CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL

CINVESTAV ZACATENCO

INGENIERÍA ELÉCTRICA

COMPUTACIÓN

Diseño e Instrumentación de un entorno visual para **e-Edita**

Tesis que presenta

Ing. Israel Méndez Martínez

Para Obtener el Grado de

Maestro en Ciencias

En la Especialidad de

Computación

Directores de Tesis:
Dr. Feliú D. Sagols Troncoso
Dr. Guillermo B. Morales Luna

México, Distrito Federal Marzo 2004

Índice general

Introducción **1**

1. Marco formal **3**
1.1. Preliminares . . . 3
1.2. Lenguajes ontológicos y *XML* . . . 3
1.2.1. Lenguajes ontológicos . . . 3
1.2.2. Extensible Markup Language (*XML*) . . . 4
1.3. Lenguajes ontológicos en la educación virtual . . . 5
1.4. **e-Edita** y su relación con las ontologías . . . 6
1.5. El lenguaje de **e-Edita** y su enlace con el entorno visual . . . 6
1.6. Consideraciones de diseño . . . 6

2. El entorno de e-Edita **9**
2.1. Características de **e-Edita** . . . 9
2.2. **e-Edita** como un entorno de especificación . . . 10
2.3. Diseño de **e-Edita** desde una perspectiva global . . . 11
2.4. Estructura del lenguaje de especificación de **e-Edita** . . . 11
2.5. Beneficios estructurales del lenguaje . . . 12
2.6. Tipos de construcciones de **e-Edita** . . . 12
2.6.1. Directivas . . . 12
2.6.2. Especificadores de reactivos . . . 12
2.7. Aplicaciones que se pueden desarrollar con **e-Edita** . . . 13
2.8. Los dos lenguajes de especificación de **e-Edita** . . . 13

3. e-Edita en la educación virtual **15**
3.1. Análisis de requerimientos desde la perspectiva de la educación virtual . . . 15
3.2. Lenguaje de Páginas Interactivas . . . 15
3.2.1. Directiva ADDRESS (Dirección) . . . 18
3.2.2. Directiva END (Fin) . . . 18
3.2.3. Directiva HEADING (Sub-título) . . . 18
3.2.4. Directiva IMAGE (Imagen) . . . 18
3.2.5. Directiva MATHIMG (Tipografía especializada) . . . 19
3.2.6. Directiva NOFOLLOW (No siguiente) . . . 19
3.2.7. Directiva NOGRADE (No calificar) . . . 20
3.2.8. Directiva SOUND (Sonido) . . . 20
3.2.9. Directiva TITLE (Título) . . . 20
3.3. Especificadores de reactivos . . . 20
3.3.1. Especificador de preguntas abiertas con respuesta específica . . . 21
3.3.2. Especificador de preguntas de opción múltiple tipo *ComboBox* . . . 21
3.3.3. Especificador de preguntas de opción múltiple tipo botón de radio . . . 22
3.3.4. Especificador de preguntas para relacionar columnas . . . 23
3.4. Lenguaje de libros interactivos . . . 23
3.4.1. Directiva CHAPTER (Capítulo) . . . 24

3.4.2. Directiva CONTENT (Contenido) 24
3.5. Operación de **e-Edita** 25
3.5.1. Traducción de páginas 25
3.5.2. Traducción de libros 25
3.5.3. Comando `generatodo` 26

4. Diseño del editor visual para e-Edita **27**
4.1. Evolución del diseño de editor visual 27
4.2. Diseño conceptual del editor visual 27
4.2.1. Creación de borradores 27
4.2.2. Diseño de la ventana principal 28
4.3. Diseño de las interfaces para los constructores de **e-Edita** 28
4.3.1. Directiva TITLE 28
4.3.2. Directiva HEADING 29
4.3.3. Directiva END 29
4.3.4. Directiva CHAPTER 29
4.3.5. Directiva CONTENT 29
4.3.6. Directivas IMAGE y SOUND 30
4.3.7. Directiva MATHIMG 30
4.3.8. Especificador de preguntas abiertas con respuesta específica 32
4.3.9. Especificador de preguntas de opción múltiple 32
4.3.10. Especificador de preguntas en modalidad de relacionar columnas 32
4.4. Diseño de íconos de **e-Edita** 32
4.5. Módulos del sistema 33
4.5.1. El editor visual 34
4.5.2. Interfaz con el S.O. Unix 35
4.5.3. Acceso desde Internet 35
4.6. Trabajo a futuro 36

5. Instrumentación del diseño del editor visual de e-Edita **37**
5.1. Instrumentación en Java 37
5.1.1. Facilidades de Java utilizadas 37
5.1.2. Organización de los archivos y directorios 38
5.1.3. Comunicación entre programas 39
5.1.4. Programas *CGI* 40
5.1.5. Íconos utilizados 40
5.1.6. Instrumentación del editor visual 41
5.1.7. Instrumentación de las interfaces para el usuario 44
5.1.8. Edición de los constructores 48
5.2. Funcionamiento de **e-Edita** 50
5.2.1. El método *getFields* 50

6. Extensión de las capacidades del editor visual **51**
6.1. Planteamiento de construcciones nuevas en **e-Edita** 51
6.2. Incorporación de directivas 51
6.2.1. Modificaciones al traductor de **e-Edita** para incorporar directivas 53
6.3. Incorporación de especificadores de reactivos 54
6.3.1. Creación del nuevo tipo de reactivo 54
6.3.2. Traducción de la especificación del reactivo a código *HTML* 55
6.4. Modificaciones al programa `EkitCore.java` 56
6.5. Adaptaciones al Applet 58
6.6. Modificaciones al programa `EkitEdita.java` 58

Conclusiones **63**

A. Componentes de `EkitEdita.java` 65

B. Instalación 67
B.1. Instalación en Windows 67
B.2. Instalación en Linux 67
B.2.1. Instalación de JRE o SDK 67
B.2.2. Configuración del navegador 68
B.3. Instalación del Editor 68

C. Compilación del editor visual 69

D. Resultado en código *HTML* de una especificación 71
D.1. Página electrónica 71
D.2. Libro electrónico 72

E. Organización del disco compacto 75

Bibliografía 77

Índice de cuadros

3.1. Modificadores del comando `pagina` 26
3.2. Programas *CGI* encargados de atender las peticiones del editor 26

4.1. Íconos utilizados en el editor visual de **e-Edita** 33
4.2. Módulos del sistema 34

5.1. Clases de Java utilizadas 38
5.2. Programas *CGI* encargados de atender las peticiones del editor visual 40
5.3. Características de los íconos de **e-Edita** 41
5.4. Extensiones al programa `EkitCore.java` 41
5.5. Pasos para construir la interfaz de la directiva MATHIMG 45

6.1. Variables del programa principal usadas en *switch(token)* 53
6.2. Atributos existentes 55

Índice de figuras

3.1. Página creada con **e-Edita** 16
3.2. Ejemplo de directiva ADDRESS 18
3.3. Ejemplo de directiva HEADING 18
3.4. Ejemplo de directiva IMAGE 19
3.5. Ejemplo de directiva MATHIMG 19
3.6. Ejemplo de directiva NOFOLLOW 20
3.7. Ejemplo de directiva NOGRADE 20
3.8. Ejemplo de directiva SOUND 20
3.9. Ejemplo de directiva TITLE 21
3.10. Ejemplo del especificador de pregunta abierta 21
3.11. Ejemplo del especificador en formato *ComboBox* 22
3.12. Ejemplo del especificador en formato botón de radio vertical 23
3.13. Ejemplo del especificador en formato botón de radio horizontal 23
3.14. Ejemplo de relacionar columnas 24
3.15. Ejemplo de un libro electrónico creado con **e-Edita** 24
3.16. Ejemplo de directiva CHAPTER y CONTENT 25

4.1. Ventana principal del editor visual para **e-Edita** 28
4.2. Interfaz para la directiva TITLE 28
4.3. Interfaz para la directiva HEADING 29
4.4. Interfaz para la directiva END 29
4.5. Interfaz para la directiva CHAPTER 29
4.6. Interfaz para la directiva CONTENT 30
4.7. Interfaces para las directivas IMAGE y SOUND 30
4.8. Editor del contenido la directiva 31
4.9. Interfaz para el ajuste de parámetros 31
4.10. Interfaz para la selección de color 31
4.11. Interfaz básica para preguntas abiertas con respuesta específica 32
4.12. Interfaz avanzada para preguntas abiertas con respuesta específica 32
4.13. Interfaz para preguntas de opción múltiple 33
4.14. Interfaz para crear preguntas de relacionar columnas 33
4.15. Página de ingreso al editor visual de **e-Edita** 35
4.16. Certificado de seguridad de **e-Edita** 36

5.1. Estructura de directorios del editor visual de **e-Edita** 39

Introducción

La educación virtual ha cobrado fuerza debido a las enormes ventajas que ofrece, por ejemplo, los alumnos pueden tomar clase sin que el maestro esté presente, es posible atender a más alumnos que en los modelos presenciales y se favorece la interacción con personas dispersas en un territorio muy amplio. En este contexto la Internet se ha convertido en el canal ideal porque ofrece una gran cantidad de medios de comunicación: texto, sonido, video, etc. Sin embargo, su uso requiere conocimientos especializados en lenguajes como *HTML* y herramientas de diseño de páginas *Web* que, en general, resultan poco accesibles para profesores y especialistas en la educación.

Este problema ha motivado el desarrollo del trabajo que aquí se presenta, el objetivo es construir herramientas de software que faciliten la creación de materiales educativos. Esto se hará extendiendo al software **e-Edita** [17], que es un conjunto de herramientas de especificación de páginas *Web*, con un editor visual accesible vía Internet.

Desde un punto de vista estrictamente técnico, el diseñar una interfaz adecuada para **e-Edita** fue el problema más desafiante. En principio tuvimos la idea de desarrollar un ambiente visual provisto de una gran cantidad de facilidades gráficas, basado en la metodología de los lenguajes visuales; sin embargo, al querer concretar este diseño nos dimos cuenta que mientras más simple e intuitiva se volvía la interfaz, más tardado era usarla. Para que el lector tenga idea de lo que estaba sucediendo, podemos tomar como ejemplo la diferencia entre *emacs* y MS-Word. En el primer editor las cosas son simples pero el usuario tiene que tener ciertos conocimientos de antemano; en MS-Word, las cosas también son simples pero aún operaciones demasiado simples toman mucho tiempo. A nosotros nos interesó desarrollar una herramienta simple que en el corto plazo ofreciera muchas ayudas al usuario neófito; pero que en el largo plazo favoreciera la productividad. En otras palabras queríamos algo que en el corto plazo se pareciera a MS-Word; pero que en el largo plazo fuera más semejante a *emacs*. La clave de todo esto está en construir una interfaz que le permita al usuario ver directamente lo que genera a un cierto "bajo nivel", de manera que en el largo plazo él pueda usar ese "bajo nivel" de manera directa.

El problema sobre el diseño de la interfaz fue difícil de plantear y tuvimos que hacer un estudio basándonos en una gran cantidad de interfaces disponibles. La solución final fue curiosamente simple: construir un editor parecido a los que se usan con LaTeX como *Scientific WorkPlace* o *WinEdt* que son sistemas provistos de íconos y elementos gráficos que directamente sustituyen el código que generan en LaTeX. Así para el usuario que escribe ocasionalmente en LaTeX el editor le permite hacerlo sin tener que aprender el lenguaje; sin embargo, quien deba escribir texto LaTeX de manera rutinaria poco a poco aprenderá las substituciones que el sistema hace y en esta medida será capaz de generar código LaTeX por sí mismo. Esto impacta directamente la productividad del usuario.

Así, la interfaz final que se planteó estuvo formada por un editor visual básico junto con íconos y elementos gráficos para incorporar los elementos sintácticos del lenguaje para especificar materiales educativos. Ciertamente se logró una solución simple y esta es una de las aportaciones más importantes de este trabajo.

A la fecha existen varias organizaciones que se dedican a la educación virtual como: Blackboard[8], Global Knowledge[13], KnowledgeOne[3], Centra[15], Unext[7], Hot Potatoes[9], KnowledgeBox[1], entre otras. En el caso de América Latina contamos con el *ILCE*[4] que es una red de educación virtual que tiene entre sus miembros a EduSat, Red Escolar, SEPa Inglés y otras.

En México se tienen dos proyectos que se complementan entre sí para la educación virtual: el proyecto *SOFIA* del *CIC-IPN* y el proyecto **e-Edita** del *CINVESTAV*. *SOFIA* es un entorno virtual

para administrar espacios de conocimiento y para dirigir a los navegantes a conseguir sus objetivos en materia educativa y **e-Edita** para especificar materiales educativos para la *Web*. Estos productos están orientados hacia los usuarios finales y los desarrolladores de materiales educativos. Pero no consiguen involucrar de una manera activa a los profesores. Ésto es otro motivo para construir el editor visual que se describe en este trabajo.

El escrito de la tesis tiene esta estructura: en el Capítulo 1 se describen los lenguajes ontológicos, se analiza la relación existente entre éstos y el de **e-Edita** con el fin de explicar los orígenes del lenguaje de **e-Edita** mismo así como las consideraciones básicas para construir un entorno visual para este lenguaje. En el Capítulo 2 se dan las características de **e-Edita** y se describe éste como un entorno de especificación, se habla de los dos lenguajes que lo conforman, se da una descripción general del traductor de **e-Edita**. En el Capítulo 3 se explica cómo se obtuvieron los requerimientos para desarrollar materiales educativos propios para la educación virtual, la estructura sintáctica del lenguaje de especificación y sus constructores fundamentales, así como algunos ejemplos. En el Capítulo 4 se muestran los detalles de diseño del editor visual así como las interfaces gráficas creadas para los constructores de **e-Edita**, se describe de forma general la interfaz con el sistema operativo Unix y el acceso al editor visual desde Internet. En el Capítulo 5 se describe cómo se instrumentó el editor visual principalmente en Java, aunque también se utilizaron lenguajes como `C`, JavaScript y `AWK`. Por último, en el Capítulo 6 se describen los pasos que se deben de seguir para agregar un nuevo constructor al lenguaje, las modificaciones al traductor de **e-Edita**, al programa `EkitEdita.java` y las adaptaciones del `Applet` que contiene a la aplicación.

Convenciones

En este trabajo de tesis se emplean diferentes tipos de letras para representar código, acrónimos, abreviaturas, etc, a continuación se describen las convenciones utilizadas:

`Teletipo`: Se utiliza para representar código fuente, nombres de programas e interfaces con el sistema operativo.

Itálicas: Se utiliza para escribir acrónimos y abreviaturas.

PEQUEÑAS MAYÚSCULAS: Se utiliza para escribir nombres de directivas.

Agradecimientos

El orden en que aparecen estos nombres es relativo; si hubiera forma pondría a todos al inicio.

A Dios:[1] Por darme la oportunidad de existir y vivir cada día un día más.

A Gloria: Mi infinitamente amada esposa, que sin su apoyo, cariño y comprensión, este trabajo no hubiera sido posible.

A Andrea: Mi adorada hija, quien, con su alegría y muestras de cariño incondicional, endulzan mi vida y me motiva a ser mejor cada día.

A mis padres: Inocencio y Eugenia, por darme la vida y a quienes debo todo lo que soy.

A mis hermanos: Nadia, Aleidi y Óscar, que me han apoyado en todo momento, y a los que quiero con todo el corazón.

A Feliú: Por todas sus enseñanzas, por su paciencia, y por el apoyo inmenso e incondicional que recibí de él.

Existe una inmensa lista de personas formada por familiares y amigos, a los que sería imposible nombrar en un espacio tan corto, a todos ellos, gracias por quererme como me quieren.

[1] Me gusta la definición de Lisa Simpson: *No se quién o qué sea Dios, sólo sé que es más poderoso que Mamá y Papá juntos.*

Capítulo 1

Marco formal

En este capítulo se introducen los lenguajes ontológicos que son la base del lenguaje de **e-Edita**. También se muestran los aspectos que relacionan estos lenguajes con la educación virtual. Se analiza la relación entre el lenguaje de **e-Edita** y el entorno visual que se desarrolla en este trabajo. El objetivo final es llegar a las consideraciones de diseño del lenguaje de **e-Edita** y de su interfaz visual así como los elementos fundamentales que debe contemplar un lenguaje ontológico para la creación de materiales para la educación virtual.

1.1. Preliminares

Las ontologías fueron desarrolladas en el área de *Inteligencia Artificial* (IA) para facilitar el intercambio y reutilización del conocimiento. Actualmente, las computadoras han dejado de ser dispositivos aislados y se han convertido en puntos de entrada a la red mundial de intercambio de información y transacciones de negocio. Se ha vuelto un asunto clave contar con apoyo en el intercambio de datos, información y conocimiento. Emplear el poder de razonamiento automático para guiar el acceso a fuentes de información requiere de representación de la semántica de tales fuentes de manera procesable por computadora, en consecuencia, se requieren metadatos que describan de una manera computable, dichas fuentes.

1.2. Lenguajes ontológicos y *XML*

Analicemos con más detalle a los lenguajes ontológicos y se estudia uno de los metalenguajes que se está convirtiendo en un estándar para la *Web* semántica: *XML*.

1.2.1. Lenguajes ontológicos

Una ontología se puede definir como: *La especificación explícita de una conceptualización.* Cuando el conocimiento de un dominio es representado en un formalismo declarativo, el conjunto de objetos que pueden ser representados es llamado el *universo de discurso*. Este conjunto de objetos, y las relaciones descriptibles entre ellos, es reflejado en el *vocabulario* con el cual se representa conocimiento. En una ontología se asocian mediante definiciones los nombres de las entidades en el universo de discurso (por ejemplo clases, relaciones, funciones o otros objetos) con cierto texto legible que las describe y con axiomas que restringen la interpretación y otorgan la característica de *bien formados* a dichos términos.

En pocas palabras, las ontologías se pueden comparar con diccionarios especializados o con un vocabulario controlado que se utiliza para describir objetos y situaciones en una aplicación dada. Así los *lenguajes ontológicos* permiten especificar con precisión el significado del contenido de una página *Web*.

Las ontologías se han clasificado por diversos autores utilizando diferentes criterios:

- GRADO DE AXIOMATIZACIÓN

  Ontologías terminológicas: Definen términos y sus relaciones en taxonomías que involucran relaciones de subtipo y supertipo y metodologías que relacionan partes con un todo, pero no incluyen axiomas y definiciones expresadas en lógica.

  Ontologías formales: Sus categorías están restringidas por axiomas y definiciones expresadas en lógica o en algún tipo de lenguaje interpretable por una computadora.

- DEPENDENCIA DE CONTEXTO

  Ontologías de dominio: Expresan conceptualizaciones que son específicas a un dominio particular, colocando restricciones en la estructura y contenido de un dominio de conocimiento mediante axiomas que se cumplen siempre entre los elementos de dicho dominio.

  Ontologías generales o de sentido común: Incluyen vocabulario relacionado a cosas, eventos, tiempo, espacio, causalidad, comportamiento, función, etc.

  Meta-ontologías o genéricas: Son similares a las de dominio, pero los conceptos que definen se consideran genéricos a través de diferentes áreas de conocimiento y por ello son reutilizables en diferentes dominios.

  Ontologías de tareas: Proveen un vocabulario sistemático de los términos usados para resolver problemas asociados con tareas particulares, ya sean dependientes o no de dominio.

- SUJETO DE CONCEPTUALIZACIÓN

  Ontologías de aplicación: Contienen todas las definiciones que son necesarias para modelar el conocimiento requerido para una aplicación particular en un dominio dado.

  Ontologías de dominio: Coinciden con la clasificación por dependencia de contexto.

  Ontologías genéricas: Coinciden con la clasificación por dependencia de contexto.

  Ontologías de representación del conocimiento: Describen las primitivas de representación usadas para formalizar conocimiento en paradigmas de representación de conocimiento; esto es, que explican la conceptualización que subyace a un formalismo de representación de conocimiento.

  Ontologías de Dominio-Tarea: Aquellas clasificadas como ontologías de tareas pero que se distinguen por ser reutilizables en un determinado dominio, pero no en otros dominios.

- ONTOLOGÍAS "TOP-LEVEL" Pretenden establecer una estructura básica, bajo la cual todos los términos en cualquier ontología existente deberían poder relacionarse.

El lenguaje de **e-Edita** no cae dentro de una clasificación específica, más bien, tiene características de más de una de éstas, por ejemplo, de *dominio* por trabajar en el ámbito de la educación virtual y de *tarea* porque provee de constructores que pueden aplicarse en contextos diferentes a la educación virtual.

1.2.2. Extensible Markup Language (*XML*)

XML es un lenguaje basado en marcas para construir documentos que contengan información estructurada. La información estructurada se compone por un contenido (palabras, imágenes, etc.) y alguna referencia del papel que juega el contenido (por ejemplo, el contenido de un encabezado tiene un significado diferente del contenido de un pie de página, que a su vez significa algo diferente que el pie de una imagen o los datos en una tabla, etc.). Casi todos los documentos tienen alguna estructura. Un lenguaje basado en marcas es un mecanismo para identificar estructuras en un documento. La especificación *XML* define una forma estándar de añadir marcas a los documentos.

XML fue creado al amparo del *Word Wide Web Consortium (W3C)* organismo encargado de supervisar el desarrollo de *WWW* partiendo de las amplias especificaciones de la Standar Generalized Markup Language *SGML*. Su desarrollo se comenzó en 1996, la primera versión salió a la luz

el 10 de febrero de 1998. La primera definición que apareció fue: *Sistema para definir, validar y compartir formatos de documentos en la Web.* Durante 1998 *XML* tuvo un crecimiento exponencial, con apariciones en medios de comunicación, referencias en páginas *Web*, soporte de software, etc. Los objetivos de *XML* son:

- *XML* debe ser directamente utilizable en Internet.
- *XML* debe soportar una amplia variedad de aplicaciones.
- *XML* debe ser compatible con SGML.
- Debe ser fácil la escritura de programas que procesen documentos *XML*.
- El número de características opcionales en *XML* debe ser absolutamente mínima o idealmente cero.
- Los documentos *XML* deben ser legibles por humanos y razonablemente claros.
- El diseño de *XML* debe ser preparado rápidamente.
- El diseño de *XML* debe ser formal y conciso.
- Los documentos *XML* deben ser fácilmente creables.
- La concisión en las marcas *XML* es de mínima importancia.

Esta especificación, junto con los estándares asociados (Unicode e ISO/IEC 10646 para caracteres, Internet RFC 1766 para identificación de lenguajes, *ISO* 639 para códigos de nombres de lenguajes e ISO 3166 para códigos de nombres de países), proporciona toda la información necesaria para entender la versión 1.0 de *XML* y construir programas de computadora que los procesen.

