha llegado a los límites de la ecuación.

Hay que notar que este escenario es recursivo: es posible contener una o varias estructuras dentro de otras estructuras y por este mecanismo el cursor podrá entrar y permitir modificar el contenido de cualquiera de ellas.

6.3.4 Seleccionar

La selección empieza cuando el usuario hace el primer movimiento del cursor en modo selección (moviendo el cursor oprimiendo al mismo tiempo la tecla Shift, o moviendo el apuntador del ratón apretando al mismo tiempo el botón izquierdo). El Selector marca esa posición con el iterador `selcursor`.

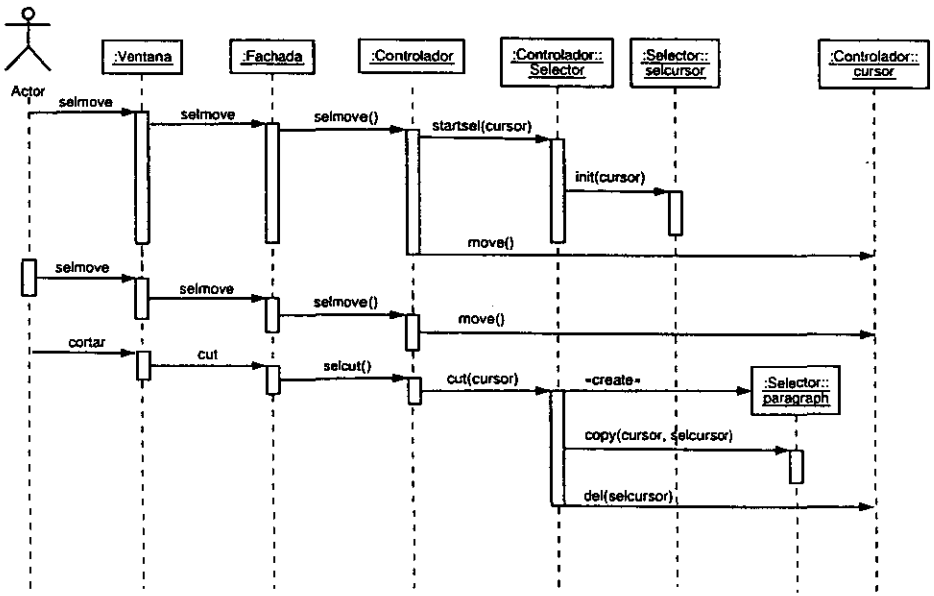

Figura 6.5: Caso Seleccionar

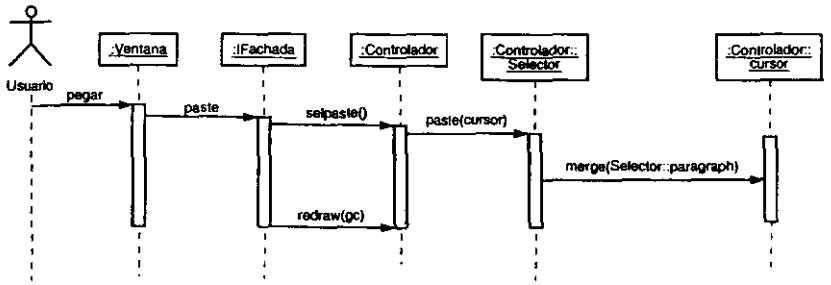

Figura 6.6: Caso Pegar

El usuario continúa navegando en modo selección. El área seleccionada está entre la posición marcada al principio por el selector y la posición actual del cursor.

La secuencia termina cuando el usuario solicita copiar o cortar. El Selector crea una instancia local de Párrafo, para almacenar el contenido entre la posición actual y la inicial de la selección. Si se selecciono cortar, seleccionada es eliminada del párrafo. El mensaje `del(selcursor)` hace que cursor elimine el contenido entre la posición actual (marcada por él mismo) y la posición marcada por `selcursor`.

Ambos `cursor` y `selcursor` son iteradores de `parrafo`. El selector anula el modo selección destruyendo `selcursor`.

Cabe mencionar que la navegación en modo selección es más simple que el caso de uso Navegación en modo normal. Se inhibe el mecanismo de edición anidada, por lo que no se llama a la función push de la pila. Por otra parte, si es posible que la posición original al inicio del modo selección (guardada en `selcursor`) esté en una estructura y el movimiento del cursor lleve a la posición actual a salirse de la estructura (y por lo tanto se llamará la función pop de la pila).

6.3.5 Pegar

El Usuario solicita pegar. El contenido del Párrafo local del selector es copiado en la posición actual de `cursor`.

Precondición Debe existir un Párrafo no vacío en Selector, producto de una selección previa (ver caso seleccionar). De otro modo, no tendrá ningún efecto sobre el párrafo la ejecución de este caso de uso.

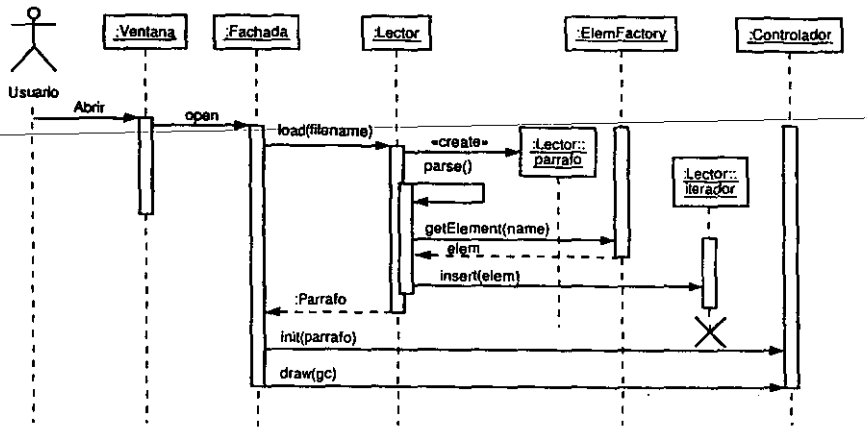

Figura 6.7: Caso Leer

6.3.6 Leer

El Usuario lee una ecuación previamente creada del disco. El parser obtiene nombres de símbolos y estructuras, que son integrados al párrafo por los mismos medios que si el usuario los insertara a través de la ventana: el parser obtiene el nombre de los objetos, da este nombre al InsetFactory para obtener a cambio una instancia a un Elemento (que no es otra cosa que un inset). El iterador local inserta los elementos en el párrafo. Una vez conluida la operación, devuelve un párrafo nuevo a la Fachada y destruye su iterador local. La fachada inicializa el Controlador con un nuevo párrafo, lo cual se manifestará con la gráfica de la ecuación en la Vista.

