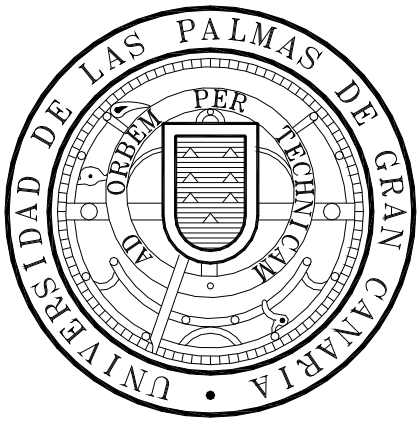


Universidad de Las Palmas de Gran Canaria

Proyecto de fin de carrera
RTHC: Conversor de RTF a HTML

Autor: Esteban Manchado Velázquez
Director: José Miguel Santos Espino

Tomo I
Las Palmas de Gran Canaria
Septiembre de 2000

Introducción

Resumen

RTHC es un conversor de documentos en formato RTF a páginas HTML. Su intención principal es conservar el texto, y, si es posible, parte del formato físico del documento original. La razón por la cual se eligieron los formatos RTF y HTML es que aquél es tremendamente utilizado, pero es un formato de texto cerrado, propiedad de una compañía privada (Microsoft corporation). Además, el proyecto es también un estudio de la comunidad del *software* libre, e indaga en sus costumbres y en su forma de publicar programas y de trabajar sobre ellos.

Técnicamente, el proyecto se ha separado en dos partes: un conjunto de clases en el lenguaje C++ para el manejo, de forma genérica, de documentos en formato RTF, y el programa conversor en sí, que usa el primer componente. Por tanto, los resultados del trabajo han sido un conjunto de clases, llamado *libFreeRTF*, un conversor de RTF a HTML, llamado RTHC, ambos con documentación asociada; una guía de referencia del formato RTF; y unos documentos que describen la comunidad del *software* libre desde el punto de vista técnico.

Organización de este documento

Este documento describe el proyecto en líneas generales, la comunidad del *software* libre, los formatos de conversión empleados, el diseño e implementación del programa en sí, las herramientas usadas y los resultados finales. Además, en los anexos se adjuntan varios documentos que pueden resultar de interés, como manuales de referencia, las licencias de los programas o artículos sobre *software* libre.

En el capítulo 1, «El proyecto RTHC», se introduce al lector al proyecto RTHC, centrándose en sus objetivos y su justificación. En el segundo capítulo, «*Software* libre», se discuten las características más relevantes del *software* libre y de su comunidad de programadores y usuarios. «El formato RTF» y «El lenguaje HTML», por otro lado, sirven de guía para comprender la naturaleza y características de los dos formatos usados en el proyecto. El capítulo 5, «El programa RTHC» discute en profundidad el diseño e implementación del programa en sí, y de las clases que lo apoyan, la *libFreeRTF*. En «Herramientas *software* usadas en el proyecto», se habla de los programas utilizados para construir el programa. Por último, los capítulos 7 y 8 terminan con las conclusiones y las referencias bibliográficas de todo el proyecto. Después de los 8 primeros capítulos, vienen los anexos, donde se incluyen un manual de referencia de RTHC, uno de las clases de *libFreeRTF*, una guía de referencia del formato RTF, un texto que explica cómo crear paquetes Debian y RPM, y algunos documentos más relevantes para el proyecto.

Índice general

1 El proyecto RTHC

1.1. Breve descripción del trabajo

El proyecto consiste en la elaboración de un programa capaz de traducir textos escritos en formato RTF *(Rich Text Format)* al formato de descripción de páginas de Internet (HTML). Se pretende además que el proyecto profundice en las técnicas y características del llamado *software libre*: por este motivo el programa se desarrollará en un entorno abierto (Linux), utilizando herramientas gratuitas. La aplicación, que está dividida en dos partes, será publicada utilizando las licencias GPL y LGPL[1].

1.2. Fundamentos del proyecto

1.2.1. Los sistemas abiertos

Desde hace décadas, en la industria informática se contraponen dos modelos de distribución de productos de *software*: los sistemas cerrados o «propietarios» y los *sistemas abiertos*. El modelo de sistemas propietarios se corresponde con el habitual sistema de producción capitalista, según el cual una empresa explota económicamente sus innovaciones, manteniéndolas en secreto y condicionando su utilización a una contraprestación económica. Es el caso de sistemas operativos como Windows o OS/400, o aplicaciones como Office o Autocad, o formatos de documentos como el GIF (para imágenes).

El concepto de *software* abierto engloba desde aplicaciones completamente gratuitas como Emacs hasta productos comerciales como Netscape, y formatos como el HTML. Todos estos sistemas tienen en común que sus especificaciones son conocidas, no secretas. Los formatos usados en sistemas abiertos son de uso gratuito. Cualquiera puede escribir aplicaciones que los procesen sin tener que pagar derechos por ello. Los sistemas abiertos más genuinos son aquellos regulados por una norma o estándar oficial (ISO, ANSI, etc.)

A pesar de la presión de la industria, una parte considerable de los sistemas informáticos actuales se basan en *software* abierto, desde los protocolos de Internet hasta el sistema operativo UNIX (ambos recogidos en normas ISO). La comunidad científica debe un gran interés a los sistemas abiertos, ya que permiten la libre difusión de las innovaciones y la ausencia de costes por el uso de herramientas de trabajo. Todo ello redunda en una mayor productividad en la generación de conocimiento.

1.2.2. El software libre

Para que un sistema de software pueda considerarse realmente abierto, es necesario disponer de los *fuentes* (instrucciones originales escritas por el programador) de todos los programas que constituyen el sistema. Un software abierto se llama *software libre* (*free software*) cuando el usuario del software está autorizado a ceder libremente los *fuentes* a cualquier otra persona, y además tiene derecho a hacer modificaciones y distribuirlas (quizás bajo ciertas restricciones). Otra denominación es «fuentes abiertos» o «fuentes libres» (*open source, free source*). Las reglas que confieren el calificativo de «libre» a una licencia (y, por tanto, a los programas que se distribuyen bajo ésta) son las «Debian Free Software Guidelines», disponibles en `http://www.debian.org`, y añadidas como anexo a esta memoria.

Richard Stallman, pionero impulsor del software libre, concibió la licencia GPL (*General Public License*), como una ingeniosa forma de asegurar la distribución y modificación de *software* libre sin problemas de que evolucione a formas *cerradas*. Quien modifique programas bajo GPL está obligado a distribuir el

[1] «General Public License» y «Library General Public License», ver anexos y `http://www.gnu.org`

nuevo código siguiendo también la GPL. Esta licencia ha tenido bastante éxito en los sistemas abiertos; durante los últimos años se han concebido licencias inspiradas en la GPL que suelen ser más permisivas con el uso comercial. Los ejemplos más claros han sido LGPL, también del proyecto GNU, que legaliza la distribución de programas obtenidos a partir de programas abiertos, mediante licencias cerradas; la Apache, que permite que las derivaciones de un programa liberado por esa licencia se distribuyan por otra cualquiera; la Artística, del intérprete Perl, que da al autor original (al poseedor de los derechos) un poco de control sobre qué se puede hacer y qué no; la QPL y la NPL, que son variaciones «comerciales» de la GPL, escritas por empresas como Troll Tech o la antigua Netscape, y que, aunque se consideran libres, desafortunadamente son incompatibles con la GPL, lo que da problemas a la hora de mezclar programas con ambas licencias.

La FSF (*Free Software Foundation*) está reconocida como la mayor defensora a nivel mundial de este modelo de programación en el que todos aprenden de todos. En su seno se encuentra el proyecto GNU, el cual pretende construir un sistema operativo completo, totalmente abierto y libre, que cualquiera pueda usar. Cualquier programador de cualquier parte del mundo puede, si tiene conocimientos suficientes, ayudar al proyecto GNU y a la comunidad entera. El resultado más visible de los esfuerzos de GNU es el sistema operativo Linux, cuyas utilidades más fundamentales se han tomado en su mayoría del referido proyecto.

1.2.3. El formato RTF

El RTF («Rich Text Format», o «Formato de texto enriquecido») es un formato de texto diseñado por Microsoft, muy popular en Windows. Lo generan programas tan comunes como Word, Wordpad o Access. RTF es en principio un formato abierto, pero no está recogido en una norma oficial, no es un auténtico estándar. Microsoft divulga un documento de especificaciones de RTF [MIC97] que actualiza a su capricho y que además es deliberadamente oscuro.

A pesar de las dificultades del formato, la comunidad usuaria de sistemas abiertos necesita de herramientas de libre distribución que permitan visualizar documentos en RTF, ya que Windows es el entorno de trabajo más usado en el mundo y la cantidad de documentación escrita en RTF es inmensa.

Un caso particular de esta necesidad es el entorno GNUstep [GNUstepWeb], uno de los proyectos de la FSF. En GNUstep se trata de crear un clon de las estaciones NeXTStep. Estas máquinas empleaban como formato de texto el RTF, con lo que los programas que manejan este formato son un elemento de gran importancia. Hasta hace poco, nadie se ha encargado de ello en el proyecto GNUstep, en parte debido a la poca claridad de la especificación suministrada por Microsoft.

1.2.4. Justificación del proyecto

El objetivo principal del proyecto es proporcionar a la comunidad informática una herramienta libre que permita visualizar archivos RTF. De esta forma, se tenderá un puente entre un sistema no totalmente abierto y el auténtico *software* libre. De las muchas formas en que se puede llevar a cabo este propósito, se ha optado por realizar un conversor entre RTF y HTML. En este apartado se explicarán los motivos de esta elección.

En primer lugar, se ha escogido un conversor de formato, en lugar de un visor directo, por estas tres razones:

- Existen visores libres para visualizar documentos en el formato de salida (HTML), por lo que escribir un visor propio de RTF sería realizar trabajo baldío.
- El conversor es más transportable a otras plataformas.
- Los documentos en RTF se convierten a un formato estándar que es legible por toda la comunidad informática. Una vez convertido un documento, éste es visualizable por cualquier persona, aunque no disponga de la herramienta conversora.

En segundo lugar, se ha elegido el lenguaje HTML como formato de salida, frente a otras alternativas, como LaTeX, PostScript o PDF. LaTeX es un formato en esencia muy similar a HTML, pero se ha descartado por ser utilizado en ámbitos muy reducidos. Por su parte PostScript y PDF están bastante extendidos, pero no tanto como el HTML. PostScript está orientado a la impresión de documentos, lo cual representa a la vez una gran ventaja y un gran inconveniente para el proyecto: la gran ventaja es que el resultado de la conversión puede ser un 100 % fiel al original. Pero el gran inconveniente es que para lograrlo hace falta escribir un *software* de complejidad análoga a la de un procesador de textos, lo cual escapa a la carga de trabajo recomendable para un proyecto de fin de carrera. Algo similar ocurre con el formato PDF.

Aunque la conversión a HTML, por simple, no sea tan fidedigna como lo sería a PostScript, abre al usuario la posibilidad de editar a mano los ficheros de salida. Algo casi impensable en PostScript. Asimismo, HTML es capaz de mantener la estructura interna del documento (títulos, secciones, hipervínculos, etc.). Esta característica será aprovechada en la aplicación del proyecto, que permitirá generar tablas de contenidos y trocear el archivo origen en varios archivos de salida, separados por secciones, algo que el conversor a HTML del propio Office no es capaz de hacer.

En el mercado ya existen herramientas que convierten documentos RTF a HTML. Para empezar, los procesadores de textos, como las *suites* Office de Microsoft y WordPerfect de Corel, son capaces de leer documentos RTF y exportarlos a HTML. El problema es la no gratuidad de estas aplicaciones y la dependencia de entornos determinados. Por otro lado, es sorprendente la poca calidad de la conversión de Office (teniendo en cuenta que es de la empresa autora de RTF).

También se ha encontrado un conversor de RTF a HTML multiplataforma, pero es un producto comercial en régimen de *shareware*, del que además no se distribuye el código fuente. Por tanto, sigue existiendo la necesidad de escribir un conversor gratuito y libre.

A lo largo de todo el tiempo invertido en el proyecto, se han descubierto varios programas libres que manejan el formato RTF, ninguno de los cuales satisface los objetivos planteados en este proyecto. Se puede leer una pequeña discusión sobre estos en el apartado 3.3.

Aún considerando que hay algunas implementaciones libres de analizadores de RTF, el programa sigue siendo muy útil a la comunidad, porque está escrito de forma explícitamente modular: es fácil de añadir a otros programas, modificarlo, mejorarlo, añadirle nuevas prestaciones, o adaptarlo a nuevos entornos o nuevas especificaciones del formato RTF, con lo que se consigue aunar los esfuerzos de la comunidad en mejorar un solo componente que se utilice en muchos programas. Esto, unido a algunos proyectos similares con el formato Word, abre a Linux las puertas de los formatos propietarios.

1.3. Objetivos del proyecto

Podemos distinguir dos tipos de objetivos: los académicos y los del producto del trabajo. Además, se han cubierto otros, como consecuencia de la experiencia del desarrollo del programa.

1.3.1. Objetivos académicos

- Perfeccionar el adiestramiento en la programación de sistemas. Mejorar las habilidades de programación en el lenguaje C++, especialmente el manejo de la biblioteca estándar de la nueva norma C++ ISO.
- Aprender a manejarse con especificaciones técnicas de formatos.
- Adiestramiento en el desarrollo de programas libres, tanto su confección como en el proceso de distribución en la comunidad usuaria.
- Mejorar las habilidades de redacción y maquetación de documentos profesionales con el sistema LaTeX, que es la norma indiscutible, por su calidad y elegancia, en entornos académicos.

1.3.2. Objetivos del trabajo

- Proporcionar a la comunidad informática una herramienta abierta y libre para visualizar documentos en formato RTF, utilizable en una amplia gama de plataformas.
- Contribuir al proyecto GNUstep, antes mencionado, gracias a la experiencia adquirida en el trabajo y los módulos del programa que puedan usarse para escribir un visor RTF en este proyecto.
- Documentar con claridad el formato RTF, ya que la especificación publicada por Microsoft es muy pobre.
- Para la ULPGC, adquirir experiencia en el desarrollo de software libre, tanto en su escritura como en su divulgación.

1.4. Medios utilizados

Los medios para llevar a cabo este proyecto fueron, entre otros:

1.4.1. Equipos físicos

- Una computadora personal tipo Pentium.
- Dispositivos adecuados que permitan la conexión a la red Internet.

1.4.2. Programas

- Sistema operativo Linux, con el sistema de ventanas «X11».
- Entorno de programación en C++ (compilador `g++` de GNU, `flex`, etc.) para el desarrollo de la aplicación.
- Entorno de documentación en LaTeX (compilador de LaTeX, previsualizadores de DVI, etc.) para realizar la documentación.
- Un visor de RTF («WordPerfect for Linux» o «Microsoft Word», por ejemplo) y uno de HTML (como «Netscape») para comprobar los resultados del programa.

1.4.3. Sólo programas libres

Es digno de mención que la gran mayoría de los programas implicados fueron enteramente libres. En realidad, todos, salvo los visores de RTF y HTML. Además, los formatos utilizados (LaTeX, TeX, flex, HTML, DVI, PostScript) son abiertos y están perfectamente documentados. La excepción, naturalmente, es el RTF, dado que justamente el problema que tratamos de resolver es que el formato en cuestión no es abierto.

Este uso casi exclusivo de programas libres es uno de los objetivos del proyecto, cumplido satisfactoriamente.

1.5. Planificación

En este apartado se detalla la planificación para afrontar el trabajo.

1.5.1. Plan de trabajo

Estas son las etapas en las que se descompuso el trabajo:

1. Documentarse sobre el formato RTF y las posibilidades del lenguaje HTML.
2. Aprender la biblioteca estándar del C++, en especial la STL (*Standard Template Library*, [MUS97]).
3. Elaborar las especificaciones de la aplicación.
4. Hacer un diseño arquitectónico de la aplicación.
5. Implementar la aplicación (de forma iterativa e incremental).
6. Documentarse sobre la creación de paquetes RPM y Debian, así como y los protocolos habituales para publicar software libre.
7. Configurar la aplicación según lo hallado en el punto anterior y divulgarla en la comunidad informática.

1.5.2. Método de desarrollo

Como en todo buen trabajo de ingeniería del software, se dividió el proyecto en etapas de análisis, diseño e implementación.

Análisis

El análisis preliminar consistió en determinar los diferentes conceptos existentes en los documentos RTF, tarea harto difícil dada la pobre documentación oficial. Como se pudo se identificó todo lo posible, y se procedió a la siguiente etapa, que era la más complicada y la que iba a condicionar el tiempo invertido en el resto del trabajo, tanto de implementación como de posibles revisiones debidas a potenciales fallos encontrados en el análisis. El resultado del proceso realizado es el capítulo 3, donde se hace una descripción del lenguaje RTF.

Diseño

El primer paso dado para elaborar el diseño arquitectónico de la aplicación fue documentarse sobre diseño, consultando tanto documentación sobre el lenguaje UML como estudiando textos de esta disciplina. Entre los primeros libros consultados estaba «The Unified Modeling Language User Guide» [BOO95], que ayudó de manera decisiva en la comprensión de este lenguaje, que es por excelencia la forma de comunicar diseños de programas en la industria informática.

Una vez revisada la documentación necesaria para conocer el lenguaje de representación a usar en la etapa de diseño, se acudió a la mejor fuente de ideas de diseño orientado a objetos para estudiarla. Dicha fuente fue el afamado libro «Design Patterns» [GAM95], empapado de conocimiento sobre ingeniería del software. Básicamente, el libro se compone de dos partes: una introducción y explicación de cómo hacer buenos diseños, con un caso de estudio analizado a fondo, y una colección de «patrones de diseño», que no son otra cosa que ideas reutilizables para incluir y combinar en nuestros diseños.

En este libro se encontró el patrón de diseño del «visitante». Este patrón resuelve la aplicación de diferentes operaciones a los nodos de un árbol, teniendo en cuenta que los nodos son, en general, de diferente tipo, y que queremos apartar las operaciones de la estructura del árbol, así como hacerlas independientes entre ellas. Vista la idea, y dándonos cuenta de que era realmente limpia, sencilla de modificar y fácilmente extensible (cosa que ayudaba a los propósitos del proyecto), decidimos mejorar sobremanera la idea original, para generalizarla, a pesar de aumentar el trabajo inicial estimado.

Implementación

La implementación del programa fue la etapa más larga, dada la envergadura del proyecto y los casos que había que tratar. También ayudó bastante a alargar este tiempo las lagunas de la documentación con la que se contaba, que hizo en diversas ocasiones que se tuvieran que corregir parte de la implementación, detalles de diseño o incluso que se modificaran o ampliaran ciertas ideas de análisis, debido sobre todo a las pruebas y los descubrimientos de naturaleza heurística, muchos de ellos inesperados.

2 *Software* libre

2.1. Un poco de historia

A pesar de que el *software* libre no es un movimiento reciente, ni mucho menos (existe como mínimo desde los 70), parece haber pasado desapercibido durante décadas. Bien es verdad que hasta hace pocos años no ha tomado cuerpo como movimiento social realmente relevante, pero es algo que «siempre» ha estado ahí y ha tenido cierta importancia.

Probablemente el hecho más importante en la historia del *software* libre, que ha provocado la situación actual, fue la decisión de Richard Stallman de iniciar la fundación del *software* libre («Free Software Foundation», o FSF) y con ella el proyecto GNU («GNU's Not Unix»). Se dice que la razón que le llevó a empezar todo esto fue que, al encontrar un problema con el controlador de una impresora, no pudo arreglarlo por no tener disponible el programa original. Su enfado creció cuando, al pedirlo al fabricante, éste se negó a dárselo. En ese momento, Stallman decidió que la industria informática tenía demasiado de negocio y demasiado poco del viejo espíritu científico y académico de colaboración, y que tenía que hacer algo por solucionarlo. Así empezó el proyecto GNU, que consiste en crear un sistema operativo completo, con todas las aplicaciones que pueda necesitar cualquier persona, y hacer que se pueda conseguir de forma completamente gratuita.

El proyecto GNU tomó como referencia al sistema operativo UNIX. Las razones fueron su gran popularidad en aquella época, además de su flexibilidad. El propio Stallman reconoce que *no* es su sistema operativo ideal, pero la mayoría de los aspectos que no eran de su agrado podían modificarse u ocultarse de alguna forma. También era el sistema operativo que más personas sabían utilizar, especialmente si nos concentrábamos en ámbitos académicos. Por todo ello, se empezaron a escribir todas las herramientas que tenían los UNIX de la época, generalmente con extensiones o mejoras de alguna clase. Una de las herramientas de GNU más potentes y renombradas es EMACS, un programa que sirve para escribir texto sin formato, aunque se le han añadido extensiones para leer el correo, navegar por Internet o un sistema de ayuda hipertextual. A la vez que se escribían todas estas herramientas, se empezó con en análisis y diseño de GNU/Hurd, el núcleo que debía sustituir al núcleo monolítico tradicional de UNIX. De hecho, Hurd son unas siglas que significan «Hird of Unix Replacing Daemons». Es decir, que se consideraba obsoleto el diseño de UNIX, y se quería reemplazar con un sistema operativo moderno, basado en las técnicas de micronúcleo.