El metalenguaje *XML* consta de cuatro especificaciones, para las cuales el propio *XML* sienta las bases sintácticas y el alcance de su implementación:

DTD (Document Type Definition): Definición formal del tipo de documento.

XSL (eXtensible Stylesheet Language): Implementa el lenguaje de estilo de los documentos escritos para *XML*

XLL (eXtensible Link Language): Define el modo de enlace entre diferentes aplicaciones creadas con *XML*.

XUA (XML User Agent): Estandarización de navegadores *XML*.

1.3. Lenguajes ontológicos en la educación virtual

Los lenguajes ontológicos tienen diversas características que pueden traducirse en ventajas (si se saben explotar) al desarrollar lenguajes para crear aplicaciones en el ámbito de la educación virtual. Éstas se describen a continuación.

Claridad: Comunica de forma efectiva el significado propuesto para cada uno de los términos.

Completitud: Definen condiciones necesarias y suficientes. Se recomienda que estén documentadas utilizando lenguaje natural.

Coherencia: Las conclusiones inferidas son consistentes con las definiciones propias de la ontología.

Extensibilidad: Soportan la definición de nuevos términos basándose en el vocabulario existente.

Mínimo compromiso ontológico: Contiene la menor cantidad de aseveraciones posibles acerca del mundo que está siendo modelado.

Diversificación de jerarquías: Aumentan el poder que otorgan los mecanismos de herencia múltiple.

Modularidad: Se pueden definir diferentes elementos como clases, relaciones y axiomas, tales definiciones se pueden agrupar en teorías que reúnen los objetos más relacionados entre si de una ontología.

Minimización de la distancia semántica entre conceptos hermanos: Conceptos similares se deben agrupar y representar como subclases de una clase son definidos usando las mismas primitivas, mientras conceptos que son menos similares se representan aparte en la jerarquía.

En el lenguaje de **e-Edita** se manifiestan sobre todo la claridad y el mínimo compromiso ontológico.

1.4. e-Edita y su relación con las ontologías

En las secciones anteriores se presentaron características de los lenguajes ontológicos y *XML*; además de la relación de éstos con la educación virtual. Ahora, es el turno de ver cómo es su relación con el lenguaje de **e-Edita**. Para empezar hay que aclarar que **e-Edita** como lenguaje ontológico es en extremo simple, ya que sólo se utiliza para especificar los elementos que forman parte de una página con contenido educativo, no los axiomas ni condiciones de correctitud sobre los mismos. En **e-Edita** no existen directivas para especificar qué tipo de contenido hay en las páginas creadas. Aunque se pueden incorporar al lenguaje, no es la finalidad de **e-Edita**.

Por el momento el uso del lenguaje de **e-Edita** como ontología es realmente limitado, sin embargo, esta característica puede modificarse en el futuro, por esta razón es que hace gran énfasis en el capítulo 6 sobre la forma de extender tanto el lenguaje básico de **e-Edita** como su interfaz visual. En la actualidad no está claro cuál será el lenguaje ontológico que ha de convertirse en estándar, creemos que *XML* es la alternativa. En realidad vemos al lenguaje de **e-Edita** como un lenguaje intermedio y preferimos en el futuro construir los traductores de *XML* (o cualquier otro lenguaje ontológico) al lenguaje de **e-Edita** y viceversa.

1.5. El lenguaje de e-Edita y su enlace con el entorno visual

A pesar de que la sintaxis de **e-Edita** es muy simple, para que un usuario la aprenda requiere de cierto entrenamiento. El objetivo de diseñar e instrumentar una interfaz visual es que el usuario se familiarice poco a poco con la sintaxis utilizada en el lenguaje. Esto funciona de la siguiente manera: cuando se está editando una página y el usuario desea insertar un constructor (ver sección 3.2 y 3.3), selecciona de la barra de herramientas el ícono asociado con la tarea que desea realizar (ver 4.1), en ese momento aparece una interfaz gráfica, la cual pide al usuario los parámetros requeridos por el constructor, una vez terminada la edición de éste, se inserta en el editor visual la sintaxis correspondiente a la instancia del mismo.

El objetivo más importante en **e-Edita** es la especificación de materiales para la *Web* y uno de sus compromisos básicos es el de promover la productividad. Si bien las interfaces gráficas simplifican el uso de sistemas de cómputo, también es cierto que a largo plazo éstas resultan tediosas, así que la idea de construir una interfaz gráfica para **e-Edita** es que los usuarios aprendan su lenguaje de especificación hasta el grado de prescindir de la interfaz y trabajar con las facilidades que cualquier editor típico ofrece.

Esto contrasta sensiblemente con otras herramientas como *Hot potatoes* [9], en la cual, la interfaz gráfica está bien diseñada y es bastante agradable a la vista pero, resulta en extremo cansado utilizarla para producir un libro electrónico con 50 o más páginas.

1.6. Consideraciones de diseño

La creación de páginas *Web* con contenidos educativos requiere de construcciones que permitan incluir textos, elementos multimedia y diversos tipos reactivos. El lenguaje de especificación de **e-**

Edita permite crear páginas con estas características. A construcciones como éstas se les conoce como *objetos de aprendizaje*. No existe una definición formal de este concepto, aquí se dan tres versiones distintas:

- Recursos modulares digitales, identificados de forma única y acompañados por meta-marcas, que pueden ser utilizados para soportar procesos de aprendizaje.
  National Learning Infraestructure Iniciative [10]
- Cualquier recurso digital que pueda ser reutilizado para soportar procesos de aprendizaje. La idea principal de los objetos de aprendizaje es partir el contenido educativo en piezas pequeñas que puedan ser reutilizadas en varios ambientes de aprendizaje, en forma similar a los principios de la programación orientada a objetos.
  David A. Wiley
- Cualquier entidad, digital o no, que pueda ser usada, reutilizada o referenciada durante el aprendizaje soportado tecnológicamente.
  Learning Object Metadata Working Group [2]

En el caso de **e-Edita** los objetos de aprendizaje son las páginas *Web* que se pueden crear, éstas permiten incorporar elementos interactivos. Las herramientas para construir tales páginas se diseñaron tomando en cuentas las necesidades de los profesionales de la educación respecto a la creación de materiales educativos accesibles vía Internet. En primer lugar se tomó en cuenta el proceso de la edición que puede requerir del uso de tipografía especializada y en segundo lugar la publicación vía Internet de los materiales. También es necesario evaluar los conocimientos de los alumnos, lo cual se logra por medio cuestionarios, éstos se definen con ayuda de los especificadores de reactivos. Todo esto se verá en detalle en el Capítulo 3.

Capítulo 2

El entorno de e-Edita

El lenguaje de **e-Edita** es el resultado de varios años de investigación y desarrollo en el ámbito de los sistemas de información en general y en particular en los sistemas orientados a la educación.

Su objetivo fundamental, desde su perspectiva más amplia, ha sido el de diseñar lenguajes de especificación de documentos para la construcción de sitios en Internet que ofrezcan altos niveles de interactividad. Los resultados han sido aplicados en contextos muy diversos que incluyen el desarrollo de formularios, encuestas interactivas, sistemas de bases de datos, sistemas de información geográfica y sistemas orientados a la educación virtual vía Internet.

En este capítulo sólo se muestran las características y estructura del lenguaje de especificación de **e-Edita**, así como los beneficios y tipos de construcciones que éste nos ofrece; todos los aspectos relacionados con la sintaxis del lenguaje explican en el Capítulo 3.

2.1. Características de e-Edita

El lenguaje de **e-Edita** permite a los creadores de páginas *Web* con contenidos educativos utilizar diversos elementos que pueden incorporar en sus textos:

Textos libres. Estas son secuencias de texto que se traducen de manera literal a *HTML*. En estos textos es posible incluir construcciones propias del lenguaje *HTML* aunque esto prácticamente no se necesita hacer.

Textos en tipografía LaTeX.[1] Sin lugar a dudas el lenguaje natural para escribir matemáticas es LaTeX, todos los matemáticos tienen que conocerlo y dominarlo porque es el estándar en las revistas internacionales, así que es de esperarse que un autor matemático lo requiera. Por desgracia existen pocas herramientas que soportan la tipografía LaTeX en páginas de Internet, entre éstas la más conocida es `latex2html` [14] pero tiene la desventaja de hacer traducciones estáticas, es decir, sus parámetros no se pueden configurar; también existe `vulcanize` [5] la cual no puede manejar ambientes LaTeX anidados y `cam1` que sólo traduce texto, no puede convertir expresiones matemáticas. **e-Edita** ofrece un soporte más completo produciendo imágenes digitales de gran calidad a partir de textos fuente en LaTeX.

Definición de problemarios interactivos. Ofrece la posibilidad de especificar preguntas que los usuarios responden. Estas preguntas tienen un carácter interactivo bidireccional. Por esta razón el autor, al especificar una pregunta, indica cuál es la respuesta correcta y alguna sugerencia para el caso en el que un usuario no la responda de manera adecuada.

Calificar problemarios. Un usuario puede solicitar que su trabajo sea calificado con objeto de obtener la retroalimentación del sistema. En este caso la respuesta que recibe el usuario

[1] **e-Edita** originalmente se desarrolló en el Departamento de Matemáticas del CINVESTAV para publicar notas de cursos interactivas en la Internet.

contiene la indicación de las preguntas que respondió mal, así como la sugerencia que el autor haya escrito para tal efecto.

Registrar calificaciones. Esto sirve en caso de requerir llevar un control de las calificaciones de los usuarios. Es decir, el usuario puede especificar que una página determinada es en realidad un examen. En este caso los usuarios deben seguir un proceso de registro donde reciben una clave de acceso que tienen que enviar junto con su examen. Todos los datos necesarios para manejar esta opción se almacenan en una base de datos y se ofrecen elementos para la producción de estadísticas, registros de calificación y boletas para los usuarios. (Esta facilidad está diseñada pero aún no se instrumenta)

Definición de bancos de preguntas. Los bancos de preguntas se construyen en bases de datos que se generan con comandos específicos de **e-Edita**. Los bancos son definidos y mantenidos por sus autores de manera que pueden agregar, modificar o eliminar preguntas. El lenguaje tiene opciones específicas para hacer que, en la producción de un examen, se incorporen preguntas elegidas aleatoriamente, que provengan de uno o varios bancos de datos. Aparte del proceso propio de la generación de exámenes hay herramientas a disposición del autor para hacer análisis de reactivos. (Ídem)

Creación de índices. Los autores tienen a su disposición diversas facilidades para indicar que en un texto ciertas palabras deben aparecer en un índice de términos. Los índices de términos tienen una naturaleza interactiva de modo tal que al desplegarse una página, todas las palabras que pertenezcan al índice se muestran en un color especial; si el usuario selecciona esta palabra entonces automáticamente se le muestran todos los demás lugares en un libro electrónico que contiene las palabras. El índice mismo es accesible tal como se hace con el índice de un libro, la diferencia es que para consultar una palabra específica hay ligas directas a los lugares donde aparece.

Enlace con bases de datos. e-Edita contiene facilidades para establecer contacto con bases de datos. Esto sería útil, por ejemplo, en un libro de historia donde se desean guardar los datos estadísticos sobre población. Al usuario se le podría invitar a seleccionar un estado de la república y un año específico para que revise datos como número de habitantes, número de personas alfabetizadas, número de mujeres, etc., en el año y el estado especificados.

2.2. e-Edita como un entorno de especificación

La palabra *especificación* tiene un significado muy particular en este trabajo, que difiere del significado tradicional en el contexto de los modelos de especificación formal. Al igual que en algunos modelos de especificación formal, entendemos por lenguaje de especificación a un lenguaje que permite describir una aplicación de manera no procedural, sino más bien descriptiva, pero a diferencia de la especificación formal no estamos propiamente interesados en la coherencia de lo que se está definiendo, más bien en traducirlo de manera directa a código en un lenguaje que pueda ser ejecutado directamente en una computadora. En otras palabras, para nosotros especificar una aplicación significa describirla en un lenguaje próximo a la aplicación misma.

e-Edita está constituido esencialmente por dos lenguajes de especificación, el primero sirve para especificar páginas *Web* individuales y el segundo para agrupar páginas en un libro electrónico. En **e-Edita** lo fundamental son los aspectos funcionales por encima de los de diseño, esto con el fin de favorecer un alto nivel de productividad. Así se logran esquemas de desarrollo que propician la comunicación entre grupos de trabajo, la posibilidad de dividir un desarrollo entre varias personas y que grupos a distancia puedan colaborar en un desarrollo común.

Los aspectos finales de diseño y presentación se abordan de manera independiente y se incorporan en el trabajo mediante el uso de páginas de estilo *HTML*. Esto permite que al final todas las páginas de un proyecto tengan exactamente el mismo estilo de presentación.

Los dos lenguajes de especificación de **e-Edita** se basan en el uso de *constructores*, a su vez éstos se dividen en *directivas* y *especificadores de reactivos*. En **e-Edita** la unidad sintáctica es el párrafo

(conjunto contiguo de líneas no vacías) cuya primera palabra corresponde a la directiva; seguida de ésta y separadas por caracteres cedilla (ç) (que se usan a manera de comas) aparecen los parámetros de las directivas.

Hay un tipo particular de párrafos que no contienen directiva, a estos se les denomina *párrafos libres*[2] y tienen una estructura especial que permiten incluir elementos interactivos (especificadores de reactivos) como: preguntas abiertas, preguntas de opción múltiple y ejercicios de relacionar columnas. Este tipo de párrafos pueden contener directamente código en *HTML*.

2.3. Diseño de e-Edita desde una perspectiva global

En esencia los lenguajes de especificación de **e-Edita** han sido diseñados con la finalidad de ofrecer un conjunto mínimo básico de construcciones para instrumentar aplicaciones que sean accesibles vía Internet.

El compromiso principal de **e-Edita** es la simplicidad, por encima de cuestiones de apariencia o de presentación. El propósito fundamental de los lenguaje de especificación de **e-Edita** es que los usuarios expresen la funcionalidad de sus aplicaciones en términos propios de la jerga que entienden los expertos en el dominio de la aplicación que se está desarrollando.

Por otra parte, el modelo básico del lenguaje de **e-Edita** permite definir con facilidad herramientas visuales para asociar objetos gráficos con la sintaxis del lenguaje, de manera que un usuario que la desconozca pueda especificar una aplicación. Esta es la característica fundamental que hemos explotado en esta tesis para la construcción del lenguaje visual que permite construir aplicaciones en el ámbito de la educación virtual.

Con **e-Edita** no se pretende eliminar la participación de personal técnico en la definición de una aplicación. Más bien, **e-Edita** es un vehículo de comunicación para que aquellos aspectos que no sean cubiertos de manera satisfactoria por el proceso de traducción del lenguaje puedan ser llevados a cabo por técnicos especialistas. El punto clave, es que los usuarios finales no tengan que tratar con aspectos técnicos de instrumentación. En este sentido podemos decir que **e-Edita** es una herramienta para propiciar la productividad a través de la comunicación entre todos los que participan en el desarrollo de una aplicación.

A la fecha **e-Edita** es un producto experimental que se ha probado en diferentes ámbitos además del de la educación virtual y los resultados han sido satisfactorios; incluso se ha construido una gran cantidad de aplicaciones industriales. Las desventajas que hasta ahora se han presentado son:

- Que las interfaces para el usuario final muchas veces son rígidas y resulta un tanto difícil mejorar sus diseños.
- Las modificaciones sobre los modelos básicos que se producen tienen que ser llevadas a cabo por un técnico especialista.
- Resulta difícil que el usuario poco familiarizado con lenguajes de programación acepte usar un lenguaje como el de **e-Edita**.

Éstas son las partes más delicadas que estamos atendiendo en nuestro trabajo: para que los diseños sean un tanto más flexibles se está trabajando con la incorporación de páginas de estilo que unifiquen las presentaciones finales. La dificultad de aceptación de los usuarios al lenguaje de **e-Edita** pretende resolverse mediante el editor visual que se presenta en este trabajo.

2.4. Estructura del lenguaje de especificación de e-Edita

Se pretende que el proceso de traducción del lenguaje de **e-Edita** a código *HTML* sea simple, lo que se logra utilizando expresiones regulares. Por esta razón, la sintaxis utilizada en el lenguaje es en extremo simple: las directivas van seguidas de sus atributos separadas por carácteres cedilla (ç),

[2]Cuando el nombre de una directiva se quiera utilizar al inicio de un párrafo libre se precede al nombre de la directiva por un carácter \

en tanto los especificadores de reactivos se escriben entre dos barras verticales (|), escribiendo cada atributo seguido de su valor separados por comas.

Esto puede parecer restrictivo en primera instancia, pero se compensa con la posibilidad de enlazar una especificación de **e-Edita** con programas escritos en algún lenguaje interpretado por Unix como los lenguajes de comandos del sistema operativo, `AWK`, Pearl, Java, TCL, etc. También es posible asociar directivas con aplicaciones más complejas del sistema operativo.

Estas características hacen de **e-Edita** un lenguaje que estimula la productividad ya que propicia una comunicación muy estrecha con Unix.

2.5. Beneficios estructurales del lenguaje

Como se vió en la sección 1.3, los lenguajes ontológicos ofrecen varios beneficios, que el lenguaje de **e-Edita** pretende satisfacer. En lo referente a la *extensibilidad*, es fácil hacer al lenguaje compatible con estándares como *XML* o *RDF*; el *mínimo compromiso ontológico* se mantiene al tener pocos constructores (evitando así la redundancia) pero que al combinarse logran un gran poder; la *claridad* se logra en todos los constructores del lenguajes, ya que comunican de forma efectiva la acción para la que están destinados y la *completitud* se manifiesta en **e-Edita** al permitir definir aplicaciones en términos que manejan los expertos del dominio (creando para ello los constructores necesarios).

Además, **e-Edita** ofrece algunas otras ventajas, por ejemplo, el proceso de traducción es muy eficiente, lo cual permite que sea ejecutado en línea por aplicaciones accesibles vía Internet y la simplicidad de su sintaxis permite que los usuarios la aprendan rápidamente.

2.6. Tipos de construcciones de e-Edita

Como se vió en la sección 2.2, en **e-Edita** existen dos tipos de construcciones disponibles para crear páginas de especificación: *Directivas* y *Especificadores de reactivos*. Cada uno de estos elementos tiene una sintaxis bien definida que se describe a continuación. Cabe señalar que los elementos entre llaves ({}) pueden repetirse cero o más veces y los elementos entre corchetes ([]) deben aparecer al menos una vez.

- *Directiva*
  <identificador>ç {parámetro}c

- *Especificador de reactivos*
  |[nomAtributo1, valor1,] {nomListaAtributos, valorLista1,
  valorLista2, /nomListaAtributos}|

2.6.1. Directivas

Las directivas de **e-Edita** utilizadas para construir aplicaciones en educación virtual permiten, entre otras cosas, insertar elementos multimedia, invocar programas `AWK`, definir títulos y contenidos. También son utilizadas en otros contextos, por ejemplo, permiten integrar mapas a las páginas electrónicas o manejar bases de datos.

2.6.2. Especificadores de reactivos

Los especificadores de reactivos son elementos que permiten incorporar diversos tipos de preguntas en las páginas electrónicas. Las cuales sirven para evaluar a los navegantes. En la sección 2.2 se vió que existen tres tipos base: *preguntas abiertas*, de *opción múltiple* y de *relacionar columnas*. Éstas se pueden dividir en subtipos (opción múltiple en formatos vertical u horizontal), o bien, se pueden utilizar para especificar preguntas de tipo verdaro-falso (preguntas abiertas con respuesta específica *V* o *F*).

2.7. Aplicaciones que se pueden desarrollar con e-Edita

A la fecha con **e-Edita** se han realizado varios trabajos que han sido recibidos favorablemente por la comunidad, en particular se escribió un libro electrónico interactivo (con más de ciento veinte páginas de Internet) para la actualización de profesores de nivel medio superior en Matemáticas Básicas. Este libro [18] es un producto funcional que está en uso y que nos ha permitido entender el enorme vacío que en materia de producción de este tipo de trabajos existe a nivel nacional.

e-Edita se ha aplicado también en contextos no matemáticos y el resultado se ha plasmado en la publicación de una gran cantidad de trabajos que abarcan desde materiales para la enseñanza de idiomas hasta la construcción de textos interactivos relacionados con áreas de capacitación técnica. Por sus orígenes y sus capacidades actuales, la vocación fundamental de **e-Edita** es la producción de materiales con contenido educativo.

2.8. Los dos lenguajes de especificación de e-Edita

La orientación fundamental de **e-Edita** es la especificación de textos de manera que se ofrezca el máximo nivel de interactividad posible. Para este fin se dispone de dos lenguajes de especificación fundamentales:

- El Lenguaje para Páginas Interactivas (LPI)
- El Lenguaje para Libros Interactivos (LLI)

El LPI permite definir las lecciones u objetos de aprendizaje que conforman un libro electrónico. El lenguaje para la especificación de libros (LLI) tiene la función de aglutinar, en un solo documento, a todas las páginas que componen un libro. Ofrece algunas de las facilidades del LPI y su funcionalidad es comparativamente menos compleja.

Capítulo 3

e-Edita en la educación virtual

En este capítulo se explica cómo se obtuvieron los requerimientos para desarrollar materiales educativos propios para la educación virtual, asimismo se explica cuál es la estructura sintáctica del lenguaje de especificación de **e-Edita** y se muestra cómo deben ser utilizados los dos tipos de constructores del lenguaje: directivas y especificadores de reactivos, por medio de dos ejemplos, en el primero se muestra una página y en el segundo un libro electrónico.

3.1. Análisis de requerimientos desde la perspectiva de la educación virtual

Para que los profesionales de la educación elaboren materiales con contenidos educativos para Internet se requiere de herramientas accesibles y simples de utilizar. El lenguaje de especificación de **e-Edita** cuenta con estas características, sin embargo, para poder acercarlo al mayor número posible de desarrolladores de materiales educativos se le pretende dotar, con el presente trabajo, de una interfaz amistosa que facilite su aprendizaje y uso.

La producción de páginas *Web* con contenidos educativos requiere cuidar dos aspectos fundamentales: diseño de la página y el contenido de la misma. El diseño se maneja por medio de páginas de estilo que dan uniformidad a las páginas creadas. En lo referente al contenido, **e-Edita** cuenta con diversos constructores que permiten insertar elementos interactivos. En las secciones siguientes se muestran con detalle los elementos de cada uno de los constructores involucrados en la edición de materiales educativos para la red Internet.