6.3.7 Escribir

El Usuario desea escribir una ecuación en disco. La Fachada envía un mensaje al Escritor con un apuntador a `párrafo`, con el cual el Escritor podrá inicializar su iterador interno. Este iterador recorrerá el párrafo para obtener los nombres de cada uno de los elementos y escribirlos en un archivo en disco.

La sintaxis que utilizará al escribir será la correspondiente a un lenguaje matemático específico. En particular, si escribe el lenguaje escogido para almacenar las ecuaciones, debe ser tal que el subsistema de lectura pueda recuperar la fórmula exactamente como era cuando se escribió.

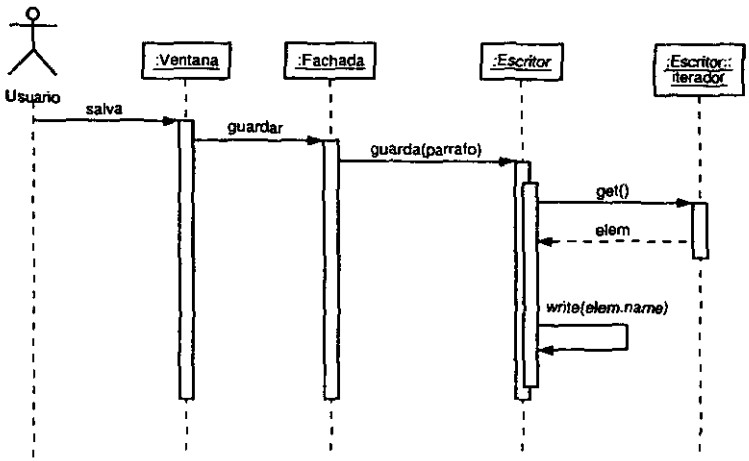

Figura 6.8: Caso Escribir

6.4 Diagramas de Clases

Una vez identificadas las clases y subsistemas, su participación en los casos de uso y las interacciones entre sus instancias, así como posibles patrones de diseño, identificamos las asociaciones entre las clases, por medio de los diagramas de clases.

En la figura 6.9 se muestra la estructura interna de una ecuación. La clase `Inset` es la clase base para representar cualquier elemento de la ecuación, ya sea un símbolo, operador o estructura. La clase `ActiveInset` encapsula la información correspondiente a una estructura. Es una subclase de `Inset`, con la propiedad adicional de que puede contener un número específico de instancias de `Paragraph`. Por último la clase `Paragraph` representa una expresión matemática al contener un número ilimitado de `Insets`. Esta última relación de agregación da lugar a una estructura recursiva, mejor definida por el patrón *Composite* (ver sección 6.5).

Únicamente la clase `Iterator` puede acceder al contenido de `Paragraph`. Los métodos `begin()` y `end()` colocan la posición del iterador al principio o al final del párrafo, respectivamente. El método `next()` mueve la posición al elemento siguiente. Estos métodos permiten que el iterador se coloque en todas las posiciones posibles dentro del contenedor. El método `getItem()` devuelve el elemento en donde se encuentra el iterador. El método `InsertItem()` permite insertar un elemento en la posición en que se encuentre el iterador.

La figura 6.10 muestra las relaciones de la clase InsetFactory con las diferentes clases derivadas de Inset. Básicamente estas relaciones definen el patrón Flyweight

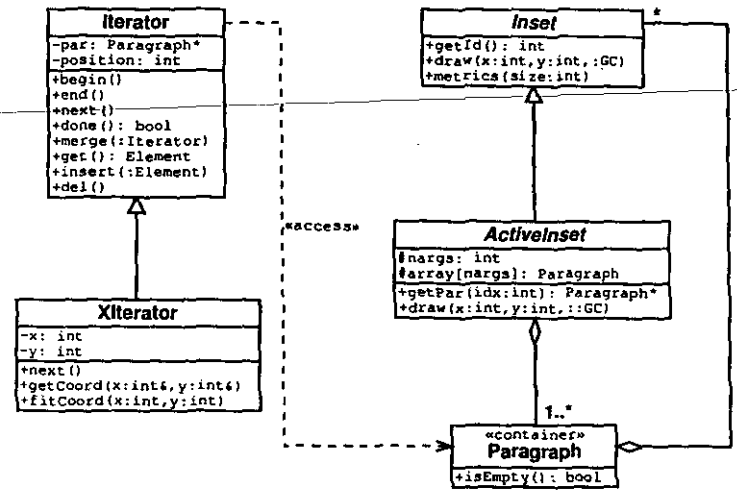

Figura 6.9: Estructura interna y clases de acceso.

([GHJV94], pág. 195). La clase abstracta `Inset` brinda una interfase con la que sus descendientes pueden actuar en estado extrínseco (dependiente del contexto). La clase `SymbolInset` contiene un espacio para el estado intrínseco (independiente del contexto), que en este caso sería la identificación del símbolo en la tabla de símbolos. Los objetos de dicha clase serían compartibles, es decir, cada símbolo tendría solamente una instancia. La clase `ActiveInset` es abstracta (ver la sección 6.2.2 en la página 40). Las instancias de las clases concretas derivadas de esta clase no son compartibles, ya que la información que contiene cada uno (párrafos concretos) es única. Por último, la clase InsetFactory es la encargada de brindar instancias concretas de insets, dado un nombre de un símbolo u objeto matemático. Si la instancia correspondiente a un nombre aún no existe, se crea una nueva.

Esta estructura también se asemeja al patrón Abstract Factory, ya que los clientes de InsetFactory obtienen únicamente apuntadores a Inset, que es abstracta, y no apuntadores a clases concretas.

En la figura 6.11 se muestra la arquitectura del sistema. La clase Fachada es la única clase *visible* externamente. La operación `dispatch()` admite como parámetro el nombre o identificador de un evento o comando que la `Fachada` interpreta y ejecuta la operación correspondiente por medio de las clases `Controller`, `Writer` y `Reader`. Otra responsabilidad de la `Fachada` es pasarle el contexto gráfico al Controlador para obtener una representación gráfica de la ecuación representada en el párrafo `par`.