El diseño e implementación del GNU/Hurd, como la mayoría de los proyectos de GNU, avanzaba con paso lento pero seguro. Esto daba como resultado que las herramientas de GNU estuvieran usándose en sistemas operativos propietarios, por lo general reemplazando a las versiones que venían con el sistema, que solían ser inferiores. Por tanto, a pesar de tener muchas de las herramientas disponibles en UNIX, todavía quedaba el núcleo del sistema operativo, todavía nadie podía tener una computadora exclusivamente cargada con *software* libre, que era el sueño de Richard Stallman.

En algún momento de la década de los 90, un finlandés llamado Linus Torvalds empezó a escribir un sistema operativo tipo UNIX, utilizando las herramientas del proyecto GNU. Ni él mismo sabía lo difícil que era escribir un núcleo, y Linus reconoció en una ocasión que esa misma circunstancia fue la que le llevó a intentarlo. Basándose en sus estudios del sistema operativo Minix, construyó las primeras versiones del núcleo Linux, que anunció en un grupo de noticias en Internet. Su mensaje a `comp.os.minix` fue algo como:

```
¿Cansado de que todo funcione en Minix? ¿Añoras horas delante del ordenador
depurando un núcleo de sistema operativo? Si es así, sigue leyendo, este
mensaje puede ser para ti...
```

El hecho de que Linux avanzara tan rápido, en parte debido a su simple diseño, propició que la FSF adoptara temporalmente a Linux como el núcleo para su sistema. Era rápido, estable y seguro (aunque

no tanto como los UNIX libres basados en BSD, según se dice), lo que le hizo el candidato perfecto para completar el sistema GNU. Mientras tanto, el proyecto GNU sigue trabajando en el Hurd, que será el núcleo de su sistema operativo en el futuro.

Así, actualmente hay dos sistemas basados en el proyecto GNU: el GNU/Linux (llamado normalmente «Linux», a secas) y el GNU/Hurd, que se puede decir que es el sistema GNU verdadero. Todavía el Hurd está en fase de desarrollo, así que los sistemas GNU/Linux son mucho más populares y muchísimo más usados. En la figura 2.1 se puede ver un diagrama temporal, que, aunque no tiene las fechas exactas, permite hacerse una idea bastante aproximada de las personas y organizaciones que han formado parte de la historia más reciente del movimiento del *software* libre.

Durante los últimos años se ha podido comprobar que el éxito de los programas libres ha actuado como un efecto dominó: cuanto más *software* libre de calidad ha habido, más programadores lo han usado y han dedicado esfuerzos a completarlo y mejorarlo. Los ejemplos más claros de programas libres de calidad que han arrasado en cuanto a uso se refiere son:

- Apache, un servidor web que actualmente potencia a más de la mitad de los servidores de toda Internet.
- MySQL, un servidor europeo de bases de datos, muy utilizado en la Red.
- PHP, un lenguaje de programación empotrado en páginas HTML que sirve, entre otras cosas, para comercio electrónico. Es similar al lenguaje ASP de Microsoft, pero es mucho más general.
- Perl, un lenguaje de programación tremendamente popular entre administradores de sistemas y programadores. Gran parte de los programas CGI para Internet están escritos en este lenguaje.
- Emacs, un programa «multiuso», del que se dice que se arranca cuando se entra en la máquina, y se sale de él cuando se va a dejar la terminal.
- Python, un lenguaje moderno, interpretado y orientado a objetos, que se está imponiendo claramente como uno de los lenguajes de programación de futuro.
- KDE, un gestor de ventanas y conjunto de utilidades asociadas, cuya meta es crear un entorno de trabajo cómodo, intuitivo y eficiente, con una interfaz moderna.
- GNOME, un proyecto similar a KDE, pero oficial de GNU.
- GCC, una colección de compiladores de C, C++, Objetive C, Ada, Fortran, etc. utilizado por muchas organizaciones.
- TEX, un sistema de maquetación de documentos completo.

Puede comprobarse analizando esta lista que el *software* libre ha contribuido a la explosión de servicios en Internet, principalmente con Apache, PHP, MySQL y PostgreSQL.

Como último apunte, vale la pena destacar que hace pocos años[1] se pueden distinguir dos facciones que apoyan al llamado *software* libre: los que siguen denominándolo «Free Software», y los que lo han empezado a llamar «Open Source». Éstos se preocupan más de hacer que los programas libres lleguen a las masas, y no son tan estrictos en cuanto a las licencias. Las cabezas más visibles de este movimiento son Eric S. Raymond, autor del «New Hacker's Dictionary», continuación del mítico «Jargon File» de Guy L. Steele, y el propio Linus Torvalds. Los que siguen usando «Free Software» para referirse al *software* libre están más preocupados por la comunidad y por conservar su «pureza», es decir, porque lo que llaman «la comunidad» se pueda seguir llamando de esa manera, y están más atentos al uso de licencias que puedan restar, a la larga, libertad al individuo para compartir en cualesquiera circunstancias los programas con sus colegas, para resolver los problemas a su manera y utilizar estos programas de la manera que le plazca. Los seguidores más visibles de este movimiento son la FSF y, en particular, Richard Stallman. En general, los seguidores del movimiento Open Source [OSSWeb] piensan que el movimiento «Free Software» es extremista, y los seguidores de éste piensan que aquél no se preocupa suficientemente por la libertad.

2.2. Diferencias entre *software* libre y propietario

La comunidad de usuarios de *software* libre siempre insiste en que lo importante es la libertad, no la gratuidad de los programas[2]. Tanto es así, que al propio Richard Stallman no le preocupa la superioridad

[1] Concretamente, desde la introducción de la expresión «Open Source», o «Fuentes abiertos»

[2] Recordemos que libre, en inglés, es «free», que puede traducirse también como «gratis». Para hacer la diferencia, Stallman propone «think free speech, not free beer», o sea «piensa en libertad de expresión, no en barra libre»

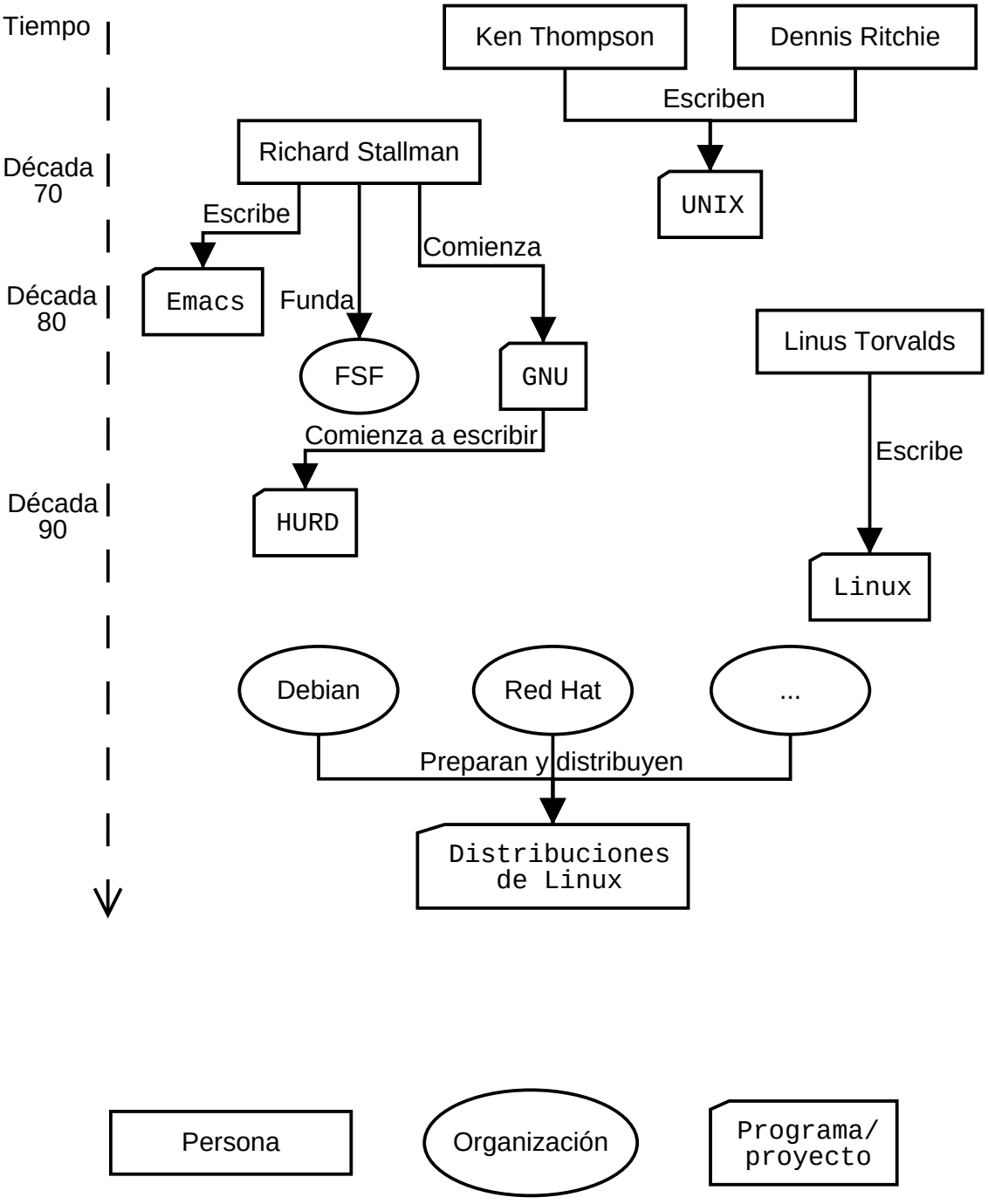


Figura 2.1: Diagrama temporal de la historia de GNU y Linux

técnica del *software* libre: dice que, aunque no fuera mejor que el «tradicional», seguiría usándolo, porque él no vende su libertad por la conveniencia de la situación. Pueden encontrarse discusiones interesantes sobre estos asuntos en la página oficial del proyecto GNU [GNUWeb], y en el artículo «The Cathedral and the Bazaar», de Eric Raymond [ESR97].

Al contrario de lo que mucha gente piensa, el *software* libre no trata de hacerlo todo gratis: simplemente se trata de otro modelo de producir programas y documentación. En vez de centrarse en hacer los programas para poder venderlos, se centra en construirlos para solucionar problemas. Por ello, muchos de los proyectos los llevan programadores a los que poco importa que sea o no fácil de utilizar. Su único afán es arreglar, y su solución la comparten con el resto de la comunidad. Esto propicia que:

- El *software* libre, normalmente, sea técnicamente mejor, ya que lo importante es arreglar el problema bien y de forma general, no suele haber intereses económicos o prisa por terminar el trabajo.
- Las interfaces de usuario suelen estar poco cuidadas, aunque en algunas aplicaciones grandes se están haciendo esfuerzos verdaderamente asombrosos.
- Los programas suelen estar especializados en pequeños problemas, y con frecuencia se ayudan de componentes externos.
- La resolución del problema no se centra en ser fácil de utilizar o resolver los problemas de forma especialmente rápida o cómoda para un usuario medio: estos asuntos se dejan a terceros. Esto conlleva también que no se cometan fallos garrafales como permitir extensiones a los programas que permitan fácilmente escribir *virus*.

Por otra parte, el hecho de que el *software* libre tenga la versión original del programa disponible, y de que se pueda modificar y se obligue a publicar de la misma manera los resultados, implica que:

- Los programas suelen ser más fiables, en tanto en cuanto hay muchas más personas «expertas» utilizando el programa, y pudiendo arreglarlo. Como dice Eric Raymond, «dados suficientes ojos, se encuentran todos los fallos»[3].
- Las modificaciones son más frecuentes, ya que cualquiera puede aportar algo al programa.
- El trabajo está muy descentralizado, y se puede repartir mejor entre muchas personas.
- Las características y prestaciones de los programas «libres» se adecúan más a los gustos y necesidades de la comunidad de usuarios activos (que aportan, ya sea a base de modificaciones o sugerencias).
- Rara vez se duplican esfuerzos innecesariamente con proyectos grandes.

Pero no todo iban a ser ventajas. Debido a sus curiosas características, el *software* libre adolece por lo general de:

- La documentación es escasa o incluso inexistente en algunos casos, si exceptuamos, por supuesto, la «mejor» documentación que se puede tener: el programa original disponible.
- Diferentes componentes del mismo programa pueden tener una calidad muy distinta, porque cualquiera puede aportar.
- Es difícil encontrar una solución «canónica» a los problemas: cada uno los resuelve a su manera, y montar un sistema completo puede ser difícil, por el mar de infinitas posibilidades de combinación que tenemos.
- Algunos trabajos, especialmente los no técnicos, como traducir programas a otros idiomas o escribir documentación, los hacen con demasiada frecuencia personas que carecen del talento o conocimientos adecuados. Esto lleva a que la documentación y las traducciones del *software* libre estén plagadas de malas traducciones y pasajes realmente incomprensibles o mal redactados.

[3] «Given enough eyeballs, all bugs are shallow»

2.3. El sistema GNU/Linux

Los sistemas operativos conocidos como «Linux» se puede decir que se llaman más propiamente «GNU/Linux». Como se describe en el apartado 2.1, Linus Torvalds empieza a escribir a principios de los 90 un núcleo tipo UNIX. Utilizando las herramientas del proyecto GNU, pudo escribirlo y ponerlo en funcionamiento, pues todas las funciones básicas, las utilidades y herramientas de programación necesarias se habían escrito para este proyecto.

Richard Stallman una vez llegó a decir algo como «un usuario de Linux es un usuario del sistema GNU que no sabe el nombre de su sistema operativo». Hay que aclarar, ante todo, que Linux es solamente un núcleo, y que lo que se conoce como «Linux» en la actualidad, se refiere al concepto de *distribución*, que es el propio núcleo, más un conjunto de utilidades y programas necesarios para que la máquina funcione, y un programa que ayude a instalarla.

2.3.1. UNIX

El sistema operativo UNIX nació a finales de los años 60, de mano de Ken Thompson, en los laboratorios Bell, de AT&T. Heredó muchas características de un antiguo proyecto de los mismos laboratorios, llamado MULTICS. Se suponía que UNIX iba a ser parecido, pero mucho más simple que MULTICS, y de ahí su nombre.

Unos pocos años más tarde, entre el 72 y el 74, Dennis Ritchie diseña el lenguaje C, un lenguaje de nivel medio, para programación de sistemas, expresamente concebido para evitar escribir UNIX en lenguaje ensamblador, ya que éste depende de la máquina. Tanto es así, que UNIX fue el primer sistema operativo en escribirse en un lenguaje que no fuera el nativo de la propia máquina en la tenía que ejecutarse.

A mediados de la década de los 70, AT&T empezó a distribuir gratuitamente UNIX por las universidades estadounidenses. Esto provocó que científicos de todo el mundo pudieran aprender cómo se hizo UNIX, añadirle utilidades que necesitaran, y en general experimentar e investigar. El hecho de no estar escrito en ensamblador le abrió muchas puertas y le permitió incrementar sobremanera su popularidad en esta etapa.

Así, UNIX crece rápidamente, pero no guiado por los intereses económicos de una compañía, sino por el interés científico de miles de programadores, que habían escrito mejoras y extensiones del sistema operativo, o nuevas utilidades, por necesidad propia.

Un punto importante en la historia de UNIX es el año 80, en el cual la Universidad de Berkeley edita su propia versión, que llama BSD (*Berkeley Software Distribution*). Esta versión es un gigantesco paso adelante respecto a las anteriores, ya que añade memoria virtual y funciones básicas de comunicación con los protocolos TCP/IP. En esta época, alrededor del 80 % de los centros de Estados Unidos utilizaba UNIX.

Unos años después de este momento, empiezan a salir compañías que sacan versiones de UNIX, como Sun. AT&T, además, decide lanzar su primera versión comercial de UNIX. Sería el Sistema V. Debido a esta «fragmentación» en UNIX, producida por el hecho de que varias organizaciones estuvieran escribiendo sus propias versiones, a mediados de los ochenta se empieza a intentar regular UNIX, con las normas POSIX, de IEEE. Este tipo de normas, de cumplirse, garantizaban la transportabilidad de las aplicaciones entre diferentes versiones del sistema operativo.

Por toda esta historia, UNIX siempre se ha tenido como el sistema operativo de científicos por excelencia, pero, dada la poca preocupación de los fabricantes por facilitar a los operarios el uso del sistema, siempre ha tenido problemas para abordar el mercado de las computadoras personales.

2.3.2. La conexión de GNU y Linux con UNIX

El propio Richard Stallman, a pesar de reconocer que no es su prototipo de sistema operativo perfecto, afirma que UNIX era la mejor alternativa a la hora de construir su sistema operativo libre, el GNU. El propio nombre del proyecto hace referencia a este hecho, ya que GNU significa «*GNU's Not UNIX*», es decir, «GNU no es UNIX». El detalle de «*no* es...» viene por algunos problemas de derechos de autor que había con el nombre, en la época en la que se empezó el proyecto.

La idea original de Stallman fue construir un sistema operativo compatible con UNIX, por varias razones:

1. Estaba demostrado, a lo largo de todos aquellos años, que la idea general detrás del sistema UNIX era buena.

2. Aprovechar las utilidades ya escritas para UNIX.

3. Todas las personas que supieran utilizar UNIX, que eran muchas en aquella época, sabrían usar, automáticamente, el sistema GNU.

4. Gran parte de los manuales escritos para UNIX servirían para GNU.

Pero todavía había muchas cosas que no estaban bien en UNIX. Principalmente, los problemas eran su interfaz de usuario y su tosco diseño. El núcleo de UNIX, tradicionalmente, se había diseñado de forma monolítica, es decir, como un solo bloque compacto. Esto daba problemas a la hora de modificarlo y añadirle funcionalidad.

Por tanto, el diseño interno del núcleo de GNU, el Hurd, iba a ser radicalmente diferente del de UNIX, a pesar de que, al exterior, iba a dar unos servicios compatibles. Tanto es así, que el diseño actual del núcleo Hurd, que aún no está terminado, permite a los diferentes usuarios cargan sus propios servicios privados, sin molestar a los demás. Es un esquema seguro que permite extender el núcleo sin tener que arrancar de nuevo la máquina ni tener acceso como administrador.

Por otro lado, el núcleo Linux era más una copia de UNIX, tanto a nivel de servicios como a nivel de diseño. Estaba basado en los estudios de MINIX que hizo Linus Torvalds, y en sus experimentos con su computadora 386. Es decir que, en principio, el núcleo Linux estaba concebido como monolítico, y por ello, el propio Linus Torvalds y el diseñador de Minix, Andrew S. Tannenbaum, tuvieron algunas discusiones algo subidas de tono en un grupo de noticias de Minix. De aquí se deduce que Linux no era más que una versión de UNIX, que poco o nada añadía al original. Simplemente era una versión libre que funcionaba en PC's, a pesar de que en la actualidad haya añadido algunas novedades. Es decir, que Linux no es ningún invento original, sino una mera versión para 386 de un sistema operativo ya existente.

2.4. Licencias para *software* libre

Lo que diferencia a un programa libre de uno propietario es la licencia con la que se distribuye. Tanto es así, que muchas licencias son incompatibles entre sí, lo cual *prohíbe* la distribución de programas creados a partir de otros programas distribuidos inicialmente mediante dos licencias diferentes (e incompatibles).

Para que se considere libre a una licencia, tiene que cumplir ciertas condiciones especificadas en la *Debian Free Software Guidelines*, en el anexo E, apartado E.1.

En el primer «Documento Halloween», uno filtrado, escrito por uno de los empleados de Microsoft, se hacía una clasificación de licencias de programas, libres y no libres, que intentaban explicar las diferencias entre unas y otras. La tabla usada para resumir la discusión está representada en la tabla 2.1. En el apartado E.5 del anexo E se puede leer la breve explicación de todas las licencias que acompañaba a la tabla. Aquí sólo se discutirán las licencias libres más importantes:

GPL Es la licencia principal del proyecto GNU. Permite la libre distribución de binarios y fuentes, vender los binarios, a precios arbitrarios, siempre que se den los fuentes, o una forma de conseguirlos. Cualquier modificación de un programa GPL *obligatoriamente* se distribuirá bajo la licencia GPL. Además, cualquier programa que se produzca enlazando programas GPL con otras licencias, tendrá que distribuirse por la licencia GPL.

LGPL Es la licencia del proyecto GNU para los paquetes de funciones y similares. Es básicamente como la GPL, sólo que permite que un programa producido enlazando LGPL con otra licencia, se distribuya por otra licencia.

BSD Permite la distribución y el uso libres, tanto de ejecutables como de ficheros fuente. Además, se permite coger programas distribuidos por la licencia BSD, y modificarlos, distribuyendo las modificaciones por otras licencias. Es decir, que uno podría modificar un programa distribuido por la licencia BSD, y hacerlo comercial, cerrándolo. Por lo general, el pequeño grupo de programadores que crea programas BSD *no* acepta contribuciones de otras personas.

Apache Es similar a la BSD, sólo que es normal que otras personas contribuyan arreglos a los programas de licencia Apache.

Artística Es una licencia libre, que permite cierto control, por parte del autor, sobre los cambios que se harán en el programa.

Además de estas licencias para programas, se ha estado experimentando, dentro y fuera del proyecto GNU, con licencias pensadas para otras entidades. Actualmente, hay licencias para «contenido» (cualquier trabajo intelectual), libros, documentación, e incluso hace poco ha surgido una pensada para trabajos artísticos, como música o poesía.