Como se vió en la sección 2.8, **e-Edita** cuenta con dos modos de edición que permiten crear tanto páginas como libros electrónicos. Las páginas pueden contener textos y elementos interactivos (creados por medio de los especificadores de reactivos). Los libros son un compendio de páginas ya creadas.

3.2. Lenguaje de Páginas Interactivas

En esta sección se describen las directivas y especificadores de reactivos que conforman el *Lenguaje de Páginas Interactivo (LPI)* de **e-Edita** utilizados para construir materiales educativos. En la Figura 3.1 se muestra un ejemplo completo que se explica con detalle en esta sección.

La página mostrada en la Figura 3.1 se generó a partir de la especificación:

```
TITLEç Título e-Edita

HEADINGç Editor de materiales educativos para la web

ADDRESSç Ing. Israel Méndez. imendez@computacion.cs.cinvestav.mx<BR>
Centro de Investigación y de Estudios Avanzados del IPN<BR>
```

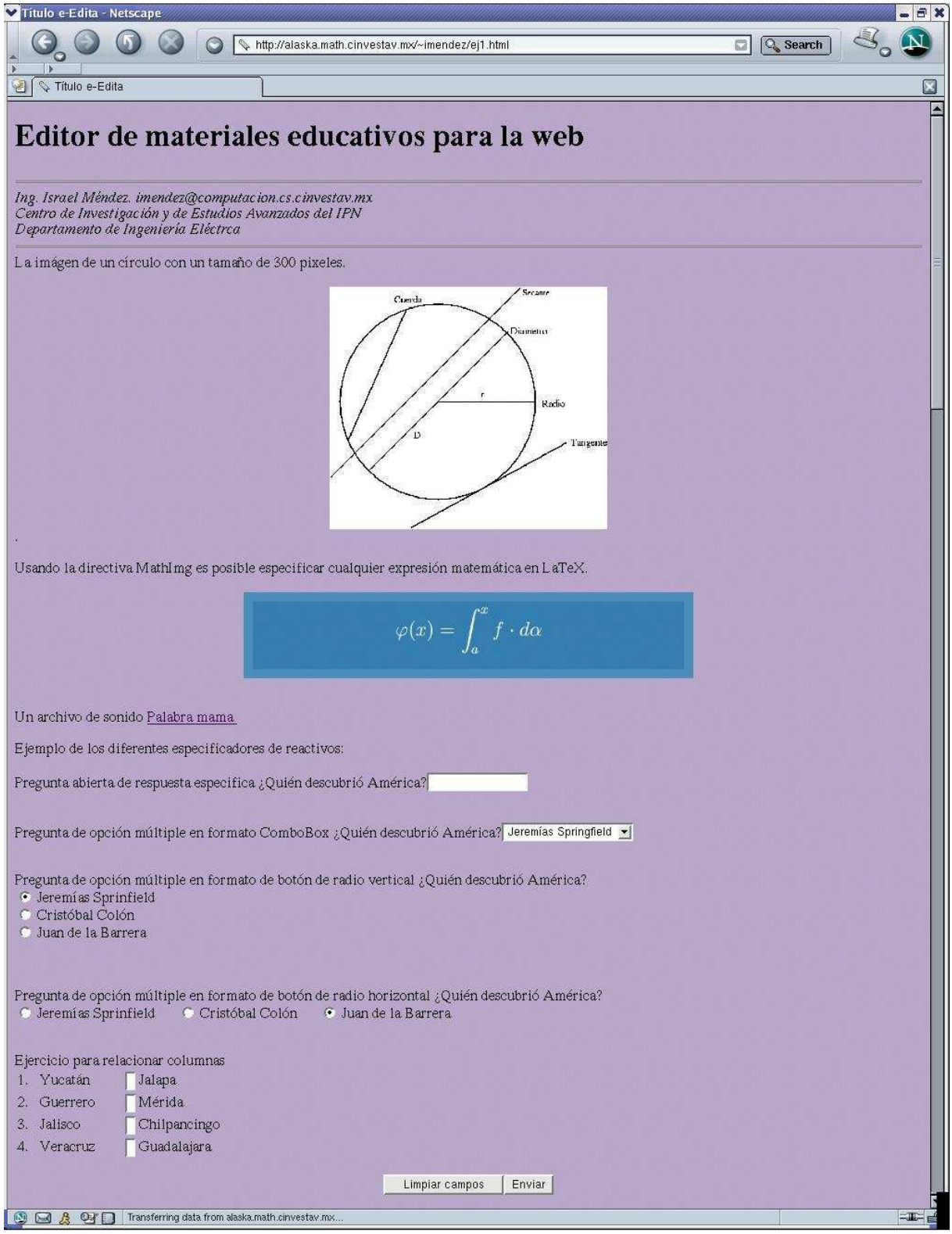

Figura 3.1: Página creada con **e-Edita**

```
Departamento de Ingeniería Eléctrca<BR>

La imagen de un círculo con un tamaño de 300 pixeles.<BR><BR>

<CENTER>

IMAGEç circulo.jpgç 300ç

</CENTER>.<BR><BR>

Usando la directiva MathImg es posible especificar cualquier expresión
matemática en LaTeX.<BR><BR>

<CENTER>

MATHIMGç \fbox{\parbox{3.5in}{ $$\varphi(x)=\int_a^x f\cdot d\alpha$$
}}ç -density 130x130 -quality 100 -crop 0x0-1-1 -border 10x10+15+15
-colorize 20/50/70 -negate ç

</CENTER><BR><BR>

Un archivo de sonido

SOUNDç mamap.wavç Palabra mamáç

<BR><BR>
Ejemplo de los diferentes especificadores de reactivos:<BR><BR>

Pregunta abierta de respuesta específica
¿Quién descubrió América? |answer, Cristóbal Colón, hint, Carabelas|
<BR><BR><BR>

Pregunta de opción múltiple en formato ComboBox
¿Quién descubrió América? |name, cmbAmerica, type, select, options,
Cristóbal Colón, Juan de la Barrera, Jeremías Springfield, /options,
answer, Cristóbal Colón, default, Jeremías Springfield| <BR><BR><BR>

Pregunta de opción múltiple en formato de botón de radio vertical
¿Quién descubrió América?  <BR>
|type, vradio, options, Jeremías Sprinfield, Cristóbal Colón, Juan de
la Barrera, /options, answer, Cristóbal Colón, default, Jeremías
Sprinfield| <BR><BR><BR>

Pregunta de opción múltiple en formato de botón de radio horizontal
¿Quién descubrió América?  <BR>
|type, hradio, options, Jeremías Sprinfield, Cristóbal Colón,
Juan de la Barrera, /options, answer, Cristóbal Colón, default, Juan
de la Barrera| <BR><BR><BR>

Ejercicio para relacionar columnas <BR>
|type, columns, questions,Yucatán, Guerrero, Jalisco, Veracruz,
/questions, options, Mérida, Chilpancingo, Guadalajara, Jalapa,
/options, hint, Estudia mas|

ENDç NOç ej2
```

3.2.1. Directiva ADDRESS (Dirección)

Con esta directiva se puede agregar a la página que se está editando información general y de contacto sobre autor. El texto contenido en esta directiva se presenta en un tipo de fuente especial (especificado en la página de estilo) y puede colocarse al inicio o al final de la página.

ADDRESSç Ing. Israel Méndez. imendez@computacion.cs.cinvestav.mx
Centro de Investigación y de Estudios Avanzados del IPN
Departamento de Ingeniería Eléctrica

Figura 3.2: Ejemplo de directiva ADDRESS

3.2.2. Directiva END (Fin)

Marca el final de la especificación, requiere de dos parámetros: el nombre de la página anterior y de la siguiente, si alguno de éstos no esta definido, se utiliza la cadena *NO* para indicar la ausencia del parámetro. En el caso de libros electrónicos esta directiva no lleva parámetros.

ENDç NOç ej1

3.2.3. Directiva HEADING (Sub-título)

Coloca un título (<texto>) dentro en la página que se está editando. La primera directiva de este tipo en la especificación introduce el título que aparece en el índice definido por la directiva *CONTENT*. Se pueden introducir tantas instancias de esta directiva como se requieran. El resultado de la especificación se muestra en la Figura 3.3

HEADINGç Editor de materiales educativos para la web

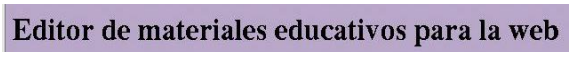

Figura 3.3: Ejemplo de directiva HEADING

3.2.4. Directiva IMAGE (Imagen)

Inserta una imagen en la página, esta directiva requiere de dos parámetros: el nombre del archivo que contiene la imagen y el tamaño en pixeles de la misma, si este último parámetro es cero, la imagen se inserta en su tamaño original.

IMAGEç circulo.jpgç 300ç

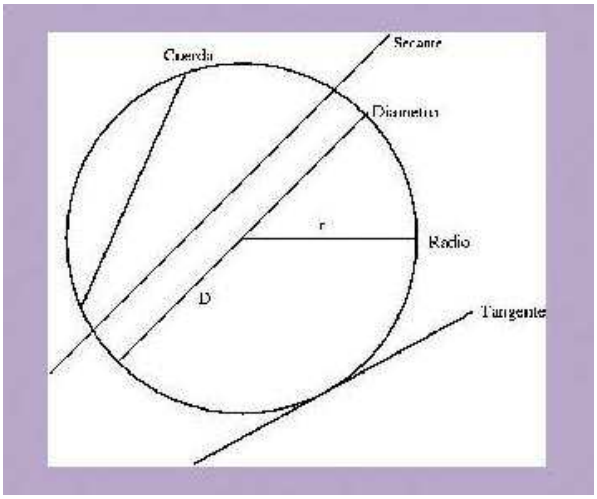

Figura 3.4: Ejemplo de directiva IMAGE

3.2.5. Directiva MATHIMG (Tipografía especializada)

La directiva MATHIMG inserta una imagen con tipografía LaTeX. Este comando genera la imagen y la deposita en el directorio `$HOME/public_html`. Los archivos así generados reciben nombres como <*nombre-página*><*número-consecutivo*>, donde <*nombre-página*> es el nombre del archivo con la especificación y <*número-consecutivo*> es un número entero con tres cifras (000, 001, y así sucesivamente). Por consiguiente, este tipo de nombres no deben ser utilizados para otros propósitos. Además esta directiva es muy versátil al trabajar con imágenes ya que opera con la misma sintaxis del comando `convert` de Linux.

En la Figura 3.5 se muestra el resultado que se obtiene con la especificación de la directiva MATHIMG.

```
MATHIMGç \fbox{\parbox{3.5in}{ $$\varphi(x)=\int_a^x f\cdot d\alpha$$
}}ç -density 130x130 -quality 100 -crop 0x0-1-1 -border 10x10+15+15
-colorize 20/50/70 -negate ç
```

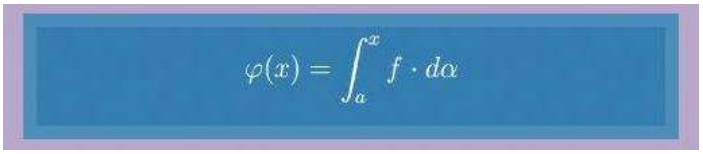

Figura 3.5: Ejemplo de directiva MATHIMG

3.2.6. Directiva NOFOLLOW (No siguiente)

Los libros electrónicos están formados por un conjunto de páginas, las cuales tienen un orden secuencial. Cuando se está editando una página que no pertenece a un libro se utiliza esta directiva para no poner las referencias a las páginas anterior y siguiente. Si en la especificación se hubiera utilizado esta directiva, en la Figura 3.1 no aparecería lo mostrado en la Figura 3.6

NOFOLLOW

Figura 3.6: Ejemplo de directiva NOFOLLOW

3.2.7. Directiva NOGRADE (No calificar)

Cuando se desea evaluar a los navegantes se utilizan reactivos en las páginas electrónicas (ver sección 3.3), sin embargo, existen páginas en las cuales no se requieren, por ejemplo, páginas de introducción. Con esta directiva no se muestran los botones de *Enviar* y *Limpiar campos* que se muestran en la Figura 3.7.

NOGRADE

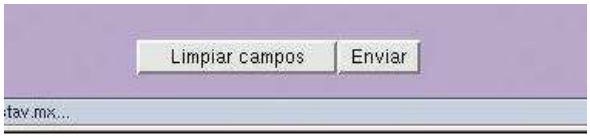

Figura 3.7: Ejemplo de directiva NOGRADE

3.2.8. Directiva SOUND (Sonido)

Esta directiva permite insertar un hipervínculo a un archivo de sonido, el cual, al ser seleccionado se reproduce.

SOUNDç mamap.wavç Palabra mamáç

Figura 3.8: Ejemplo de directiva SOUND

3.2.9. Directiva TITLE (Título)

Coloca un título (<texto>) en la parte superior de la ventana del navegador utilizado para visualizar la página, cada documento debe llevar una sola de estas directivas y debe aparecer al principio del archivo. Se recomienda que todas las páginas que forman parte de un capítulo lleven el mismo título. El resultado de la especificación se muestra en la Figura 3.9

TITLEç Título de la ventanaç Página de estilo

3.3. Especificadores de reactivos

En las subsecciones anteriores se mostró la sintaxis de las directivas de **e-Edita** ahora es el turno de explicar la sintaxis de los especificadores de reactivos disponibles. Existen tres tipos de reactivos:

Figura 3.9: Ejemplo de directiva TITLE

preguntas abiertas con respuesta específica, *preguntas de opción múltiple* y *preguntas para relacionar columnas*. A su vez las preguntas de opción múltiple se dividen en dos categorías: tipo *ComboBox* y tipo *botón de radio*. Por último las de botón de radio se dividen en *formato horizontal* y *vertical*. Esto da un total de cinco tipos base, cada uno de los cuales se explica a continuación.

3.3.1. Especificador de preguntas abiertas con respuesta específica

Tenemos dos formas sintácticas para definir este tipo de pregunta[1]:

- |answer, <respuesta>, hint, <sugerencia> |
- |name,<nombre>, type, text, heading, <título del campo>, answer, <respuesta>, default, <valor por omisión>, hint, <sugerencia> |

La primera es una forma abreviada y la segunda una forma completa; pero el objetivo de ambas es el mismo: introducir campos interactivos de texto en la página *HTML* final para que el usuario responda preguntas abiertas con un texto específico. En esta sintaxis <respuesta> es el texto de la respuesta correcta y <sugerencia> es un texto que se muestra en caso de que se escriba una respuesta errónea. En la forma sintáctica completa <nombre> es el texto asociado al nombre interno del campo en *HTML* y <valor por omisión> es el texto que por omisión se despliega en el campo. Las palabras *name, type, answer, default* y *hint* son etiquetas que deben escribirse tal como aparecen al lado izquierdo de la categoría sintáctica que le corresponda, aunque su orden relativo es independiente.

En la Figura 3.1 la especificación es la siguiente:

```
¿Quién descubrío América?
|answer, Cristóbal Colón ,hint, Carabelas|
```

y se obtiene:

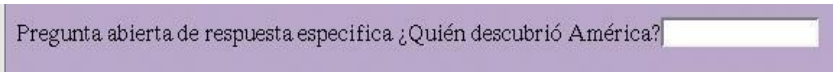

Figura 3.10: Ejemplo del especificador de pregunta abierta

3.3.2. Especificador de preguntas de opción múltiple tipo *ComboBox*

En las preguntas de este tipo se muestra al usuario un cuadro con las posibles respuestas, tiene la ventaja de ocupar poco espacio dado que éste sólo se despliega al hacer clic sobre él. Para definir instancias se utiliza la siguiente sintaxis:

[1] Los textos entre paréntesis angulares <> denotan categorías sintácticas, los textos subrayados deben escribirse tal como aparecen. Los caracteres | son parte de la sintaxis (no metacaracteres del formulismo BNF)

|name, <nombre>, heading, <título del campo>, type, select, options, <opción 1>, <opción 2>, . . . , <opción n>, /options, answer, <respuesta>, default, <valor por omisión>, hint, <sugerencia> |

Podemos observar que los cambios en esta sintaxis con respecto a la utilizada para definir preguntas abiertas con respuesta específica son pocos: la etiqueta *type* aquí tiene el valor de *select* lo cual indica que se trata de una pregunta que se despliega en formato "comboBox" de *HTML*, la etiqueta *options* sirve para introducir posibles respuestas de la pregunta. Para poder identificar donde terminan las opciones usamos como delimitador */options*. En caso de que se desee utilizar el carácter coma (,) en una opción se utiliza el código \, para hacer la distinción.

En el ejemplo 3.1 la especificación correspondiente es:

```
¿Quién descubrió América?
|name, cmbAmerica, type, select, options, Cristóbal Colón, Juan de la Barrera, Jeremías
Springfield, /options, answer, Cristóbal Colón, default, Jeremías Springfield|
```

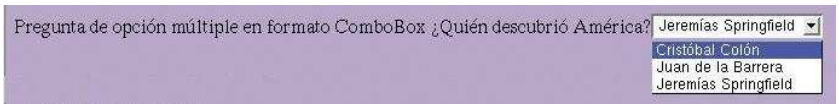

Figura 3.11: Ejemplo del especificador en formato *ComboBox*

3.3.3. Especificador de preguntas de opción múltiple tipo botón de radio

Este tipo de pregunta tiene la característica de mostrar todas sus opciones de respuesta, lo cual puede consumir mucho espacio, sobre todo si se utiliza el formato vertical. El colocar las opciones en formato vertical u horizontal depende, entre otras cosas, del tamaño de cada opción, por ejemplo: si las opciones son palabras el formato horizontal es adecuado, si las opciones son enunciados es mejor el formato vertical, sin embargo, el usuario puede elegir el que más le agrade. Para definir instancias de ésta se utiliza la sintaxis:

|name, <nombre>, type, hradio, options, <opción 1>, <opción 2>, . . . , <opción n>, /options, answer, <respuesta>, default, <valor por omisión> |

Aquí la etiqueta *type* puede tomar el valor *hradio* (o bien *vradio*), indica que se trata de una pregunta que se despliega en formato de *botón de radio* de *HTML*, horizontal (o vertical). La etiqueta *answer* se utiliza para definir la respuesta correcta. Cada una de las opciones deben de colocarse entre las etiquetas *options* y */options*, la etiqueta *default* indica si alguna de la opciones estará seleccionada por omisión.

En la Figura 3.1 la especificación es:

```
¿Quién descubrió América?
|type, vradio, options, Jeremías Sprinfield, Cristóbal Colón,
Juan de la Barrera, /options, answer, Cristóbal Colón, default,
Jeremías Sprinfield|
```

La especificación de la pregunta de opción múltiple en formato vertical es:

```
¿Quién descubrió América?
|type, hradio, options, Jeremías Sprinfield, Cristóbal Colón,
```

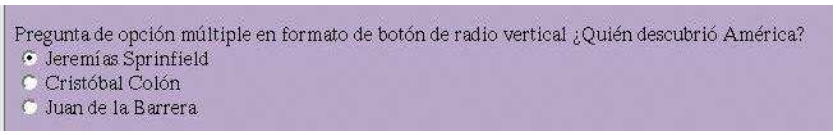

Figura 3.12: Ejemplo del especificador en formato botón de radio vertical

```
Juan de la Barrera, /options, answer, Cristóbal Colón, default,
Juan de la Barrera|
```

Figura 3.13: Ejemplo del especificador en formato botón de radio horizontal

3.3.4. Especificador de preguntas para relacionar columnas

Las preguntas que involucran relacionar dos columnas son comúnmente utilizadas en las evaluaciones, en el ejemplo se muestra un ejercicio donde los alumnos deben asociar diferentes estados de la república con su correspondiente capital. En el lenguaje de especificación de **e-Edita** se utiliza la siguiente sintaxis para definir instancias de éstas.

|name, <nombre>, heading, <encabezado>, type, columns, questions, <pregunta 1>, <pregunta 2>, ..., <pregunta n>, /questions, options, <opción 1>, <opción 2>, ..., <opcion n>, hint, <sugerencia> |

En este caso la etiqueta *type* tiene el valor *columns* para indicar que se trata de un reactivo para relacionar columnas, entre las etiquetas *questions* y */questions* de escriben los elementos de la columna de la izquierda, los cuales representan las preguntas y entre las etiquetas *options* y */options* se colocan los elementos de la columna de la derecha, que representan las opciones para la columna de la derecha. Por último con la etiqueta *hint* se define una sugerencia para el grupo de preguntas.

La especificación que se muestra a continuación es para definir un ejercicio de este tipo:

```
|type, columns, questions, Yucatán, Guerrero, Jalisco, Veracruz,
/questions, options, Mérida, Chilpancingo, Guadalajara, Jalapa,
/options, hint, Estudia mas|
```

En la Figura 3.14 se muestra el resultado que se obtiene al utilizar esta especificación, el ejemplo es sobre un ejercicio de relacionar algunos estados de la República Mexicana con sus respectivas capitales.

3.4. Lenguaje de libros interactivos

En el *Lenguaje de Libros Interactivos (LLI)* también se pueden utilizar las directivas del LPI: ADDRESS, TITLE, HEADING, IMAGE y END. Se puede observar que en la Figura 3.15 la ventana está dividida en dos partes, en la parte izquierda se presenta la tabla de contenido y en la parte derecha una portada para el libro electrónico, siendo ésta un archivo con extensión `.html` y con el mismo nombre que el archivo de especificación `.l` pero precedido por la letra *p*.