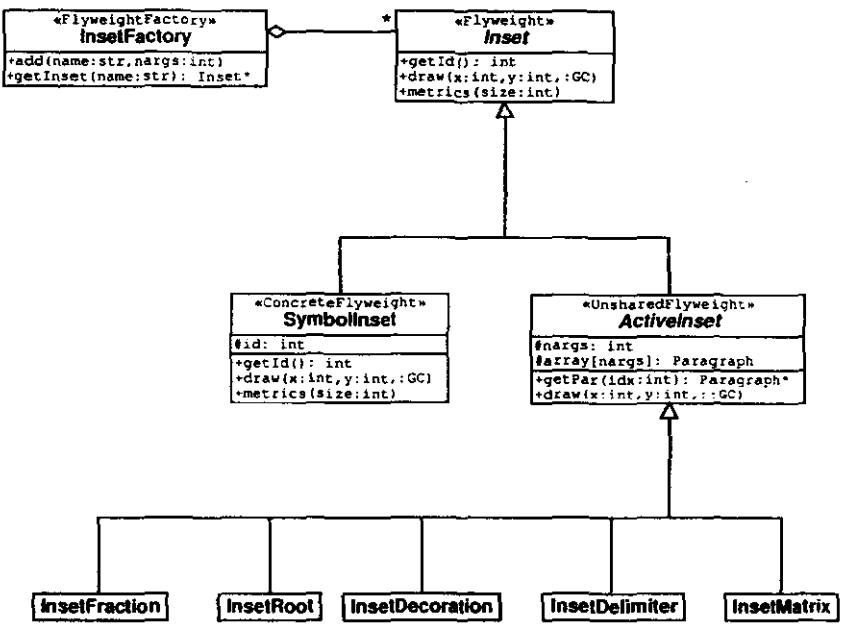

Figura 6.10: Administración de inserciones.

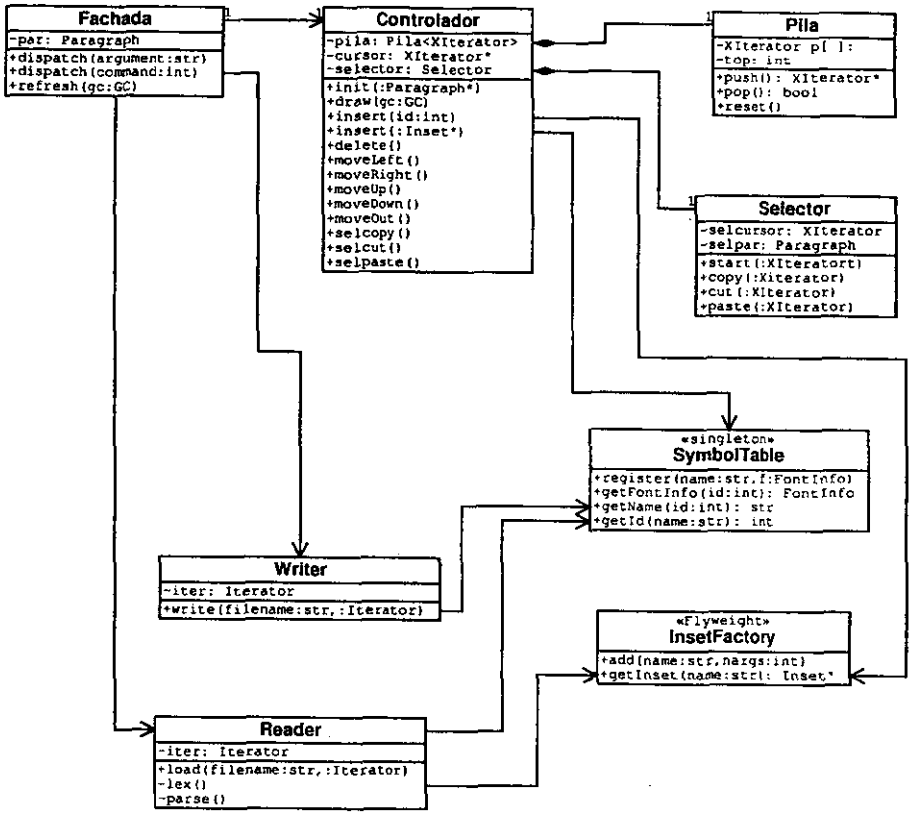

Figura 6.11: Arquitectura del sistema.

La clase `Controller` coordina todas las operaciones de edición, con la ayuda de las clases `Pila` y `Selector`. El atributo `cursor` es un iterador que mantiene una posición única dentro del párrafo, y las coordenadas correspondientes. Para realizar las operaciones de inserción de un símbolo o una estructura utiliza las clases `InsetFactory` y `SymbolTable` para obtener una referencia a un Inset y el identificador de un símbolo.

La clase `Reader` se encarga de la recuperación de una fórmula almacenada en disco o cualquier otro medio. Utiliza las clases `InsetFactory` y `SymbolTable` con el mismo propósito que el controlador. La clase Writer usa `SymbolTable` para obtener el nombre de un elemento, dado su identificador.

6.5 Patrones de diseño utilizados

Varias de las estructuras y relaciones desarrolladas en este capítulo corresponden a los siguientes patrones de diseño [GHJV94]:

Adapter Convierte la interfase de una clase en otra interfase esperada por el cliente. Permite que puedan trabajar juntas clases que de otro modo serían incompatibles (subsistema Despliega).

Abstract Factory Brinda una interfase para crear familias de objetos relacionados entre sí sin especificar sus clases concretas (clase InsetFactory).

Composite Compone objetos en estructuras de árbol para representar jerarquías parte-todo. Permite tratar uniformemente a objetos individuales y a composiciones de objetos (clases Iterador y Párrafo).

Facade Define una interfase de alto nivel que hace más fácil de usar a un subsistema complejo (clase Fachada).

Flyweight Usa compartimiento para soportar un gran número de objetos de grano fino eficientemente (clase InsetFactory).

Iterator Brinda un medio de acceder a los elementos de un contenedor secuencialmente sin exponer su representación interna (clase Iterador).

Singleton Se asegura que una clase tiene una sola instancia y brinda un punto de acceso global a ella (clases SymbolTable e InsetFactory).

Capítulo 7

Resultados

7.1 Implementación

Se implementó el sistema conforme al diseño mostrado en el capítulo anterior. El lenguaje de programación que se usó fue C++, por lo que teóricamente, el sistema puede ser compilado en cualquier plataforma que cuente con un compilador C++ estándar.

Durante el análisis y diseño del sistema se utilizó el paradigma orientado a objetos para definir la arquitectura del sistema a un nivel de abstracción relativamente alto. Para la implementación de las estructuras de datos el paradigma de la Programación Genérica [Aus98] resultó conveniente para evitar reescribir código. Para las estructuras de datos básicas, se utilizaron contenedores y algoritmos de la biblioteca STL [MS96], la cual se incluye en la instalación de C++ estándar. La estrategia de implementación de la STL, aparte del reúso de código desarrollado por expertos y depurado durante años, es la separación de los contenedores de datos de los métodos de acceso (iteradores) y los algoritmos, eliminando las dependencias directas entre estos componentes. Esto permite una gran flexiblidad, ya que se pueden probar distintos algoritmos con los mismos contenedores, y viceversa.

En los subsistemas de lectura y escritura decidimos utilizar el lenguaje LaTeX [Lam94], dado que se pudo reusar un *parser* capaz de leer este lenguaje, tomado de la versión empírica de Mathed.

Como archivos auxiliares se requirieron las fuentes con el conjunto de caracteres matemáticos de LaTeX y AMS [GMS94] en los formatos T1 (postscript) y True Type, para poder desplegar los símbolos matemáticos estándar de LaTeX (apéndice B).