Característica	Gratuito	Redistribuible	Uso ilimitado	Código fuente disponible	Código fuente modificable	Aceptan contribuciones externas	Todos los derivados deben ser libres
Comercial							
Programas de prueba	X	X					
Uso no comercial	X	X					
Shareware	X	X					
Binarios gratuitos («*Freeware*»)	X	X	X				
Funciones gratuitas	X	X	X	X			
Software libre (Estilo BSD)	X	X	X	X	X		
Software libre (Estilo Apache)	X	X	X	X	X	X	
Software libre (Estilo Linux/GNU)	X	X	X	X	X	X	X

Cuadro 2.1: Clasificación de las licencias de programas informáticos

2.5. El negocio del *software* libre

Como se dijo en el apartado 2.2, el movimiento del *software* libre no es uno ingenuo en el que se quiera compartir todo y rechazar sistemáticamente el dinero u otras compensaciones de otro tipo. En realidad, casi todos los componentes de la comunidad buscan compensaciones, ya sean de tipo económico (indirecto), personal, social o simplemente técnico. Para mostrar la heterogeneidad de la comunidad, Eric S. Raymond escribió en «Homesteading the Noosphere»[4] [ESR98]:

> La ideología de la cultura del *software* libre de Internet (en lo que los *hackers* dicen que creen) en un asunto complicado en sí mismo. Todos los miembros están de acuerdo en que el *software* libre (esto es, los programas libremente distribuibles y que se puede modificar para hacer evolucionar y modificarlo según las necesidades del momento) es algo bueno y se merece un esfuerzo colectivo significativo. Este acuerdo efectivamente define quién es miembro de esta cultura. Sin embargo, las razones por las que diferentes individuos y varias subculturas tienen este convencimiento varían considerablemente.
>
> Un grado de variación es el extremismo; si el desarrollo del *software* libre se ve como los medios más convenientes para conseguir algún fin (buenas herramientas, juguetes divertidos y juegos interesantes que practicar) o como un fin en sí mismo [...]
>
> Otro grado de variación es la hostilidad contra los programas comerciales o las compañías que parecen dominar ese mercado. [...]

No sólo buscan algún tipo de compensación los integrantes de la comunidad: además, se puede decir que el *software* libre es todo un modelo de hacer negocios, es decir, *no* va en contra de la economía. De hecho, muchos llegan a afirmar que el modelo es más activo económicamente, y más capitalista que el tradicional, porque promueve la competición y anima a más personas a participar en los negocios, por hacer más difíciles los monopolios y los controles unilaterales de los programas.

Hay varios argumentos a favor de que el modelo de negocios del *software* libre no sólo no destruye puestos de trabajo, sino que de hecho puede llegar a crearlos:

1. En general, dificulta mucho las posiciones excesivamente dominantes, el control de los formatos de fichero utilizados y los monopolios.

[4] Traducción libre del original

2. Permite que haya técnicos especialistas en adaptar programas existentes a compañías, o en dar apoyo técnico muy específico. Por ejemplo, hace posible que una organización contrate a una persona para adaptar un programa a sus máquinas o a sus circunstancias, en vez de tener que escribir el programa completo.

3. Reparte el trabajo, haciendo que pueda haber involucradas muchas partes en un mismo proyecto (en la interfaz, en el corazón del programa, en el empaquetado y facilidad de instalación, en la interoperabilidad con otras aplicaciones, en la adaptación final a las máquinas donde se ejecutará, etc.)

4. Anima a innovar e investigar, puesto que reduce la conveniencia de inventar una y otra vez la rueda y sacar continuamente las mismas aplicaciones con unas pocas mejoras.

5. Fomenta la libre competencia y la atención a los usuarios, por la imposibilidad de escudarse en secretos, y la presión del mercado por diseñar productos mejores.

Por tanto, el movimiento del *software* libre favorece a usuarios, programadores, aficionados, y a la mayoría de los empresarios. Se puede decir que el único grupo que no se ve favorecido por el movimiento son las grandes empresas de *software*, que ven peligrar su poder, para verlo repartido por toda la industria. No es algo de lo que haya que huir, ya que hace avanzar a la técnica, nos da problemas interesantes que resolver y recupera el viejo espíritu científico y académico de compañerismo.

Como ejemplo de proyecto que apoya todo esto, podemos citar a Cosource ([CSWeb]), que mantiene una página para hacer de intermediarios entre organizaciones que necesitan programas y programadores dispuestos a trabajar en proyectos libres por dinero. Parafraseándoles de su propia página:

> Cosource.com es una página de colaboración y de subastas inversas que permite que clientes internacionales y programadores de *software* libre trabajen conjuntamente para financiar el desarrollo de soluciones innovadoras.
>
> El proceso resumido es:
>
> 1. Los miembros piden nuevos paquetes de *software* libre, mejoras o documentación, y dan los requisitos funcionales completos según las instrucciones dadas en el formulario «Enviar petición».
> 2. Los programadores envían propuestas para desarrollar el paquete, nombrando a una «Autoridad», o tercera parte, además de un presupuesto (precio) y un calendario del proyecto.
> 3. Los miembros examinan las propuestas, y podrán elegir financiar una o más... la mínima aportación es de 10 dólares americanos.
> 4. La primera propuesta que consiga reunir suficientes respuestas como para cubrir el presupuesto, gana, y empieza a desarrollarse.
> 5. Una vez que la Autoridad declare completo al proyecto, se publica... y entonces todos los miembros que asignaron fondos al proyecto pagan a sus desarrolladores con tarjeta de crédito.
> 6. Cosource.com, entonces, paga al desarrollador y a la Autoridad por su trabajo, excepto un justo porcentaje.
>
> Los miembros se benefician:
>
> - Teniendo la posibilidad de especificar que paquete libre necesitan.
> - Compartiendo los costes de desarrollo con otros miembros, quizás ahorrándose hasta un 99 % que si pagaran a los programadores ellos mismos.
> - Eliminando su implicación en contratos de negocios tediosos y asuntos de pagos.
>
> Los programadores se benefician:
>
> - Haciéndose un sitio (los menos experimentados) en el mundo del *software* libre, llevando trabajos de pequeña escala, pero pagados.
> - Permitiendo a los programadores más experimentados, potencialmente, ganarse la vida escribiendo paquetes libres.
> - Teniendo una oportunidad de sondear un mercado potencial antes de enviar una propuesta para llevar a cabo un trabajo.

> Y, por supuesto, la comunidad gana más programas libres.
> ¿Está preparado para hacer que Cosource.com trabaje para usted?

Para terminar, se mostrarán algunas de las posibles fuentes de ingreso de las compañías que se dedican al *software* libre:

- Vender los productos empaquetados en caja, preferentemente con los manuales impresos en papel.
- Ofrecer apoyo técnico telefónico a usuarios.
- Ofrecer apoyo técnico a organizaciones.
- Adaptar programas existentes organizaciones.
- Vender equipos informáticos con sistemas operativos libres preinstalados y preparados para su uso inmediato.
- Mantenimiento y mejora de programas.
- Mantenimiento de sistemas informáticos.

Podemos citar a algunas de las compañías que se dedican a distribuir Linux, como SuSE, Caldera, RedHat o Stormix, como ejemplos de empresas que han tenido éxito trabajando con este modelo. Estas organizaciones han demostrado que es perfectamente posible ganarse la vida, e incluso hacerse rico, sin intentar evitar que los usuarios de sus programas los compartan con sus colegas y puedan verlos, mejorarlos, y aprender examinándolos, y sin entorpecer el avance de la ciencia.

2.6. La comunidad de programadores

La comunidad del *software* libre es, en ciertos aspectos, bastante convervadora: sigue convenciones históricas, mantiene costumbres de los antiguos expertos, y, en general, siempre intentan seguir los consejos y tradiciones de los que estaban antes. Si todo funcionaba, debe ser que no lo hacían tan mal, después de todo. Intentan a toda costa no reinventar nada, y no pensar arrogantemente que se puede venir con una solución radicalmente nueva y mejor si el problema han estado resolviéndolo exitosamente de una determinada manera durante décadas. Al fin y al cabo, «el origen cultural de la mayoría de los *hackers* les enseña que desear satisfacción con el ego es una motivación mala (o al menos inmadura)» [ESR98].

Muchas de estas costumbres a la hora de escribir programas, decidir sus nombres, plantear soluciones, etc. están cubiertas en otro documento escrito por Eric S. Raymond, llamado «Software Release Practice HOWTO» [ESR99]. Este documento es de imprescindible lectura para aquellos que quieran conocer las costumbres de los programadores de *software* libre o bien quieran ellos mismos crear más programas de este tipo. En él se detallan prácticas y costumbres que facilitan a todos construir y utilizar el programa, y a los programadores ayudar a mejorarlo y a hacerlo crecer. Citando el resumen del documento[5]:

> Este COMO describe buenas prácticas de publicación de proyectos libres en Linux. Siguiendo estos consejos, hará lo más fácil posible a los usuarios utilizar su programa y construirlo, y a otros programadores entenderlo y cooperar para mejorarlo. Este documento es de lectura requerida para los programadores novatos. Los más experimentados deberían echarle un vistazo cuando estén a punto de publicar algún nuevo proyecto. Se modificará periódicamente para reflejar la evolución de lo que se entiende por buenas costumbres.

Esto no significa, en absoluto, que se nieguen a evolucionar o a descartar vestigios del pasado cuando es necesario o simplemente conveniente. De hecho, la comunidad del *software* libre es capaz de escribir desde cero un programa que funcionaba perfectamente, por ser conveniente a la larga, bien por profundos cambios de diseño, o simplemente por buscar eficiencia o simplicidad.

[5] Traducción libre del original en inglés

2.7. Cómo producir *software* libre

Teniendo en cuenta las características tan particulares y marcadas de la comunidad del *software* libre, es natural pensar que, por respeto, es conveniente respetar al menos parte de sus costumbres. Esto ayuda a que todo sea más homogéneo, y de alguna forma concede a la comunidad el trabajo que ha estado realizando durante años.

Por tanto, vale la pena tener en cuenta las siguientes reglas y consejos para escribir y publicar software libre:

1. Intentar no escribir un programa que ya está escrito. Ya hay suficientes repeticiones por la Red. Si hay algo similar a lo que queremos hacer, y es posible, debemos intentar modificarlo primero.

2. Escribir el programa, si es posible, en algún lenguaje de programación para el que hayan buenos compiladores/intérpretes libres. Preferiblemente, C o algún lenguaje interpretado.

3. Anunciar el proyecto, una vez haya una versión «visible», en foros como `freshmeat.net` [FMWeb] o en grupos de noticias como `comp.os.linux.announce` (si el proyecto es para Linux). Si el proyecto, por sus características (estar escrito en Perl o en Python, ser científico, etc.) tiene otros sitios apropiados en Internet donde anunciarlo, duplicar los anuncios.

4. Hacer una página web para el proyecto, con información sobre él, noticias, enlaces relacionados, preguntas comunes, etc. Hay máquinas en Internet que dan páginas gratuitamente, especialmente si la página es de un proyecto de *software* libre.

5. Crear paquetes con los fuentes que se descompriman en un único directorio. Si utilizamos las herramientas `autoconf` y `automake`, tenemos éste y otros problemas resueltos de entrada.

6. Si es posible, crear una lista de correo para el intercambio de ideas y opiniones entre los programadores.

7. Si es posible, crear paquetes fácilmente instalables, para facilitar a los novatos el aprovechamiento del programa.

2.7.1. Los paquetes

El último punto no se sigue con demasiada frecuencia. En realidad, la mayoría de los programas sólo se distribuyen en su forma original. Esto es ideal para programadores pero incómodo para los «usuarios finales». El trabajo de facilitar la instalación del programa suele correr a cargo de otros, por lo que el trabajo se distribuye, y se hace mejor y más rápido. Por lo general, la forma fácil de gestionar los programas instalados en una máquina son los sistemas de paquetes, que son ficheros que contienen programas completos, o trozos, junto con información sobre sus requisitos o su contenido. Principalmente, hay dos sistemas de paquetes usados en las diferentes distribuciones de Linux: RPM y Debian. Éste lo utilizan la distribución Debian, la Storm Linux y la Corel Linux; aquél, RedHat Linux, Mandrake, Caldera, SuSE, y otras. Además de estos sistemas de paquetes, hay una forma de empaquetar los programas, mucho más primitiva, que utiliza la distribución Slackware. Consiste en comprimir los programas tal cual quedan instalados, sin información sobre su contenido ni requisitos de instalación. A estos efectos, no consideramos la forma de empaquetar programas de Slackware como un sistema de paquetes.

2.7.2. Formatos de paquetes

Tal y como se acaba de comentar en el anterior apartado, hay principalmente dos formatos de paquetes en las distribuciones de Linux: el RPM y el Debian[6]. Tristemente, las distribuciones de Red Hat, o sea, las que usan el formato RPM, no se han puesto de acuerdo en algunos aspectos, lo que hace que ciertos paquetes RPM no sirvan para todas las distribuciones que comparten formato de paquetes. Esto hace que el RPM, a pesar de ser utilizado por la mayoría de las distribuciones, no sea tan universal como pretende.

En cambio, por ahora, las distribuciones basadas en Debian (Storm Linux, Corel y Libranet) han cambiado aspectos suficientemente independientes de los paquetes, como para que los de una sirvan para los de otra.

[6] También está el de Slackware, que no consideramos un «paquete» en la mayoría de los sentidos, y los SLP, de la distribución Stampede Linux, pero es muy minoritaria, y sólo esa distribución los usa

Aunque para ciertos proyectos (por ejemplo, el actual) no es necesaria la existencia de paquetes, por estar más pensados para programadores, siempre es conveniente tener disponible una versión fácil de instalar. Para ello, se ha estudiado cómo construir tanto paquetes RPM como paquetes Debian. Aunque en el anexo D se describe con mucha más profundidad, aquí se hará un pequeño resumen.

Paquetes RPM

Para construir un paquete RPM debemos tener en cuenta dos ficheros: uno general, el `/etc/rpmrc`, y uno por cada paquete, el fichero *spec*. En el primero va información sobre el empaquetador y otros datos que en principio son comunes a todos los paquetes producidos en la misma máquina. En el fichero *spec* se describen los contenidos del paquete, y se especifican las instrucciones para compilarlo e instalarlo. Además, en éste hay una lista de los ficheros que compondrán el paquete final.

Los pasos que hay que dar para construir un paquete RPM son:

1. Comprobar de que el fichero `/etc/rpmrc` está perfectamente configurado para el sistema.
2. Hacer los cambios necesarios a los fuentes del programa para que compilen en la máquina en la que se va a construir el paquete.
3. Hacer un parche con los cambios hechos en el paso anterior para que los fuentes compilaran correctamente.
4. Escribir el fichero *spec* para el paquete.
5. Asegurarse de que todo está donde debe estar.
6. Construir el paquete utilizando la orden `rpm`.

Paquetes Debian

Los paquetes Debian se pueden construir con las propias herramientas de la organización que mantiene la distribución. Una de ellas es `deb-make`, que prepara unos fuentes para crear un paquete Debian de ellos. Aunque se considera obsoleta, por haber herramientas más modernas disponibles (todo un conjunto de ellas), sigue funcionando perfectamente, al menos para pequeños proyectos con pocas pretensiones.

Otra de las herramientas necesarias para construir el paquete en sí, una vez preparado, es el *dpkg-buildpackage*. Este programa se encarga de crear un paquete Debian a partir de un árbol de directorios con los fuentes de un programa, ya preparado para crear un paquete Debian.

Por tanto, y siempre que hayamos utilizado las herramientas `autoconf` y `automake` (si no, puede que tengamos que hacer algunas modificaciones), tendremos el paquete listo después de haber tecleado:

```
$ deb-make
.... [Salida del deb-make]
$ dpkg-buildpackage
.... [Salida del dpkg-buildpackage]
```

Tal y como se comenta en el anexo D, apartado D.2, hay otras herramientas que nos pueden ayudar a construir paquetes Debian, que además se consideran más modernas. Los dos paquetes de herramientas principales, sin contar con el `deb-make`, son `devscripts` y `debhelper`. Sobre todos estos programas se pueden encontrar información en la propia distribución Debian, y en la referencia bibliográfica [SAN99].

3 El formato RTF

RTF es uno de los formatos nativos del Microsoft Word. Teóricamente, todas y cada una de sus opciones son representables en este formato. Teniendo en cuenta que las opciones de Word se han ido añadiendo a lo largo de los años, el RTF es bastante desordenado.

3.1. Características generales

Las características más relevantes, en lo que respecta al trabajo realizado, son las siguientes:

1. Es uno de los formatos nativos de Microsoft Word y del Wordpad, es decir, cualquier característica del programa se puede plasmar en un documento en formato RTF. Esto, además, implica que todas las computadoras actuales que utilicen Windows pueden producir documentos de este tipo.
2. Es un formato de texto, como el HTML, el LaTeX o el PostScript, y no como el otro formato nativo del Word. Esto facilita mucho el análisis de los documentos, y la creación y depuración de analizadores informáticos.
3. Se intuye que se puede convertir directamente a un documento binario de Word. De ser cierto, esto hace que, teniendo un analizador de RTF, se pueda conseguir fácilmente uno de documentos Word.
4. Hay un documento oficial de Microsoft de especificación del formato, aunque está plagado de errores y omisiones.
5. Está completamente enfocado a la descripción física del documento, no a describir su estructura lógica. Prima la velocidad de interpretación del formato, no su generalidad. Esto dificulta mucho su conversión a otros formatos, al menos si están basados en la estructura lógica del contenido, como el HTML.

El formato RTF existe, como mínimo, desde el Word 6. Las versiones anteriores de RTF, en caso de existir, son completamente irrelevantes para el proyecto, dado que la primera versión de Word que se impuso como la norma de uso fue la 6, que funcionaba en Microsoft Windows 3.1. A lo largo de los años, el RTF ha ido sufriendo variaciones y extensiones, a la par que el formato Word. Pero, mientras que en el programa Microsoft Word se ha diferenciado siempre muy bien entre versiones incompatibles del formato principal, no se ha hecho una diferenciación clara entre las diferentes versiones de RTF. Además, por lo que se ha podido descubrir investigando, los cambios del RTF de Word 6 al del Word 97, la siguiente versión, son bastante desconcertantes, de tal manera que, excepto teniendo mucho cuidado, un analizador del RTF producido por el Word 6 no sabría interpretar un documento RTF creado con Word 97.

Esto crea muchos conflictos a la hora de diseñar un analizador de documentos RTF: el formato está lleno de casos particulares, incorporados durante varios años, el diseño del programa tiene que irse adaptando a los «caprichos» de diseño del RTF, y es difícil encontrar generalidad y abstracción en un lenguaje tan condicionado por las circunstancias (el diseño del programa que iba a manejarlo y la necesidad de rapidez para mostrarlo de forma WYSIWYG[1]).

Pero, aunque parezca mentira, esos no son las mayores dificultades a la hora de programar un analizador de documentos RTF. Por si fuera poco, el formato está muy pobremente documentado y la especificación oficial está llena de lagunas, omisiones e incluso fallos estrepitosos. Estos dos hechos unidos hacen perder las esperanzas a muchos programadores, o al menos les incitan a hacer tan solo implementaciones parciales y claramente insuficientes, que no terminan de arreglar el problema. Toda esta situación

[1] *What You See Is What You Get*, es decir, «lo que ves es lo que tienes»; es un paradigma de diseño de programas de redacción de texto, en el que el programa tiene que mostrarlo tal y como va a quedar al final. Es cómodo para el usuario, pero tiene muchos inconvenientes

se agrava teniendo en cuenta que cada cierto tiempo la norma RTF avanza y crece, y la especificación, aunque madura con la norma, nunca llega a documentarla toda.

3.2. ¿Por qué RTF?

Como se ha podido inducir de toda la discusión anterior, el formato RTF es uno de los dos formatos nativos de Microsoft Word, el programa para redactar texto más popular del mundo, actualmente. Esto, por supuesto, no implica que sea el mejor, simplemente que es el que más personas usan. Teniendo en cuenta la naturaleza de los formatos de ficheros informáticos, está claro que, para poder intercambiar documentos con otras personas, tiene que utilizarse el mismo formato, o al menos uno que ambos extremos comprendan.

Esto ha hecho que muchos hayan intentado construir programas que manejaran tanto el formato RTF como el Word. Y, debido a las razones argumentadas al principio del capítulo, muchos de ellos han fracasado o se han quedado cortos.

Por otro lado, si el objetivo del proyecto era entender la mayor parte de ficheros en formato cerrado que hay hoy en día en el mundo, ¿por qué no escoger el formato Word, que al fin y al cabo es el formato que utiliza por defecto el programa, y, por ende, el más usado? Por varias razones:

1. Porque es más fácil de transportar y modificar por personas, al estar compuesto de texto ASCII, utilizando exclusivamente caracteres de 7 bits[2].

2. Al ser más sencillo, hay más programas que lo entienden y pueden crear documentos en el formato, aunque sea a nivel primitivo. Esto acelera la adopción del formato RTF en plataformas heterogéneas.

3. Debido a la misma razón, hay más personas dispuestas a ayudar, o que hayan tenido algo de experiencia previa con el formato. Por lo tanto, tendrá mucho más éxito como proyecto de *software* libre.

4. Al ser el RTF uno de los formatos nativos del Word, uno puede fácilmente traducir sus documentos en formato Word a formato RTF, sin pérdida de información, por lo que no entender directamente el formato Word no es un problema tan grave.