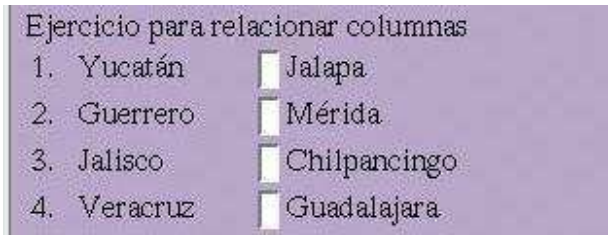

Figura 3.14: Ejemplo de relacionar columnas

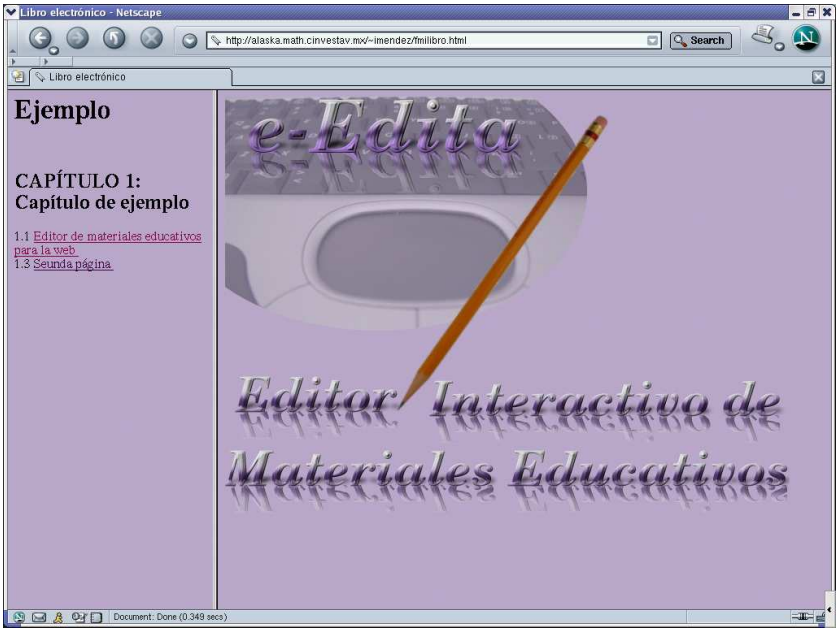

Figura 3.15: Ejemplo de un libro electrónico creado con **e-Edita**

```
TITLEç Libro electrónico

HEADINGç Ejemplo

CHAPTERç Capítulo de ejemplo

CONTENTç ej1ç ej2

END
```

3.4.1. Directiva CHAPTER (Capítulo)

Especifica el título de un capítulo dentro de un libro electrónico. El resultado de la especificación se muestra en la Figura 3.16.

CHAPTERç Capítulo de ejemplo

3.4.2. Directiva CONTENT (Contenido)

Especifica las páginas que conforman un capítulo para formar la tabla de contenido de un libro electrónico. El texto que aparece en la tabla es el definido en la primera instancia de la directiva

HEADING de la página (ver sección 3.2.3). El resultado de la especificación se muestra en la Figura 3.16

CONTENTç ej1ç ej2

Donde ej1 y ej2 son los nombres de los archivos que contienen la especificación de cada uno de los capítulos.

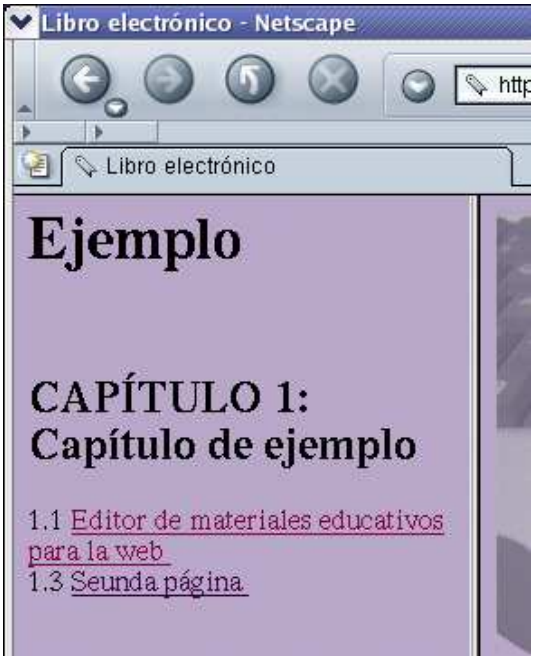

Figura 3.16: Ejemplo de directiva CHAPTER y CONTENT

3.5. Operación de e-Edita

En esta sección se describen las herramientas de **e-Edita** desde el punto de vista del sistema operativo, es decir, cómo se lleva a cabo la traducción de páginas y libros electrónicos desde la línea de comandos del sistema y cómo utilizar y para qué sirve el comando `generatodo`.

3.5.1. Traducción de páginas

Una vez terminada la especificación de una página es momento de traducirla, ésto se logra utilizando el comando `pagina`, el cual requiere como parámetro el nombre del archivo de especificación sin la terminación `.e`. Los modificadores del comando `pagina` se muestran en el Cuadro 3.1.

```
[user@localhost espec]$pagina mipagina
```

La mayoría de los modificadores mostrados en el Cuadro 3.1 no son utilizados desde la línea de comando, más bien, son utilizados por los programas *CGI* que invocan las traducciones.

3.5.2. Traducción de libros

Cuando se han editado las páginas que conforman un libro electrónico, éstas se unen por medio de un archivo de especificación con terminación (`.l`); utilizando la directiva *content* se definen los archivos (`.e`) que forman a los capítulos. Los archivos (`.l`) se traducen con el comando `libro`, con el cual genera la página principal y la tabla de contenido del libro, los modificadores de este comando se muestran en el Cuadro 3.2

-m 0	Traduce la especificación sin generar los MATHIMG
-m 3,5	Traduce la especificación generando los MATHIMG del 3 al 5
-p	Califica una página
-o	Atender servicio "al vuelo"
-e	Escribir los posibles errores en la página
-h	Especifica el directorio *home* del servidor
-f	Especifica el nombre del archivo a traducir
-t	Especifica el nombre del *host*
-l	Especifica el *login* del usuario
-q	Especifica la calidad de los MATHIMG
-s	Indica que se va a utilizar una página de estilo
-z	No pone tilde (~) en las direcciones IP

Cuadro 3.1: Modificadores del comando `pagina`

```
[user@localhost espec]$libro milibro
```

-m	Resalta en color la liga de la sección que se esta mostrando
-n	No numera los capítulos ni las secciones
-c	No pone la palabra *Capítulo* en las directivas *chapter*, resulta útil cuando no se están editando libros

Cuadro 3.2: Programas *CGI* encargados de atender las peticiones del editor

3.5.3. Comando `generatodo`

En las subsecciones anteriores se mostró cómo traducir por separado las especificaciones de páginas y libros electrónicos (archivos con terminación `.e` y `.l` respectivamente), sin embargo, existe un comando que permite traducir todas las especificaciones que se encuentran en un directorio, éste es `generatodo`.

El comando `generatodo` no requiere argumentos y se utiliza directamente en la línea de comandos del sistema operativo Unix. Desafortunadamente este comando no esta instrumentado para el *Applet* del editor visual.

```
[user@localhost espec]$generatodo
```

Los comandos para traducir las especificaciones de **e-Edita** son muy simples, pero no por eso dejan de ser poderosos, permitiendo inclusive, reducir el tiempo de traducción (que se logra no generando los elementos MATHIMG). Dado que el editor visual (motivo de este trabajo) tiene como objetivo primordial que los usuarios aprendan la sintaxis del lenguaje, no se instrumentaron interfaces que permitan invocar estos comandos de forma directa.

Capítulo 4

Diseño del editor visual para e-Edita

En este capítulo se presenta el diseño del editor visual así como las consideraciones que se hicieron en su conceptualización. También se muestra el diseño de las interfaces utilizadas y los módulos en los que está dividido el sistema. Los detalles sobre la instrumentación del diseño presentado se abordan en el siguiente capítulo.

4.1. Evolución del diseño de editor visual

El diseño del editor visual fue una parte fundamental para lograr una buena planeación de la instrumentación del mismo. Este diseño sufrió varias modificaciones desde su concepción original. La primera parte del diseño consistió en analizar los requerimientos del usuario, es decir, qué se necesita para crear páginas de especificación de **e-Edita** utilizando herramientas visuales. En la segunda parte se estudió cada uno de los constructores relacionados con la educación virtual con la finalidad de obtener los parámetros configurables por el usuario. Con esta información se crearon los borradores de las interfaces para el usuario y ventana principal del editor visual, más adelante se muestran estos diseños.

4.2. Diseño conceptual del editor visual

El editor visual está dividido en dos partes, la primera corresponde a la interfaz principal donde se encuentra el área de edición y los elementos de interacción con el usuario, la segunda corresponde a las interfaces de cada uno de los constructores de **e-Edita**. En las subsecciones siguientes se explica cómo se crearon los borradores y se muestra la versión final de cada una de las interfaces gráficas instrumentadas.

4.2.1. Creación de borradores

Las interfaces gráficas del editor visual de **e-Edita** se diseñaron analizando páginas ya construidas con el lenguaje de especificación de **e-Edita**, estudiando en qué forma éstas podrían ser construidas utilizando elementos visuales.

Para cada uno de los constructores de **e-Edita** (excepto para las directivas *NOFOLLOW* y *NOGRADE* que no requieren parámetros) se crearon borradores de sus interfaces tomando en cuenta los argumentos requeridos por cada uno de ellos, diseñando la distribución de los componentes de manera que fuera intuitivo para el usuario el uso de los mismos.

En el caso de las interfaces que requieren un número variable de argumentos, como es el caso de preguntas de opción múltiple y el contenido de un capítulo, se decidió definir un número por defecto (configurable) de opciones, éstos son seis y diez respectivamente.

Durante el proceso de instrumentación de las interfaces gráficas, algunos borradores sufrieron pequeñas modificaciones debido a que algunas de las facilidades de Java permitían instrumentar la

interfaz de forma más clara que el diseño original y en algún caso debido a parámetros que no se habían considerado en los borradores iniciales.

4.2.2. Diseño de la ventana principal

La interfaz gráfica del editor visual está diseñada de forma estándar, es decir, tiene una apariencia similar a los procesadores de palabras, con un área de edición, barras de menús y herramientas.

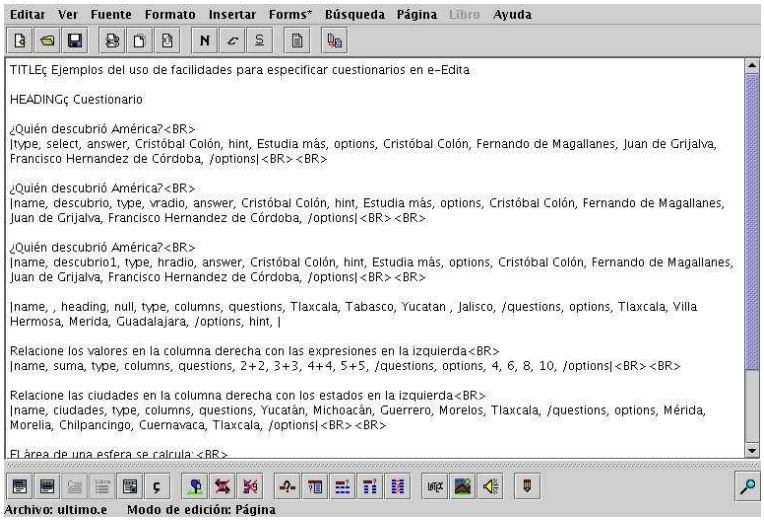

Figura 4.1: Ventana principal del editor visual para **e-Edita**

Se puede observar en la Figura 4.1 que en la parte superior se encuentra la barra de menús y herramientas estándar, en la parte central aparece el área de edición y en la parte inferior se tiene la barra de herramientas de **e-Edita**. El significado de cada uno de estos íconos se explica en la sección 4.4.

4.3. Diseño de las interfaces para los constructores de e-Edita

Para diseñar la interfaz apropiada de cada uno de los constructores de **e-Edita** se analizaron los elementos que los conforman así como los parámetros que requieren. En las subsecciones siguientes se muestran los diseños de las interfaces gráficas instrumentadas.

4.3.1. Directiva TITLE

La directiva TITLE (ver sección 3.2.9) utiliza dos argumentos: el título de la página que se está editando y el nombre de la página de estilo que se desea utilizar. Como ambos valores son de tipo *cadena* la interfaz para esta directiva sólo requiere dos campos de texto.

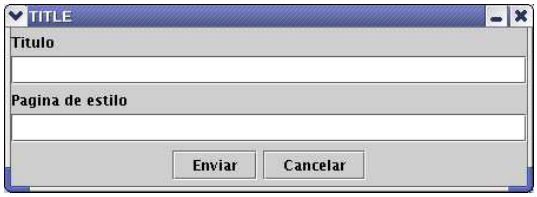

Figura 4.2: Interfaz para la directiva TITLE

4.3.2. Directiva HEADING

Esta directiva (ver sección 3.2.3) sirve para introducir un título dentro de la página que se está editando, tiene como parámetro una *cadena* que especifica el título. Así la interfaz para esta directiva consta de un solo campo de texto.

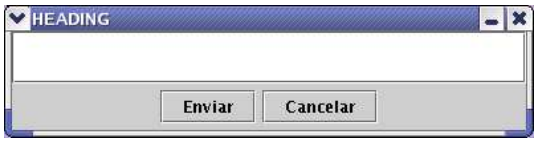

Figura 4.3: Interfaz para la directiva HEADING

4.3.3. Directiva END

La interfaz de esta directiva (ver sección 3.2.2) es muy similar, en lo que al diseño se refiere, a la directiva TITLE. Utiliza dos argumentos que indican la página siguiente y anterior en relación a la página actual.

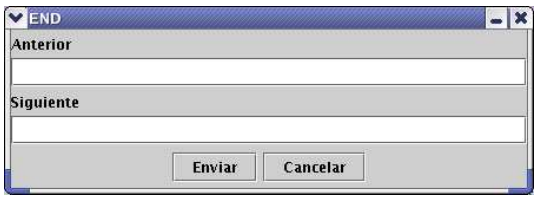

Figura 4.4: Interfaz para la directiva END

4.3.4. Directiva CHAPTER

En la sección 3.4.1 se explicó que esta directiva sirve para especificar nombres de capítulos en la edición de libros electrónicos, al igual que la directiva HEADING sólo requiere de un parámetro que es el nombre del capítulo.

Figura 4.5: Interfaz para la directiva CHAPTER

4.3.5. Directiva CONTENT

Esta directiva (ver sección 3.4.2) permite especificar los nombres de las páginas electrónicas que forman parte un mismo capítulo. El texto que aparece en el índice creado por esta directiva es el especificado en la primer directiva HEADING de cada página incluida. Esta interfaz proporciona por defecto espacio para diez elementos.

Figura 4.6: Interfaz para la directiva CONTENT

4.3.6. Directivas IMAGE y SOUND

Estas directivas (ver secciones 3.2.4 y 3.2.8) tienen como finalidad insertar en la página que se está editando un elemento multimedia, ya sea una imagen o un sonido, requieren como parámetro el nombre del archivo que se desea insertar y en el caso de *IMAGE* el tamaño horizontal de la imagen en pixeles. Estas directivas tienen una interfaz gráfica casi idéntica, que consta de un navegador de archivos y una barra para ajustar el tamaño de la imagen en caso necesario.

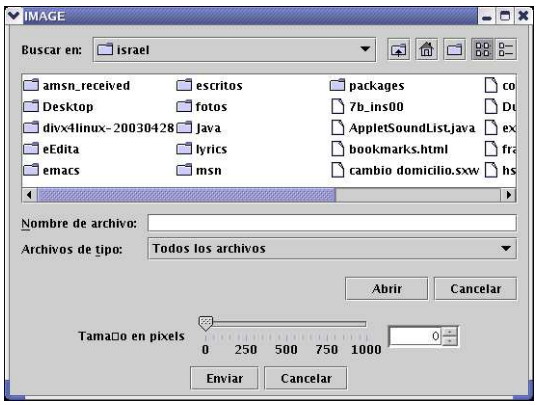

Figura 4.7: Interfaces para las directivas IMAGE y SOUND

4.3.7. Directiva MATHIMG

Esta directiva sirve para incorporar tipografía LaTeX en un documento. Por su nivel de complejidad utiliza varios parámetros (ver sección 3.2.5). Con el fin de simplificar el diseño de su interfaz ésta se dividió en tres partes: cuerpo de la directiva, ajuste de parámetros y selección de color. En las Figuras 4.8, 4.9 y 4.10 se muestran, respectivamente, cada una de estas partes.

Con esto se completa el diseño de las interfaces visuales para cada una de los elementos del lenguaje de especificación de **e-Edita** relacionados con la educación virtual. Ahora continuamos con las interfaces para especificar reactivos.

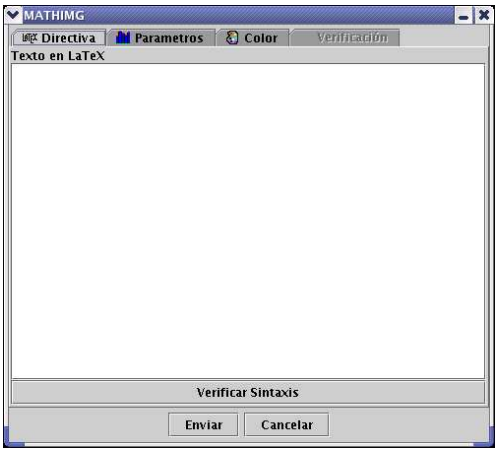

Figura 4.8: Editor del contenido la directiva

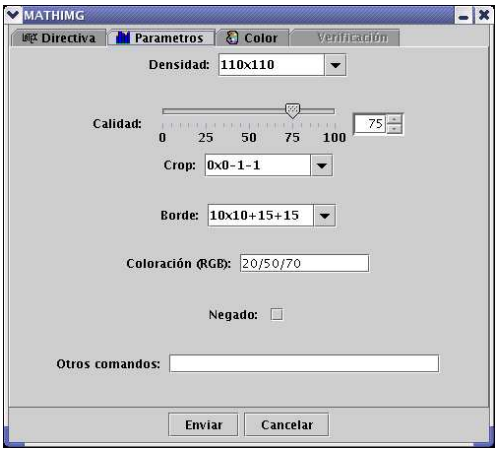

Figura 4.9: Interfaz para el ajuste de parámetros

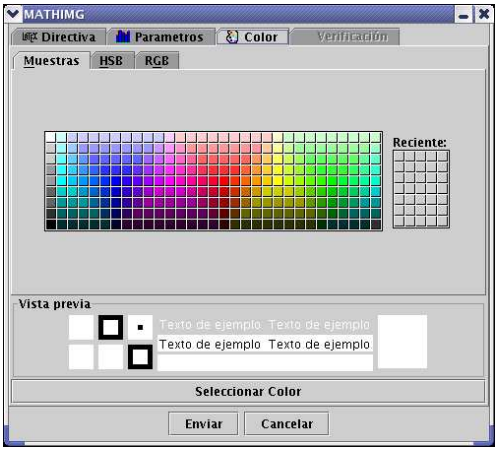

Figura 4.10: Interfaz para la selección de color

4.3.8. Especificador de preguntas abiertas con respuesta específica

Existen dos formas para definir este especificador (ver sección 3.3.1), así su interfaz se ha dividido en dos partes: la básica y la avanzada. La diferencia entre ambas radica en el número de parámetros que se pueden configurar con cada una de ellas.

La interfaz básica consta de dos componentes: la respuesta a la pregunta y una sugerencia. Ésta se muestra en la Figura 4.11.

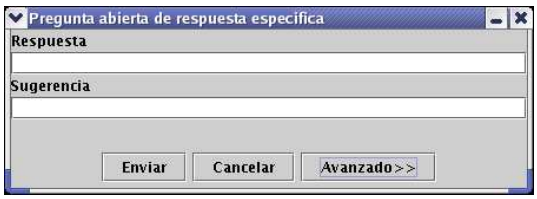

Figura 4.11: Interfaz básica para preguntas abiertas con respuesta específica

La interfaz avanzada muestra elementos para configurar de forma más precisa al reactivo, asignándole un nombre específico, un encabezado para la pregunta y una opción por defecto. Ésta se muestra en la Figura 4.12

Figura 4.12: Interfaz avanzada para preguntas abiertas con respuesta específica

4.3.9. Especificador de preguntas de opción múltiple

Hay varios especificadores de reactivos que manejan la modalidad de opción múltiple: preguntas de tipo *ComboBox* (ver sección 3.3.2) y *RadioButton* (ver sección 3.3.3)en sus modalidades horizontal y vertical. Salvo diferencias mínimas estos especificadores requieren los mismos parámetros por lo que se utiliza una misma interfaz para todos ellos, ésta se muestra en la Figura 4.13.

4.3.10. Especificador de preguntas en modalidad de relacionar columnas

Por último, para especificar reactivos en la modalidad de relacionar columnas (ver sección 3.3.4) se debe proporcionar la lista de valores de la primera y segunda columna. La interfaz para hacerlo tiene la estructura que se muestra en la Figura 4.14.

4.4. Diseño de íconos de e-Edita

El editor visual de **e-Edita** cuenta con una barra de herramientas y menús provistos de íconos para accesar a cada una de las interfaces gráficas presentadas en la sección anterior. La asociación entre las interfaces y los íconos que las representan se muestra en el Cuadro 4.1.

Figura 4.13: Interfaz para preguntas de opción múltiple

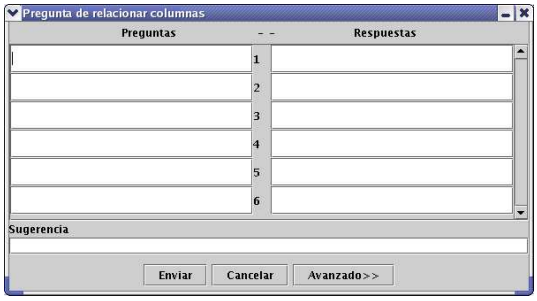

Figura 4.14: Interfaz para crear preguntas de relacionar columnas

	Directivas		
	TITLE		HEADING
	ADDRESS		NOFOLLOW
	NOGRADE		MATHIMG
	IMAGE		SOUND
	CHAPTER		CONTENT
	END		
	Especificadores de reactivos		
	Pregunta abierta		Pregunta ComboBox.
	Pregunta RadioButton vertical		Pregunta RadioButton horizontal
	Edición		
	Modo página		Modo libro
	Traducir documento		Vista previa del documento
	Inserta cedilla		Edita constructor

Cuadro 4.1: Íconos utilizados en el editor visual de **e-Edita**

4.5. Módulos del sistema

Para facilitar el diseño del sistema, éste se dividió en tres módulos, cada uno de ellos realiza una tarea específica. La descripción de los mismos se muestra en el Cuadro 4.2 y se detallan a

continuación.

El editor visual	Entorno en el cual interactúa el usuario
La interfaz con el S.O.	Atiende las peticiones que genera el editor visual
El acceso desde Internet	Diseño del portal de acceso al editor visual

Cuadro 4.2: Módulos del sistema

4.5.1. El editor visual

El componente más importante y objeto principal del trabajo es el Editor Visual para **e-Edita**. A continuación se describen los aspectos relacionados con su diseño, tales como la descripción general de su funcionamiento, las políticas de administración utilizadas y el entorno con el que interactúa el usuario.

Funcionamiento del editor visual

Los eventos que maneja el editor visual para **e-Edita** se dividen en dos categorías: los de edición encargados de agregar un nuevo constructor a la página de especificación que se está editando y los de control que realizan tareas específicas como traducir, guardar o mostrar el documento actual. Ambas categorías pueden ser invocadas por medio de los botones en las barras de herramientas o las opciones en el menú.