Finalmente, los únicos componentes dependientes de la plataforma son los que corresponden al subsistema de interfase gráfica. Por lo tanto es posible conectar el núcleo del editor a diferentes bibliotecas de interfases gráficas, cada una con diferente aspecto y funcionalidad.

7.2 Interfase gráfica

La figura 7.1 muestra la interfase gráfica usada en la versión estable de LyX. Utiliza la herramienta de GUI (Graphical user interface) Xforms[1] bajo el sistema de ventanas X window [SG92]. En la figura se muestra, en la parte superior izquierda, la barra de herramientas, con íconos de las estructuras matemáticas más conocidas. Abajo se encuentra la barra de menús de símbolos, con el menú de caracteres griegos abierto. A la derecha se encuentra una lista con las funciones matemáticas más comunes.

En la figura 7.2 se muestra la interfase gráfica usada en el prototipo actual de Mathed. La barra de herramientas está en la parte superior. En lugar de usar un menú de símbolos se usan tarjetas, como puede verse en la figura. Esta interfase utiliza la herramienta de GUI GTK+[2] y se planea utilizarla como interfase gráfica en la integración de Mathed a aplicaciones que usen esta herramienta de GUI y las bibliotecas del ambiente de escritorio GNOME[3]. El proyecto GNOME intenta ser la respuesta a la falta de un ambiente de escritorio amigable bajo la plataforma Unix y una base para el desarrollo de aplicaciones de oficina.

7.3 Integración a un sistema

El cliente inmediato para la integración de este sistema es el procesador de documentos LyX[4], en el cual la versión empírica de Mathed ha estado integrada desde principios de 1996 (ver la sección 1.1). En la figura 7.3 se muestra una imagen de LyX con la guía del usuario [Tea98] parcialmente mostrada en la pantalla. En la figura 7.4 se muestra el resultado impreso del mismo documento. En estas dos imágenes se ilustra el carácter *WYSIWYM* de LyX: contrariamente a lo que sucede con el concepto *WYSIWYG*, el usuario trabaja sobre el contenido de su texto, no en la presentación. De eso se encarga LaTeX, como un proceso en el *background* de LyX. Esta filosofía, que es la misma del editor de ecuaciones, puede resumirse de la siguiente manera: dar al usuario comodidad y flexiblidad para que se concentre en el contenido de su documento y sus ecuaciones, no en la presentación final de las mismas.

Nota: estas dos últimas imágenes fueron tomadas de la versión estable de LyX, la integración de la nueva versión de Mathed a la versión de desarrollo de LyX[5] aún no se concluye.

Otros sistemas a los que se puede integrar Mathed es a otros procesadores de palabras, a hojas de cálculo, a programas de graficación científica, etc.

[1] `http://bragg.phys.uwm.edu/xforms`
[2] `http://www.gtk.org`
[3] `http://www.gnome.org`
[4] `http://www.lyx.org`
[5] `http://www.devel.lyx.org`

Figura 7.1: Interfase gráfica en LyX.

Figura 7-2: Interfaz gráfica implementada en GTK

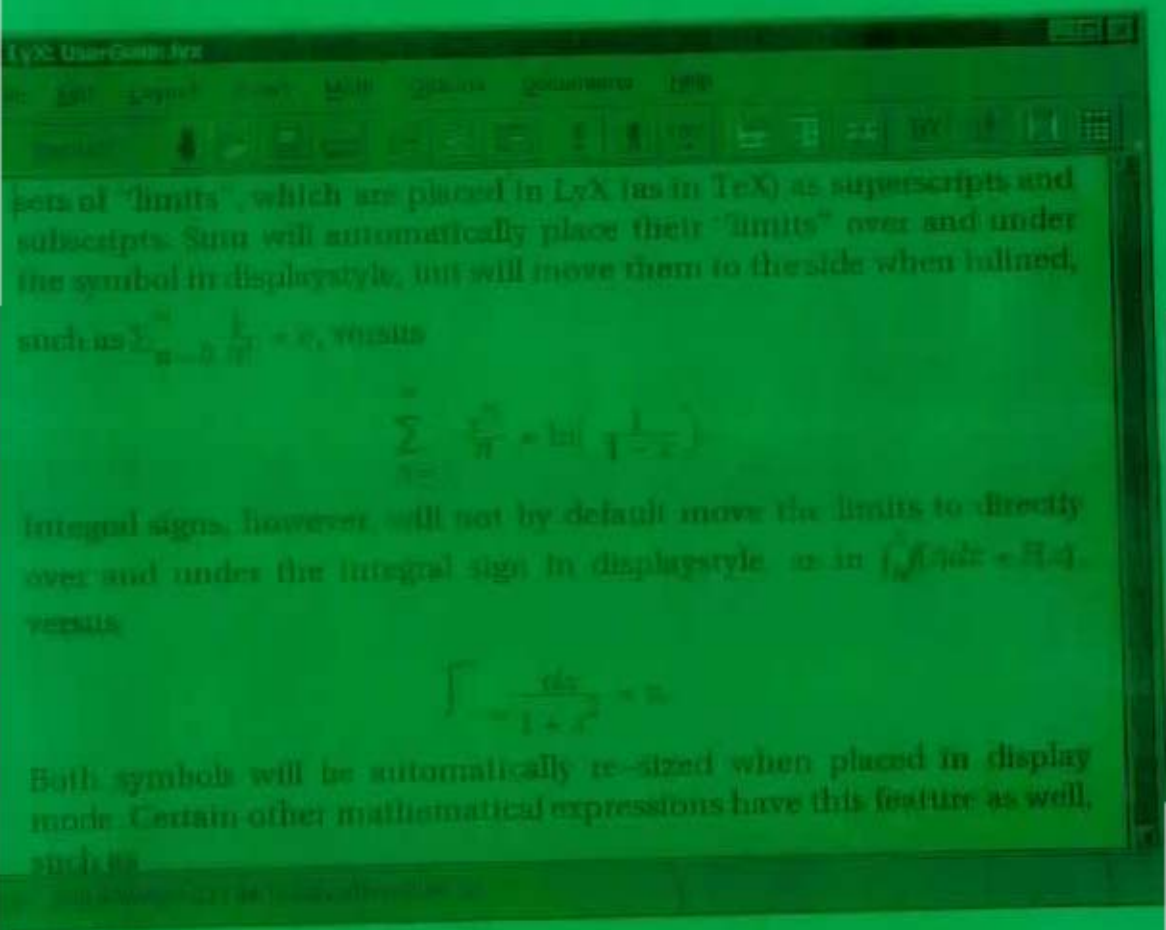

Figura 7.3: Pantalla de LyX

Sum will automatically place them "limits" over and under the symbol in the displayle, but will move them to the side when inlined, such as $\sum_{n=0}^{\infty} \frac{1}{n^2} = 6$ versus

$$\sum_{n=1}^{\infty} \frac{x^n}{n} = \ln\left(\frac{1}{1-x}\right).$$

Integral signs, however, will not by default move the limits to directly over and under the integral sign in displaystyle, as in $\int_a^b f(t)dt = F(x)$ versus

$$\int_{-\infty}^{\infty} \frac{dx}{1+x^2} = \pi$$

Both symbols will be automatically resized when placed in display mode. Certain other mathematical expressions have this feature as well, such as

$$\lim_{x \to \infty} f(x)$$

which will place the $x \to \infty$ underneath the "lim" in display mode. Left out in inlined mode, $\lim_{x \to \infty} f(x)$.