3.3. Programas de libre distribución que manejan RTF

Desde antes de empezar el trabajo del proyecto, se buscaron por la Red programas de libre distribución que trataran RTF, y que cumplieran los objetivos marcados para el proyecto. Al ver que no había ninguno de esas características, se mantuvieron para tener diferentes implementaciones de referencia, y para demostrar el sentido que tiene haber trabajado en el proyecto RTHC.

La lista de programas encontrados que manejan documentos RTF es:

AbiWord Un pequeño redactor de textos, libre, empezado por la compañía AbiSource. Reconoce RTF, entre otros formatos, pero todavía está muy *inmaduro*.

GNUstep Es una implementación preliminar de un conversor de RTF al formato interno de GNUstep. Sus mayores desventajas son no ser completo, estar escrito en Objective-C, un lenguaje poco conocido, y ser un conversor a un formato interno de GNUstep, y no a un formato de texto extendido como HTML.

Ted Un programa pequeño, que sólo maneja RTF. Se ha intentado hacer funcionar varias veces, sin éxito. Su interfaz, por si fuera poco, está mal diseñada y es bastante pedestre, y su analizador de RTF tampoco parece diseñado para ser fácilmente reutilizable.

rtf2html Un conversor de dominio público, bastante antiguo, obsoleto, y difícil de entender y modificar. A pesar de que entiende listas y la estructura lógica del texto, no entiende tablas ni puede partir en varios ficheros de salida el original RTF.

Pathetic Writer Un redactor de textos del paquete ofimático Siag Office. Su análisis del RTF es medianamente bueno, aunque no está pensado para modificarlo.

[2] Esto no es estrictamente cierto, véase apartado 3.5

Lexi Word Processor Un programa muy reciente, escrito en Java. No se ha llegado a probar, aunque dada su juventud, no debe ser muy sofisticado.

StarOffice El paquete ofimático de Sun, un programa «libre», según la licencia GPL. Es de gran envergadura. Necesita una máquina bastante moderna para ejecutarse bien, y debe ser bastante difícil modificar el programa. Recientemente, Sun anunció su «liberación», aunque todavía no se ha publicado, ni se hará hasta el 13 de octubre de este mismo año.

Entre los programas que tratan de alguna forma documentos en formato RTF, ya sea produciéndolos, leyéndolos o ambas cosas, podemos decir que no hay ninguno verdaderamente libre y práctico a la hora de adaptarlo a nuevas necesidades, nuevas normas del formato o, simplemente, a otras circunstancias.

A pesar de que durante la realización del proyecto la compañía Sun «liberó» el programa StarOffice, que es probablemente el más utilizado en Linux para leer documentos en formato Word y RTF, en la página de preguntas frecuentes de `http://www.openoffice.org`, la dirección de Sun para este nuevo proyecto, puede leerse algo casi *inquietante*: para que Sun acepte contribuciones de personas ajenas a su empresa, exigen que se pasen a Sun los derechos de autor. Esto hace peligrar la «libertad» de StarOffice, ya que significa que, en cualquier momento, la compañía Sun, como única propietaria de *todos* los derechos de autor sobre el programa, podría cambiar la licencia unilateralmente. Por todo ello, el programa que más se acercaba a nuestro objetivo (aunque es monstruosamente grande, y por tanto será difícil adaptar el analizador de documentos RTF para otros programas) no se puede considerar completamente libre a esos efectos.

3.4. Método de trabajo

Como puede leerse en la referencia del formato que viene en el anexo C de este documento, gran parte del conocimiento sobre el RTF plasmado en esta memoria se ha obtenido investigando, examinando documentos de ejemplo creados con el propio propósito de la investigación, e intentando adivinar las reglas de compartamiento de Microsoft Word a la hora tanto de leer como de escribir documentos RTF.

Todo este trabajo fue muy poco gratificante, sobre todo cada vez que se encontraban casos particulares, contradicciones con la especificación oficial de la que se partía o lagunas en ésta. Lo peor de estos casos era cuando el comportamiento del Microsoft Word no era secundado por otros programas. En todo momento, sin embargo, la prioridad absoluta era comprender y analizar correctamente los documentos producidos por el redactor de Microsoft, que es la norma hoy en día.

Ante todo, este ingrato trabajo se llevó a cabo con los ánimos que infunde saber que es necesario para dar un producto realmente útil a la comunidad del *software* libre, a la que tanto debemos los usuarios de sistemas y programas de este tipo.

3.5. Descripción léxica

Los elementos más básicos que constituyen un documento RTF son:

Palabras de control una serie de caracteres alfabéticos, precedidos de una barra invertida y seguidos, optativamente, por un parámetro numérico (que puede ser negativo).

Símbolos de control una barra invertida seguida de un solo caracter no alfabético.

Caracter especial una barra invertida seguida de un apóstrofo y dos caracteres hexadecimales.

Grupos elementos contenidos entre llaves.

Texto todo lo demás.

En realidad, también se distingue un sexto tipo de símbolo al analizar documentos RTF, que es el contenido binario: se distingue por ser contenido arbitrario, codificado con 8 bits. Esto, por supuesto, contradice el planteamiento inicial de RTF, que estaba pensado para poder enviarse sin problemas por canales de 7 bits. Este último tipo de símbolo se utiliza para datos como imágenes que, aunque se pueden codificar perfectamente con 7 bits, el uso de 8 facilita en cierta manera el tratamiento.

Como llamativa omisión en la especificación oficial, podemos destacar el hecho de que en ningún sitio en las 164 páginas de ésta se puede leer de forma suficientemente explícita que los saltos de líneas no

tienen ningún significado en los documentos RTF, es decir, *ni siquiera* separar palabras. Esto es un hecho realmente sorprendente y poco intuitivo, por lo que debería haberse hecho notar explícitamente en la especificación.

3.6. Estructura de un documento RTF

Los documentos RTF tienen dos partes: la cabecera y el cuerpo. En la primera parte se especifican los tipos de letra y estilos a usar; en la segunda va el contenido del documento. La cabecera tiene una sintaxis relativamente rígida, al menos según la especificación oficial. Sin embargo, se han descubierto algunos fallos y omisiones.

Para el cuerpo, desafortunadamente, no existe una gramática oficial. Se puede decir que el análisis de un documento en este formato es una tarea casi *heurística*, ya que no hay una forma fija y establecida de representar elementos comunes como listas numeradas. Al ser de tan bajo nivel de abstracción, y basarse excesivamente es descripciones «físicas» de cómo debe quedar el resultado final, los productores de documentos RTF tienen la libertad de crearlos como prefieran. Aunque esto puede verse como una ventaja a la hora de diseñar y construir un escritor RTF, es claramente una gran desventaja para cualquier analizador de RTF, incluidos los que tienen que leer este tipo de documentos para luego escribirlos.

Así, los elementos léxicos comentados en el apartado 3.5 se agrupan en el documento RTF prácticamente sin orden establecido. Uno puede encontrarse cualquier combinación de palabras de control, texto y grupos, aunque ciertas combinaciones tendrán significados definidos, como las tablas o las listas. Las especificaciones oficiales tan sólo definen cierta estructura para algunas de estas combinaciones y para (la mayor parte de) la cabecera del documento, proponiendo para ello una descripción en seudo-BNF incompleta e inexacta.

La casi inexistente sintaxis de RTF hace que diseñar un analizador sea complicado. Por si fuera poco, el propio Microsoft Word, la implementación que se ha seguido como modelo, incumple bajo ciertas condiciones la especificación oficial. Esto refuerza el manto de incomprensión y misteriosa oscuridad que rodea a los formatos cerrados en general, y al formato RTF en particular. El primer enfoque que se intentó dar, para simplificar la construcción del analizador sintáctico, de usar *yacc*, resultó inviable. Después de hacer varias pruebas, se optó por escribir a mano el analizador, ya que las reglas sintácticas parecen a veces improvisadas, y añadidas con prisas para cumplir fechas en algún apretado calendario.

A continuación se estudian algunos casos particulares de la descripción sintáctica del formato RTF: los más interesantes son los estilos, las listas, las tablas, los marcadores y los campos. Los últimos, además, se tratan de una manera particularmente pedestre. Naturalmente, se puede encontrar una descripción completa de cada uno de estos conceptos en el anexo correspondiente.

3.6.1. Los estilos

Los estilos también, en cierta manera, están ausentes del vocabulario RTF. Aunque existen, el Word los trata como una simple marca, y necesita de *toda* la definición de éste para representarlo correctamente en pantalla. Esto, como de costumbre, choca con la especificación oficial, que dice que se copia la definición del estilo por compatibilidad con los lectores RTF antiguos. O eso, o la propia Microsoft considera obsoletos sus propios productos.

3.6.2. Las listas

En RTF no existe en sí el concepto de listas de elementos: para representarlas, se limitan a numerar los párrafos, y a inventar un par de palabras de control que indican la profundidad lógica de cada uno de los elementos.

3.6.3. Las tablas

Las tablas tampoco existen por sí solas en RTF. Sólo existe el concepto de fila, lo que lleva a la curiosa, casi absurda, situación de que no podemos tener dos tablas seguidas, porque se convertirían en una sola. Las tablas, por añadidura, no se pueden anidar, lo que, a pesar de agradecerse a la hora de leer RTF, no se agradece en absoluto a la hora de escribir, porque quita flexibilidad. Lo más grave de este asunto es que Microsoft Word intenta evitar que el usuario anide las tablas, ocultando la barra de menús correspondiente, pero falla estrepitosamente cuando se intenta copiar y pegar una tabla dentro de otra: la tabla se descompone, y el resultado es una mezcla entre la tabla original y la pegada, que obviamente no satisface las necesidades del usuario.

3.6.4. Los marcadores

Los «marcadores», en RTF, son marcas que podemos poner a ciertas partes del documento, para poder hacer referencias a éstas mediante un nombre. Su uso más obvio son las referencias cruzadas.

Lo primero que choca de los marcadores de RTF es que hay dos construcciones diferentes para indicar el principio y el final del texto referenciado. Esto provoca que las zonas «referenciables» puedan solaparse, lo que no es legal en, por ejemplo, HTML. Si uno medita sobre ello se da cuenta de que, en las circunstancias en las que se usa el Microsoft Word, lo más lógico es permitir que estas zonas se solapen, pero esto esta decisión hace incómodo el análisis de este tipo de documentos, y a la larga es poco conveniente tener este tipo de características en los formatos de los documentos.

Las marcas tienen la forma:

```
{\*\bkmkstart etiqueta1}Texto referenciable{\*\bkmkend etiqueta1}
```

3.6.5. Los campos

Los campos en Microsoft Word representan texto que no se ha escrito directamente, sino que tiene una forma alternativa de *calcularse*. Por ejemplo: a través de una variable, de la fecha, etc. Estos campos se marcan en RTF con un grupo que empieza con la palabra de control `\field`. Dentro de este grupo hay dos grupos más, que son el grupo `\fldinst` y el `\fldrslt`.

El primero contiene las instrucciones para poder construir el contenido del campo (no tiene por qué ser texto). Estas instrucciones tienen un formato especial de texto, algo similar a las llamadas de funciones en los lenguajes de programación. Estas «funciones» *no* están documentadas en absoluto en la especificación oficial, aunque hay dos o tres ejemplos en los que aparecen estas referencias, pero completamente fuera de lugar.

El segundo grupo, el `\fldrslt` contiene el último resultado hallado. El hecho de tener el último resultado hallado hace más rápido mostrar el texto, aunque es un gasto grande de espacio, y hace al formato más confuso.

3.7. Consideraciones de implementación

El formato RTF está más enfocado a programas que a personas, dado su bajo nivel de abstracción y sus constantes referencias a medidas físicas para ubicar los diferentes elementos que componen el documento en las páginas. Por ello, la interpretación semántica, entendiendo por ello interpretación más allá de lo fijado, es casi nula.

El caso más interesante de interpretación semántica son los grupos RTF, representados por llaves. Su significado semántico es que todos los atributos que hayan aparecido dentro de un grupo se cierran implícitamente al terminar éste. Esto implica tener que llevar una cuenta, por cada grupo, de los atributos que se han abierto, lo que a su vez implica tener que distinguir de alguna manera entre palabras de control independientes y de atributo.

La propia especificación sugiere que se mantenga una pila de estado para el analizador RTF. Cada uno de estos estados debe tener:

- Una lista de atributos abiertos, distinguiendo además entre los de carácter, párrafo y apartado. Esta diferenciación se debe a la existencia de palabras de control como `\plain`, `\pard` o `\sectd`, que cierran los atributos de carácter, párrafo y apartado respectivamente.
- El destino para el que el contenido del grupo va a parar. Puede ser el destino por defecto (texto normal), una nota al pie, un campo, etc.
- Algún tipo de estado interno, por ejemplo para ignorar el texto del grupo.

Es importante recordar que toda esta interpretación y estas estructuras de datos *no* son responsabilidad de *libFreeRTF*, sino del analizador, en este caso de cada uno de los visitantes. Dado que toda esta discusión es más consejos y recomendaciones que imposiciones, se ha creído conveniente *no* incluir ninguna de estas estructuras de datos como atributos de la clase base `RTFVisitor`.

Otro detalle importante que vale la pena comentar es el análisis léxico. Excepto por el manejo de los caracteres de 8 bits, que puede dar algunos problemas, el análisis léxico de un documento RTF revela que es trivial escribir con la ayuda de *lex* un analizador léxico: los simbolos están perfectamente definidos, son pocos, y las reglas para reconocerlos, simples.

4 El lenguaje HTML

HTML es el lenguaje utilizado para describir las páginas de Internet, las páginas *web*. Al principio se creó como un mero instrumento de intercambio de información entre científicos: estaba diseñado para describir el *contenido*, no el formato físico de éste. Con los años, y después de la liberalización de la Red por parte del Gobierno de los Estados Unidos, el lenguaje HTML empezó a evolucinar para introducir cada vez más posibilidades de controlar el aspecto físico de los documentos. La referencia definitiva para el lenguaje HTML se encuentra en la página del consorcio de la *web* (véase [W3CWeb]).

4.1. ¿Por qué HTML?

HTML es una norma aceptada por toda la industria informática: todos los sistemas operativos modernos tienen navegadores de Internet, y es un lenguaje escrito como texto, lo que facilita la manipulación posterior. Otros formatos, como PostScript o PDF, carecen de aceptación suficiente o habrían hecho excesivamente difícil tanto la creación del conversor como la manipulación y aprovechamiento posteriores de los documentos producidos.

A pesar de que el HTML está bastante extendido, el programa no limita necesariamente al usuario a trabajar con HTML estricto: con las plantillas de salida del programa, uno puede crear automáticamente programas en Perl, páginas dinámicas con PHP, o cualquier otro tipo de documento. El texto en sí saldrá convertido a HTML, pero el «envoltorio» de éste se puede configurar al gusto de usuario.

4.2. Método de trabajo

En líneas generales, el HTML usado fue HTML 3.2, una norma bastante antigua, pero relativamente flexible y potente, y que entiende cualquier navegador actualmente. Por un lado, no era positivo aprovechar las extensiones más modernas del lenguaje, porque dejaría a algunos navegadores «a medias» a la hora de interpretar el documento traducido. Por otro lado, en algunas ocasiones era más preciso y más cómodo utilizar HTML 4.0 para ciertos aspectos puntuales de la traducción, como los tamaños de las letras usadas.

Para aprovechar la emergente norma de hojas de estilo de HTML 4.0, se han añadido algunas opciones al programa. Esto permite al usuario utilizar hojas de estilo definidas por él, para personalizar los resultados de las conversiones. Con este enfoque se intentó conseguir tanto la flexibilidad de las hojas de estilo como la seguridad de una norma antigua y por tanto ya consolidada como la HTML 3.2.

4.3. Descripción léxica

Los elementos más básicos que constituyen un documento HTML son:

Contenido caracteres, separados por espacio en blanco. Al principio, tenían que ser caracteres codificables con 7 bits, pero más tarde se ha permitido usar codificaciones establecidas como ISO-8859-1, o latin-1. Ej.: `esto es texto`.

Espacio en blanco espacios, tabuladores o saltos de línea, que se usan para separar palabras.

Entidades caracteres *ampersand* («&») seguidos de caracteres alfabéticos y terminados en un punto y coma. Ej.: `&aacute;`.

Etiquetas son una serie de caracteres alfabéticos, encerrados entre los símbolos «menor que» y «mayor que». Optativamente, estas etiquetas pueden tener propiedades, que se separan con espacio del nombre de la etiqueta, y son, a su vez, caracteres alfabéticos. Si las propiedades tienen parámetros, éstos se especifican con un signo igual y el valor entre comillas. Ej.: `<table border width="100%>`.

4.4. Descripción sintática

La especificación completa de la gramática de HTML puede encontrarse en [W3CWeb]. Está completamente definida en SGML. La *World Wide Web Consortium* incluso tiene marcadas las etiquetas y propiedades desaconsejadas y obsoletas. Es un documento totalmente irreemplazable como referencia, aunque no se puede usar como manual básico de aprendizaje.

Al contrario que con el RTF, las etiquetas suelen dan información sobre el significado de su contenido, y no instrucciones sobre cómo formatearlo. Esto es una ventaja en la mayoría de los casos, ya que nos da información extra sobre el documento, que podemos usar como creamos conveniente, y porque reducimos la dependencia del entorno de visualización. Es decir, que si especificamos que un texto es un título o un ejemplo, no sólo podremos crear índices automáticamente o representar el contenido como queramos, sino que además tendremos la posibilidad de usar los recursos disponibles para formatear el documento de la mejor forma posible. Con la proliferación actual de los dispositivos móviles, como teléfonos o asistentes digitales, la descripción sin formato se hace cada vez más interesante.

A la hora de analizarlo, sin embargo, hay que tener en cuenta que la mayoría de los documentos HTML están *mal* construidos, porque los suelen hacer personas sin suficientes conocimientos, o porque los programas que los crean no son todo lo buenos que deberían. Esto hace que la ventaja de lo estricto del formato se difumine mucho, ya que los analizadores siempre tienen que ser tolerantes a fallos, y éstos se producen, por si fuera poco, con mucha frecuencia.

A pesar de esto, como nuestro cometido será *producir*, y no *analizarlo*, esto no era ningún problema.

4.5. Comparación con el formato RTF

En general, el HTML marca la semántica de los datos, no su aspecto. Normalmente este enfoque es mejor para intercambiar documentos, aunque no permite especificar de forma precisa el formato. El RTF, por otro lado, es un formato diseñado eminentemente para su uso con el programa Microsoft Word. Aunque teóricamente RTF se creó para intercambiar datos, su manera de describir el texto es tan poco abstracta que en la práctica es poco recomendable para intercambiar información entre entornos heterogéneos.

Esto produce un «choque generacional» muy fuerte entre ambos formatos, lo que complica la traducción de uno a otro, y más, si estamos traduciendo del formato físico al abstracto. Este es el caso del proyecto, en el que, al pasar de un formato con poca información a uno con mucha, tenemos que «recuperar» información[1], normalmente con medidas heurísticas.

Los casos más llamativos de estas dificultades de traducción son los atributos físicos de las letras usadas, las tablas, las listas y las referencias cruzadas. El mejor caso de traducción es la mayoría del texto, que es exactamente igual, y los caracteres extendidos, que básicamente los podemos traducir con una simple tabla.

4.5.1. Atributos físicos

En HTML, que es un lenguaje orientado a la descripción precisa de la información, las etiquetas que fijan atributos del contenido, lo rodean, con lo cual siempre queda explícitamente marcado el principio y el fin de éstas. En RTF, en cambio, las palabras de control (el equivalente a las etiquetas HTML) sólo marcan el principio de su actuación, y se cierran implícitamente de varias maneras. Esto da algunos problemas en la traducción, ya que, a pesar de que muchas de las etiquetas se pueden traducir literalmente, hallar los cierres conlleva mantener una pila con todos los atributos abiertos. Por si fuera poca la complicación, en teoría hay también cierres explícitos de las palabras de control, que podrían aparecer en cualquier momento. Esto es un problema gravísimo, ya que en HTML los entornos tienen que ir contenidos unos en otros, *no se pueden solapar*, con lo que hay muchos casos que podrían dar muchos problemas en la traducción. Por ejemplo, el siguiente texto en RTF:

[1] Algo en realidad imposible, según los postulados de la Teoría de la Información

```
...\i\b negrita y cursiva, \ul subrayado también, \i0 el resto en negrita
    y subrayado...
```

Debería traducirse al siguiente HTML, debido a que los entornos de actuación de las diferentes etiquetas no se pueden solapar:

```
...<i><b>negrita y cursiva, <u>subrayado también, </u></b></i><b><u>el resto
    negrita y subrayado...
```

En realidad, este problema no fue tan grande en el proyecto, ya que Microsoft Word y otros programas de los cuales se han sacado ficheros RTF no usan de esa manera las palabras de control. Por lo general, ese mismo texto habría estado escrito como:

```
...\b{\i negrita y cursiva, \ul subrayado también, }\ul el resto negrita y
    subrayado...
```

Con lo cual, la traducción es mucho más sencilla:

```
...<b><i>negrita y cursiva, <u>subrayado también, </u></i><u>el resto negrita
    y subrayado...
```

4.5.2. Las tablas

En RTF, el concepto de tablas *no existe*. Se limita a una colección de filas seguidas. Dado que, como es natural, en HTML *sí* que se marcan explícitamente las tablas, esto da también bastantes problemas a la hora de traducir.