Cuando se invoca un evento de la categoría de edición se presenta al usuario alguna de las interfaces gráficas mostradas en la sección 4.3, el usuario usa está interfaz para especificar los parámetros que se requieran y al terminar se inserta en el editor visual el texto con la especificación correspondiente en el lenguaje de **e-Edita**. Por otra parte, cuando se invoca un evento de control no se observa ningún cambio en el editor visual, sino que se ejecuta una acción específica como abrir, guardar o traducir un documento, abrir una ventana del navegador para ver como queda la página terminada, o bien, para cambiar el modo de edición actual del editor visual (ver sección 2.8).

Políticas de administración de usuarios

Dado que el editor visual estará disponible en Internet y podrá ser utilizado por varios usuarios simultáneamente es necesario establecer políticas de administración que permitan asegurar su buen funcionamiento. La primera política que se debe establecer es ¿Quién puede tener acceso al editor visual?, Cualquier persona puede tener acceso, siempre que se registre y obtenga su *nombre de usuario* y *contraseña* para ingresar al sistema.

La segunda política es ¿Cuánto espacio tendrá asignado cada usuario?, Cada usuario que se registre recibe automáticamente 2Mb de espacio en el servidor para almacenar y publicar sus especificaciones, sin embargo, esta cantidad puede incrementarse mediante una solicitud al administrador.

Cada usuario tendrá por defecto dos directorios: `espec` en el cual se guardan todos los archivos de especificación generados y `public_html` donde se guardan las páginas *HTML* traducidas así como las imágenes que incluyen.

La tercera política es ¿Cuándo dar de baja a un usuario?, para hacer más eficiente el uso de espacio en el servidor es necesario verificar que los usuarios hagan uso del espacio asignado de forma constante y responsable. En el primer punto se verifica que el usuario entre al sistema y en segundo que publique información relacionada con la educación virtual.

Entorno del usuario

Los directorios `espec` y `public_html` de un usuario particular se usan, respectivamente para guardar las páginas de especificación y las páginas *HTML* generadas por el traductor de **e-Edita**.

En el directorio `espec` se almacenan únicamente las especificaciones de páginas electrónicas (archivos con extensión `.e`) y libros electrónicos (archivos con extensión `.l`), mientras que en directorio

`public_html` se guardan los archivos con extensión `.html` generados a partir de las especificaciones del directorio `espec`, además *todas* las imágenes y sonidos que se deseen incluir en las páginas *HTML*. También deben colocarse en este directorio todos los archivos que se requieran para atender las peticiones de un usuario. Esto incluye programas, scripts de comando, etc.

4.5.2. Interfaz con el S.O. Unix

En la sección 4.5.1 se explicó que existen dos categorías de eventos que pueden ser generados por el editor visual, sin embargo, sólo los eventos de control requieren que se interactúe con el sistema operativo, esto se hace por medio de los programas *CGI* que se explican en la sección 5.1.4.

Hay otras formas de interactuar con el sistema operativo, que se basan fundamentalmente en el uso de scripts interpretados por programas como Java, `AWK`, Pearl, etc., o por programas desarrollados directamente. A la fecha se tiene instrumentada la directiva "AWK" que toma como parámetro el nombre del archivo `.awk` que ha de procesar la página de especificación donde aparezca. Cuando el programa *CGI* `califica1.out` procesa una petición con esta primitiva, automáticamente transfiere al programa `.awk` los parámetros de la petición en un formato adecuado y deja que sea tal programa quien responda la petición. El programa `.awk` genera la especificación de la respuesta que a su vez es traducida por el programa `pagina`, de manera automática y el resultado se devuelve al usuario que hizo la petición. Esta forma de procesamiento permite utilizar prácticamente todo el poder de las herramientas de Unix. No se permite procesar comandos de Unix de manera directa por razones de seguridad.

4.5.3. Acceso desde Internet

El acceso al editor visual de **e-Edita** por medio de Internet se hace a través del portal [20], como se explicó en la sección 4.5.1 es necesario que cada usuario cuente con un *nombre de usuario* y una *contraseña* para tener acceso al sitio, al entrar al portal aparece la ventana mostrada en la Figura 4.15 donde se pide al usuario que proporcione esta información.

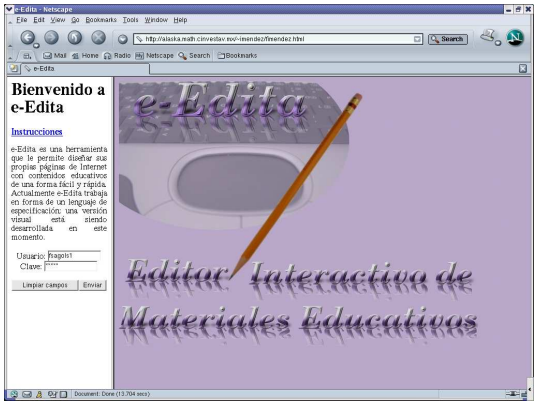

Figura 4.15: Página de ingreso al editor visual de **e-Edita**

Una vez validada esta información el navegante tiene acceso al editor visual. Por defecto al iniciar su sesión de trabajo se abre de forma automática el último documento de especificación con el que se trabajó en la sesión anterior. En caso de ser la primera vez que el usuario entra al sistema se presenta un documento en blanco.

Certificado de seguridad

Con el objetivo de insertar archivos personalizados de sonido e imágenes a las páginas de especificación de **e-Edita** es necesario exportarlos desde la máquina del cliente al servidor. Para seleccionar

un archivo es necesario leer el disco duro del cliente, pero esta acción está inhabilitada por defecto en los *applets* por razones de seguridad. Por tal motivo fue necesario crear un certificado seguridad con el cual se garantiza al usuario que sólo se extraerá de su disco duro los archivos que él ha seleccionado para su exportación y ninguna otra información.

Así el acceso al editor visual requiere que se acepte el certificado de seguridad que se presenta al navegante, este se muestra en la Figura 4.16.

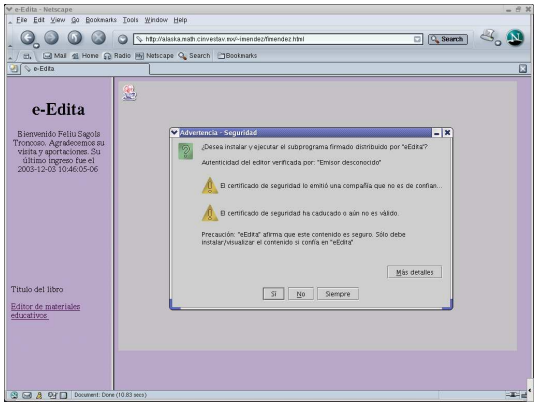

Figura 4.16: Certificado de seguridad de **e-Edita**

4.6. Trabajo a futuro

En el contexto de la educación virtual, se pueden agregar nuevos especificadores de reactivos al lenguaje de especificación de **e-Edita**, por ejemplo, para definir crucigramas o preguntas abiertas que no puedan ser calificadas de manera automática, sino por un evaluador humano.

Además con **e-Edita** se pueden desarrollar aplicaciones que van más allá del contexto de la educación virtual, abarca campos tan diversos como: *Sistemas de Información Geográfica y Generación de Encuestas*, entre otros. Es posible introducir el editor visual en cada uno de estos dominios de aplicación instrumentando las interfaces adecuadas.

Capítulo 5

Instrumentación del diseño del editor visual de e-Edita

En este capítulo se describe cómo se llevó a cabo la instrumentación del diseño del editor visual para **e-Edita**, en el proceso se emplearon varios lenguajes de programación como Java, C, `AWK` y JavaScript, y diversas herramientas del sistema operativo UNIX como el compilador de LaTeX el programa *sed* y el comando *convert*.

5.1. Instrumentación en Java

En el capítulo anterior se mostró el diseño del editor visual desde la concepción de las ideas hasta la estructura de cada una de las interfaces gráficas para el usuario, ahora es el turno de mostrar cómo se instrumentó el diseño utilizando el lenguaje de programación Java. Se eligió este lenguaje debido a las facilidades que ofrece para la instrumentación del diseño propuesto, por ejemplo, la gran versatilidad que se obtiene con las ventanas, el número de componentes que se puede utilizar y la facilidad para establecer conexiones con los servidores involucrados por medio de la red Internet.

5.1.1. Facilidades de Java utilizadas

El lenguaje de programación Java ofrece una gran cantidad de clases que permiten que la instrumentación sea rápida y eficiente. En particular se utilizaron las facilidades para la creación de componentes, manejo de ventanas, control de eventos y administración de conexiones con el servidor donde se encuentra el editor visual [19].

El cuadro 5.1 muestra la relación de clases utilizadas. En total son nueve paquetes de los cuales cinco son para el manejo de ventanas y los restantes para controlar eventos, excepciones y conexiones con los servidores. La descripción detallada de estos componentes aparece en [6].

Usando estas bibliotecas se crearon los tres programas fundamentales del editor visual:

- El *Applet* básico del editor visual (programa `EkitApplet.java`) que es el programa que utiliza el navegante vía Internet.
- El controlador general del editor visual (programa `EkitCore.java`).
- Las interfaces que interactúan con el usuario final (programa `EkitEdita.java`).

Estos programas son explicados en las secciones 5.1.6 y 5.1.7, por ahora sólo se presenta la estructura de los directorios donde están almacenados.

awt	Abstract Window Toolkit	
BorderLayout	Color	Cursor
Dimension	Frame	Graphics
Image	Insets	ToolKit
awt.event		
ActionEvent	ActionListener	KeyEvent
io	Input/Output	
BufferedReader	File	FileInputStream
FileOutputStrean	FileWriter	InputStream
InputStreamReader	IOException	ObjectInputStream
ObjectOutputStream	Reader	DataInputStream
DataOutputStream		
net		
URL	URLConnection	URLEncoder
MalformedURLException		
util		
Enumeration	Hashtable	Locale
ResourceBundle	Vector	
swing		
AbstractAction	Action	ButtonGroup
Icon	ImageIcon	JApplet
JButton	JCheckBox	JColorChooser
JComboBox	JComponent	JDialog
JEditorPane	JFileChooser	JFrame
JLabel	JMenu	JMenuBar
JMenuItem	JOptionPane	JPanel
JRadioButton	JScrolPane	JSlider
JSpinner	JSplipPane	JTabbedPane
JTextArea	JTextField	JTextPane
JToggleButton	JToolBar	KeyStroke
SpinnerNumberModer		
swing.event		
CaretEvent	CaretListener	DocumentEvent
DocumentListener	UndoableEditEvent	UndoableEditListener
swing.text		
AttributeSet	BadLocationException	ChangedCharSetException
DefaultEditorKit	DefaultStyledDocument	Element
PlainDocument	Position	Style
StyleConstants	StyledDocument	StyledEditorKit
swing.undo		
UndoManager	CannotUndoException	

Cuadro 5.1: Clases de Java utilizadas

5.1.2. Organización de los archivos y directorios

Para entender mejor cómo se instrumentó el diseño es necesario conocer la estructura de directorios utilizada para programar el editor visual. Como en todo paquete de Java se utiliza el nombre del dominio URL estructurado de acuerdo con las recomendaciones dadas en el capítulo 5 de la referencia [6], el cual es *mx.cinvestav.math*. La estructura del directorio se muestra en la Figura 5.1.2.

En el directorio raíz (`eEdita`) se encuentran los archivos con extensión *.csh* para Linux y *.bat* para Windows necesarios para compilar el editor visual. Los directorios `mx`, `cinvestav` y `math` son para cumplir con el nombre de dominio estructurado. En el directorio `ekit` se encuentran los

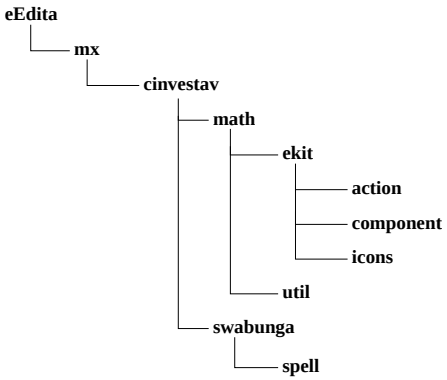

Figura 5.1: Estructura de directorios del editor visual de **e-Edita**

programas `EkitCore.java`, `EkitEdita.java` y `EkitApplet.java` encargados del funcionamiento del editor visual. En el directorio `icons` se encuentran todos los iconos utilizados por el editor visual. En `components` se encuentran las clases extendidas de algunos componentes con características especiales. En `action` están las clases que procesan acciones de edición y estilo del editor visual. En `util` están las facilidades para manejar múltiples idiomas. En `swabunga` están las clases utilizadas para utilizar el corrector ortográfico.

Esta información se proporciona con la finalidad de que futuros programadores que deseen ampliar el editor visual puedan ubicar rápida y fácilmente los programas que deben modificar para instrumentar nuevas capacidades.

5.1.3. Comunicación entre programas

Como se explicó en el apartado anterior, los programas que conforman al editor visual de **e-Edita** (`EkitCore.java`, `EkitEdita.java` y `EkitApplet.java`) son parte de un mismo paquete de Java; sin embargo, la invocación de métodos entre estos programas no se hace de forma directa.

La comunicación entre métodos pertenecientes a programas diferentes se lleva a cabo mediante la creación de una instancia del programa al cual pertenece el método que se desea invocar, por ejemplo, una tarea común es invocar métodos del programa `EkitEdita.java` (interfaces de usuario) desde el programa `EkitCore.java` (centro de control de los eventos del editor visual). Para crear la instancia del programa `EkitEdita.java` se escribe el siguiente código en el programa `EkitCore.java` en el punto donde se desee hacer la invocación.

```
EkitEdita ekitEdita = new EkitEdita();
```

Una vez creada la instancia, ésta se utiliza por medio de la variable `ekitEdita`. Por ejemplo, para invocar un método se utiliza un código de la forma:

`ekitEdita.` <**método**>

donde <**método**> es el nombre del método que se desea invocar.

Para invocar métodos pertenecientes al programa `EkitCore.java` desde el programa `EkitEdita.java` se siguen los mismos pasos de forma análoga.

Si al invocar a un método de otro programa se pasa como parámetro el argumento *this*, ya no es necesario crear la instancia del programa fuente para invocar a sus métodos, pues se está pasando como parámetro una instancia del programa. Esto resulta muy útil cuando desde un programa *fuente* se invoca un método de un programa *auxiliar* que se sabe invocará a su vez un método del programa *fuente*.

Esto es necesario porque en principio Java sigue la filosofía de abstracción de datos que impide tener acceso a los componentes de un objeto de manera directa. Lo que se logra con el esquema que

se ha explicado es hacer que la comunicación entre programas sea más eficiente. Es importante tener esto en cuenta para poder entender el código fuente de los programas.

5.1.4. Programas *CGI*

Sin duda una parte muy importante para el funcionamiento del editor visual son los programas *CGI* encargados de atender las peticiones que éste manda al servidor. Estos programas se crearon tomando como modelo los ejemplos en lenguaje `C` que aparecen en la referencia [16], la relación de los mismos aparece en el cuadro 5.2.

`califica1`	Califica las respuestas dadas por el usuario
`verilatex`	Verifica que la sintaxis LaTeX de un MATHIMG sea correcta
`writefile`	Guarda un archivo de especificación en el servidor
`compilefile`	Traduce el archivo de especificación activo

Cuadro 5.2: Programas *CGI* encargados de atender las peticiones del editor visual

En las siguientes subsecciones se explica con más detalle el funcionamiento de cada uno de estos programas.

Programa `califica1`

Programa *CGI* encargado de calificar las respuestas del navegante, envía a su salida estándar un archivo *HTML* con la relación de aciertos y la calificación obtenida.

Programa `verilatex`

Programa *CGI* encargado de verificar que una expresión LaTeX incrustada en una instancia de la directiva MATHIMG sea sintácticamente correcta. Esencialmente este programa construye un archivo .tex que contiene la expresión que se desea validar y después compila este archivo invocando al compilador de LaTeX. El programa genera la cadena "Expresión Latex correcta" en caso de que la expresión dada sea sintácticamente correcta o bien una cadena con los errores devueltos por el compilador de LaTeX.

Programa `writefile`

Una de las funciones básicas del editor visual para **e-Edita** es guardar las páginas de especificación que generan los usuarios, esta tarea la lleva a cabo el programa *CGI* `writefile`.

Programa `compilefile`

Para ver el resultado de una especificación como página *HTML*, es necesario traducir el archivo que la contiene, esto se logra con el programa *CGI* `compilefile`.

5.1.5. Íconos utilizados

Los íconos utilizados en el editor visual de **e-Edita** (ver sección 4.4) fueron creados utilizando el programa "The GIMP" que viene con la distribución 8.0 de RedHat Linux, éstos tienen las siguientes propiedades:

Siempre que se quiera crear nuevos constructores en los lenguajes de especificación de **e-Edita** es importante diseñar íconos apropiados que deben tener las características mostradas en el cuadro para conservar la uniformidad en la apariencia del editor visual. Las acciones asociados con estos íconos deben ser programadas tomando en cuenta las consideraciones que al respecto se hacen en el capítulo 4.

Formato de imagen tipo GIF
Tamaño de imagen de16×16 pixeles
Formato RGB de 256 colores
Tamaño de archivo promedio de 100 Bytes

Cuadro 5.3: Características de los íconos de **e-Edita**

5.1.6. Instrumentación del editor visual

El editor visual está basado en el programa de software libre *Ekit* [11], el cual originalmente fue diseñado para editar páginas *HTML* en Internet por medio de un *Applet*. Como se explicó en la sección 5.1.2, el archivo encargado de controlar el funcionamiento interno del editor visual es el programa `EkitCore.java`, el cual sufrió varias modificaciones para adaptarlo a nuestras necesidades, entre ellas: se eliminaron las facilidades para introducir código *HTML*, se redefinió el modo de trabajo con archivos (abrir, guardar y crear nuevo) y por último se agregó la funcionalidad necesaria para trabajar con **e-Edita**. En concreto se hicieron las siguientes extensiones:

E.1.	Ampliación del menú para introducir los constructores de **e-Edita**
E.2.	Creación de la barra de herramientas con los constructores de **e-Edita**
E.3.	Programación para atender los eventos provenientes del menú y de la barra de herramientas de **e-Edita**
E.4.	Programación de los métodos para invocar programas *CGI's*

Cuadro 5.4: Extensiones al programa `EkitCore.java`

En la extensión E.1 se agregaron a la barra de menús las opciones "Página" y "Libro" que contienen los constructores necesarios de **e-Edita** para especificar páginas o libros electrónicos. Los pasos involucrados en esta extensión son:

E.1.1 **Definición los nuevos menús:** Se logró declarando dos instancias del componente *JMenu*, una para "Página" llamada `jMenuPage` y la otra para "Libro" llamada `jMenuBook`.

```
private JMenu jMenuPage;
private JMenu jMenuBook;
```

E.1.2 **Definición de cadenas para identificar los menús:** Esto con la finalidad de invocarlos desde el *Applet*

```
public static final String KEY_MENU_PAGE   = "page";
public static final String KEY_MENU_BOOK   = "book";
```

E.1.3 **Creación del componente** *JMenu***:** En éste se incluyó el título ("Página" o "Libro") que debía aparecer en la barra de menús

```
jMenuPage      = new JMenu("Página");
jMenuBook      = new JMenu("Libro");
```

E.1.4 **Creación de los elementos del menú:** En este último paso se definió y creó el menú y una instancia de *JMenuItem*, se asoció la acción que atiende los eventos así como un acceso rápido y se añadió al menú correspondiente. En el ejemplo se muestra lo que se hizo para agregar la directiva TITLE.

```
JMenuItem jmiTitle   = new JMenuItem("Titulo");
jmiTitle.setActionCommand("insertatitle");
jmiTitle.addActionListener(this);
jmiTitle.setAccelerator(KeyStroke.getKeyStroke('T',
                        java.awt.Event.ALT_MASK, false));
if(showMenuIcons)
  jmiTitle.setIcon(getEkitIcon("Title"));
jMenuPage.add(jmiTitle);
```

En el programa `EkitCore.java` después del comentario

```
/* eEdita Menu */
/*)(*/
/* Create the Page menu */
```

aparecen las acciones que se llevaron a cabo para incorporar cada uno de los constructores a sus respectivos menús.

En la extensión E.2 se creó la barra de herramientas de **e-Edita**, la cual contiene los íconos asociados a sus constructores. Los pasos que se siguieron son:

E.2.1 **Definición de la barra de herramientas:** Se logró declarando una instancia del componente *JToolBar* con el nombre `jeEditaToolBar`

```
private JToolBar jeEditaToolBar;
```

E.2.2 **Creación de la barra de herramientas:** Se especificó la orientación de la barra y sus propiedades: una barra horizontal que no se puede mover

```
jeEditaToolBar = new JToolBar(JToolBar.HORIZONTAL);
jeEditaToolBar.setFloatable(false);
```

E.2.3 **Creación de los botones que conforman la barra:** En el ejemplo se muestra como se definió el botón para la directiva TITLE. Primero se creó el botón. Luego se le asoció un *ToolTip*, la acción a ejecutarse al presionar el botón y un *Listener*. Finalmente se añadió a la barra de herramientas y se asoció el botón con un identificador para invocarlo desde el *Applet*.

```
jbtnTitle        = new JButtonNoFocus(getEkitIcon("Title"));
jbtnTitle.setToolTipText("Directiva TITLE");
jbtnTitle.setActionCommand("insertatitle");
jbtnTitle.addActionListener(this);
jeEditaToolBar.add(jbtnTitle);
htTools.put(KEY_TOOL_TITLE, jbtnTitle);
```

En el programa `EkitCore.java` después del comentario

```
/* Create the e-Edita toolbar */
/*)(*/
```

aparecen las acciones que se llevaron a cabo para definir los botones asociados con los constructores e incorporarlos a la barra de herramientas de **e-Edita**.

La extensión E.3 consistió en programar los elementos encargados de atender las peticiones generadas desde la barra de herramientas o los menús de **e-Edita**. Sólo fue necesario realizar un paso:

E.3.1 **Modificación del método** *actionPerformed* Se modificó el método *actionPerformed(ActionEvent)* agregando un código que identifica el comando que se desea ejecutar así como las acciones asociadas. La parte substancial del código agregado es la siguiente:

```
String command = ae.getActionCommand();
EkitEdita ekitEdita = new EkitEdita();
int caretPos = jtpMain.getCaretPosition();

 else if(command.equals("insertatitle"))
  {
   styDoc.insertString(0, "\n\n", jtpMain.getInputAttributes());
   jtpMain.setCaretPosition(0);
   ekitEdita.editTEWindow("TITLE", this);
  }
```

En la primera línea se obtiene el identificador del comando involucrado en la petición, en la segunda línea se crea una instancia del programa `EkitEdita.java` para invocar los métodos de ese programa y en la tercera línea se obtiene la posición actual del cursor dentro del área de edición. En las líneas que vienen a continuación se identifica el comando `insertatitle` y se procede, en caso necesario, a ejecutar el cuerpo del `if` cuya primera línea inserta dos retornos de carro en el editor visual y la segunda línea posiciona el cursor al inicio del área de edición. Al final, se invoca el método *editTEWindow(String, EkitCore)* del programa `EkitEdita.java` que se encarga de mostrar la interfaz gráfica para construir la directiva TITLE.