Figura 7-4: Versión impresa del mismo documento.

Capítulo 8

Conclusión y perspectivas

Durante el desarrollo y mantenimiento del Mathed empírico se tuvieron los siguientes problemas:

1. En un principio no se conocían todos los requisitos importantes. Se diseñó el primer prototipo con base en la experiencia de usar el editor de ecuaciones del Scientific Word (ver la sección 2.4.2). Al ir conociendo los requerimientos de los usuarios fue necesario introducir cambios no contemplados desde el principio.

2. Después de acumular muchos cambios sobre la versión original, al menos en un par de ocasiones fue necesario modificar la arquitectura del sistema, lo que llevó a reescribir grandes porciones de código fuente e introducir errores.

3. Durante estos 3 años, algunas de las estructuras de clases han tenido que ser refinadas para mejorar su funcionalidad.

En contraste, el editor matemático desarrollado en este trabajo satisface los requisitos listados en la sección 4.1, simplemente porque el proceso de desarrollo fue dirigido por estos mismos requisitos, en la forma de casos de uso y requerimientos especiales. Por lo mismo, desde la fase de elaboración se esbozó una arquitectura apropiada y resistente a cambios. Por otra parte, el diseño de las estructuras de clases fue sencillo gracias a la aplicación de los patrones de diseño adecuados. La mejor manera de escogerlos fue estudiando su semejanza con las estructuras de clases que se desarrollaron durante los 3 años de mantenimiento del Mathed empírico.

El Mathed empírico estuvo listo para su uso en menos de 3 meses pero llevó 3 años refinarlo al punto de cumplir con los requisitos exigidos en esta tesis. Si se usara el Proceso Unificado desde el principio, probablemente tomaría más tiempo desarrollar una versión lista para usarse; tan solo el encontrar la lista de requisitos apropiada, tomaría unas semanas. Pero la calidad esperada del producto final se alcanzaría en unos cuantos meses.

El proceso empírico de desarrollo de software tiene en común con el Proceso Unificado el ser iterativo e incremental, pero en una forma desorganizada e indisciplinada. El uso de una metodología orientada a objetos robusta permite hacer este proceso de desarrollo mucho más eficiente.

En el futuro se añadirán las siguientes mejoras al programa:

- Sistema de *corrección de errores* por medio de dar marcha atrás (*undo*) a modificaciones hechas por el usuario.
- Actualmente el editor puede leer y escribir en el modo matemático de LaTeX. Es conveniente añadir la capacidad de leer y generar MathML, ya que éste se perfila como el nuevo estándar en lenguajes de comunicación de expresiones matemáticas.
- Adopción de CORBA[1] para la *integración* de Mathed a otras aplicaciones que soporten este protocolo, como las que pertenecen al proyecto GNOME.

La nueva versión de Mathed será más robusta y flexible que la *versión empírica* y será usada en un número mayor de aplicaciones.

[1] CORBA permite a diferentes aplicaciones interaccionar entre sí por medio de la distribución de objetos. Para más información, visitar `http://www.corba.org`.

Apéndice A

Glosario

Términología usada en este trabajo

Contents markup En notación MathML se usa para describir objetos y funciones matemáticos. Se relaciona con el contenido y significado de una expresión más que con su notación.

Contexto gráfico Término usado en herramientas gráficas para indicar que contiene la información necesaria sobre dónde y cómo se realizarán las operaciones de graficación.

Ecuación Expresión matemática.

Fuente Anglicismo de tipo.

Markup Es una indicación de un componente lógico o del formato del texto o cualquier información que pueda ser interpretada automáticamente. Se usa en sistemas de preparación de documentos.

Presentation markup En la especificación MathML se usa para describir notación matemática. Esta relacionado con la presentación más que con el contenido.

WYSIWYG *What You See Is What You Get.* Lo que se ve en la interfaz interactiva es lo mismo que lo que se verá en la copia impresa del documento.

WYSIWYM *What You See Is What You Mean.* Lo que se ve en la interfaz interactiva no es lo mismo que lo que se verá en la copia impresa del documento, pero da información suficiente sobre el contenido del documento.

Apéndice B

Símbolos matemáticos comunes

α	θ	o	τ
β	ϑ	π	υ
γ	ι	ϖ	ϕ
δ	κ	ρ	φ
ϵ	λ	ϱ	χ
ε	μ	σ	ψ
ζ	ν	ς	ω
η	ξ		

Γ	Λ	Σ	Ψ
Δ	Ξ	Υ	Ω
Θ	Π	Φ	

Tabla B.1: Caracteres griegos

$\pm$	$\cap$	$\diamond$	$\oplus$
$\mp$	$\cup$	$\bigtriangleup$	$\ominus$
$\times$	$\uplus$	$\bigtriangledown$	$\otimes$
$\div$	$\sqcap$	$\triangleleft$	$\oslash$
$*$	$\sqcup$	$\triangleright$	$\odot$
$\star$	$\vee$	$\bigcirc$	$\circ$
$\wedge$	$\dagger$	$\bullet$	$\setminus$
$\ddagger$	$\cdot$	$\wr$	$\amalg$

Tabla B.2: Operadores binarios

$\leq$	$\geq$	$\equiv$	$\models$
$\prec$	$\succ$	$\sim$	$\perp$
$\preceq$	$\succeq$	$\simeq$	$\mid$
$\ll$	$\gg$	$\asymp$	$\parallel$
$\subset$	$\supset$	$\approx$	$\bowtie$
$\subseteq$	$\supseteq$	$\cong$	$\neq$
$\sqsubseteq$	$\sqsupseteq$	$\doteq$	$\frown$
$\in$	$\ni$	$\propto$	$=$
$\vdash$	$\dashv$	$<$	$>$
$:$	$\smile$		