Las «tablas» de RTF son colecciones de filas cuyo principio lo marca la palabra de control `\trowd`. Luego, le sigue una descripción de todos los atributos de las celdas interiores a la fila, y después el contenido en sí de éstas. El principio del contenido lo marca la palabra de control `\pard`; el final de cada celda, la palabra `\cell`; el final de toda la fila, la palabra `\row`. Una tabla básica RTF, de dos filas por tres columnas, es algo como:

```
\trowd\trql\clbrdrt\brdrs\brdrw1\brdrcf0\brsp55\clbrdrb\brdrs\brdrw1\brdrcf0
\brsp55\clbrdrl\brdrs\brdrw1\brdrcf0\brsp55\cellx3324\clbrdrt\brdrs\brdrw1
\brdrcf0\brsp55\clbrdrb\brdrs\brdrw1\brdrcf0\brsp55\clbrdrl\brdrs\brdrw1
\brdrcf0\brsp55\cellx6648\clbrdrt\brdrs\brdrw1\brdrcf0\brsp55\clbrdrb\brdrs
\brdrw1\brdrcf0\brsp55\clbrdrl\brdrs\brdrw1\brdrcf0\brsp55\clbrdrr\brdrs
\brdrw1\brdrcf0\brsp55\cellx9972
\pard\intbl\pard\plain \intbl\s4\sa120\i\b\qc 1.1
\cell\pard\plain \intbl\s4\sa120\i\b\qc 1.2
\cell\pard\plain \intbl\s4\sa120\i\b\qc 1.3
\cell\row\pard
\trowd\trql\clbrdrb\brdrs\brdrw1\brdrcf0\brsp55\clbrdrl\brdrs\brdrw1\brdrcf0
\brsp55\cellx3324\clbrdrb\brdrs\brdrw1\brdrcf0\brsp55\clbrdrl\brdrs\brdrw1
\brdrcf0\brsp55\cellx6648\clbrdrb\brdrs\brdrw1\brdrcf0\brsp55\clbrdrl\brdrs
\brdrw1\brdrcf0\brsp55\clbrdrr\brdrs\brdrw1\brdrcf0\brsp55\cellx9972
\pard\intbl\pard\plain \intbl\s3\sa120 2.1
\cell\pard\plain \intbl\s3\sa120 2.2
\cell\pard\plain \intbl\s3\sa120 2.3
\cell\row
```

En HTML, las tablas tienen una sintaxis muy estricta. Las tablas son colecciones de filas, rodeadas por la etiqueta `<table>`

; cada fila es una colección de celdas, y está rodeada por la etiqueta `<tr>`

; cada celda es contenido HTML arbitrario (incluso otras tablas, lo cual no es posible en RTF) rodeado por la etiqueta `<td>`

. Así, una tabla básica en HTML, de dos filas por tres columnas, sería:

```
<table>
  <tr>
    <td>1.1</td>
    <td>1.2</td>
    <td>1.3</td>
```

```
  </tr>
  <tr>
    <td>2.1</td>
    <td>2.2</td>
    <td>2.3</td>
  </tr>
</table>
```

El procedimiento es algo como:

1. Al encontrar una fila y no estar en modo tabla, abrimos una tabla, una fila y una celda en HTML, y cambiamos a modo tabla.

2. Guardamos todas las definiciones de celda encontradas en alguna estructura dispuesta a tal efecto.

3. Al encontrar la palabra `\pard`, empezamos a sacar por pantalla normalmente lo que encontremos.

4. A cada `\cell`, cerramos los atributos abiertos, cerramos la celda, y, si todavía quedan celdas por procesar, abrimos otra.

5. Cuando encontremos el `\row`, cerramos la fila.

6. Ahora, buscamos lo siguiente en el documento RTF: si es otra fila, abrimos otra fila en HTML y seguimos en modo tabla; si no, quitamos el modo tabla, y cerramos la tabla que teníamos abierta.

Como se puede ver, hay que hacer muchas comprobaciones y tiene cierto componente heurístico. Como la traducción no es directa, hay que mantener estructuras de datos especiales y variables de estado para poder analizar correctamente una tabla.

4.5.3. Las listas

Con las listas pasa algo parecido al caso de las tablas: HTML las describe perfectamente, mientras que RTF apenas lo hace, pero sí especifica su formato físico.

En este caso, RTF las trata como «párrafos numerados». Al principio de cada uno de estos, pone una marca especial de elemento de una lista, junto con el símbolo o texto a usar al principio de éste (un bolo, un número incrementado automáticamente, etc.). En realidad, el problema es parecido al de las tablas. La diferencia más importante es que *no* hay marca de final de lista, por lo que nunca se sabe cómo gestionar correctamente el estado *dentro-de-lista*. Entre versiones de Word, para empeorar la situación, la forma de escribirlas varía bastante. Una lista no numerada en RTF podría ser algo como:

```
{\pntext\pard\plain\f3\fs20\lang1034\cgrid \loch\af3\dbch\af0\hich\f3 \'b7\tab}
\pard \fi-360\li360\nowidctlpar\widctlpar\jclisttab\tx360{\*\pn \pnlvlblt
\ilvl0\ls1\pnrnot0\pnf3\pnindent360\pnhang{\pntxtb \'b7}}\ls1\adjustright
{\lang1034
Primer elemento de no numerada
\par {\pntext\pard\plain\f3\fs20\lang1034\cgrid \loch\af3\dbch\af0\hich\f3
\'b7\tab}}\pard \fi-360\li360\nowidctlpar\widctlpar\jclisttab\tx360{\*\pn
\pnlvlblt\ilvl0\ls1\pnrnot0\pnf3\pnindent360\pnhang{\pntxtb \'b7}}\ls1
\adjustright {\lang1034 Segundo }{
\lang1034 elemento de no numerada}{\lang1034
\par {\pntext\pard\plain\f3\fs20\lang1034\cgrid \loch\af3\dbch\af0\hich\f3
\'b7\tab}}\pard \fi-360\li360\nowidctlpar\widctlpar\jclisttab\tx360{\*\pn
\pnlvlblt\ilvl0\ls1\pnrnot0\pnf3\pnindent360\pnhang{\pntxtb \'b7}}\ls1
\adjustright {\lang1034 Tercer }{
\lang1034 elemento de no numerada}{\lang1034
\par {\pntext\pard\plain\f3\fs20\lang1034\cgrid \loch\af3\dbch\af0\hich\f3
\'b7\tab}}\pard \fi-360\li360\nowidctlpar\widctlpar\jclisttab\tx360{\*\pn
\pnlvlblt\ilvl0\ls1\pnrnot0\pnf3\pnindent360\pnhang{\pntxtb \'b7}}\ls1
\adjustright {\lang1034 Y as\'ed
 sucesivamente
\par }
```

En HTML, en cambio, las listas tienen marcas explícitas de principio y fin, y una marca por cada elemento. Dependiendo de si la lista es numerada o no (o bien, *ordenada* o no, utilizando vocabulario de HTML), la marca de la lista será `<ol>`
o `<ul>`
respectivamente. Así, un ejemplo de lista en HTML sería:

```
<ul>
  <li>Primer elemento de no numerada
  <li>Segundo elemento de no numerada
  <li>Tercer elemento de no numerada
  <li>Y as&iacute; sucesivamente
</ul>
```

Como no se consiguió resolver el problema de las listas, se ha dejado la traducción casi literal. Aunque no se conserva la estructura de la lista, sí se producen unos resultados muy parecidos al original.

4.5.4. Las referencias cruzadas

En RTF existen las referencias cruzadas como caso particular de los campos. En HTML también existen las referencias cruzadas, aunque de otra manera. Hay que tener en cuenta que RTF se diseñó para documentos con páginas, mientras que HTML no. Por otro lado, Microsoft Word no tiene primitivas para «saltar» de una parte del documento a otra, así que en RTF no tenía sentido añadir unas estructuras a tal efecto. Debido a estas diferencias fundamentales de naturaleza, algunas conversiones de referencias cruzadas no tienen mucho sentido. Por ejemplo, se podría traducir perfectamente una frase como:

```
... En la página 24 se puede encontrar la tabla que explica ...
```

Pero en HTML no tendría sentido, porque no existe el concepto de página. Afortunadamente, el RTF siempre guarda el último resultado calculado, que es el que utiliza para representar en pantalla el documento. Por tanto, el número 24, aunque carece de sentido en la traducción, se convierte a HTML, y no queda la referencia vacía. Este comportamiento facilita mucho la traducción, ya que no hay que entender e implementar necesariamente las instrucciones de creación y actualización de campos[2], sino que uno puede saltarse las instrucciones y traducir o interpretar los últimos resultados calculados.

En la traducción final se ha optado por sustituir las referencias cruzadas por hiperenlaces, para por lo menos dar la oportunidad al lector de la página traducida de saber de dónde salió el texto que está leyendo.

4.6. Visores de HTML disponibles

Tal y como se dijo en este mismo capítulo, el lenguaje HTML es universal, en el sentido de que virtualmente cualquier sistema operativo que corra bajo cualquier máquina tiene disponible un visor de este tipo de documentos. Estos navegadores datan de fechas muy dispares, y tienen un conocimiento muy distinto de las normas HTML.

Los navegadores más famosos son:

Netscape Navigator Hay versiones para Microsoft Windows, para Mac y para diferentes versiones de UNIX, incluida Linux.

Internet Explorer Probablemente el navegador más moderno, y sin duda el más usado actualmente. Pertenece a la compañía Microsoft, y actualmente sólo hay versiones para Microsoft Windows y para Mac.

Opera Un pequeño programa, que lucha por hacerse un hueco entre los que tradicionalmente han sido los grandes (los dos anteriores). Tiene fama de pequeño, rápido, y de ser el que mejor cumple todas las normas. No utiliza extensiones inventadas, y cumple las recomendaciones de la *World Wide Web Consortium*.

Mozilla La versión libre de Netscape. Está basado en el componente *Gecko*, y parece que, en el futuro, será uno de los grandes. Teóricamente, reconoce las últimas adiciones a la norma HTML.

[2] Que, por cierto, *no* están especificadas en el documento oficial de Microsoft

Mosaic Un navegador gratuito, bastante antiguo y obsoleto.

Lynx Un pequeño navegador, en modo texto, que ignora la mayoría de las etiquetas.

Links Un programa, que, aunque funciona en modo texto y por tanto no reconoce las imágenes, sí que reconoce un subconjunto bastante grande del HTML actual.

Mnemonic Un programa, escrito en C++, y con un diseño bastante logrado, que pretende a largo plazo convertirse en una de las alternativas más importantes en el mercado de los navegadores. Su estado actual es bastante primitivo, aunque se pretende que siga todas las normas HTML del momento.

5 El programa RTHC

En este capítulo es explican todos los aspectos técnicos del programa, tanto relacionados con su elaboración, como relacionados con la funcionalidad ofrecida a otros programadores.

5.1. Especificaciones y objetivos

Antes de empezar, es importante destacar que el proyecto ha visto notablemente incrementada su envergadura. Al principio, lo que se pretendía construir no era más que un conversor entre dos formatos; pero durante el trabajo de diseño la idea fue evolucionando, y se ha terminado escribiendo un componente genérico que facilite la construcción de programas que manejen RTF. Así, el objetivo original ha quedado como un mero caso particular, o casi como una simple demostración de que el componente mencionado funciona y es viable su utilización.

5.1.1. Especificaciones iniciales

Las especificaciones del programa eran las siguientes: había que construir un conversor de documentos escritos en formato RTF a documentos en el lenguaje HTML, que cumpliera lo siguiente:

1. Se conserva el texto y el formato básico del original.
2. El programa es fácil de comprender, porque debe servir como modelo de implementación de un analizador de documentos RTF.
3. El programa es fácilmente extensible y adaptable a nuevas circunstancias.

Además de sus características «intrínsecas», debía permitir las siguientes posibilidades como mínimo:

1. Poder trocear un documento grande en varios ficheros, para evitar las salidas en HTML poco manejables.
2. Se deben conservar la estructura lógica del texto original.
3. Se debe poder personalizar de alguna manera el HTML producido.

La última opción podía enfocarse de varias maneras: permitiendo mediante opciones cambiar la forma en que el HTML final quedaría, delegando en un fichero de configuración la traducción de ciertos caracteres especiales, o utilizando documentos-plantilla, que definieran el «envoltorio» para el texto convertido, es decir, el aspecto final que tendrían las páginas HTML creadas. La primera posibilidad se descartó, por ser engorrosa y no tan flexible como la tercera. Las otras dos se implementaron en el programa final.

Por otro lado, el objetivo era permitir que los documentos RTF pudieran leerse desde cualquier sistema operativo y en cualquier máquina[1]. Como el formato de salida es HTML, el objetivo se consigue plenamente. Por tanto, a pesar de que se sabía que en la conversión se iba a perder información de formato físico, este hecho no era preocupante. La razón para construir el conversor y documentar el formato RTF era aportar a la comunidad del *software* libre una vía para entender documentos escritos en un formato cerrado.

[1] Virtualmente no hay sistemas operativos para los que no se hayan escrito navegadores de Internet, o al menos simples analizadores de HTML

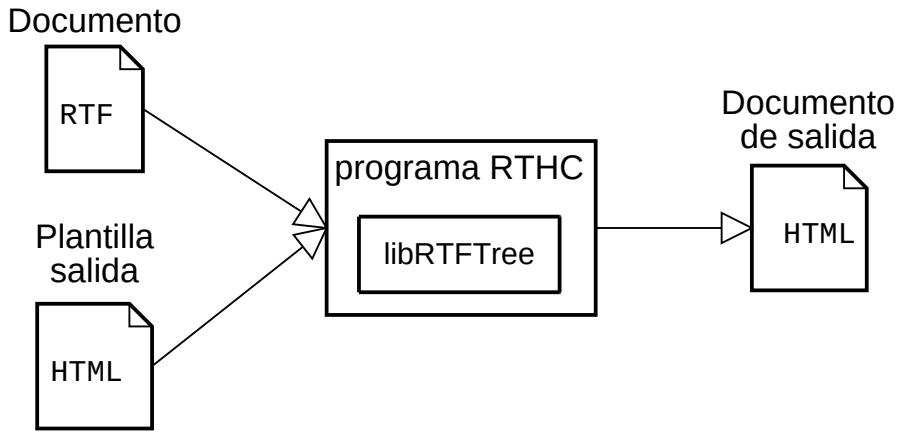


Figura 5.1: Diagrama de funcionamiento general de RTHC

5.1.2. Extensibilidad

El programa, tal y como está escrito, permite muchos cambios y adaptaciones. De hecho, está diseñado con ese mismo propósito en mente. Todas estas modificaciones pueden dividirse en tres grandes grupos:

1. Mejoras al programa conversor RTHC. Esto es posible gracias al modular diseño de la aplicación, que trata a cada entidad RTF con una función diferente, por el patrón de diseño «visitante».
2. Adaptaciones y mejoras a las clases de *libFreeRTF*. Es sencillo debido a que la funcionalidad se ha intentado separar lo máximo posible. La clase que construye los árboles y la que los almacena, por ejemplo, son diferentes, lo que garantiza que los cambios de una no afecten a la otra.
3. Nuevos programas basados en *libFreeRTF*. El mismo patrón de diseño del visitante permite fácilmente crear programas que recorran un árbol RTF con cualquier motivo. Además, si la funcionalidad es parecida a la de un visitante ya disponible, se puede hacer heredar a la nueva clase y sobrecargar los métodos necesarios, en vez de escribir la clase desde cero.

En el capítulo 7 se dan algunas ideas de cada uno de estos grupos.

5.2. Arquitectura general

A la hora de diseñar el programa, y después de considerar diversas alternativas, se concluyó que lo mejor era dividirlo en dos partes:

1. Un conjunto de clases (llamado *libFreeRTF*) que analizaran un documento RTF, construyeran una representación en memoria, y facilitaran de alguna manera el recorrido de esta estructura, y
2. Un caso particular de los infinitos usos que se le pueden dar al conjunto de clases, consistente en el objetivo original del proyecto, un conversor de RTF a HTML.

Se tratan separadamente las dos partes en los siguientes apartados.

5.3. *libFreeRTF*

Como componente de manejo de RTF, la colección de clases *libFreeRTF* tenía que ser:

- Simple, para que el esfuerzo total de comprender su forma de uso y es el de construir el programa con *libFreeRTF* no fuera mayor que escribir el programa sin la ayuda de ésta.
- Eficiente, para no entorpecer el uso final de la representación intermedia.
- Extensible, para permitir adaptarla a nuevas necesidades, no contempladas en el diseño original, y para dar la posibilidad de modificarla fácilmente para manejar nuevas versiones de la norma RTF.

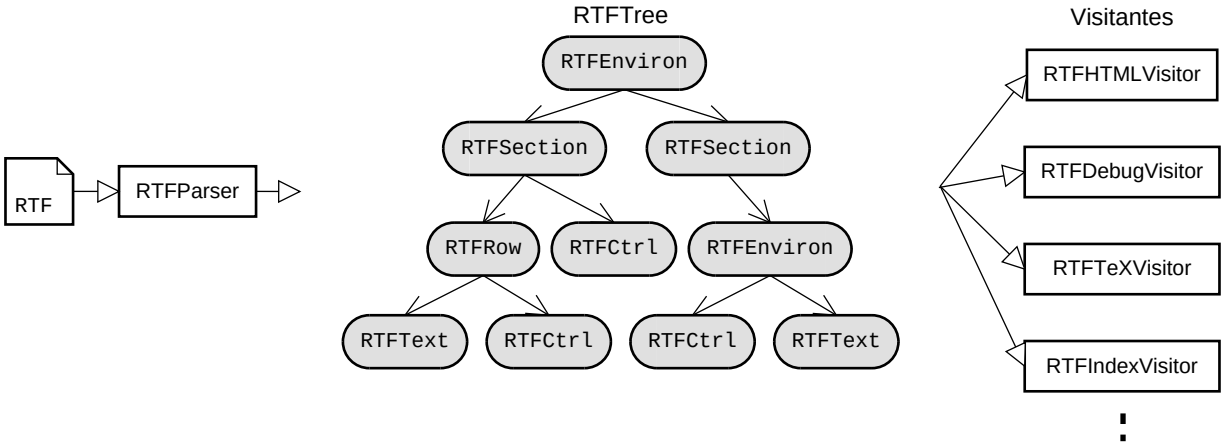


Figura 5.2: Arquitectura general de *libFreeRTF*

- Completa, para asegurarnos de que ninguna aplicación carecía de información del documento original.

Por ello, y tal y como se comentó en el apartado 1.5.2, se acudió al libro «Design Patterns», en busca de ideas de diseño e inspiración. En este libro se encontró el patrón del visitante, que es el núcleo de las clases de *libFreeRTF*. En principio, para facilitar el aprendizaje y uso de estas clases, se pensó en crear un iterador al estilo STL, que recorriera el árbol nodo a nodo. Los visitantes, en ese caso, recibirían un nodo, cuyo contenido tendrían que procesar. A la larga, se vio que no era posible recorrer y procesar un árbol RTF de esta manera, dado el desorden y el anárquico diseño del formato.

Darse cuenta de este fallo llevó bastante tiempo, y provocó que tuviera que descartarse una buena parte del programa que se había hecho hasta el momento. Este es el ejemplo más patente de error de análisis debido a documentación incompleta y, sobre todo, lenguaje confuso, que hizo que se tuviera que corregir el primer diseño.

Por todo ello, se tuvo que descartar la idea inicial, para implementar un modelo que ha demostrado ser mucho más versátil, que es el del recorrido a cargo de los visitantes. Por ese modelo, los visitantes son los encargados de recorrer el árbol, es decir, conceptualmente los visitantes ya no reciben un nodo, sino la raíz de un subárbol a recorrer. De esta forma, no sólo tienen que procesar la información del nodo, sino recorrer, en caso necesario, sus «hijos». Esto implicaba una ventaja adicional, consistente en poder ignorar subárboles enteros simplemente dejando sin visitar los hijos de un nodo dado.

5.3.1. Visión general de las clases

Las clases que componen *libFreeRTF* son:

`RTFTree` La representación en forma de árbol del documento RTF. Es la clase principal.

`RTFVisitor` Un «visitante» genérico de árboles RTF. Todos los visitantes se heredan de esta clase.

`RTFParser` Es el «constructor» de árboles RTF. Al principio se pensó en incluir la creación de estos árboles en el constructor de los mismos, pero por dar mayor flexibilidad, se decidió posteriormente en separar la forma de construir el árbol del objeto en sí.

`RTFLexer` Es el analizador léxico que utiliza RTFParser para leer el documento original.

`RTFWordsLexer` Analizador léxico para el fichero `rtf.words`, que guarda la configuración de la clase `RTFParser`.

`RTFContents` Es la clase base de una jerarquía que incluye todos los tipos de nodos que puede haber en un árbol RTF.

`SectionNumber` Guarda un número de apartado. Esta clase es independiente del formato RTF, de ahí que no tenga `RTF` como primeros caracteres de su nombre.

`RTFRefTable` Es un índice de referencias cruzadas.

`RTFData` Es la clase que guarda lo que define una palabra o símbolo de control RTF.