El código asociado con las interfaces gráficas (*Heading*, MATHIMG, etc) del resto de los constructores tiene una estructura similar a la que se ha mostrado.

Por último, la extensión E.4 consistió en construir los métodos encargados de invocar a los programas *CGI* ubicados en el servidor. Su propósito es establecer la conexión con el servidor. Los pasos esenciales para hacerlo son:

E.4.1 **Creación de la instancia** *URL***:** Java contiene en sus bibliotecas objetos especiales (*URL*) para establecer conexiones de Internet. En estos objetos se especifica la dirección *URL* del sitio que se desea contactar.

E.4.2 **Creación de la instancia URLConnection:** Formando también parte de las bibliotecas de Java se tienen objetos pertenecientes a la clase *URLConnection* que permiten conectarse con el sitios especificado en una instancia de tipo *URL*.

En los método `writeFile(String, String)` y `compileFile(String, String)` aparecen ejemplos sobre cómo crear las instancias descritas en los pasos E.4.1 y E.4.2. Para presentar la metodología seguida se presentan fragmentos del código del método `writeFile(String, String)`. El método `compileFile(String, String)` tiene una estructura análoga.

E.4.3 **Establecimiento de la conexión con el servidor e invocación del programa *CGI*:** En el método `writeFile(String, String)` se programaron los comandos para establecer la conexión y para llamar al programa *CGI* `writefile`. La parte medular de este código es:

```
// Parte A
  String cgibin = "cgi-bin/writefile?textfile=";
  String espec = "&espec=/home/imendez/public_html/"+
                  EkitApplet.strUserName+"/espec";
  String filename = "&filename="+strFileName;

// Parte B
  strEncodedFileStr = URLEncoder.encode(strFile, "ISO-8859-1");

//Parte C
  URL urlAlaska = new URL("http://alaska.math.cinvestav.mx");
  URL urlWriteFile = new URL(urlAlaska, cgibin+
                                        strEncodedFileStr+
                                        espec+filename);

  URLConnection urlAlaskaConnection =
               urlWriteFile.openConnection();
```

En la `Parte A` se crean cadenas con la dirección del programa *CGI*, la primera variable del programa, el directorio del usuario y la variable donde se almacena un nombre de archivo. En la `Parte B` se codifica el texto del archivo en formato ISO-8859-1 que permite que caracteres especiales (caracteres acentuados y eñes) viajen por la Internet. En la `Parte C` se crean las instancias de la *URL* del servidor y de la conexión al mismo.

En general los programas *CGI* permiten construir el código que en el servidor instrumenta la funcionalidad de una aplicación accesible vía Internet. Si bajo alguna circunstancia fuese necesario agregar una nueva característica funcional de este tipo al editor visual entonces sería necesario respetar la estructura de programación que se ha explicado para garantizar la coherencia del sistema.

5.1.7. Instrumentación de las interfaces para el usuario

Como se explicó en la subsección anterior, para crear el editor visual de **e-Edita** se tomó como base el programa `Ekit` [11], al cual (entre otras modificaciones) se le agregó el programa `EkitEdita.java` donde se encuentran instrumentadas todas las interfaces. Una explicación detallada sobre cómo instrumentar estas interfaces para el usuario se da en la sección 6.6.

Entre estas interfaces la más interesante es la que corresponde a la directiva MATHIMG por su complejidad y generalidad, por esta razón se presenta la explicación de los pasos que se siguieron para conformar su interfaz del usuario (mostrados en el cuadro 5.5). Los demás constructores fueron tratados de manera parecida.

En el paso I.1 se declararon las variables necesarias para obtener los parámetros relativos a la geometría de la imagen; y los componentes que conforman la interfaz gráfica como íconos y en su mayoría páneles.

I.1.1 **Declaración de variables de tipo `int`:** Estas tienen la finalidad de almacenar la posición de los parámetros *density, quality, crop, border, colorize* y *negate*.

I.1.	Declaración de variables y componentes locales
I.2.	Verificación de la variable `bEditConstructor`, la cual indica si trata un caso de edición de directiva.
I.3.	Creación de la lista de opciones para manipular la geometría de la imagen
I.4.	Programación de los componentes *JSpinner* y *JSlider*
I.5	Creación de las instancias de los componentes locales
I.6	Configuración de los páneles para: edición de directiva, modificación de parámetros de la geometría y cambio de color
I.7	Asociación de comandos a los botones de la ventana principal y exhibición de la misma

Cuadro 5.5: Pasos para construir la interfaz de la directiva MATHIMG

I.1.2 **Declaración de componentes tipo `ImageIcon`:** Los cuales sirven para guardar los íconos que se muestran en la pestaña de los páneles.

I.1.3 **Declaración de vectores:** En estos vectores se almacena la lista de opciones de los parámetros *density, crop* y *border.*

I.1.4 **Declaración de constantes:** Utilizadas para dar valor al parámetro *quality*, tienen la finalidad de indicar el valor mínimo, máximo y promedio de la calidad de la imagen.

I.1.5 **Declaración de páneles:** Se divide en dos partes, los páneles que forman las pestañas principales de la interfaz gráfica y los páneles donde se piden los valores de los parámetros al usuario.

En el paso I.2 se verifica el valor de la variable *booleana* `bEditConstructor` para saber si se está editando al constructor o si se trata de una instancia nueva del mismo. Si es un caso de edición se guardan los valores de la directiva para que más adelante sean presentados al usuario.

I.2.1 **Almacenamiento el texto LaTeX anterior:** El texto anterior se almacena en la variable *strLatex* por medio del método *parseLatexString(String)*

```
strLatex = parseLatexString(strLatex);
```

I.2.2 **Obtención de la posición los parámetros:** Esto con la finalidad de verificar que el parámetro exista, la posición se determina con el método *getIndexInOption(String).*

I.2.3 **Verificación del rango de valores de *quality*:** Para evitar que el usuario inserte un valor que no es apropiado para el parámetro *quality.*

```
// Convierto un tipo String a un int
QLTY_INIT =  (Integer.valueOf(arrFields[nQuality+1])).intValue();
if ( QLTY_INIT > 100 )
  QLTY_INIT = 100;
if ( QLTY_INIT < 0)
  QLTY_INIT = 0;
```

En el paso I.3 se crearon listas de opciones para los parámetros *density, crop* y *border* que son parte de la lista de parámetros para manipular la geometría de la imagen. También se crearon los componentes para los parámetros *colorize* y *negate.*

I.3.1 **Creación de la lista de valores:** Si la variable *nDensity* tiene un valor diferente a -1 quiere decir que existe un valor predefinido para este parámetro, en caso contrario se crea una lista de posibles valores para los parámetros con la finalidad de que el usuario experimente el resultado que obtiene al utilizar cada uno de ellos.

```
if ( nDensity != -1)
   vcDensity.add(arrFields[nDensity+1]);
vcDensity.add("110x110");    vcDensity.add("110x120");
vcDensity.add("120x110");    vcDensity.add("120x120");
vcDensity.add("130x110");    vcDensity.add("130x120");
```

I.3.2 **Creación de los componentes tipo** *ComboBox***:** Ahora que se tiene la lista de opciones es necesario colocarla en un *ComboBox* para que el usuario pueda visualizarla:

```
jcbDensity = new JComboBox(vcDensity);
```

I.3.3 **Modificación del** *ComboBox***:** Por último, para que el usuario tenga la posibilidad de escribir un valor que no aparezca en la lista se modifica la propiedad *setEditable* del *ComboBox*.

```
jcbDensity.setEditable(true);
```

Estos pasos se aplican de forma análoga para *crop* y *border*.

En el paso I.4 se programaron los componentes que permiten ajustar el valor del parámetro *quality* visualmente, ya sea con una barra de desplazamiento (componente *JSlider*) o bien con el teclado, particularmente con las flechas del cursor (componente *JSpinner*).

I.4.1 **Creación de la barra de desplazamiento:** Con esta barra se puede ajustar el valor del parámetro *quality* utilizando el ratón.

```
jslQuality = new JSlider(JSlider.HORIZONTAL,
                         QLTY_MIN, QLTY_MAX, QLTY_INIT);
jslQuality.addChangeListener(jsListener);

jslQuality.setMajorTickSpacing(25);
jslQuality.setMinorTickSpacing(5);
jslQuality.setPaintTicks(true);
jslQuality.setPaintLabels(true);
```

I.4.2 **Creación del** *Spinner***:** Este componente permite hacer la modificación utilizando las flechas del cursor.

```
jspQuality = new JSpinner(new SpinnerNumberModel(
                          QLTY_INIT, QLTY_MIN, QLTY_MAX, 1));
jspQuality.addChangeListener(jspListener);
currentSpinnerType = QUALITY_SPINNER;
quality = QLTY_INIT;
```

En el paso I.5 se crearon instancias de los componentes declarados en el paso I.1 y se les agregaron los elementos definidos en los pasos I.3 e I.4.

I.5.1 **Construcción de páneles:** Donde el usuario introduce los valores de los parámetros de la directiva MATHIMG, cada uno de estos páneles está conformado por dos componentes, una etiqueta que indica el parámetro que se está editando y un componente editable para asignarle un valor.

```
jpDensity.add(new JLabel("Densidad: "));
...
jpQuality.add(jslQuality);
jpDensity.add(jcbDensity);
```

I.5.2 **Integración de los páneles:** Todos los páneles creados en el paso anterior deben de colocarse en un solo contenedor para crear la pestaña correspondiente a la edición de parámetros.

En el paso I.6 se configuraron los páneles que se han de agregar a la ventana principal de la interfaz en forma de pestañas.

```
//Agrego las pestañas
jtpLatex.addTab("Directiva", iconLatex, jpDirectiva,
                "Contenido de la directiva");
jtpLatex.addTab("Parámetros", iconParameters, jpParametros,
                "Parámetros de la directiva");
jtpLatex.addTab("Color", iconColor, jpColorChooser,
                "Selector de color");
jtpLatex.addTab("Verificación", iconOK, jpCheckResult,
                "Resultado de la verificación del texto Latex");
```

Por último, en el paso I.7 se crean los botones que aparecen en la ventana principal de la interfaz, se les asocian los identificadores de comando correspondientes y se muestra la ventana al usuario.

I.7.1 **Creación de botones:** Que controlan a la interfaz gráfica presentada al usuario.

```
vcButtons.add(jbtnSend  =customButton("Enviar","returnlatex"));
vcButtons.add(jbtnCancel=customButton("Cancelar","cancel"));
```

El primer botón creado tiene asociado el comando "`returnlatex`" que se encarga insertar la directiva en el editor visual y el segundo tiene asociado el comando "`cancel`" que cierra la ventana ignorando los cambios.

I.7.2 **Visualización de la interfaz:** Por último se agregan a la interfaz los botones creados en el paso anterior, la ventana se hace visible

```
jpButtons = customButtonPanel(vcButtons);
jfMainFrame.getContentPane().add(jpButtons, BorderLayout.SOUTH);
jfMainFrame.show();
```

En esta directiva es necesario verificar la sintaxis del texto en LaTeX asociado; el programa *CGI* `verilatex` se encarga de realizar esta tarea. Los pasos involucrado en la invocación son:

1. Crear una cadena con la ubicación del programa *CGI* y el nombre de la primera variable que recibe este programa

```
String cgibin = "cgi-bin/verilatex?latexexp=";
```

2. Crear otra cadena donde se especifique el segundo parámetro que recibe el programa, éste indica el directorio del usuario

```
String espec = "&espec=/home/imendez/public_html/"+
               EkitApplet.strUserName+"/espec";
```

3. Codificar con formato UTF-8 la expresión LaTeX para que pueda viajar por Internet.

```
strLatexExp = URLEncoder.encode(jtaLatex.getText(), "UTF-8");
```

4. Crear una instancia de la *URL* del servidor

```
URL urlAlaska = new URL("http://alaska.math.cinvestav.mx");
```

5. Crear una instancia de la *URL* del programa *CGI*

```
URL urlVeriLatex = new URL(urlAlaska, cgibin+strLatexExp+espec);
```

6. Crear la conexión a la *URL* especificada en el paso anterior

```
URLConnection urlAlaskaConnection = urlVeriLatex.openConnection();
```

7. Crear una instancia de la clase *BufferedReader* para leer la salida que arroje el programa

```
BufferedReader in = new BufferedReader(new InputStreamReader
                    (urlAlaskaConnection.getInputStream()));
```

5.1.8. Edición de los constructores

El método encargado de atender las peticiones del usuario para la edición de constructores es *editConstructor(String, int, EkitCore)* en el cual se realizan las acciones necesarias para modificar los constructores previamente insertados en el editor visual, la estructura general del método es la siguiente:

```
editConstructor(String strConstructor, int caretPos,
                EkitCore ekitcore)
 {
  // Parte A
   Definicion de variables locales...
   if (strConstructor == null)
     {
       strParagraph = ekitcore.getParagraph(caretPos);
       newCaretPos = ekitcore.newPos;
       if (Es una Directiva)
          strConstructor = strParagraph;
       if (Es un especificador de reactivos)
```

```
            Posicion del cursor al inicio del especificador
        }

    // Parte B
    bEditConstructor = true;
    oldDirectiveLength = strConstructor.length();
    getFields(strConstructor, strSeparators, true,
              strParentesis);

    // Parte C
    if ( arrFields[0].compareToIgnoreCase("newConstructor")==0 )
      newMethodWindow("newConstructor", ekitcore);
    if ( arrFields[5].compareToIgnoreCase("newConstructor")==0 )
      newMethodWindow("newConstructor", ekitcore);
}
```

Este método trabaja así: En la `Parte A` después de declarar todas las variables locales que serán utilizadas se pregunta si el parámetro de tipo *String* llamado *strConstructor* es nulo, lo que indica que no se ha resaltado el constructor en el editor visual y en tal caso se asume que el cursor está posicionado sobre un constructor, entonces el método se encarga de delimitar al mismo, en caso de que la cadena no sea nula se asume que se ha pasado como parámetro el constructor que se desea modificar.

En el primer caso (cuando no se ha pasado como parámetro el constructor) es necesario aislarlo del párrafo en el cual se encuentra el cursor. Esto se hace con el método *getParagraph(int)* definido en el programa `EkitCore.java`, el parámetro de tipo *int* que se pasa al método sirve para indicar la posición del cursor dentro del texto, misma que indica el párrafo a procesar, una vez obtenido el párrafo surgen dos posibilidades, la primera, que se trate de una directiva y la segunda que sea un especificador de reactivos. En caso de ser una directiva se sabe por lo descrito en la sección 2.6.1 que ésta ocupa un párrafo completo; por tanto el párrafo aislado contiene la especificación de la directiva que se está buscando, así párrafo es almacenado en la variable *strConstructor*. En caso de ser un especificador de reactivos, se sabe por lo descrito en la sección 2.6.2 que está delimitado al inicio y al final por barras verticales (|) las cuales sirven de guia para saber donde empieza y donde termina el especificador de reactivos. Para delimitar el especificador de reactivos se emplea el método *getFields,* una vez que se ha delimitado el especificador de reactivos su valor el es guardado en la variable *strConstructor*.

Ahora se tiene almacenado en la variable *strConstructor* el constructor que ha de ser modificado, en el siguiente paso (`Parte B`) se enciende la bandera llamada *bEditConstructor* que indica a los demás métodos que se está editando un constructor. Se calcula la longitud de la cadena que contiene al constructor (instrucción *oldDirectiveLength = strConstructor.length()*) con la finalidad de pasarlo como parámetro al método que escribirá el constructor modificado en el editor visual, este parámetro indica la cantidad de caracteres que han de ser borrados del editor visual antes de insertar el nuevo constructor.

Para poder mostrar al usuario una ventana de edición del constructor es necesario obtener los valores correspondientes a cada uno de los nombres de atributo del constructor que se esté modificando, esto se logra aplicando el método *getFields(String, String, boolean, String)* a la variable *strConstructor*, para separar los nombres de atributo de sus valores.

Una vez que se ha aplicado el método *getFields*, se pasa a la `Parte C` en la cual, se sabe que se está editando una directiva porque el nombre de la misma esta almacenado en la variable *arrFields[0]*, o en caso de estar editando un especificador de reactivos su tipo estará almacenado en *arrFields[5]*. Sin importar el tipo de constructor que se esté editando llama a la interfaz de usuario correspondiente al constructor especificado, ya sea con el nombre de la directiva o bien con el tipo del especificador.

La última parte en el proceso de edición de los constructores se realiza dentro de los métodos correspondientes a las interfaces gráficas de los mismos, dentro del método se verifica el valor de la bandera *bEditConstructor*, si el valor es verdadero se trata de un caso de edición del constructor, por

tanto al momento de crear los campos de interacción para el usuario (que generalmente son campos de texto del tipo *JTextField*) no se deben de crear en blanco, sino que deben de tomar como valor de inicio el contenido de *arrFields* en el campo correspondiente al nombre del atributo pertinente.

5.2. Funcionamiento de e-Edita

En esta sección se describe de forma general cómo funciona el traductor de **e-Edita**: el programa `pagina.c`, el cuál, básicamente es un traductor del lenguaje de especificación de **e-Edita** a código *HTML*.

Éste programa instrumenta a un autómata que realiza el análisis sintáctico de un archivo de especificación (`.e`), analizando cada constructor (con ayuda de la función *getFields* que se describe más adelante) y realizando la traducción de dicha especificación al código *HTML* correspondiente.

5.2.1. El método *getFields*

Una de las funciones más importantes del programa `pagina.c` es `getfields` que se encargada de separar los parámetros que conforman a los constructores en *tokens*, a continuación se explica de forma general el funcionamiento de la misma.

Primero es importante conocer los parámetros que recibe la función, para ello se muestra su prototipo:

```
void getFields(char *line, char field[][], int *n, char *separador, char *parenthesis)
```

donde:

`char *line`: Es la cadena que contiene al constructor que se está analizando.

`char field[][]`: Es el arreglo donde se almacenan los parámetros que conforman al constructor.

`int *n`: Es un contador que indica cuantos parámetros hay almacenados en el arreglo `field[][]`.

`char *separador`: Es una cadena en la que se especifican los caracteres que han de emplearse como separadores.

`char *parentesis`: Esta cadena contiene los caracteres que son empleados como paréntesis.

Separa la cadena `line` usando como separador a cualquiera de los caracteres contenidos en la cadena `separador` los cuales se consideran como separadores independientes Si la variable paréntesis es diferente de `'\0'` entonces una vez que aparezca el primer carácter paréntesis todos los demás caracteres se dejan en una misma separación hasta que llegue otro carácter paréntesis.

Por ejemplo, si llegar la secuencia:

|a|b|c$a|l|g|o$d|e|f

y se invocara esta función con *separador="|", paréntesis="$"entonces se obtendría la separación:

a b c$a|l|g|o$d e f

mientras que con separador="|", paréntesis='\0'se obtiene:

a b c$a l g o$d e f

El análisis sintáctico y la traducción a código *HTML* se basan en la función `getFields`, la cual, debido a la simplicidad de la sintaxis del lenguaje, lleva a cabo su tarea de forma muy eficiente, permitiendo así la traducción de especificaciones el línea. Ésto resulta muy útil en la evaluación de cuestionarios ya que permite generar rápidamente páginas con la calificación del usuario, además, los usuarios que desarrollen aplicaciones pueden generar una vista previa de su trabajo desde una máquina remota en tiempos razonablemente cortos.

Capítulo 6

Extensión de las capacidades del editor visual

En este capítulo se explica el proceso para agregar nuevas construcciones al lenguaje de especificación de **e-Edita**. Este proceso es fundamental porque a partir de él se pueden extender las capacidades funcionales del lenguaje; asimismo es importante porque todas las herramientas de software que se han construido, incluyendo al editor visual, pueden ser aplicadas en contextos que rebasan con mucho al educativo. Así siguiendo los métodos expuestos en este capítulo es posible construir entornos de especificación para el desarrollo de aplicaciones accesibles vía Internet en ámbitos como: generación y procesamiento de encuestas, procesamiento de formatos, comercio electrónico, etc.

Nos concentramos fundamentalmente en las modificaciones que se deben de realizar en los programas de **e-Edita**, presentamos un ejemplo sencillo sobre cómo llevar a cabo estas modificaciones para incorporar algunas construcciones nuevas y damos algunos criterios útiles para el diseño de las mismas.

Cabe señalar que este capítulo está presentado desde un punto de vista muy técnico, es decir, está destinado a aquellas personas que desean hacer modificaciones al editor visual o bien al traductor de **e-Edita**. Preferentemente, quien lea este capítulo debe tener conocimientos previos del lenguaje Java y haber hecho una lectura de los diferentes programas que conforman el sistema (`pagina.c`, `EkitCore.java`, `EkitEdita.java` y `EkitApplet.java`). Por razones de espacio no se incluyen todos los detalles de la programación, sin embargo, se da una explicación general de las modificaciones que deben llevarse a cabo y se bosquejan algunos ejemplos de éstas.

6.1. Planteamiento de construcciones nuevas en e-Edita

Como se explicó en la sección 2.6 en **e-Edita** hay dos construcciones sintácticas fundamentales: las *directivas* y los *especificadores de reactivos*. Las primeras permiten introducir construcciones de *HTML* estáticas mientras que las segundas sirven para especificar campos interactivos dinámicos; es decir, aquellos que utiliza el usuario para responder sus cuestionarios.

Cada una de estas construcciones tiene una forma específica para ser incorporada en **e-Edita** y tiene sus propias consideraciones de diseño.

6.2. Incorporación de directivas

De acuerdo a lo explicado en la sección 2.6 las directivas siguen un modelo estático posicional fijo, donde los atributos se escriben en secuencia, separados por el carácter ç. El primer atributo es un identificador compuesto solamente por caracteres alfabéticos que se asocian de manera única con cada directiva.

¿Cómo diseñar la sintaxis de una directiva? En principio quien defina la nueva sintaxis tiene la libertad de hacerlo como desee siempre que respete las convenciones dadas en la sección 2.6. Sin

embargo, en el diseño de las construcciones que actualmente tiene **e-Edita** se han seguido ciertos esquemas que resulta provechoso conocer.