Tabla B.3: Relaciones binarias

$\leftarrow$	$\longleftarrow$	$\uparrow$
$\Leftarrow$	$\Longleftarrow$	$\Uparrow$
$\rightarrow$	$\longrightarrow$	$\downarrow$
$\Rightarrow$	$\Longrightarrow$	$\Downarrow$
$\leftrightarrow$	$\longleftrightarrow$	$\updownarrow$
$\Leftrightarrow$	$\Longleftrightarrow$	$\Updownarrow$
$\mapsto$	$\longmapsto$	$\nearrow$
$\hookleftarrow$	$\hookrightarrow$	$\searrow$
$\leftharpoonup$	$\rightharpoonup$	$\swarrow$
$\leftharpoondown$	$\rightharpoondown$	$\nwarrow$
$\rightleftharpoons$		

Tabla B.4: Flechas

$\ldots$	$\cdots$	$\vdots$	$\ddots$
$\aleph$	$\prime$	$\forall$	∞
$\hbar$	$\emptyset$	$\exists$	$.$
$\imath$	∇	$\neg$	$\mid$
$\jmath$	$\surd$	$\flat$	$\triangle$
ℓ	$\top$	$\natural$	$\clubsuit$
$\wp$	$\bot$	$\sharp$	$\diamondsuit$
$\Re$	$\|$	$\backslash$	$\heartsuit$
$\Im$	$\angle$	∂	$\spadesuit$

Tabla B.5: Símbolos miscelánea

Apéndice C

Licencia Pública General[1]

Version 2, junio de 1991.

NOTA IMPORTANTE: Esta es una traducción de cortesía únicamente, que cubre la obligación de hacer entender de la manera mas clara posible a quién se le entrega un programa GPL sus derechos y obligaciónes sobre el uso y la disposición del mismo. Esta NO es una traducción oficial de la "General Public License" de la Free Software Foundation[2].

Esta traducción trata de mantener el mismo espíritu de la licencia original en inglés, pero no ha sido revisada minuciosamente por abogados, y la Free Software Foundation no la avala ni la aprueba como substituto legal de la auténtica versión en Inglés. en caso de cualquier controversia, la licencia GPL original en inglés y su interpretación oficial será la que deberá aplicarse.

Preámbulo

Para la mayoría de los software se idean licencias que le quitan a usted la libertad de compartirlos y cambiarlos. En cambio la licencia Pública General GNU tiene como objetivo garantizarle a usted la libertad de compartir y cambiar software libre, asegurarse de que el software es libre para todos sus usuarios. Esta licencia Pública General es aplicable a la mayoría del software de la Free Software Foundation así como a cualquier otro programa cuyos autores se comprometan a usarlo. (Algunos de los programas

[1]Traducción del inglés, Julio de 1996. Prof. Carmen Arizmendi, Programa de Formación de Traductores El Colegio de México, A.C. Camino al Ajusco #20, México 10740 DF, MEXICO, `http://www.colmex.mx`

[2]`http://www.gnu.org`

de la Free Software Foundation están cubiertos por la Licencia Pública General para Bibliotecas GNU). Usted puede aplicarlo a sus programas también.

Cuando hablamos de software libre, nos referimos a la libertad y no al precio. Nuestras Licencias Públicas Generales están pensadas para asegurar que usted tiene la libertad de distribuir copias del software libre (y cobrar por el servicio si así lo desea) que pueda usted recibir el código fuente de los programas o bien que lo obtenga si lo quiere usted, que pueda usted cambiar el software o bien usar partes del mismo en nuevos programas libres y que sepa usted que puede hacer todas estas cosas.

Para proteger sus derechos, es necesario que impongamos restricciones que impidan que le nieguen a usted dichos derechos o bien que le pidan que renuncie usted a los mismos. Estas restricciones se traducen en ciertas responsabilidades que debe usted asumir si distribuye usted copias del software o si lo modifica usted.

Por ejemplo, si distribuye usted copias de un programa de esa naturaleza, ya se gratis o por una cuota, deberá usted conceder a los receptores todos los derechos que usted tiene. Deberá asegurarse, asimismo, de que reciban o puedan recibir el código fuente. Y deberá mostrarles estas estipulaciones a fin de que conozcan sus derechos.

Protegemos sus derechos mediante dos medidas: (1) con el copyright del software y (2) proporcionánlole esta licencia que le permite legalmente copiar, distribuir y/o modificar el software.

Asimismo, como una manera de protección de cada autor y de la nuestra propia, queremos estar seguros de que todos entienden que no existe garantía para este software libre. Si el software es modificado por otra persona y lo hace circular, queremos que los que lo reciben sepan que lo que obtienen no es el original, de modo que los problemas que hayan introducido otras personas no repercutan en el prestigio de los autores.

Por último, cualquier programa libre se ve amenazado constantemente por las patentes de software. Deseamos evitar el peligro de que los redistribuidores de un programa libre obtengan individualmente licencias patentadas que los convierta en realidad en los propietarios del programa. Para impedirlo, hemos dejado muy claro que cualquier patente debe autorizarse para el uso libre de todo el mundo o no autorizarse.

A continuación detallamos los términos y las condiciones precisas para copiar, distribuir o modificar.

LICENCIA PUBLICA GENERAL GNU. TERMINOS Y CONDICIONES PARA COPIAR, DISTRIBUIR O MODIFICAR.

0. Esta Licencia es aplicable a cualquier programa u otra obra que contenga una leyenda, colocada ahí por el dueño del copyright, que especifique que puede ser

distribuido de acuerdo con los términos de esta Licencia Pública General. El "Programa", más abajo mencionado, se refiere a cualquer programa y obra de esa naturaleza, y una "obra basada en el Programa" significa, ya sea el Programa o cualquier obra derivada bajo la ley copyright: es decir, una obra que contenga el Programa o una porción del mismo, ya sea una copia exacta o que contenga modificaciones y/o una traducción a otra lengua, (Más adelante, se incluye sin limitación alguna la traducción en el término "modificación"), Todo aquel que recibe la licencia será denominado "usted".
Otras actividades que no sean la de copiar, distribuir y modificar no están cubiertas por esta Licencia, quedan fuera de su alcance. No está restringido el acto de correr el Programa, y el producto del Programa está cubierto únicamente si su contenido constituye una obra basada en el Programa (independientemente de que se haya hecho corriendo el Programa). Si ello es cierto o no depende de lo que hace el Programa.

1. Usted puede copiar y distribuir copias exactas del código fuente del Programa tal y como lo recibe, en cualquier medio, con tal que publique usted de manera conspicua y apropiada en cada copia una leyenda apropiada de copyright y de no garantía; de que mantenga usted intactas todas las advertencias referentes a esta Licencia y al hecho de la ausencia de cualquier garantía; y de que proporcione usted a cualquier otro receptor del Programa una copia de esta Licencia junto con el Programa.
   Usted puede cobrar una cuota por el acto físico de transferir una copia, y puede usted, como una opción, ofrecer protección de garantía a cambio de el dinero que recibe.