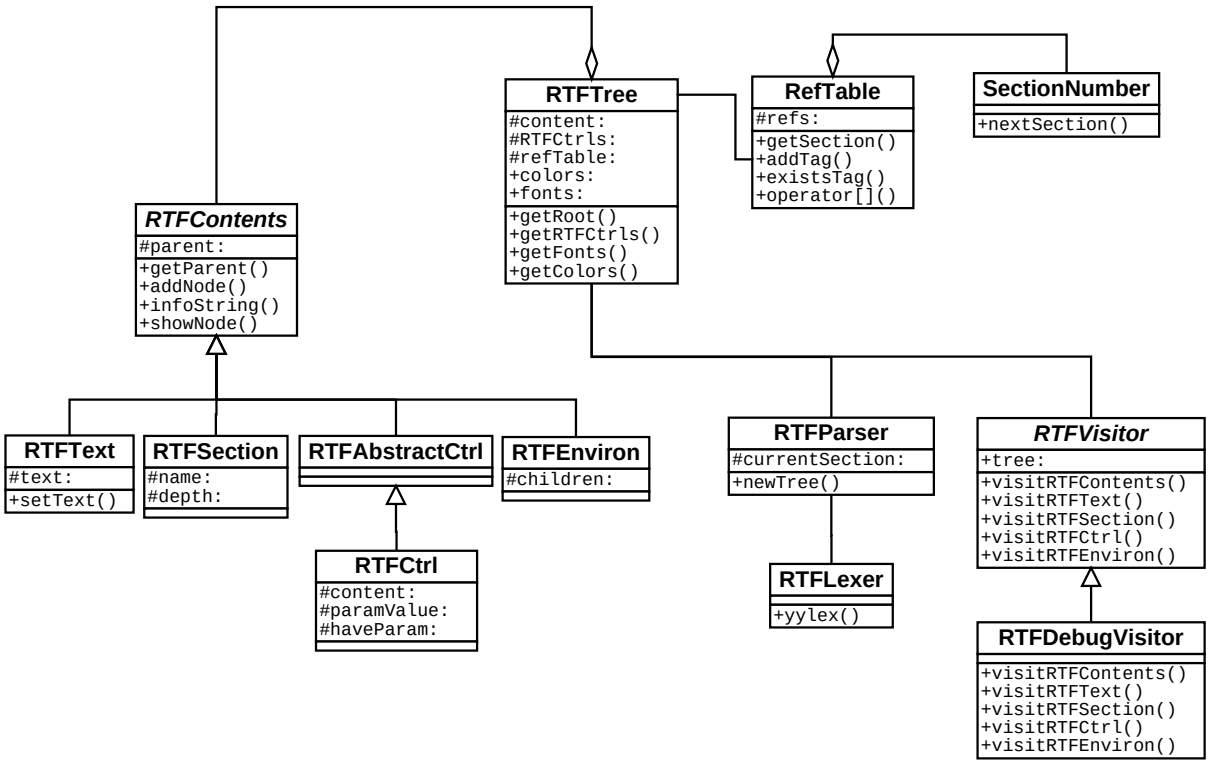


Figura 5.3: Diagrama UML simplificado de *libFreeRTF*

Para el programador que quiera utilizar la *libFreeRTF*, las únicas que importan son la clase `RTFTree`, la `RTFParser` y las jerarquías `RTFVisitor` y `RTFContents`. Las dos primeras servirán para representar y construir el árbol, y las dos últimas para recorrerlo y representar su contenido. Estas cuatro se describen más detenidamente en los siguientes apartados.

5.3.2. `RTFContents` y sus herederas

La clase `RTFContents` representa los tipos de datos que pueden coexistir en un árbol RTF. Tiene algunos métodos comunes, como `setParent`, `getParent` o `infoString`.

Cada una de sus herederas es un concepto de RTF. Muchos de estos conceptos se corresponden directamente con un símbolo léxico, aunque no es necesario. Algunos de los ejemplos más claros de conceptos RTF son un trozo de texto (clase `RTFText`), una fila de una tabla (clase `RTFRow`), un cambio de destino (`RTFDestChange`) o una marca referenciable (`RTFTag`).

5.3.3. `RTFTree`

Esta clase es el centro de todas. Representa, como se ha dicho, el documento RTF en forma de árbol, en memoria. Sus métodos permiten acceder a su nodo raíz y a sus propiedades, tanto para consultarlas como para establecerlas. Lo que aquí se llama propiedades son datos globales del documento, como el autor, los tipos de letra usados, la fecha de última modificación, los estilos definidos, el número de caracteres del documento original, los colores, etc.

5.3.4. `RTFVisitor` y sus herederas

`RTFVisitor` y sus clases herederas son las encargadas de recorrer los árboles (o, lo que es lo mismo, documentos) RTF para aplicarles cierta operación. Dado que en principio los *visitantes* están pensados para recorrer «estáticamente» los documentos, la operación más común es la de conversión a otros formatos. En realidad, cualquier operación que sea de filtrado, como contar cierto número de elementos, «limpiar» el documento original de partes que no nos interesan, etc. es susceptible de ser resuelta fácilmente con *libFreeRTF*.

Con *libFreeRTF* vienen dos ejemplos útiles de visitantes: la clase `RTFPlainTextVisitor` y `RTFDebugVisitor`. El primero es un conversor de RTF a texto ISO-8859-1 sin formato, y el segundo es un depurador de

árboles RTF, muy útil cuando se hacen modificaciones de implementación a la clase `RTFParser`, que los construye, o cuando algo va mal.

`RTFPlainTextVisitor`

El objeto de esta clase es convertir los documentos RTF en simples textos sin formato, que cualquiera puede entender. Se le podrían hacer muchas mejoras, pero, teniendo en cuenta que es un ejemplo, no valía la pena complicarlo, porque (1) haría más difícil entender el ejemplo, y éste tiene que ser legible y simple, y (2) no es el objetivo del proyecto, y las mejoras requieren algo de tiempo.

Básicamente, el comportamiento de los visitantes `RTFPlainTextVisitor` es sacar por pantalla el texto del árbol RTF que recorren, e ignorar prácticamente todo lo demás. Una de las pocas cosas que se consideraron en la creación de la clase fue ignorar el texto que había que ignorar por tener que interpretarse de otra manera. Por lo demás, el `RTFPlainTextVisitor` se dedica a «tirar a la basura» casi todo lo que encuentra a su paso.

`RTFDebugVisitor`

La otra clase incluida en *libFreeRTF* es `RTFDebugVisitor`, un visitante que se dedica a mostrar la estructura del árbol que va recorriendo. Los entornos anidados los muestra con llaves, como en C, y con un ligero aumento de sangrado, para diferenciarlo fácilmente. Hay que tener en cuenta que el afán de este visitante es servir a personas, a programadores, y no a máquinas, aunque no sería descabellado procesar automáticamente el resultado de un objeto `RTFDebugVisitor`, ya que la salida es bastante homogénea.

El comportamiento del visitante se puede resumir en:

1. Imprimir toda la información relativa a la cabecera
2. Por cada nodo encontrado en el árbol, hacer:
   a) Se imprime su tipo
   b) Si tiene hijos, hacer:
      1) Imprimir una llave abierta
      2) Incrementar el sangrado en dos espacios
      3) Por cada uno de los hijos, recorrerlo recursivamente
      4) Imprimir una llave cerrada

Un ejemplo de la salida producida por un visitante `RTFDebugVisitor` es:

```
Hijos de RTFEnviron {
 Paso por un RTFCtrl (\paperw11906)
 Paso por un RTFCtrl (\paperh16838)
 ...
 Paso por un RTFCtrl (\endnhere)
 Paso por un RTFCtrl (\sectdefaultcl)
 Paso por un RTFEnviron
 Hijos de RTFEnviron {
  Paso por un RTFStar
  Paso por un RTFCtrl (\pnseclvl1)
  Paso por un RTFCtrl (\pnucrm)
  Paso por un RTFCtrl (\pnstart1)
  Paso por un RTFCtrl (\pnindent720)
  Paso por un RTFCtrl (\pnhang)
  Paso por un RTFEnviron
  Hijos de RTFEnviron {
   Paso por un RTFCtrl (\pntxta)
   Paso por un RTFText (.)
  }
 }
 Paso por un RTFEnviron
 Hijos de RTFEnviron {
  Paso por un RTFStar
  Paso por un RTFCtrl (\pnseclvl2)
  Paso por un RTFCtrl (\pnucltr)
```

```
  Paso por un RTFCtrl (\pnstart1)
  Paso por un RTFCtrl (\pnindent720)
  Paso por un RTFCtrl (\pnhang)
  Paso por un RTFEnviron
  Hijos de RTFEnviron {
   Paso por un RTFCtrl (\pntxta)
   Paso por un RTFText (.)
  }
 }
...
}
```

5.3.5. El fichero `rtf.words`

El fichero de configuración `rtf.words` indica qué palabras reservadas deben entenderse forzosamente, junto a su tipo (independientes, de atributo de carácter, de atributo de párrafo, o de atributo de apartado). Es necesario que *libFreeRTF* sepa a qué tipo pertenecen las palabras de control, ya que no hay ninguna forma de saberlo con tan solo mirar la palabra en sí, y no tiene sentido que cada aplicación por separado tenga que estar preocupándose de eso. Por ello, el propio árbol guarda las palabras que son independientes, y las de atributo de diferentes tipos.

Un ejemplo del citado fichero podría ser:

```
# Palabras de control independientes
RTFCtrl
\widctlpar
\lang
\cgrid
\li

# Palabras de control de entorno
RTFCharAttribute
\b
\i
\f
\fs
RTFParAttribute
\qc
\qj
# No definimos atributos de apartado
```

libFreeRTF tiene unos valores por defecto de cada uno de estos conjuntos de palabras, para obviar la necesidad de que haya un buen fichero de configuración y se encuentre. Los valores precompilados en el programa se pueden cambiar fácilmente con la utilidad `createDefaults`, que está en el directorio `RTFTree/util`. Al igual que `createDefaultMaps` (véase apartado 5.4.3), sólo necesita un fichero de configuración válido para generar un fichero cabecera, que será incluido por la aplicación. Al recompilar, los valores por defecto se actualizarán.

5.4. RTHC

El conversor en sí, cuyas siglas se corresponden con las palabras «RTF To HTML Converter», es un pequeño programa que lee y procesa todas las opciones que se le pasan, y hace algunas comprobaciones. El trabajo en sí lo hace un nuevo visitante creado solamente con este propósito, la clase `RTFHTMLVisitor`.

5.4.1. El visitante `RTFHTMLVisitor`

Este componente es el más importante de la aplicación. Es una clase derivada de `RTFVisitor`, que se utiliza para recorrer un árbol, creado por el módulo principal, y traducirlo a HTML. Las capacidades principales del conversor son poder partir en varios ficheros la salida, y permitir la modificación o adición de traducciones de símbolos RTF a HTML mediante un fichero de configuración.

Internamente, la clase `RTFHTMLVisitor` se compone de:

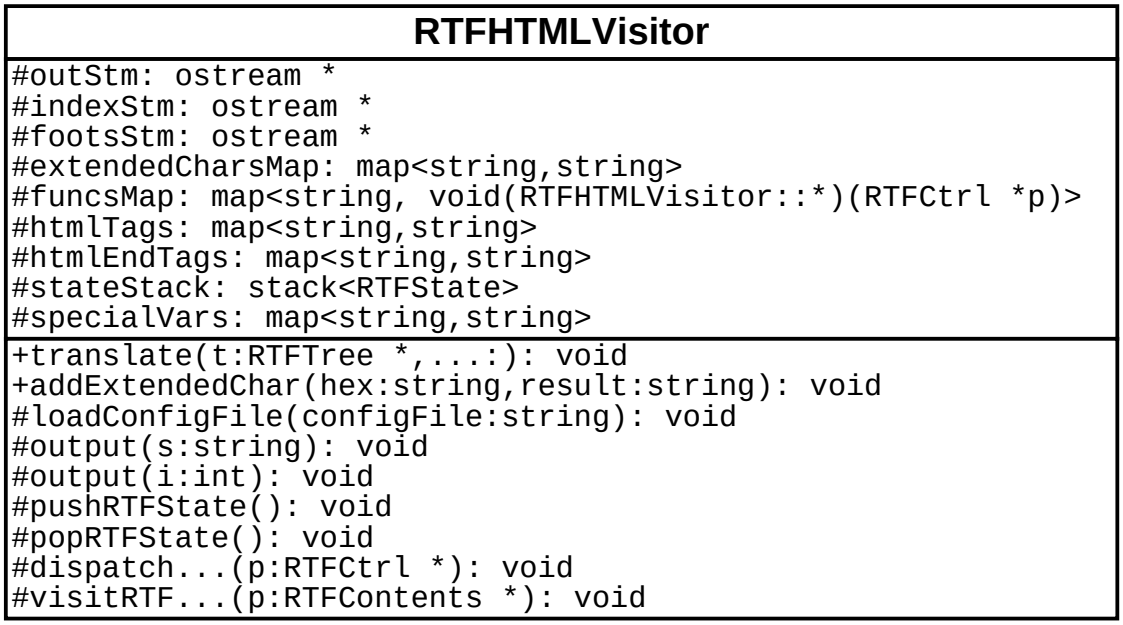


Figura 5.4: Diagrama UML de la clase `RTFHTMLVisitor`

- Métodos públicos, para utilizar sus servicios o actualizar el estado interno, que condiciona la salida producida.
- Atributos protegidos, que guardan toda la información interna del conversor, como su estado temporal o sus características.
- Funciones protegidas de visita, heredadas todas de `RTFVisitor`.
- Funciones protegidas de proceso de palabras de control, creadas para facilitar la modificación de las acciones llevadas a cabo ante cada palabra de control.
- Funciones protegidas de apoyo, creadas para modularizar el contenido de la clase.

En los siguientes apartados se describen con detenimiento todos estos puntos.

Métodos públicos

Aparte del constructor por defecto y el destructor, hay dos métodos públicos: el método `translate`, para invocar los servicios de la clase (traducir a HTML el árbol dado), y el método `addSpecialChar`, que añade una traducción más de caracteres especiales RTF a HTML. Esto permite modificar cómo se produce la salida del conversor.

El método `translate` se declara de la siguiente manera:

```
void translate (RTFTree *t, string configFile,
        map<string,string> options, string outputFile = "");
```

El significado de los parámetros es:

`RTFTree *t` Un puntero al árbol RTF a traducir a HTML.

`string configFile` El nombre del fichero de configuración, por defecto llamado `rthcrc`. En éste se pueden especificar las traducciones a HTML de los caracteres «especiales» RTF, y las traducciones de ciertas palabras de control RTF.

`map<string,string>options` Las opciones pasadas al conversor. Se trata de pares de objetos `string` (nombre de la opción y valor). En estas opciones se pueden especificar desde las plantillas de salida hasta la profundidad del «corte» en el documento original RTF, para crear más de un fichero HTML de salida.

`string outputFile` La raíz que se va a utilizar para construir los nombres de los diferentes ficheros de salida. Por defecto vale un objeto `string` vacío, lo que indica que la salida tiene que sacarse por pantalla. Naturalmente, este caso es incompatible con la opción de profundidad de corte del documento.

El método `addSpecialChar` se declara así:

```
void addSpecialChar (string hex, string result);
```

Los parámetros significan:

`string hex` Es el caracter, en hexadecimal, cuya traducción quiere añadirse a la lista interna de traducciones del visitante.

`string result` Es la traducción, que es HTML arbitrario.

Su cometido no es más que introducir el par de caracter y traducción en una lista interna llamada `specialCharsMap`.

Opciones La correspondencia entre objetos `string options` almacena las opciones que se le pasan al visitante. Teóricamente, los objetos que indizan la correspondencia son los nombres de las diferentes opciones, y los objetos indizados son los valores de las susodichas.

Las opciones posibles que se pueden pasar al conversor son:

`use-css` Especifica la lista de hojas de estilo a usar, separadas por comas.

`cut-depth` Profundidad de corte del documento.

`template` El nombre de la plantilla a utilizar.

`with-index` Si se desea producir un índice, además de las páginas con la conversión del documento.

`with-jsindex` Si se desea producir un índice flotante en JavaScript, además de las páginas con la conversión del documento.

Atributos

Todos los atributos de la clase `RTFHTMLVisitor` son protegidos. Sólo uno de ellos, `specialCharsMap`, es «directamente» accesible desde fuera, mediante el método `addSpecialChar`. La lista de todos los atributos de la clase son los siguientes:

`ostream *outstm` Es el flujo de datos usado para producir toda la salida, ya sea a fichero(s) o a la pantalla.

`map<string,string>specialCharsMap` La lista de traducciones de caracteres especiales RTF a HTML.

`map<string, void (RTFHTMLVisitor::*)(RTFCtrl *p)>funcsMap` Una correspondencia entre palabras de control y métodos de la clase que se encargan de procesar esas palabras.

`map<string,string>htmlTags` Correspondencia entre palabras de control RTF y sus traducciones directas a HTML, si las tienen.

`map<string,string>htmlEndTags` Correspondencia entre palabras de control RTF (que se aplican en un entorno) y sus cierres en HTML, si son especiales.

`stack<RTFState>stateStack` Pila de estado necesaria para poder recorrer un documento RTF, sea cual sea su representación.

`nCells` Número de celdas que quedan por visitar en la fila actual, cuando se recorren tablas.

`inParagraph` Variable de estado que indica si se está «dentro» de un párrafo.

`inSectionName` Variable de estado que indica si se está recorriendo el nombre de un apartado del documento original.

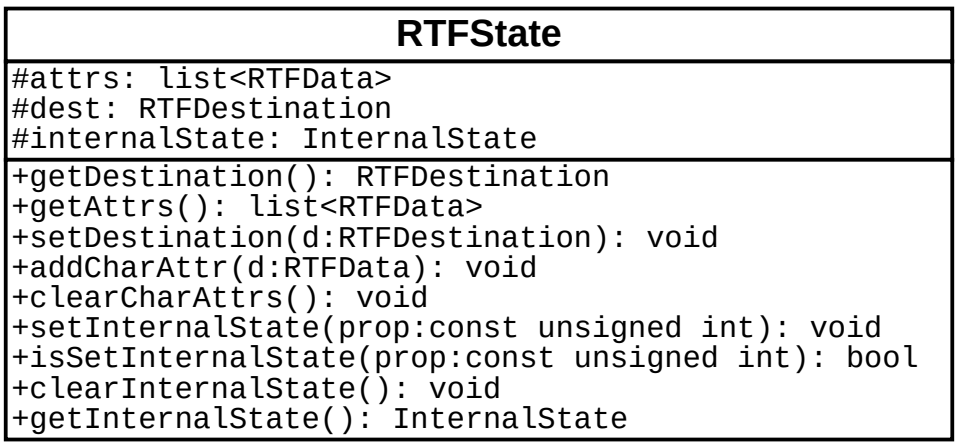


Figura 5.5: Diagrama UML de la clase `RTFState`

La clase `RTFState` La clase `RTFState` representa un *estado* RTF. La propia especificación recomienda que se utilicen estos estados para analizar documentos RTF (véase [MIC97] y apartado 3.7). Los atributos de esta clase son:

`attrs` Una «pila» de atributos abiertos en ese entorno. Los atributos son de tipo `pair<RTFData, AttrType>`, de tal forma que siempre se guarda el tipo de cada atributo de la lista.

`dest` Un objeto de tipo `RTFDestination`, que no es más que un sinónimo de la clase `string`, que indica a qué destino ha de mandarse el texto encontrado dentro del grupo.

`internalState` Un entero que guarda un conjunto de propiedades. Actualmente sólo guarda si hay que ignorar el texto del grupo, y si se está dentro de un grupo de la palabra `listtext`.

Sus métodos sirven para acceder a sus atributos internos de diferentes maneras:

`getDestination` Devuelve un objeto `RTFDestination` con el destino del estado.

`getCharAttrs` Devuelve una lista con los atributos de carácter abiertos en el grupo. Hay otros dos métodos análogos para los atributos de párrafo y apartado, llamados `getParAttrs` y `getSectAttrs`..

`setDestination` Establece el destino del estado.

`addCharAttr` Añade a la lista de atributos abiertos de carácter uno más. Hay otros dos métodos más, análogos, llamados `addParAttr` y `addSectAttr`.

`popCharAttr` Quita el último atributo de carácter de la lista. Como siempre, hay dos métodos `popParAttr` y `popSectAttr`.

`clearCharAttrs` Borra completamente la lista de atributos abiertos de carácter. `clearParAttrs` y `clearSectAttrs` hacen lo propio con los atributos de párrafo y de apartado.

`setInternalState` Añade una propiedad determinada al estado interno actual del estado RTF.

`isSetInternalState` Devuelve verdadero si el estado interno especificado está activo actualmente en el estado RTF.

`clearInternalState` «Limpia» el estado interno del estado RTF.

`getInternalState` Devuelve el estado interno entero.

Funciones de visita

En la clase `RTFHTMLVisitor` todas las funciones de visita están sobrecargadas. Los métodos más complicados e importantes son `visitRTFCtrl` y `visitRTFSymbol`, que se encargan de recibir las palabras y símbolos de control y actuar en consecuencia.

Para hacer su trabajo, `visitRTFCtrl` examina el atributo `funcsMap`, que contiene la correspondencia entre las palabras de control y los métodos a ejecutar cuando se encuentran. Si encuentra la palabra actual en `funcsMap`, sólo hay que ejecutar el método apropiado. Si no la encuentra, entonces aplica el método general, que es:

1. Buscar una posible traducción a HTML en `htmlTags`. Si no se encuentra, entonces la palabra de control actual se ignora completamente, y se sigue analizando el resto documento.
2. Si se encuentra, se comprueba si la palabra está en alguna de las listas de atributos, ya sea de carácter, párrafo y apartado. Si no se encuentra en ninguna de estas, se da por sentado que es una palabra de control independiente.
3. Según el tipo de palabra encontrada, si es un atributo, se mete en la lista correspondiente, para poder cerrarlo si se cierra implícitamente.