1. El paso fundamental que decide la pertinencia de una directiva es saber si ésta aporta realmente los elementos que se desean dentro del contexto de la aplicación. Por ejemplo, en la enseñanza del idioma inglés es común el uso de crucigramas para enseñar a los alumnos palabras nuevas, hasta el momento en **e-Edita** no se pueden especificar crucigramas utilizando las directivas existentes, por tanto, en este contexto sería necesario agregar una directiva que permita especificar crucigramas, la cual tendría un nombre apropiado, digamos *crossword*[1].

2. El conjunto de atributos debe ser elegido de tal forma que tengan una interpretación estricta en los términos funcionales de la directiva. Por ejemplo, para especificar una pregunta de opción múltiple la sintaxis que se explica en la sección 3.3 utiliza sólo elementos funcionales básicos (la lista de opciones) sin tener que indicar si va a utilizar una tabla *HTML* específica o no. La idea es parecida a lo que se hace en LaTeX donde un autor se enfoca en la definición general de los tipos de texto y su contenido sin decir exactamente cómo se verán estos en el texto impreso final.

3. El procesamiento de la directiva debe de poder hacerse utilizando herramientas existentes o que se puedan construir en UNIX, de manera que todos los atributos que se requieran pasar como parámetros a un comando de UNIX deben de colocarse como un solo atributo. Por ejemplo, en el diseño de la directiva MATHIMG se tenía que especificar un gran número de parámetros para indicar la geometría y presentación de la imagen final; bien pudieron haberse definido atributos como COLOR_DEL_FONDO, COLOR_DEL_BORDE, CALIDAD_JPG, ANCHO_BORDE, etc., sin embargo esto hubiera complicado innecesariamente el uso de la directiva. Dado que se decidío utilizar el comando "convert" (disponible en UNIX y Linux) sólo se requiere pasar un parámetro con todos los argumentos necesarios para especificar la geometría de la imagen (ver 3.2.5).

Desde el punto de vista técnico, incorporar una directiva nueva implica los siguientes pasos:

Análisis sintáctico de la directiva y transformación a *HTML*. Esto se refiere al proceso para identificar a la nueva directiva dentro de un archivo de especificación (.e) y transformar lo especificado a *HTML* para construir la página final. Las modificaciones relacionadas con este proceso se realizan dentro del programa `pagina.c`.

Programación de la interfaz de usuario en el editor visual. Esto tiene que ver con los cambios necesarios al programa `EkitCore.java` para incorporar un nuevo ícono dentro del editor visual, el sub-programa interactivo para que el usuario defina instancias de la directiva y su transformación correspondiente al lenguaje de especificación.

Siendo más puntuales, la realización de estos pasos involucra modificar algunos de los programas que conforman a **e-Edita**, afortunadamente el diseño modular de los mismos permite que las modificaciones afecten solamente programas muy bien identificados:

- `pagina.c`. Traductor del lenguaje de especificación de **e-Edita**.
- `EkitCore.java`. Programa de control del editor visual. Se encarga de coordinar el funcionamiento de los menús y las barras de herramientas.
- `EkitEdita.java`. Conjunto de interfaces gráficas para la especificación interactiva de las formas sintácticas de **e-Edita**.
- `EkitApplet.java`. *Applet* para acceder desde un navegador de Internet al editor visual junto con sus componentes básicos.

En lo que sigue se exponen, de manera detallada, las modificaciones que se deben hacer en cada uno de estos programas.

[1] Optamos por usar palabras en Inglés para evitar confusiones con las palabras del Español; de esta manera queda claro al usuario que estas palabras tienen un uso especial dentro del lenguaje de especificación

6.2.1. Modificaciones al traductor de e-Edita para incorporar directivas

El programa `pagina.c` está escrito en lenguaje ANSI C y, como todo traductor, se encuentra dividido en dos partes:

- El código para realizar el análisis sintáctico y
- la parte dedicada a la asociación de las construcciones especificadas con su código *HTML* equivalente.

Ambas partes son particularmente simples debido a la estructura rígida del lenguaje de especificación 2.2, por lo que una revisión general del programa principal de `pagina.c` permite identificar cada una de estas partes así como los paradigmas básicos de programación que se han utilizado.

La simplicidad es una virtud que nos permite exponer la relación de cambios al programa `pagina.c` sin tener que hacer una justificación detallada de los mismos:

1. **Incrementar en uno la variable** *NTOKENS* la cual sirve para indicar cuantos elementos sintácticos se han declarado en el programa.
2. **Agregar identificador** Al final del arreglo *tokens*, es importante que sea al final y no en cualquier otro lugar del arreglo porque a cada *token* se le asigna un número consecutivo para identificarlo.
3. **Declarar el nuevo token** Con la estructura de código:

   `#define` identificador *No.*

   donde identificador es el nombre de la directiva y *No.* es el número consecutivo relacionado al identificador.
4. **Código del nuevo identificador** En el programa principal en la instrucción:

   switch(token)

   agregar

   `case` identificador: <**Código**>

   donde identificador es el nombre de la directiva y <**Código**> es el código encargado de procesar la directiva. Este código puede aprovechar las variables que se presentan en el cuadro 6.1 que son accesibles desde el programa principal.

nfields	Indica el número de atributos en la directiva
field[i]	con $i = 0, \ldots, nfields - 1$
	El valor del atributo i-ésimo en la directiva

Cuadro 6.1: Variables del programa principal usadas en *switch(token)*

En general lo que se hace es una traducción literal al código *HTML*. En caso de que se requiera hacer una construcción más compleja en *HTML* como crear o modificar una imagen, hacer una consulta a una base de datos o alguna otra tarea parecida, es mejor buscar un programa del sistema operativo UNIX dedicado a realizar estas tareas, o bien, generar un programa propio, en ambos casos se recomienda invocar los programas de manera directa con los procedimientos *popen* o *system*.

6.3. Incorporación de especificadores de reactivos

En la sección anterior se mostraron los pasos que se deben seguir y las modificaciones para incorporar una nueva directiva a **e-Edita**, en esta sección se muestran las modificaciones pertinentes para incorporar un nuevo especificador de reactivos. Recordemos de la sección 2.6.2 la sintaxis de un especificador de reactivos:

$$|\text{nombreAtributo1, valor1, nombreAtributo2, valor2,}\cdots|$$

Crear un nuevo especificador de reactivos significa construir un nuevo valor para el atributo *type*. Así, lo esencial para incorporar un nuevo reactivo es interpretar su tipo y los parámetros que lo acompañan.

La decodificación e interpretación de los reactivos se lleva a cabo en el procedimiento *interactionField* del programa `pagina.c`. El diseño de *interactionField* es muy interesante porque el mismo código que sirve para generar los reactivos en *HTML* sirve para calificarlos. Este comportamiento es posible debido a que el programa *CGI* encargado de recibir las peticiones para calificar una página proveniente del navegante (programa `califica1.out`) construye la especificación de la página que se ha de regresar e invocar al programa `pagina.c` junto con la lista de asignaciones a los campos interactivos asociados con los reactivos. De manera más precisa, para interpretar correctamente el nuevo reactivo es necesario hacer lo siguiente.

- Crear y dar un nombre adecuado al nuevo tipo de reactivo
- Verificar si los atributos existentes (especificados en la tabla 6.2) son suficientes para el nuevo tipo de especificador (hay que entender que estos nombres tienen un significado especial bien definido).
- Si hay que crear un nombre de atributo nuevo, se tiene que ver de qué clase es. Existen dos clases: los *simples* que tienen un sólo valor como *hint* y *answer*, y los *compuestos* que tienen una lista de longitud variable de valores como *options* y *answers*.
- Modificar la función *interactionField* para que sea capaz de reconocer el nuevo tipo de especificador de reactivos.

Cada uno de estos pasos involucra modificar diferentes partes del programa `pagina.c`. Por un lado estas modificaciones dependen del comportamiento que se desea para el reactivo nuevo; pero por otro lado se debe respetar la estructura general del programa. Por esta razón la descripción de las modificaciones al programa `pagina.c` que se presentan mas adelante deben ser consideradas como recomendaciones básicas para preservar tal estructura.

6.3.1. Creación del nuevo tipo de reactivo

La creación del nuevo tipo de reactivo consiste en dar al mismo un nombre adecuado que permita relacionarlo fácilmente con la función para la cual esta siendo creado. Este nombre es precisamente el que se va usar como valor del atributo *type* en la sintaxis para crear una instancia del reactivo nuevo. Un ejemplo para un tipo nuevo de reactivo son las preguntas abiertas, las cuales es imposible calificar en forma automática, más bien, las debe de calificar un evaluador humano. El crear una instancia de este tipo de pregunta implica especificar el correo electrónico de la persona encargada de evaluar las respuestas dadas por los navegantes, sin embargo, ninguno de los atributos descritos en la tabla 6.2 sirve para especificar un correo electrónico, por tanto se hace necesario crear un atributo nuevo.

Para agregar un nuevo atributo al programa `pagina.c`, se modifica la función *interactionField* dentro del ciclo correspondiente a la instrucción

while (j<nsubfield)

se tiene que agregar una instrucción de la forma:

name	Nombre de la instancia
heading	Reservado
default	Opción por omisión
answer	Respuesta correcta
options	Lista de opciones
hint	Ayuda para el usuario
questions	Lista de preguntas
type	Tipo de reactivo

Cuadro 6.2: Atributos existentes

```
if(strcasecmp(subfield[j], <nombre_atributo> )==0)
  {
    <código>
  }
```

donde **<nombre_atributo>** es el nombre del atributo nuevo y **<código>** es el código que almacena en variables adecuadas los valores de dicho atributo.

En este punto es importante considerar si el atributo es simple o compuesto. En el caso de que sea simple se puede utilizar como modelo el código para procesar el atributo *name* (buscar la secuencia

```
if(strcasecmp(subfield[j], ``NAME'') == 0)
```

para localizar el código en el programa `pagina.c`). Si el atributo es compuesto entonces se puede usar como modelo el código para procesar el atributo *options* (buscar la secuencia

```
if(strcasecmp(subfield[j], ``CMDOPTIONS'')==0))
```

6.3.2. Traducción de la especificación del reactivo a código *HTML*

El propósito esencial es producir el código *HTML* que se debe generar para instrumentar el comportamiento deseado del reactivo. Este proceso se hace tomando los valores de las variables asignadas en el proceso descrito en la subsección anterior. Aquí también es posible incorporar cambios en la interpretación *HTML*.

Esencialmente agregar en el programa `pagina.c`, debajo del comentario

```
/* NUNCA REMOVER ESTE COMENTARIO
   En esta secci'on se construyen en HTML
   los campos interactivos asociados con los reactivos
*/
```

un código de la forma:

```
 if( strcasecmp(rdic.type, <tipo de reactivo>) )
   {
     printf( <código de salida> );
   }
```

donde **<tipo reactivo>** es el nuevo tipo de reactivo y **<código de salida>** es el código que envía a la salida estándar las instrucciones en *HTML* que se requieran para generar el campo interactivo.

En esta construcción **<tipo de reactivo>** es el nombre del nuevo tipo de reactivo que se está agregando a **e-Edita**.

Para construir las instrucciones *HTML* de **<código de salida>**se pueden aprovechar los valores almacenados en *rdic* cuya estructura es:

```
typedef struct tdicvar
  char name[FIELD_LEN]; // Nombre de la variable o campo de interacción
  char *input;          // Valor que da el usuario
  char *answer;         // Respuesta correcta asociada al reactivo
  char *type;           // Tipo del reactivo
  char *status;         // Indica el estado en reactivos CheckBox y Radio
  char *value;          // Valor de retorno del reactivo
  char *valuedefault;   // Valor por omisi'on del atributo
  char *hint;           // Sugerencia que se da al contestar mal
  char *heading;        // Reservado
  char *image;          // Reservado
 tdicvar;
```

También se pueden usar las variables:

```
char option;    // Lista de opciones en el atributo OPTIONS
int noptions;   // Número de opciones en la lista
char questions; // Lista de preguntas en el atributo QUESTIONS
int nquestion;  // Número de opciones en la lista
```

Por supuesto que también se pueden utilizar las variables definidas propiamente para procesar el nuevo tipo.

6.4. Modificaciones al programa `EkitCore.java`

El programa `EkitCore.java` está dividido en tres secciones, en la primera se encuentra el constructor de la clase principal del editor visual, aquí se crean los menús, barras de herramientas y el área de edición. En la segunda se encuentra el código de atención a los eventos activados al presionar alguno de los botones del editor visual; estas acciones son "escuchadas" por el método *actionPerformed(ActionEvent)* y se dividen en dos tipos: las relacionadas directamente con la edición de texto y las encargadas de insertar un nuevo constructor. En la tercera sección está el código que instrumenta la funcionalidad del editor visual propiamente; desde este código se invocan los programas *CGI*, los métodos de búsqueda y los métodos encargados de atender el cambio de contexto entre los diferentes modos de edición. (Modo página y modo libro ver sección 2.8).

A diferencia del programa `pagina.c`, donde es importante distinguir entre directivas y especificadores de reactivos, para el programa `EkitCore.java` tal distinción es innecesaria. Por esta razón, dentro del ámbito de `EkitCore.java` hablamos de manera genérica de *constructores* en lugar de directivas o especificadores de reactivos. Las acciones que se deben de realizar para agregar un nuevo constructor son:

1. El primer paso es asociar un botón nuevo al constructor que se desea agregar.

   ```
   private JButtonNoFocus jbtnNewButton;
   ```

2. El segundo paso es definir una cadena que identifique al constructor nuevo.

   ```
   public static final String KEY_TOOL_NEWKT  = "newkt";
   ```

3. El siguiente paso es identificar a qué modo de edición pertenece el constructor nuevo, el cual puede estar asociado al modo de edición libro, al modo de edición página o ambos (ver sección 2.8). Una vez identificado se añade al menú correspondiente. Para crear una instancia del constructor se agrega debajo del comentario

```
/* Build the menus */
```

un código de la forma:

```
JMenuItem jmiNewConstructor = new JMenuItem("NewConstructor");
jmiTitle.setActionCommand("newcommand");
jmiTitle.addActionListener(this);
jmiTitle.setAccelerator(KeyStroke.getKeyStroke('X',
                        java.awt.Event.ALT_MASK, false));
if(showMenuIcons)
  jmiTitle.setIcon(getEkitIcon("NewConstructorIcon"));
```

Para agregar la instancia al menú pagina

```
jMenuPage.add(jmiNewConstructor);
```

o bien, para agregar la instancia al menú libro

```
jMenuBook.add(jmiNewConstructor);
```

4. A continuación se incorpora el constructor a la barra de herramientas de **e-Edita** una instancia del botón que se definió arriba, para ello se agrega debajo del comentario

```
/* Create the toolbar */
```

código de la forma:

```
jbtnNewConstructor = new JButtonNoFocus(getEkitIcon(
                                          "NewConstructorIcon"));
jbtnNewConstructor.setToolTipText("Constructor NewConstructor");
jbtnNewConstructor.setActionCommand("newcommand");
jbtnNewConstructor.addActionListener(this);
jeEditaToolBar.add(jbtnNewConstructor);
hteEditaTools.put(KEY_TOOL_NEWKT, jbtnNewConstructor);
```

5. Se asocia el botón a un vector para indicar si éste pertenece al menú "página" o al menú "libro". Para el primer caso

```
vcPageIcons.add(jbtnNewDirective);
```

y para el segundo caso

```
vcBookIcons.add(jbtnNewDirective);
```

6. Por último se define el identificador del constructor que se asocia a su botón en el editor visual así como el código que se ejecuta cuando éste se presiona.

```
if(command.equals("newcommand")) {
  Acciones antes de invocar a la nueva directiva...
  ekitEdita.newConstructorWindow(par'ametros, this);
 }
```

Al invocar al método *newConstructorWindow* que está definido en el programa `EkitEdita.java` es importante pasar como parámetro *this* (es decir, el contexto del programa actual) ya que éste sirve para invocar métodos pertenecientes a `EkitCore.java` desde el programa `EkitEdita.java`.

6.5. Adaptaciones al Applet

Después de incorporar un constructor nuevo al programa `EkitCore.java` se tiene que modificar su *Applet*, éste se encuentra en el programa `EkitApplet.java`. Esencialmente se deben invocar las barras de herramientas y menús definidos en `EkitCore.java` y colocarlos en un lugar apropiado dentro del *Applet*.

Las barras de herramientas invocadas desde el *Applet* deben ser creadas agrupando los botones relacionados a los constructores que se deseen utilizar, para agruparlos se guardan dentro de un vector las cadenas asociadas con dichas herramientas:

```
Vector vcTools = new Vector();
vcTools.add(EkitCore.KEY_TOOL_NEW);
vcTools.add(EkitCore.KEY_TOOL_OPEN); ...
```

Por medio de la función *getCustomToolBar(Vector, Boolean)* se añade la barra de herramientas con las funciones especificadas en el vector *vcTools*.

```
this.getContentPane().add(ekitCore.getCustomToolBar(
                                  vcTools, showToolBar),
                                  BorderLayout.NORTH);
```

También existe el vector *vceEditaTools* cuyo objetivo es separar las herramientas de **e-Edita** del resto de las funciones básicas del editor visual, tal como copiar, pegar, negritas, etc.

6.6. Modificaciones al programa `EkitEdita.java`

Hasta el momento sólo se ha mostrado cómo modificar el programa que controla las funciones básicas del editor visual y cómo reflejar esta modificación en el *Applet* para mostrar el botón del constructor correspondiente en la barra de herramientas, pero no se ha dicho nada sobre cómo programar la interfaz visual. En esta sección se aborda el tema.

El código asociado con las ventanas de interacción que son presentadas al usuario están en el programa `EkitEdita.java`, la idea fundamental para construir estas ventanas es generar un marco base sobre el cual se agregan los componentes necesarios para editar al constructor. Dicho marco es un *JFrame* llamado *jfMainFrame* y sobre él se añaden los campos de interacción que utiliza el usuario para dar valor a los parámetros de los constructores. Existen muchos campos predefinidos que pueden ser útiles al programador, en el apéndice A se da la lista completa de los componentes existentes y una breve descripción de los mismos.

Para lograr la mayor uniformidad posible en la presentación de las interfaces de usuario, el marco base esta instrumentado con una clase llamada *EditaWindow* en la que se asigna el nombre de la ventana, se agrega el panel para los botones de control y se establece el tipo de distribución de los componentes. Esta clase es:

```
    class EditaWindow {
        EditaWindow(String construct, EkitCore ekitcore)
         {
            vcButtons   = new Vector();
            ekitCore    = ekitcore;
            constructor = construct;
            jfMainFrame = new JFrame(construct);
            jpButtons   = new JPanel(new FlowLayout());

            jfMainFrame.getContentPane().setLayout(new
                                        BorderLayout());
         }
   }
```

El argumento *construct* indica con que constructor se está trabajando, además de ser el titulo de la ventana principal, el argumento *ekitcore* es la referencia al programa principal desde donde se invoca esta ventana nueva, dicho argumento es muy importante porque se utiliza para invocar las funciones del programa `EkitCore.java` que sirven para insertar el constructor nuevo.

La variable *vcButtons* es un vector que contiene los botones asociados al funcionamiento de la ventana, usualmente estos botones son: "Enviar", "Cancelar" y "Avanzado". La lista puede hacerse más grande o más pequeña según se requiera. La variable *ekitcore* sirve para invocar funciones contenidas en el programa `EkitCore.java`. La variable *jfMainFrame* contiene la ventana principal sobre la cual se construye la interfaz de usuario y *jpButtons* es el panel donde se colocan los botones que se guardaron en el vector *vcButtons*.

Esta clase se utiliza dentro del método que instrumenta la interfaz del usuario. En el siguiente ejemplo se muestra la estructura de un método cuya finalidad es pedir un texto al usuario.

```
public void newMethodWindow(EkitCore ekitcore)
 {
  String strNewString;
  JPanel jpNewPanel = new JPanel(new FlowLayout());
  Lista de otros componentes locales...

  new EditaWindow("Constructor", ekitcore);

  currentWindowType = QUESTION_WINDOW;

  jpNewPanel.add(new JLabel("Valor"));
  jpNewPanel.add(new JTextField(10));
  Otros paneles...

  vcButtons.add(jbtnSend   = customButton("Enviar","returnquestion"));
  vcButtons.add(jbtnCancel = customButton("Cancelar","cancel"));

  jpButtons = customButtonPanel(vcButtons);
  jfMainFrame.getContentPane().add(jpNewPanel, BorderLayout.CENTER);
  jfMainFrame.getContentPane().add(jpButtons, BorderLayout.SOUTH);
  jfMainFrame.show();
 }
```

En la primera parte del método se definen componentes que sólo serán utilizados localmente. Con la instrucción *new EditaWindow("Constructor", ekitcore)* se crea la base para la interfaz visual como se explicó anteriormente, la instrucción

```
currentWindowType = QUESTION_WINDOW}
```

es una bandera que permite identificar con qué tipo de ventana se está trabajando, esto con la finalidad de asignar el tamaño de ventana apropiado para cada constructor y de procesar sus campos. En la siguiente parte se agregan al *JPanel* "jpNewPanel" definido una etiqueta y un campo de texto para que el usuario introduzca información. En el siguiente par de líneas se definen los botones asociados a la interfaz que se está programando, en este caso los botones "Enviar" y "Cancelar", los cuales son creados mediante el método *customButton(String, String)* que recibe dos cadenas: la primera etiqueta al botón que se esta generando y la segunda cadena se asocia al comando que ha de ejecutarse cuando el botón sea presionado. La línea

```
jpButtons = customButtonPanel(vcButtons)
```

genera el panel con los botones previamente definidos. En las siguientes dos líneas se agregan los paneles creados a la ventana principal y la última línea sirve para que se haga visible la ventana.