2. Puede usted modificar su copia o copias del Programa o de una parte del mismo formando así una obra basada en el Programa y copiar y distribuir dichas modificaciones o bien trabajar de acuerdo con los términos especificados en la Sección 1, a condición de que cumpla usted asimismo con estas condiciones:

   (a) Debe usted hacer que los archivos modificados lleven una advertencia prominente en que se asiente que cambió usted los archivos e incluya la fecha en que se hizo cualquier cambio.

   (b) Debe usted hacer que cualquier obra que distribuya usted o que publique, que, ya sea totalmente o en parte, contenga o se derive del Programa o de alguna de sus partes, sea también una licencia que pueda ser utilizada total o parcialmente por terceras personas sujetándose a los términos de esta Licencia.

   (c) Si el programa modificado normalmente lee las órdenes interactivamente al ser ejecutado, debe usted hacer que, al correr el programa para dicho

uso interactivo de la manera más corriente, desde el inicio imprima usted o despliegue un anuncio que incluya una advertencia apropiada de copyright y una advertencia de no garantía (o bien, especificando que usted proporciona la garantía) y que los usuarios pueden redistribuir el programa bajo estas condiciones, y diciéndole al usuario cómo puede ver una copia de esta Licencia. (Excepción: si el Programa original en sí, es interactivo pero normalmente no imprime un anuncio de esa naturaleza, su obra basada en el Programa no tendrá que cumplir con el requisito de la impresión de un anuncio).

Estos requisitos se aplican a la obra modificada en su conjunto. Si algunas secciones identificables de dicha obra no se derivan del Programa, y pueden ser consideradas como obras razonablemente independientes y aparte en sí mismas, entonces esta Licencia y los términos de la misma no se aplicarán a esas secciones cuando las distribuya usted como obras aparte, Empero, cuando distribuya usted las mismas secciones como parte de un todo, que es una obra basada en el Programa, la distribución de ese todo debera conformarse a los términos de esta Licencia, cuyos permisos concedidos a otros detentores de la licencia se extienden al conjunto total y, por consiguiente, a cualquier parte sin importar quien la haya escrito.
Por ende, no es la intención de esta sección reclamar derechos o impugnar sus derechos sobre obra escrita completamente por usted; más bien, lo que intenta es ejercer el *derecho de control sobre la distribución de obras derivadas o colectivas* que se basan en el Programa.
Es más, el hecho único de agregar otra obra que no está basada en el Programa *(o una obra basada en el Programa) dentro de un medio de almacenamiento* o de distribución no implica que esa otra obra quede dentro del alcance de esta Licencia.

3. Puede usted copiar y distribuir el Programa (o una obra basada en él, según lo especificado en la Sección 2) en código de salida o en forma ejecutable de acuerdo con los términos expresados en las Secciones 1 y 2 a condición de que satisfaga usted una de las siguientes condiciones:

   (a) aAcompañarlo del código fuente completo correspondiente en forma legible por máquinas, el cual deberá ser distribuido bajo los términos de las Secciones 1 y 2 arriba mencionadas utilizando un medio usado habitualmente en el intercambio de software; o,

   (b) Acompañarlo de un ofrecimiento escrito, válido por lo menos por tres años, de proporcionar a cualquier tercero interesado mediante el cobro exclusivamente del gasto hecho al proceder físicamente a la colocación en un medio y

distribución de la fuente, una copia completa, en forma legible por máquinas, del correspondiente código fuente, para su distribución bajo los términos especificados en las Secciones 1 y 2 arriba mencionadas, en un medio utilizado habitualmente para el intercambio de software; o,

(c) Acompañarlo de la información que recibió usted con respecto al ofrecimiento de distribución del correspondiente código fuente. (Esta alternativa es permitida únicamente en la distribución no comercial y solamente si recibió usted el programa en código de salida o en forma ejecutable junto con dicho ofrecimiento, de acuerdo con la Subsección b arriba mencionada).

El código fuente de una obra significa la forma preferida de la obra para hacerle modificaciones. Para una obra ejecutable, el código fuente completo significa todo el código fuente para todos los módulos que contiene, más cualquier archivo de definición asociado a la interfase, más los documentos utilizados para controlar la compilación y la instalación del ejecutable. Sin embargo, como excepción especial, el código fuente distribuido no necesariamente debe incluir algo de lo que normalmente es distribuido (ya sea en forma fuente o en forma binaria) con los principales componentes (compilador, kernel, etc.) del sistema operativo en el que corre el ejecutable, a menos que ese componente sea parte del ejecutable. Si la distribución del ejecutable o del código de salida se hace ofreciendo el acceso para copiar desde un lugar designado, ofreciendo en seguida acceso equivalente para copiar el código fuente desde ese mismo lugar cuenta como distribución del código fuente, aunque terceras personas no son obligadas a copiar la fuente junto con el código de salida.

4. No puede usted copiar, modificar, sublicenciar o distribuir el Programa si no es siguiendo las disposiciones que rigen expresamente en esta Licencia. Cualquier intento de copiar, modificar, sublicenciar o distribuir el Programa de otra manera será nulo y automáticamente pondrá fin a sus derechos bajo esta Licencia. Sin embargo, los que hayan recibido copias, o derechos de usted bajo esta Licencia no perderán sus licencias mientras sigan cumpliendo totalmente las obligaciones aquí impuestas.

5. No se le exige que acepte usted esta Licencia, ya que no la ha firmado. Sin embargo, ninguna otra cosa le otorga permiso para modificar o distribuir el Programa o sus obras derivadas. Estas acciones están prohibidas por la ley si no acepta usted esta Licencia. Por consiguiente, al modificar o distribuir el Programa (o cualquier obra basada en el Programa) está usted indicando que acepta esta Licencia para hacerlo, así como todos sus términos y condiciones para el copiado, la distribución o modificación del Programa o de obras basadas en él.

6. Cada vez que redistribuye usted el Programa (o cualquier obra basada en el Programa), el receptor recibe automáticamente una licencia para copiar, distribuir o modificar el Programa de acuerdo con estos términos y condiciones que el otorgante original de la licencia concede. No puede usted imponer ninguna otra restricción al ejercicio de los derechos otorgados aquí al receptor. No es usted responsable de hacer cumplir esta Licencia a terceros.