Funciones de proceso de palabras de control

Para facilitar el manejo de las palabras de control, que intrínsecamente son casos particulares, se ideó una forma de asociar a estas palabras un método a ejecutar. Esta forma fue mantener dentro de cada visitante `RTFHTMLVisitor` un atributo *funcsMap*, ya descrito en el apartado 5.4.1. Éste hace corresponder métodos de la clase `RTFHTMLVisitor` a objetos `string`. Así, cuando encontremos una palabra de control, lo primero será comprobar si a ésa le corresponde una acción concreta. Si es así, simplemente ejecutaremos el método del objeto actual, y terminaremos el trabajo.

Todas estas funciones de proceso de palabras de control tienen que ser, forzosamente, métodos definidos en la clase. Recibirán un único parámetro, de tipo `RTFCtrl *`. Los citados métodos, por legibilidad, tienen nombres que empiezan por `dispatch`, y son:

`dispatchIgnoreText` Se encarga actualizar el estado interno del visitante, para que se empiece a ignorar el texto que se encuentre, hasta que se termine el entorno actual.

`dispatchCell` Se encarga de actualizar el estado del conversor al encontrar un final de celda.

`dispatchFont` Se encarga de producir una etiqueta para cambiar el tipo de letra actual.

`dispatchFontSize` Produce una etiqueta para cambiar el tamaño de letra actual.

`dispatchCrossRef` Se encarga de resolver una referencia cruzada.

Funciones de apoyo

Además de todos los métodos comentados, los visitantes `RTFHTMLVisitor` definen algunos métodos de uso general. La mayoría de éstos se definen por legibilidad o para facilitar el mantenimiento, centralizando en un solo método tareas comunes.

`initFuncsMap` Inicializa el atributo `funcsMap`.

`initDefaultHtmlTags` Inicializa a los valores por defecto las conversiones entre palabras de control RTF y etiquetas HTML.

`loadConfigFile` Carga el fichero de configuración del programa.

`output` Produce salidas por pantalla, a menos que el indicador de ignorar texto esté activado.

`specialChar` Devuelve la conversión de dos dígitos hexadecimales a la traducción final a HTML.

`convertEntities` «Filtra» un texto, recibido como parámetro, para convertir todos los caracteres que son literales en RTF, pero que en HTML tienen una entidad especial para representarlos.

`flushAttrs` Cierra todos los atributos que hayan quedado abiertos en el entorno superficial.

`pushRTFState` Añade un nuevo estado a la pila del analizador.

`popRTFState` Termina el estado superficial de la pila del analizador.

`endHtmlTag` Devuelve el final de una etiqueta *HTML* dada.

`closeAttr` Devuelve el final HTML asociado a una *palabra de control RTF*.

`visitNodeChildren` Visita a todos los hijos del nodo dado.

`visitWithTemplate` Visita un árbol con la ayuda de una plantilla de salida, para generar el HTML.

`getTagRef` Obtiene una referencia completa HTML (con fichero, si hace falta) a partir de una etiqueta RTF.

5.4.2. Opciones del programa

RTHC tiene tres formas de recibir opciones: los parámetros en la llamada en línea de órdenes, las diferentes directivas en los ficheros de configuración y las plantillas de salida, que indican cómo se ha de producir el HTML final (o, a esos efectos, cualquier cosa que se nos ocurra). En los siguientes apartados se discuten los diferentes opciones según su tipo.

Línea de órdenes

Para llamar al programa, hay que especificar el nombre del fichero a traducir, y optativamente la raíz de los nombres de los ficheros de salida. Las opciones que se pueden pasar son:

`--help` Muestra la ayuda en la pantalla.

`--with-index` Hace un índice en un fichero aparte.

`--with-jsindex` Hace un índice flotante en JavaScript en un fichero aparte.

`--use-css=1.css,2.css` Define qué hojas de estilo se utilizarán. Se puede especificar cualquier número de ellas, separadas por comas.

`--cut-depth=d` Especifica a qué nivel se tiene que cortar el documento para generar varios ficheros. Si no se especifica, o el parámetro es 0, no se corta y queda toda la salida en una sola página HTML.

`--with-template=t` Utiliza la plantilla `t` para sustituir y producir las salidas finales.

`--config-file=c` Usa `c` como fichero de configuración.

`--use-stdout` Usa el flujo base de salida para producir los resultados (no los escribe en un fichero).

Estas opciones se leen con la clase `GetOpt` (véase apartado 5.6.

Fichero de configuración

El fichero de configuración de la aplicación se llama por convención `rthcrc`, siguiendo así las costumbres de UNIX en cuanto a nombres de estos ficheros. Éste se busca en varios directorios, incluido el indicado por la variable de entorno `$HOME`. Cuando se busque aquí, se buscará el nombre `.rthcrc`, y no `rthcrc`.

En el fichero de configuración pueden especificarse dos tipos de conversiones:

1. Las conversiones de los caracteres «especiales» RTF, que son los que no se pueden representar con 7 bits. En RTF se representan como \'*hh*, siendo *hh* dos caracteres hexadecimales.
2. Las etiquetas HTML asociadas al principio y al final de ciertas palabras de control RTF. Esto da mucha flexibilidad para personalizar la salida HTML del programa.

Cada conversión ocupa una línea entera, es decir, que el fichero no es de «formato libre». Cada caracter *salto de línea* indica que la indicación de conversión ha terminado.

Para especificar conversiones de caracteres, sólo hay que especificar las dos cifras hexadecimales, sin la barra invertida y el apóstrofo, y la conversión a HTML, que puede ser cualquier texto. Por lo general, la conversión será una entidad HTML (algo como `&aacute;`) o un solo caracter.

En las traducciones de palabras de control, en general, pueden especificarse dos series de caracteres HTML: la de comienzo y la de final. Para que tenga sentido especificar el final, las palabras de control a traducir tienen que haber sido especificadas como de atributos de caracter, párrafo o apartado en el fichero de configuración de *libFreeRTF* al construir el árbol. Si la palabra es de atributo, pero no tiene final, el programa intentará coger el comienzo y cerrarlo.

La descripción léxica del fichero de configuración es:

NEWLINE Salto de línea, para separar las traducciones.

HEXSTRING Dos cifras hexadecimales, en mayúsculas o minúsculas.

RTFCTRL Una palabra de control RTF sin parámetro numérico (ver descripción léxica del formato RTF, en 3.5).

CONTENT «Contenido», que puede ser una serie de caracteres entre comillas dobles o una serie de caracteres sin espacios.

La gramática del fichero de configuración es:

Fichero Una o más líneas.

Línea Una traducción de caracter especial o de palabra de control, y un símbolo NEWLINE.

Traducción de caracter especial Un símbolo HEXSTRING y uno CONTENT.

Traducción de palabra de control Un símbolo RTFCTRL y uno o dos símbolos CONTENT. En caso de que haya un solo símbolo CONTENT, se tomará como traducción de comienzo (si la palabra está definida como de atributo) o como traducción total (en caso contrario). Si hay dos, se tomarán como traducciones de comienzo y de final, en caso de que la palabra esté definida como de atributo; si no lo está, se ignorará la segunda palabra, y la primera se tomará como traducción total.

La acción o interpretación para cualquier otra combinación de símbolos no está definida, aunque la implementación intenta a toda costa subsanar posibles errores humanos, y «ser comprensivo» con el usuario, intentando dar una explicación lógica a lo que encuentra.

Para aclarar toda la discusión anterior, un ejemplo de `rthcrc`:

```
aa      &mientidad;
ef      "Caracter normal"
f3      |       # Sustituimos por una barra vertical
\nowidctlpar    "Paso por un \\nowidctlpar"
\i      <i>     </i>
\b      <b>     # Omitimos, libFreeRTF adivina que el cierre es </b>
\img    "<img src=\"foo.jpeg\">"    # Siempre la misma imagen
```

Plantillas de salida

Las plantillas de salida son la forma que tienen los usuarios de RTHC de especificar cómo quieren que sea el aspecto de las páginas HTML resultantes. Consisten en páginas HTML con construcciones del tipo `$rthc_contents`, que serán sustituidas por diferentes partes del documento original.

Para elegir el formato de las variables especiales de las plantillas de salida de RTHC se eligió una convención ya profundamente arraigada en la comunidad del *software* libre, que es la forma de escribir las variables en Perl y en PHP: el dólar seguido del nombre de ésta. Además, para evitar conflictos de nombres, todas las variables que utiliza RTHC empiezan por los caracteresn `rthc_`, haciendo virtualmente imposible las confusiones.

La lista de variables que pueden referenciarse en una plantilla de salida son:

`$rthc_contents` La variable que guarda el texto del documento. Si el documento original va a partirse en varios ficheros HTML, esta variable representa el trozo de texto que vaya a cada fichero de salida. Es decir, que para todos los ficheros HTML producidos se utiliza la misma plantilla.

`$rthc_css` Guarda la lista de hojas de estilo a usar, especificadas con la opción `--use-css` (ver página 41).

`$rthc_title` La que guarda el título del documento RTF.

`$rthc_author` La que guarda el nombre del autor del texto original.

`$rthc_subject` Almacena el tema del texto.

`$rthc_operator` Guarda el nombre de la persona que escribió el texto.

`$rthc_prev_file` El nombre del anterior fichero.

`$rthc_curr_file` El nombre del fichero actual.

`$rthc_next_file` El nombre del siguiente fichero. Si no hay siguiente, la cadena vacía.

`$rthc_index_file` El nombre del fichero de índice, o el primero, si no hay.

En general, uno puede hacer referencias a cualquier variable: si no está entre las contempladas, simplemente se buscará el campo correspondiente en la hoja de información, y, si existe el campo, se copiará el resultado a la salida.

5.4.3. Valores por defecto

Para evitar la necesidad de ficheros de configuración, se han compilado unos valores por defecto a la aplicación. Naturalmente, estos valores pueden modificarse con los ficheros de configuración, y es muy fácil incluso cambiar qué valores por defecto tendrá el programa.

Los valores por defecto para RTHC están en el fichero cabecera `defaultMaps.h`. Este fichero se genera automáticamente, a partir de un fichero de configuración normal y corriente, y el programa `createDefaultMaps`, sito en el directorio `RTFHTMLVisitor/util`. Para utilizarlo, no hay más que poner en el mismo directorio un fichero de configuración llamado `rthcrc` (se incluye uno ya), y ejecutar el programa sin parámetros. Después de esto, habrá un fichero `defaultMaps.h`, que se puede copiar al directorio anterior para recompilar la aplicación con nuevos valores por defecto.

5.5. La clase `PathOpen`

La clase `PathOpen` es una pequeña clase que busca ficheros en una lista de directorios dada. Esta clase es ideal para buscar ficheros de configuración, ya que generalmente pueden estar en varios directorios diferentes.

Básicamente, la funcionalidad de la clase es añadir directorios a la lista, borrar un directorio dado y buscar un fichero en la lista de directorios. Además de esta funcionalidad básica, se ha dado la posibilidad, habilitada por defecto, de añadir un punto antes del fichero a buscar, cuando se busca en un directorio cuyo nombre coincide con la variable de entorno `$HOME`. Esto es para seguir las costumbres de UNIX, que pone un punto al principio de los nombres de los ficheros de configuración cuando están en la raíz de la cuenta del usuario, para «esconderlos» y que no estorben a la vista.

Los métodos de la clase son:

`void addPath (string )` Añade a la lista de directorios donde buscar el parámetro dado. Si termina en el caracter «/», se quita.

`void removePath (const string & )` Quita un directorio de la lista.

`vector<string>getSearchPaths () const` Devuelve la lista actual de subdirectorios en un vector STL, para examinarlos.

`void setPersonalConfFiles (const bool & )` Establece si se desea o no añadir un punto a los ficheros a buscar cuando se busque en `$HOME`.

`bool getPersonalConfFiles () const` Devuelve la opción actual de ficheros de configuración personales.

`string findFile (const string & )` Busca el fichero en la lista interna. Si lo encuentra, devuelve un objeto `string`, que será la suma del directorio donde lo encontró y el fichero buscado (probablemente, la ruta absoluta del fichero, pero no necesariamente).

Esta clase se escribió expresamente para RTHC, aunque, al ser genérica, no se incluye en *libFreeRTF*, sino que se deja aislada. Se distribuye, como es lógico, por la licencia LGPL.

5.6. La clase `GetOpt`

Aparte de todas las clases de la *libFreeRTF* y de `PathOpen`, hay una clase más de la que merece la pena hablar por varias razones: la clase `GetOpt`. `GetOpt` se encarga de recoger todas las opciones que le lleguen al programa de una forma sencilla. Es decir, es la clase que simula el comportamiento de las funciones `getopt` del lenguaje C. Otra de las ventajas de utilizar un componente y no leerlas a mano es que, al delegar la responsabilidad en una clase, es más fácil añadir opciones, borrarlas, o cambiarlas.

Al empezar el proyecto y empezar a documentarse sobre paquetes con funciones o clases útiles para el trabajo, se buscó una manera de leer opciones de forma fácil, algo así como una clase que se comportara como las funciones `getopt`. Era requisito imprescindible que manejara las opciones largas, estilo GNU[2]. Después de mucho buscar, se encontró en la *libg++* la clase `GetOpt`, que entendía las opciones cortas. Por otro lado, en la *libc* se puede encontrar el programa original de toda la familia de funciones `getopt`.

[2]Opciones con dos guiones en vez de uno, en vez de con un guión y una sola letra, como `--with-frames`

Debido a la libertad que da la licencia LGPL, bajo la que se distribuyen tanto la *libg++* como la *libc*, se pudo fácilmente adaptar la función `getopt_long`, que entendía funciones largas, para convertirla en un método más de la clase `GetOpt`. Así, se consiguió mejorar la antigua clase para añadir más funcionalidad, y casi sin esfuerzo. Esto es *completamente impensable* en un programa comercial, donde no tenemos ni (1) acceso a los fuentes ni mucho menos (2) permiso para modificarlos y distribuirlos modificados, en caso de poder acceder de alguna forma a ellos. Muchas personas sienten que esta falta de libertad es algo a todas luces intolerable y contra lo que hay que luchar. Como mínimo, hay que concederles que en la mayoría de, si no en todas, las situaciones es *muy conveniente* tener el programa original disponible, poder compilarlo con optimizaciones para nuestra máquina, modificarlo, mejorarlo, arreglarlo, adaptarlo y aprender leyéndolo. Y lo mejor de todo es que ahora toda la comunidad puede aprovecharse del trabajo de una persona, `GetOpt` no se tendrá que adaptar una y otra vez: se comparten los progresos, se hace avanzar a la ciencia.

En cuanto a su descripción técnica, la clase `GetOpt` tiene dos métodos públicos:

`operator()` El operator de paréntesis, que recibe un parámetro y devuelve un entero. En el parámetro guarda qué opción ha sido elegida, y devuelve el caracter `'?'` si no se reconoció la opción. Se llama hasta que devuelva la constante `EOF`, que indica el final de las opciones.

`GetOptLong` El método añadido, que analiza las opciones largas. Funciona exactamente igual que el operador de paréntesis.

Además de los métodos, tiene cuatro atributos públicos interesantes, necesarios para la comunicación con el programa que utiliza sus servicios:

`int optind` Indica el índice del primer argumento que no es una opción. Muy útil para seguir analizando los parámetros que no sean opciones, como nombres de ficheros.

`char *optarg` El parámetro de la opción actual, para las opciones de la forma `--cut-depth=4`.

`int nargc` Una copia del número de argumentos pasados al programa.

`char **nargv` Una copia de los argumentos pasados al programa, reordenados si las opciones no se especificaron todas las principio.

Las opciones largas, que son las «nuevas», se especifican en el constructor del objeto `GetOpt`, mediante un puntero a una estructura `option`, definida como:

```
struct option {
    const char *name;
    int has_arg;
    int *flag;
    int val;
};
```

Este puntero es en realidad un puntero a un vector de este tipo. El significado de cada uno de estos campos es el siguiente:

`name` Es el nombre de la opción, es decir, lo que hay que especificar en la línea de órdenes, pero sin los dos guiones iniciales. Por ejemplo, para la opción `--with-index`, `name` valdría `with-index`.

`has_arg` Este parámetro entero indica si tiene argumento obligatorio (1), optativo (2) o no tiene en absoluto (0).

`flag` Indica cómo ha de indicarse la recepción de la opción. Si vale `NULL`, se devolverá el valor de la opción (el campo `val`). Si el puntero no es igual a `NULL`, se supone que apunta a un entero, que se carga con el valor `val` cada vez que se recibe la opción.

`val` Es el «valor» de la opción, devuelto o valor a cargar en cierta variable (véase campo `flag`).

5.7. El sistema de internacionalización *gettext*

Gettext es un sistema de internacionalización (internationalization, o simplemente «i18n», para los ingleses), creado por GNU. Está basado en la norma NLS creada por Uniform, e implementada por Sun en su versión de UNIX, el Solaris. Este sistema permite aislar del resto los mensajes a utilizar en el programa, de tal forma que se pueden traducir fácilmente. Esta facilidad se basa en la no necesidad de tocar el programa en sí, y, por supuesto, en no tener que saber nada de informática para traducir los mensajes.

Los conceptos importantes que constituyen el universo de *gettext* son los siguientes:

po Portable Object, traducción independiente de la máquina, preparada y modificada por personas.

pot Portable Object Template, plantilla con los mensajes a traducir, independiente de la máquina, preparada y modificada (normalmente) por programas, que buscan los nuevos mensajes y los ponen en este fichero.

mo Machine Object, versión del fichero original *po*, preparada para máquinas. No se actualizan directamente, sino sobreescribiéndolos con nuevas versiones sacadas de los originales *po*.

Principalmente, tenemos que llevar a cabo dos tareas para traducir nuestros programas: preparar el programa para traducir, y añadir las traducciones que convengan. Para modificar el programa, los pasos son:

1. Se modifica el programa, a mano, para que use llamadas a las funciones de *gettext* cada vez que vaya a utilizar un mensaje susceptible de ser traducido.
2. Mediante el programa `xgettext` se sacan los mensaje del programa, y se aíslan en un fichero *pot*, que se llamará *nombre_programa*`.pot`.
3. Se prepara el programa para que, al construirse, se enlace con las funciones de *gettext* y pueda funcionar el sistema de traducción, con la utilidad `gettextize`. Se crearán los directorios `intl` y `po`.

Una vez tenido el programa preparado para utilizar el sistema *gettext*, es trivial añadir nuevas traducciones. Los pasos a dar, en caso de usar *autoconf* y *automake* para construir nuestro programa, son:

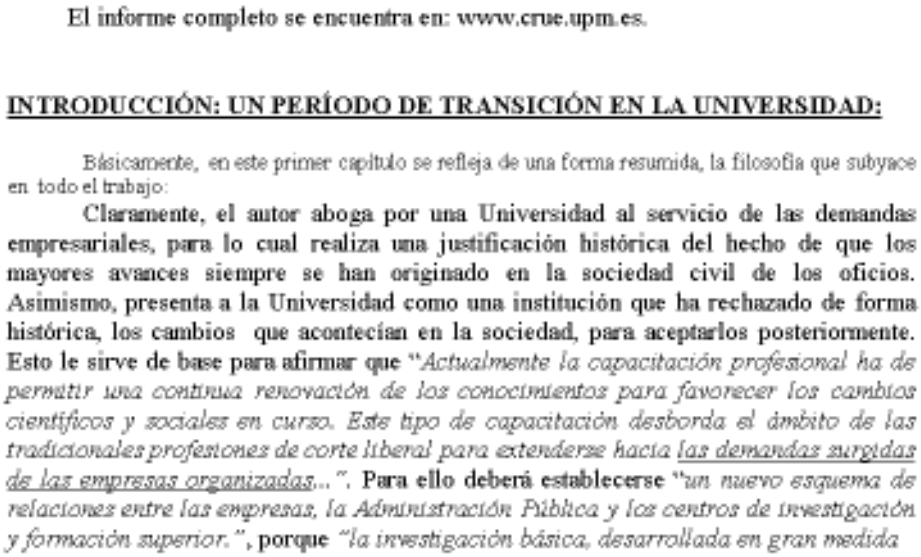
1. Copiar el fichero `.pot` a un fichero llamado *nombre_idioma*`.po`, y se traducen los mensajes.
2. Copiar el fichero final *po* al directorio `po` del programa, y se añade una referencia al fichero en la variable `ALL_LINGUAS` del fichero `configure.in`.
3. Se vuelve a construir el programa y se instala la nueva versión, para actualizar los ficheros de traducciones.

Como prueba de la facilidad de actualización y adición de traducciones a un programa basado en *gettext*, se ha traducido RTHC, en unos pocos días, a varios idiomas, entre ellos el castellano y el italiano.