El método encargado de crear los botones es:

```
private JButton customButton(String label, String cmd) {
    JButton jbtnTemp = new JButton(label);
    jbtnTemp.setActionCommand(cmd);
    jbtnTemp.addActionListener(this);
    return jbtnTemp;
}
```

en esta construcción, *cmd* representa el código que se ejecuta cuando se presiona el botón. En Java existe un método especial llamado *actionPerfomed(ActionEvent)* que se encarga de ejecutar acciones específicas cuando ocurre algún evento, en este caso, cuando se presiona un botón.

```
public void actionPerformed(ActionEvent ae)
 {
  if (command.equals("cancel"))
   {
    jfMainFrame.dispose();
   }
  else if (command.equals("returnquestion"))
   {
    getDirectiveParameters(QUESTION_WINDOW);
    ekitCore.obtainDirective(cadena a insertar, bEditDirective,
                             oldDirectiveLength);
    jfMainFrame.dispose();
   }
 }
```

El método *actionPerformed* se encarga de "escuchar" cuando un botón es presionado, en este ejemplo se pregunta si el botón presionado tenía asociado el comando *cancel* o el comando *returnquestion*, en el primer caso la acción a realizar es simplemente cerrar la ventana que estaba abierta. En el segundo caso, que corresponde a la situación cuando el botón presionado tiene asociado el comando *returnquestion*, se invoca al método *getDirectiveParameters(QUESTION_ WINDOW)* que se encarga de obtener los parámetros correspondientes al tipo de ventana especificado, en este caso el tipo es *QUESTION_ WINDOW*. A continuación se invoca al método *obtainDirective(String, boolean, int)* el cual está definido en el programa `EkitCore.java`.

```
void obtainDirective(String cadena, boolean deleteOld,
                     int deleteChars)
{
   try {
     int caretPos = jtpMain.getCaretPosition();
     styDoc.insertString(caretPos, cadena,
```

```
                        jtpMain.getInputAttributes());
      jtpMain.setCaretPosition(caretPos + cadena.length());
      if ( deleteOld )
         styDoc.remove(jtpMain.getCaretPosition(),
                       deleteChars);
   }
   catch (Exception e){System.out.err("");}
    return;
}
```

Este método recibe tres parámetros: la cadena que se va a insertar en el editor visual, una bandera que indica si la instancia del constructor se está insertando por primera vez o no y la longitud de la instancia del constructor anterior (está sólo se usa en la edición de constructores). La cadena a insertar en el editor visual es precisamente la especificación de la instancia del constructor nuevo. Cuando la bandera *bEditDirective* se encuentre encendida, se borra del editor visual la cantidad de caracteres indicada por el parámetro *oldDirectiveLength* para que sean reemplazados por la nueva cadena, en caso de que la bandera esté apagada no se borra ningún carácter del editor visual y simplemente se inserta la cadena.

Conclusiones

Al inicio de este trabajo de tesis, en mi primer contacto con las herramientas de **e-Edita**, al ver los archivos de especificación creados con este lenguaje pensé, "Qué cosa mas extraña"; sin embargo, después de revisar un par de especificaciones y las páginas generadas por las mismas, me di cuenta de la sencillez y poderío de las herramientas de edición de **e-Edita**. Sin embargo, había un pequeño "problema": no había interfaz, todo se trabajaba en modo texto, lo cual favorece la productividad, pero la mayoría de la gente no lo ve así, ellos piensan: su uso ha de ser muy complicado y si hay íconos interactivos no entiendo. Esto me motivó a construir esta interfaz visual.

Con la investigación del estado del arte pude constatar que la educación virtual está bastante avanzada en los países del primer mundo y que en Latinoamérica aún se encuentra en desarrollo. Con esta tesis pretendemos aportar nuestro granito de arena en la creación de aplicaciones que ayuden al progreso de la educación virtual en México. Esperamos que con ayuda de esta interfaz visual las herramientas de **e-Edita** lleguen a un gran número de profesionales de la educación, lo que permitirá incrementar la cantidad de materiales educativos disponibles en la Internet.

Al iniciar este trabajo creía que la programación de aplicaciones accesibles vía Internet era complicada; sin embargo no es así. Me di cuenta de que lenguajes como Java hacen que esta programación sea muy natural, además los programas *CGI* utilizados fueron programados en lenguaje `C`, un viejo conocido. Se emplearon también otras herramientas como *AWK* (antecedente de *Pearl*), que aunque ya es una herramienta antigua, resulta asombrosa por su simplicidad y poder.

Por último, en esta tesis se habló de las aplicaciones de **e-Edita** en la educación virtual, sin embargo, con este lenguaje de especificación se pueden crear aplicaciones para la *Web* que van más allá, por ejemplo, manejo de bases de datos y visualización de mapas, los cuales cuentan con sus propias directivas. Una posible extensión al trabajo es incorporarlas al editor visual, creando así un entorno más poderoso.

Apéndice A

Componentes de `EkitEdita.java`

Los componentes predefinidos en el programa `EkitEdita.java` se describen a continuación, el objetivo de mostrarlos es que los programadores conozcan su uso y así poder reutilizarlos eficientemente.

Todos estos componentes tienen el atributo *private*

`JFrame jfMainFrame`: Ventana principal sobre la cual se construye la interfaz del usuario.

`JPanel jpButtons`: Panel donde se colocan los botones de *Acepar*, *Cancelar*, *Avanzado* y *Ayuda*.

`JPanel jpComponents`: Panel en el que se colocan los paneles de componentes estándar y avanzados.

`JPanel jpStandard`: Panel donde se colocan los elementos básicos de la interfaz.

`JPanel jpAdvanced`: Panel donde se colocan los elementos avanzados de la interfaz.

`JTextArea jtaAddress`: Área de texto para diversos fines.

`JTextArea jtaLatex`: Área de texto para procesar las directivas MATHIMG.

`JTextArea jtaCheckResult`: Área de texto donde se despliega el resultado regresado por el compilador de LaTeX al verificar la sintaxis de un MATHIMG.

`JTextField jtfReturnText`: Campo de texto que es utilizado en las directivas de un sólo argumento.

`JTextField jtfAnswer`: Campo de texto donde se almacena la respuesta a una pregunta.

`JTextField jtfHint`: Campo de texto donde se almacena la sugerencia de una pregunta.

`JTextField jtfName`: Campo de texto donde se guarda el nombre asignado al campo de interacción.

`JTextField jtfHeading`: Campo de texto donde se almacena el encabezado del campo de interacción.

`JTextField jtfDefault`: Campo de texto donde se guarda el valor por omisión de una pregunta.

`JTextField jtfBack`: Campo de texto para hacer referencia al capítulo anterior, utilizado en la directiva END.

`JTextField jtfNext`: Campo de texto para hacer referencia al capítulo siguiente, utilizado en la directiva END.

`JTextField jtfTitle`: Campo de texto donde se guarda el titulo que es asignado a una página.

`JTextField jtfStyle`: Campo de texto donde se indica que página de estilo ha de ser utilizada.

`JTextField jtfColorize`: Campo de texto donde se específica la colorización de un MATHIMG.

`JTextField jtfOtherCmds`: Campo de texto el el cual el usuario puede añadir más parámetros al comando `convert` utilizado para generar los MATHIMG.

`JTextField jtfChapter`: Campo de texto que se utiliza para asignar nombre a los capítulos.

`JComboBox jcbCrop`: Caja de opción múltiple que se utiliza para guardar el parámetro *crop* del comando convert.

`JComboBox jcbBorder`: Caja de opción múltiple que se utiliza para almacenar el parámetro *border* del comando convert.

`JComboBox jcbDensity`: Caja de opción múltiple que se utiliza para almacenar el parámetro *density* del comando convert.

`JButton jbtnSend`: Botón que se utiliza para introducir en el editor la directiva que se esta editando.

`JButton jbtnCancel`: Botón que se utiliza para cancelar la edición de la directiva.

`JButton jbtnAdvanced`: Botón que se utiliza para mostrar los elementos avanzados de una directiva.

`JButton jbtnSintax`: Botón que se utiliza para verificar la sintaxis del contenido LaTeX de un MATHIMG.

`JButton jbtnSelColor`: Botón que se utiliza para seleccionar el color que ha de utilizarse en un MATHIMG.

`JFileChooser jfc`: Seleccionador de archivos

`JTabbedPane jtpLatex`: Panel tabular donde se disponen los diversos componentes de la directiva MATHIMG.

`JColorChooser jccLatex`: Selector de color para darle un fondo a un MATHIMG.

`JCheckBox jcbNegate`: Caja para establecer el valor del parámetro *negate* del comando `convert`.

`JSlider jslQuality`: Barra de desplazamiento utilizada para asignar un valor al parámetro *quality* del comando convert.

`JSpinner jspQuality`: *Spinner* utilizado junto con la barra jslQuality.

`JSpinner jspOptionsNumber`: *Spinner* para ajustar el número de opciones que han de utilizarse por defecto en las preguntas de opción múltiple.

Apéndice B

Instalación

Para que el editor funcione adecuadamente en el navegador, el usuario debe instalar `JRE` (*Java Runtime Enviroment*) y en caso de querer hacer modificaciones al editor, lo que implica compilar la aplicación, es necesario instalar `SDK` (*Software Developers Kit*). Cabe señalar que `JRE` forma parte de `SDK`, es decir, al instalar el `SDK` ya no es necesario el `JRE`.

B.1. Instalación en Windows

La instalación en Windows se realiza de la siguiente manera:

1. Descargar del sitio de Sun Microsystems[12] el archivo:
   `j2sdk-1_4_2_03-windows-i586-p.exe` o
   `j2re-1_4_2_03-windows-i586-p.exe` en su versión más reciente.

2. Ejecutar este archivo y seguir las instrucciones que aparecen en pantalla

3. Reiniciar la computadora.

B.2. Instalación en Linux

En el sistema operativo Unix, la instalación debe realizarse en dos pasos: instalar `JRE` o `SDK` según sea el caso y configurar el navegador para su uso. En este caso se dan las instrucciones para configurar Netscape.

B.2.1. Instalación de JRE o SDK

1. Descargar del sitio de Sun Microsystems[12] el archivo:
   `j2sdk-1_4_2_03-linux-i586-rpm.bin` o
   `j2re-1_4_2_03-linux-i586-rpm.bin` en su versión más reciente

2. Hacer el archivo ejecutable con el comando `chmod +x` *archivo*

3. Ejecutar el archivo

4. Instalar el *RPM* que se genere

5. En la cuenta del usuario ejecutar el comando:
   `export NPX_PLUGIN_PATH=/usr/java/jdk1.3.1_07/jre/plugin/i386/ns4`

B.2.2. Configuración del navegador

1. Entrar al directorio de *plugins'* de Netscape `cd /usr/local/netscape/plugins`
2. Crear una liga del *plugin* de Java al directorio:
   `ln -s /usr/java/j2sdk1.4.1_02/jre/plugin/i386/ns610/libjava/plugin_oji.so`
3. Reiniciar dos veces el navegador debido a un error.

Es importante señalar que los números en los nombres de los directorios cambiarán según la versión de `JRE` o `SDK` que se este instalando.

B.3. Instalación del Editor

Una vez instalado el `JRE` o `SDK` es momento de instalar el editor visual, para ello sólo se debe descomprimir un archivo: `eEdita.tar.gz`, éste creará el directorio `eEdita` donde se encuentran todos los módulos que lo conforman.

Apéndice C

Compilación del editor visual

Cuando se realice algún cambio al editor visual ya sea modificando o agregando componentes al mismo, la aplicación debe de recompilarse para que los cambios surtan efecto. Se han creado cuatro archivos, que se utilizan por pares dependiendo de la plataforma que se este utilizando: los *scripts* `MakeEkit.csh` y `MakeEkitApplet.csh` para Linux y los archivos por lotes `MakeEkit.bat` y `MakeEkitApplet.bat` para Windows.

Estos archivos compilan de forma automática todos los módulos que conforman el sistema. El archivo `MakeEkit` compila al editor visual en su formato *StandAlone*, el cual es muy útil para realizar pruebas con el editor sin necesidad de abrirlo desde el navegador. El archivo `MakeEkitApplet` compila al editor en su formato de *Applet*, para lo cual crea un archivo `.jar` que contiene a la aplicación y que es firmado con un certificado de seguridad, el cual indica al usuario que el *Applet* que está ejecutando es de una fuente confiable.

La forma de ejecutar estos archivos es la siguiente:

Linux
[user@localhost eEdita]$./MakeEkit.csh [user@localhost eEdita]$./MakeEkitApplet.csh
Windows
c:\eEdita\ >MakeEkit.bat c:\eEdita\ >MakeEkitApplet.bat

Apéndice D

Resultado en código *HTML* de una especificación

En este apéndice se muestra el código *HTML* generado a partir de las especificaciones mostradas en las secciones 3.4 y 3.2.

D.1. Página electrónica

A continuación se muestra el código *HTML* generado por la especificación que se muestra en la sección 3.2.

```
<HTML>
<HEAD>

<TITLE>Título e-Edita</TITLE>

</HEAD>

<BODY TEXT="#000000" BGCOLOR="#FFFFFF" LINK="#400040" VLINK="#800040"
ALINK="#800080"
BACKGROUND="http://alaska.math.cinvestav.mx/~imendez/back.gif">

<FORM NAME="form_ej1" ACTION="/cgi-bin/califica1.out" METHOD="POST">

<INPUT TYPE=HIDDEN NAME=_version VALUE="1.0">
<INPUT TYPE=HIDDEN NAME=_filename VALUE="ej1">
<INPUT TYPE=HIDDEN NAME=_home VALUE="/home/imendez">
<INPUT TYPE=HIDDEN NAME=_hostname VALUE="alaska.math.cinvestav.mx">
<INPUT TYPE=HIDDEN NAME=_logname VALUE="imendez">

<H1>Editor de materiales educativos para la web</H1>

<HR> <ADDRESS> Ing. Israel Méndez. imendez@computacion.cs.cinvestav.mx<BR>
 Centro de Investigación y de Estudios Avanzados del IPN<BR>
 Departamento de Ingeniería Eléctrca<BR>
</ADDRESS> <HR>

La imágen de un círculo con un tamaño de 300 pixeles.<BR><BR>

<CENTER>

<img src="http://alaska.math.cinvestav.mx/~imendez/circulo.jpg"
width=300 hight=300 ALT="El circulo ">

</CENTER>.<BR><BR>

Usando la directiva MathImg es posible especificar cualquier expresión
 matemática en LaTeX.<BR><BR>

<CENTER>

<img  src="http://alaska.math.cinvestav.mx/~imendez/ej1000.jpg" ALT="">
</CENTER><BR><BR>
```

```
Un archivo de sonido

<a href="http://alaska.math.cinvestav.mx/~imendez/mamap.wav"> Palabra
mama </A>

<BR><BR>
 Ejemplo de los diferentes especificadores de reactivos:<BR><BR>

Pregunta abierta de respuesta especifica ¿Quién descubrió América?
<INPUT TYPE=TEXT NAME=grade0 VALUE="" SIZE=15 MAXLENGTH=15>
<BR><BR><BR>

Pregunta de opción múltiple en formato ComboBox ¿Quién descubrió
 América?
<SELECT NAME=cmbAmerica ><OPTION>Cristóbal Colón<OPTION>Juan de la
 Barrera<OPTION SELECTED>Jeremías Springfield</SELECT>
<BR><BR><BR>

Pregunta de opción múltiple en formato de botón de radio vertical
 ¿Quién descubrió América?  <BR>
<INPUT TYPE=RADIO NAME=grade2 VALUE="Jeremías Sprinfield" CHECKED>
 Jeremías Sprinfield <BR> <INPUT TYPE=RADIO NAME=grade2
 VALUE="Cristóbal Colón" > Cristóbal Colón <BR> <INPUT TYPE=RADIO
 NAME=grade2 VALUE="Juan de la Barrera" > Juan de la Barrera
<BR><BR><BR><BR>

Pregunta de opción múltiple en formato de botón de radio horizontal
 ¿Quién descubrió América?  <BR>
<INPUT TYPE=RADIO NAME=grade3 VALUE="Jeremías Sprinfield" > Jeremías
 Sprinfield      <INPUT TYPE=RADIO NAME=grade3
 VALUE="Cristóbal Colón" > Cristóbal Colón     
 <INPUT TYPE=RADIO NAME=grade3 VALUE="Juan de la Barrera" CHECKED>
 Juan de la Barrera     
<BR><BR><BR>

Ejercicio para relacionar columnas <BR><table>

<tr><td ALIGN=RIGHT> 1.  </td>
<td>Yucatán</td><td>      </td>
<td ALIGN=RIGHT><INPUT TYPE=TEXT NAME=grade400 VALUE="" SIZE=1
MAXLENGTH=1 ></td><td>Jalapa</td></tr>

<tr><td ALIGN=RIGHT>2.  </td>
<td>Guerrero</td><td>      </td>
<td ALIGN=RIGHT><INPUT TYPE=TEXT NAME=grade401 VALUE="" SIZE=1
MAXLENGTH=1 ></td><td>Mérida</td></tr>

<tr><td ALIGN=RIGHT>3.  </td><td>Jalisco</td>
<td>      </td>
<td ALIGN=RIGHT><INPUT TYPE=TEXT NAME=grade402 VALUE="" SIZE=1
MAXLENGTH=1></td><td>Chilpancingo</td></tr>

<tr><td ALIGN=RIGHT>4.  </td>
<td>Veracruz</td><td>      </td>
<td ALIGN=RIGHT><INPUT TYPE=TEXT NAME=grade403 VALUE="" SIZE=1
MAXLENGTH=1 ></td><td>Guadalajara</td></tr> </table>

<P><CENTER><INPUT TYPE=RESET NAME=_b1 VALUE="Limpiar campos"><INPUT
TYPE=SUBMIT NAME=_b2 VALUE="Enviar" ></CENTER> </FORM>

<p class="cen"><b><span class="rojo">|<span class="rojo">|</span> <a
href="http://alaska.math.cinvestav.mx/~imendez/ej2.html"
onClick="parent.frames[0].history.go(0);parent.frames[0].history.go(0)">
Siguiente </a><span class="rojo">|<span class="rojo"></span></b></p>
</BODY>

</big>

</HTML>
```

D.2. Libro electrónico

A continuación se muestra el código *HTML* generado por la especificación que se muestra en la sección 3.4

```
<html>
<head>
```

```
<title>Libro electrónico</title>
</head>
<frameset cols="25%,**">
<frame src="http://alaska.math.cinvestav.mx/~imendez/milibro.html">
<frame src="http://alaska.math.cinvestav.mx/~imendez/pmilibro.html"
name = "view_window">
</frameset>
<html>
```

En el archivo `milibro.html` está el código *HTML* correspondiente a la tabla de contenido del libro y en el archivo `pmilibro.html` se encuetra el código *HTML* de la portada del libro, cabe señalar que la portada no se genera automáticamente, más bien, ésta la genera el usuario. Ahora mostramos el código *HTML* generado por el traductor de **e-Edita** correspondiente al archivo `milibro.html`

```
<HTML>
<HEAD>
<TITLE>Libro electrónico </TITLE>
</HEAD>

<BODY TEXT="#000000" BGCOLOR="#FFFFFF" LINK="#400040" VLINK="#800040"
ALINK="#0000FF"
BACKGROUND="http://alaska.math.cinvestav.mx/~imendez/back.gif">

<H1> Ejemplo  </H1>

<BR><H2>CAPÍTULO 1: Capítulo de ejemplo </H2>

<P> 1.1 <A HREF = "http://alaska.math.cinvestav.mx/~imendez/ej1.html"
target="view_window"> Editor de materiales educativos para la web
</A>

<BR> 1.3 <A HREF = "http://alaska.math.cinvestav.mx/~imendez/ej2.html"
target="view_window"> Seunda página </A><BR><BR><BR>

</BODY>
</HTML></BODY>
</HTML>
```

Apéndice E

Organización del disco compacto

Este CD está organizado de la siguiente manera:

```
CD
 |-- + codigo_fuente
 |      + editor
 |      + cgi
 |-- + escrito
 |      + fuentes_tex
 |      a tesis.ps
 |      a tesis.pdf
 |-- + instalacion
 |      + java
 |         a INSTJAVA
 |      a config
 |      a instala
 |      a lib-bin.taz.gz
 |-- + manual_usuario
 |      a manual.pdf
 |      a manual.ps
 |-- a LEAME

+ Directorio
a Archivo
```

En el directorio `codigo_fuente` se encuentran los archivos fuente en Java (directorio `editor`) y en C (directorio `cgi`) del sistema desarrollado

En el directorio `escrito` se encuentra la tesis en formato PDF y PS así como los fuentes de ésta en formato tex (directorio `fuentes_tex`)

En el directorio `instalacion` se encuentra un script para instalar el sistema (archivo instala) y los archivos necesarios para instalar la máquina virtual de java (directorio `java`) para linux y windows, éstos deben instalarse manualmente, las instruciones están en el archivo INSTJAVA

En el directorio `manual_usuario` se encuentra el manual del sistema desarrollado

Bibliografía

[1] Pearson Broadband. *KnowledgeBox.* http://www.knowledge.com, 2002.

[2] IEEE Learning Technology Standards Committee. *Learning Object Metadata Working Group.* http://itsc.ieee.org/wg12, 2003.

[3] Leading Way Corporation. *KnowledgeOne.* http://www.leadingway.com, 2003.

[4] Instituto Latinoamericano de la Comunicación Educativa. *ILCE.* http://www.ilce.edu.mx, 2002.

[5] Mark-Jason Dominus. *Vulcanize - Converter.* http://www.plover.com/ mjd/vulcanize.html, 2002.

[6] Bruce Eckel. *Thinking in Java 2ed.* Prentice-Hall, 2002.

[7] Unext Invest in Knowledge. *Unext.* http://www.unext.com, 2003.

[8] Blackboard Inc. *Blackboard.* http://www.blackboard.com, 1997-2004.

[9] Half-Baked Software Inc. *Hot Potatoes.* http://web.uvic.ca/hrd/halfbaked, 2002.

[10] National Learning Infraestructure Initiative. *NLII.* http://www.educause.edu.nlii, 2003.

[11] Howard Kistler. *Ekit.* http://www.hexidec.com, 2003.

[12] Sun Microsystems. *Java 2 Platform Standard Edition (J2SE).* http://java.sun.com, 1994-2004.

[13] Global Knowledge Network Inc. *Global Knowledge.* http://www.globalknowledge.com, 2004.

[14] LateX2HTML Development Repository. *latex2html.* http://www.latex2html.org, 2001.

[15] Centra Software. *Centra.* http://www.centra.com, 2004.

[16] Richard Stones and Neil Matthew. *Beginning Linux Programming.* Wrox Press, 1997.

[17] Feliú D. Sagols Troncoso. *e-Edita.* http://alaska.math.cinvestav.mx/ demo/e-edita.html, 2001.

[18] Feliú D. Sagols Troncoso. *Matemáticas Básicas, Curso para profesores.* http://alaska.math.cinvestav.mx/ matbas/fmatbas.html, 2001.

[19] Feliú D. Sagols Troncoso. *Servidor Alaska.* http://alaska.math.cinvestav.mx, 2001.

[20] Feliú D. Sagols Troncoso and Israel Méndez Martínez. *Editor visual para e-Edita.* http://alaska.math.cinvestav.mx/ imendez/fimendez.html, 2001.