7. Si como consecuencia de una sentencia de un tribunal o de un alegato de violación de patentes o de cualquier otro motivo (no limitado a cuestiones patentarias) se le imponen a usted condiciones (ya sea por orden de un tribunal, por acuerdo o de otro modo) que contradicen las condiciones de esta Licencia no por ello queda usted excusado de las condiciones de esta Licencia. Si no puede usted distribuir sin cumplir simultáneamente con sus obligaciones impuestas por esta Licencia, así como con otras obligaciones pertinentes, entonces, consecuentemente, no puede usted distribuir el Programa. Por ejemplo, si una licencia patentaria no permitiera la redistribución del Programa, libre de regalías, a aquellos que recibieran copias directa o indirectamente a través de usted, en ese caso la única forma de complir con esa licencia y con esta Licencia sería abstenerse por completo de distribuir el Programa. Si alguna porción de esta sección, por alguna circunstancia particular, es juzgada nula o imposible de ejecutar, el resto de la sección es aplicable y la sección como un todo también está pensada como aplicable en otras circunstancias. Esta sección no se propone inducirle a usted a infringir patentes u otros reclamos de derechos de propiedad o a impugnar la validez de cualquiera de esas reclamaciones; esta sección no tiene más objetivo que el de proteger la integridad del sistema libre de distribución de software, el cual se lleva a cabo mediante reglas de licencias públicas. Muchos son los que han contribuido generosamente a la amplia extensión de software distribuido a través de ese sistema, confiando en la aplicación consistente de dicho sistema; es el autor/donante quien habrá de decidir si él o ella desea distribuir software a través de otro sistema diferente y la persona que obtiene la licencia no puede imponer esa dicisión. Esta sección intenta dejar completamente claro lo que es considerado como una consecuencia del resto de esta Licencia.

8. Si la distribución y/o uso del Programa se halla restringida en algunos países, ya sea por patentes o por interfases con copyright, el detentor original del copyright que pone el Programa bajo esta Licencia puede añadir una limitación explícita de distribución geográfica excluyendo a dichos países, de modo que se permita la distribución sólo en y entre los países que no hayan sido excluídos. En tales casos, esta Licencia incorpora la limitación como si se hallara escrita en el cuerpo de esta Licencia.

9. La Free Software Foundation puede publicar versiones revisadas o nuevas de la Licencia Pública General de vez en cuando. Dichas versiones nuevas serán esencialmente similares a la presente versión, pero pueden variar en los detalles al abordar problemas e intereses nuevos. Cada versión recibe un número distintivo de versión, Si el Programa especifica un número de versión de esta Licencia que se le aplica y a "cualquier versión posterior" tiene usted la opción de adherirse a los términos y condiciones ya sea de esa versión o de cualquier otra versión posterior que publique la Free Software Foundation.
   Si el Programa no especifica un número de versión de esta Licencia, podrá usted escoger cualquier versión que haya publicado la Free Software Foundation.

10. Si desea usted incorporar partes del Programa en otros programas libres cuya distribución se somete a condiciones distintas, escriba usted al autor pidiendo su autorización. Para el software cuyo copyright pertenece a la Free Software Foundation, escriba usted a Free Software Foundation; a veces hacemos excepciones. Nuestra decisión se guiará por las dos metas que tenemos la de conservar el estatus libre de todos los derivados de nuestro software libre y la de promover el compartimiento y reutilización del software en general.

NO GARANTIAS

11. COMO EL PROGRAMA TIENE UNA LICENCIA LIBRE DE GASTOS, NO HAY GARANTIA PARA EL PROGRAMA, HASTA DONDE LO PERMITE LA LEY APLICABLE. CON EXCEPCION DE LOS CASOS EN LOS QUE POR ESCRITO SE DECLARE LO CONTRARIO. LOS DETENTORES DEL COPYRIGHT Y/O OTROS PARTICIPES PROPORCIONAN EL PROGRAMA "TAL CUAL" SIN GARANTIA DE NINGUNA CLASE, YA SEA EXPLICITA O IMPLICITA, INCLUYENDO, PERO NO LIMITADO A, LAS GARANTIAS DE COMERCIABILIDAD IMPLICITAS Y LA ADECUACION A UN PROPOSITO PARTICUALAR. ES USTED QUIEN TIENE QUE CORRER TODO EL RIESGO EN LA CALIDAD Y EL RENDIMIENTO DEL PROGRAMA. EN EL CASO DE QUE EL PROGRAMA RESULTE DEFECTUOSO, ASUMIRA USTED EL COSTO DEL SERVICIO, REPARACION O CORRECCCION.

12. EN NINGUN CASO, A MENOS QUE ASI LO REQUIERA LA LEY APLICABLE O SE HAYA ACORDADO POR ESCRITO, UN DETENTOR DEL COPYRIGHT O CUALQUIER OTRO QUE PUEDA MODIFICAR Y/O REDISTRIBUIR EL PROGRMA TAL Y COMO ESTA PERMITIDO, SERA RESPONSABLE ANTE USTED POR DAÑOS, INCLUSO POR DAÑOS GENERALES, ESPECIALES, INCIDENTALES O CONSECUENTES SURGIDOS POR

EL USO O LA INCAPACIDAD DE USO DEL PROGRAMA (INCLUYENDOSE, PERO NO LIMITANDOSE A, LA PERDIDA DE DATOS O A DATOS QUE RESULTAN INEXACTOS O A PERDIDAS SUFRIDAS POR USTED O POR TERCEROS O BIEN PORQUE EL PROGRAMA NO PUEDA OPERAR CON NINGUN OTRO PROGRAMA), AUN CUANDO EL POSEEDOR DEL COPYRIGHT O LA OTRA PARTE HAYAN SIDO ADVERTIDOS DE LA POSIBILIDAD DE DICHOS DAÑOS.

FIN DE LOS TERMINOS Y CONDICIONES.

Bibliografía

[AIS77] Christopher Alexander, Sara Ishikawa, and Murray Silverstein. *A Pattern Language.* Oxford University Press, 1977.

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques and Tools.* Addison-Wesley, 1986.

[Aus98] Matthew H. Austern. *Generic Programming and the STL.* Professional Computing. Addison-Wesley, 1998.

[Boo94] Grady Booch. *Object-Oriented Analysis and Design with applications.* Benjamin/Cummings, 2nd edition, 1994.

[BRJ99] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide.* Addison-Wesley, 1999.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns.* Professional Computing. Addison-Wesley, 1994.

[GMS94] Michael Gossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion.* Addison-Wesley, 1994.

[J+92] Ivar Jacobson et al. *Object-Oriented Software Engineering.* Addison-Wesley, 1992.

[JBR99] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process.* Addison-Wesley, 1999.

[Knu94] Donald Knuth. *The TeX book.* Addison-Wesley, 1994.

[Lam94] Leslie Lamport. *LaTeX: A Document Preparation System.* Addison-Wesley, 2nd edition, 1994.

[MS96] David R. Musser and Atul Saini. *STL Tutorial and Reference Guide.* Addison-Wesley, 1996.

[R+91] James Rumbaugh et al. *Object-Oriented Modeling and Design.* Prentice Hall, 1991.

[Ray97] Eric S. Raymond. The cathedral and the bazaar. http://www.kde.org/food/cathedral/cathedral-paper.html, 1997.

[RJB99] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual.* Addison-Wesley, 1999.

[SG92] Robert W. Scheifler and James Gettys. *X Window System.* Digital Press, 3rd edition, 1992.

[Tea98] The LyX Team. *The LyX User's Guide,* 1998.