5.8. Resultados finales del trabajo

Al final del trabajo, se ha terminado con lo siguiente:

1. Un paquete con los fuentes del programa RTHC y de las clases que componen *libFreeRTF*. Este paquete se ha construido automáticamente, debido al uso de las herramientas *autoconf* y *automake*, que se puede usar para distribuir el programa en versión fuentes (la manera convencional de ditribuir programas por la red).
2. Un paquete para las distribuciones Debian y las basadas en ella, como la Storm Linux de Stormix, la Corel Linux de Corel, y la Libra de Libra systems. Es muy fácil de instalar para alguien que no se maneje bien con conceptos de programación, por lo que es la vía ideal de distribución para que el programa se use mucho. Además, de esta forma, es más fácil que entre en una distribución. Se tienen tanto el paquete de fuentes como el de binarios.



El informe completo se encuentra en: www.crue.upm.es.

INTRODUCCIÓN: UN PERÍODO DE TRANSICIÓN EN LA UNIVERSIDAD:

Básicamente, en este primer capítulo se refleja de una forma resumida, la filosofía que subyace en todo el trabajo:

Claramente, el autor aboga por una Universidad al servicio de las demandas empresariales, para lo cual realiza una justificación histórica del hecho de que los mayores avances siempre se han originado en la sociedad civil de los oficios. Asimismo, presenta a la Universidad como una institución que ha rechazado de forma histórica, los cambios que acontecían en la sociedad, para aceptarlos posteriormente. Esto le sirve de base para afirmar que "*Actualmente la capacitación profesional ha de permitir una continua renovación de los conocimientos para favorecer los cambios científicos y sociales en curso. Este tipo de capacitación desborda el ámbito de las tradicionales profesiones de corte liberal para extenderse hacia las demandas surgidas de las empresas organizadas...*". **Para ello deberá establecerse** "*un nuevo esquema de relaciones entre las empresas, la Administración Pública y los centros de investigación y formación superior.*", **porque** "*la investigación básica, desarrollada en gran medida*

Figura 5.6: Ejemplo de salida de texto simple (RTF)

3. Un paquete RPM, para las distribuciones de RedHat, Mandrake, SuSE, Caldera, etc. Es muy fácil de instalar, al igual que el paquete Debian. Al igual que con los paquetes Debian, se tiene la versión final, del ejecutable, y la de los fuentes.

4. Una comunicación con la organización de las conferencias Hispalinux, para intentar presentar y exponer el proyecto en las conferencias más importantes sobre Linux y *software* libre que hay a nivel nacional.

5. Varias contribuciones de traducciones a diferentes idiomas del programa, para que desde su primer día de publicación lo puedan disfrutar cómodamente personas de diferentes idiomas maternos.

5.9. Ejemplos de salida

Como ejemplos de las salidas producidas por RHTC, se presentan aquí algunos pasajes de documentos RTF escogidos para hacer pruebas. Estos, a su vez, servirán también como prueba de lo enmarañado e intrincado del formato RTF, en contraposición a la simplicidad y elegancia del formato final, el HTML.

5.9.1. Traducciones de texto simple

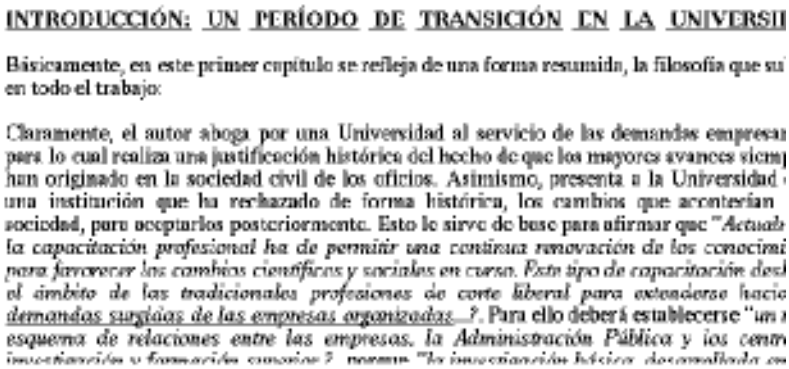
Es lo más fácil de conseguir, y también lo más importante de todo. A pesar de esto, se intentó trabajar un poco más de lo estrictamente necesario, y diferenciar los párrafos unos de otros, y además el primer párrafo de los demás. Esto permitiría hilar más finamente que en RTF, y permitir, utilizando hojas de estilo, cambiar el sangrado de la primera línea de los primeros párrafos y de los demás.

En las figuras 5.9.1 y 5.9.1 se pueden ver el original y la traducción de un documento de texto simple. Se mantienen todos los atributos físicos del texto, además de los cambios de párrafo y estilos (ver apartado siguiente).

5.9.2. Traducciones de estilos

Los estilos fueron algo más difíciles de reconocer. Sin embargo, se reconocen perfectamente, tanto los de párrafo como los de carácter. Uno de los puntos más importantes era reconocer los estilos que daban una estructura lógica al documento, que son los llamados «Título 1», «Título 2», etc.

En las figuras 5.9.2 y 5.9.2 se puede ver un pequeño documento de prueba, con títulos de apartados, reconocidos perfectamente. Como se puede comprobar, en la traducción no sólo se conserva la estructura del documento, sino además el aspecto físico de los estilos.



INTRODUCCIÓN: UN PERÍODO DE TRANSICIÓN EN LA UNIVERSII

Básicamente, en este primer capítulo se refleja de una forma resumida, la filosofía que su
en todo el trabajo:

Claramente, el autor aboga por una Universidad al servicio de las demandas empresa
para lo cual realiza una justificación histórica del hecho de que los mayores avances cient
han originado en la sociedad civil de los oficios. Asimismo, presenta a la Universidad
una institución que ha rechazado de forma histórica, los cambios que acontecían
sociedad, para aceptarlos posteriormente. Esto le sirve de base para afirmar que "*Actual
la capacitación profesional ha de permitir una continua renovación de los conocimi
para favorecer los cambios científicos y sociales en curso. Este tipo de capacitación des
el ámbito de las tradicionales profesiones de corte liberal para extenderse hacia
demandas surgidas de las empresas organizadas...*". Para ello deberá establecerse "*un
esquema de relaciones entre las empresas, la Administración Pública y los centr
investigación y formación superior*", porque "*la investigación básica, desarrollada en

Figura 5.7: Ejemplo de salida de texto simple (HTML)

Título 1

Titulillo

Texto normal

Titulillolillo

Más texto normal

Titulillolillolillo

El refi.

Figura 5.8: Ejemplo de salida de estilos (RTF)

Título 1

Titulillo

Texto normal

Titulillolillo

Más texto normal

Titulillolillolillo

El refi.

Figura 5.9: Ejemplo de salida de estilos (HTML)

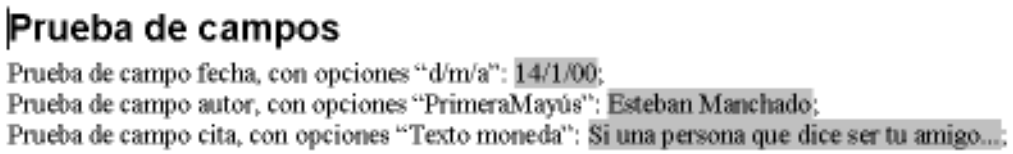


Figura 5.10: Ejemplo de salida de campos (RTF)

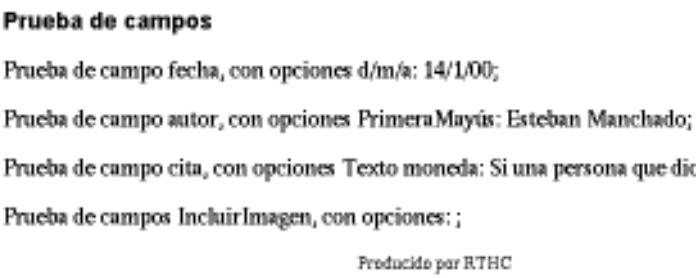


Figura 5.11: Ejemplo de salida de campos (HTML)

5.9.3. Traducciones de campos

Los campos son bastante fáciles de traducir, debido a que el propio Word almacena en un campo el último resultado hallado. Sin embargo, para los dos más importantes, las referencias y los hiperenlaces, se ha hecho algo más que traducir simplemente el resultado calculado: se ha producido un enlace, que lleve al texto original o a la URL indicada en el parámetro del hiperenlace.

5.9.4. Traducciones de tablas

Las tablas han dado bastantes problemas, por tres razones:

1. Tienen muchísimos parámetros, lo que hace difícil cubrir todas las posibilidades.
2. El propio Microsoft Word incumple en muchas ocasiones la propia especificación.
3. Las tablas se describen de una manera demasiado física, y es casi imposible traducirlas a un formato abstracto como HTML.

Aún así, las tablas más simples se reconocen sin problemas, y como prueba de ello, se presenta un ejemplo en las figuras 5.9.4 y 5.9.4.

5.9.5. Traducciones de listas

Las listas también han dado muchos problemas, por razones similares a las de las tablas: se incumplen las especificaciones, y sobre todo su descripción es demasiado física e incoherente como para poder convertirla fácilmente a una lista de HTML.

En las figuras 5.9.5 y 5.9.5 puede verse un ejemplo de listas, que no ha salido del todo bien. Las profundidades de entorno diferentes no se llegan a analizar correctamente, y, al ignorarse el texto insertado automáticamente, los números de la lista salen mal.

5.9.6. Traducciones con la opción `--cut-depth`

Para ilustrar esta opción, se usará un documento con la estructura marcada mediante los estilos con nombres como «Título 1», «Título 2», etc. Para cada ejecución, se dará la relación de los apartados pertenecientes a cada página.

Figura 5.12: Ejemplo de salida de tablas (RTF)

Figura 5.13: Ejemplo de salida de tablas (HTML)

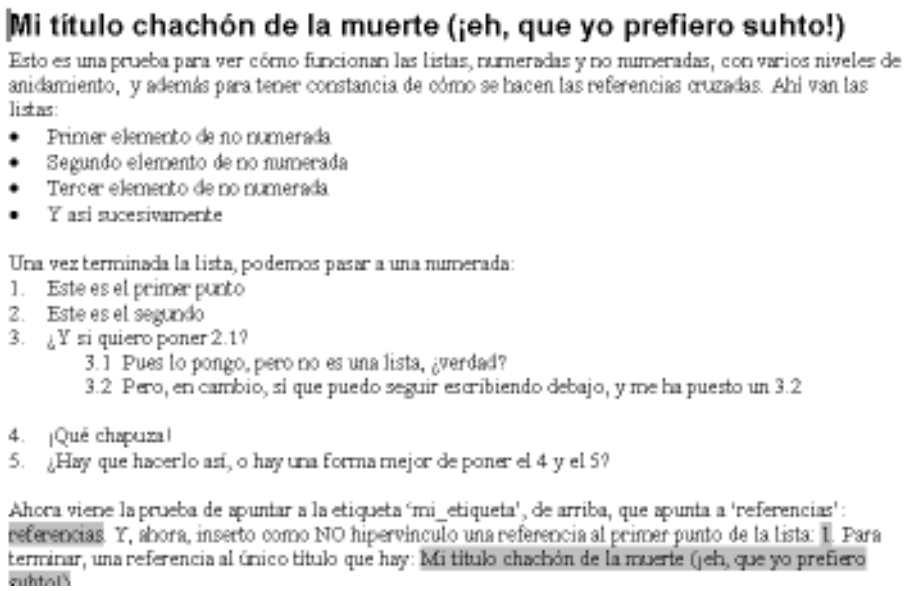


Figura 5.14: Ejemplo de salida de listas (RTF)

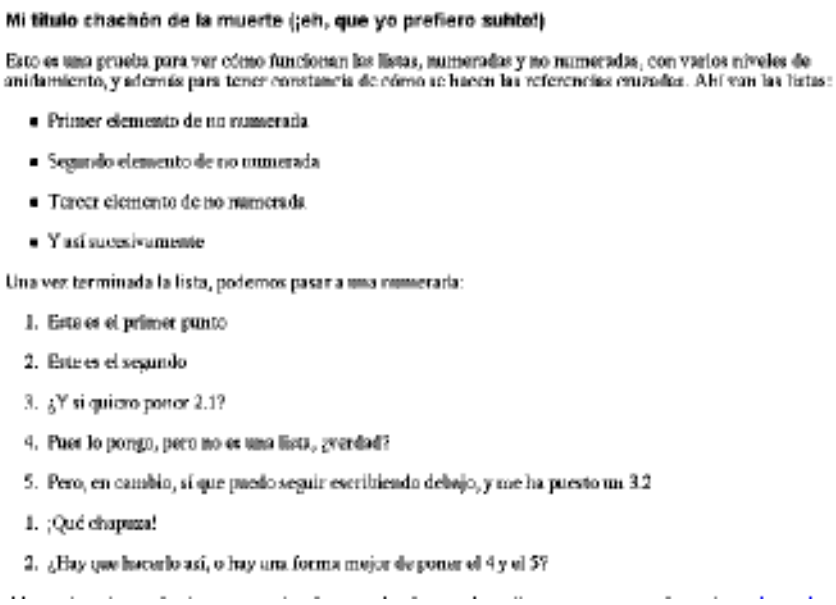


Figura 5.15: Ejemplo de salida de listas (HTML)

En la primera prueba, el documento original tenía los siguientes apartados:

1. Nota
2. Historia Interna
3. Historia Externa
4. La Estructura Del Quenya: Un breve repaso
   a) Fonología elemental
   b) El Sustantivo
   c) El Artículo
   d) El verbo
   e) El adjetivo
   f) Los participios
   g) Pronombres
5. Apéndice: Ejemplos De Sustantivos Quenya completamente declinados

Al invocar a RTHC de la siguiente forma:

```
rthc --cut-depth=2 Helge.rtf
```

se produjeron 13 ficheros, cuyos nombres se detallan a continuación:

1. `index-1.html`
2. `index-2.html`
3. `index-3.html`
4. `index-4.html`
5. `index-4.1.html`
6. `index-4.2.html`
7. `index-4.3.html`
8. `index-4.4.html`
9. `index-4.5.html`
10. `index-4.6.html`
11. `index-4.7.html`
12. `index-5.html`
13. `index-fn.html`

Los tres primeros ficheros contenían los tres primeros apartados respectivamente; el cuarto, el principio del apartado «La Estructura Del Quenya: Un breve repaso»; el quinto y sucesivos, hasta el fichero `index-4.7.html`, los subapartados del apartado cuarto; el fichero `index-5.html` contenía el último apartado; por último, el fichero `index-fn.html` contenían las cuatro notas al pie definidas por todo el documento.

Haciendo otra prueba con el mismo fichero, pero esta vez cortando por el nivel 1, con la orden

```
rthc --cut-depth=1 Helge.rtf
```

los resultados fueron similares. La lista de ficheros en este caso fue, como cabría esperar:

1. `index-1.html`
2. `index-2.html`
3. `index-3.html`

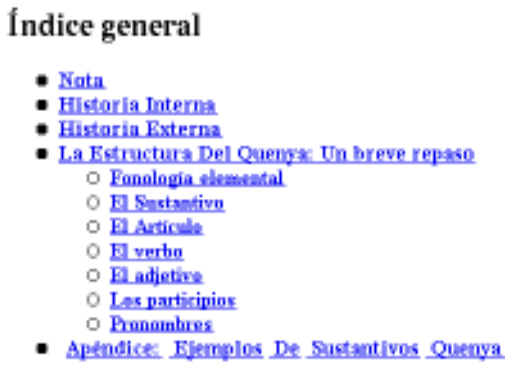


Figura 5.16: Ejemplo de índice de la opción `--with-index`

4. `index-4.html`

5. `index-5.html`

6. `index-fn.html`

La única diferencia entre ambos resultados es el cuarto fichero, que, además del texto perteneciente al apartado cuarto, contiene a todos sus subapartados, porque estamos cortando a un nivel menos profundo.

5.9.7. Traducciones con la opción `--with-index`

La opción `--with-index` crea automáticamente un índice para el documento, con enlaces a todos los apartados. Esto permite una navegación rápida y eficaz, e incluso conseguir fácilmente crear documentos con marcos sin necesidad de que el programa los cree automáticamente.

Una página de índice tiene el aspecto mostrado en la figura 5.9.7, aunque es importante destacar que las hojas de estilo usadas las incluye también esta página, con lo que se puede personalizar fácilmente su aspecto. Para poder hilar más fino, las etiquetas producidas en esta página tienen una clase especial, de tal forma que uno puede cambiarle el aspecto a las listas de esta página, con hojas de estilo, sin cambiárselo a las del documento convertido.

5.9.8. Traducciones con la opción `--with-jsindex`

El índice flotante es similar al índice normal. La única diferencia entre los dos es que, al construir un índice flotante, en todas las páginas producidas, se pone un pequeño programa en JavaScript que carga una ventana flotante, donde se carga el nuevo índice. Además, cuando uno pulsa en cualquier apartado en la ventana flotante, es la ventana principal la que cambia, así que siempre se tiene el índice flotante disponible para seguir navegando por el documento. Dado que es flotante es exactamente igual que el anterior, no vale la pena mostrarlo.

5.10. Tareas pendientes

Después de terminar el proyecto, los entes RTF no reconocidos son:

1. Las listas. Aunque se representan bien, no se consiguió conservar completamente (o, mejor dicho, *recuperar*, porque no está plasmado completamente en el formato RTF) la descripción lógica de las listas de más de un nivel de anidamiento.

2. Las imágenes y diagramas. Probablemente no es excesivamente difícil, pero sí muy tabajoso, y, dado que no eran ninguna prioridad en el proyecto, se dejaron de lado.

3. Los campos. La mayoría de éstos se ignoran. El único que se utiliza es el de referencias cruzadas. La documentación oficial *no* describe los campos en profundidad, y hacer un estudio completo de todos habría llevado mucho tiempo probando todos y cada uno en el Microsoft Word.

4. Los cambios de destino. Los únicos que realmente están implementados son las notas al pie, las más importantes. En HTML, después de todo, no se pueden representar las cabeceras y pies de páginas, puesto que este concepto no existe en el lenguaje.

6 Herramientas *software* usadas en el proyecto

Tal y como se anticipó en el apartado 1.4.3, la mayoría de las herramientas utilizadas para elaborar tanto el proyecto como la memoria fueron libres. En los siguientes apartados se ve una descripción de estas herramientas, haciendo incapié en las que resultaron nuevas, y se aprendieron durante estos meses.

6.1. Herramientas de programación

Las herramientas principales para escribir el programa en sí fueron:

1. Compilador de C++, entre otros
2. Conjunto de funciones y clases básicas del sistema
3. Herramientas de gestión del proyecto
4. Herramientas de gestión de cambios
5. Sistema de traducción semi-automática

Excepto la herramienta de gestión de cambios y el compilador de C++, el resto se aprendió a utilizar haciendo el proyecto.

Para la compilación en sí se usaron dos programas: *g++*, el compilador de C++ de GNU, y *flex++*, la versión de GNU de *lex*, que permite crear clases en C++ en vez de funciones en C. A pesar de que se tenía algo de experiencia con *flex*, nunca se había usado «en serio» *flex++* para producir una clase en C++, lo que provocó que se tuviera que investigar un poco para conseguir toda la funcionalidad requerida por la aplicación. En cuanto al propio C++ y a la funcionalidad básica del sistema para programar en este lenguaje, lo más importante que se aprendió fue:

La clase `string` Una clase para manejar series de caracteres, limpia y eficiente, que ahorra muchos de los quebraderos de cabeza que daba el uso de vectores de caracteres en el lenguaje C.

Excepciones Las excepciones son en realidad una característica del compilador. Permiten manejar los errores de una forma mucho más limpia y general, de tal forma que el programa queda más legible, y los errores se cazan mejor y más rápidamente.

Las clases de la STL STL es un conjunto de clases plantilla, que pertenecen a la norma C++ desde hace unos años. Nos permiten resolver la mayoría de los problemas de estructuras de datos dinámicas, como pilas, listas, vectores, etc. sin necesidad de preocuparnos por crear nosotros mismos las clases. Al ser plantillas, además, sirven para cualquier tipo de datos existente, y, con la ayuda de unos algoritmos genéricos, pertenecientes a la propia STL, podemos ordenar las estructuras, recorrerlas, procesarlas fácilmente, etc.

Como último comentario, se puede nombrar a la pequeña utilidad *ctags*, que crea un fichero especial, de «etiquetas», que pueden leer algunos programas, como *vim* o *emacs*, y que sirven para saltar en cualquier momento a cualquier función o clase definida en el programa. Esto facilita mucho la programación, ya que permite, desde cualquier punto de cualquier fichero, acceder directamente a la declaración de una clase, a la definición de una función, etc.

Las herramientas usadas para gestionar el proyecto fueron `autoconf` y `automake`, y por tanto `make`, implícitamente. Estas dos herramientas, utilizadas juntas, nos permiten deshacernos de mucho del trabajo que antes tenía que hacerse a mano, como construir los ficheros *Makefile* del proyecto, llevar la cuenta de las dependencias, etc. Por si fuera poco, estas herramientas están pensadas para trabajar con muchas otras herramientas, como el *gettext*, los diferentes compiladores más populares, etc., con lo que el uso de éstas se hace casi automático y transparente para el programador. Por último, también realizan automáticamente tareas tan comunes y pesadas como crear los objetivos necesarios para instalar el programa, para crear una distribución del programa en fuentes (la normal en Internet)...

Para llevar el control de los cambios producidos al programa (control de versiones) se usó la herramienta *RCS*, con la que se tenía algo de experiencia. Hay otra, más usual en Internet, llamada *CVS*, que es más potente, pero (1) no era necesaria para el proyecto, y (2) era demasiado complicada.

Por último, para hacer las traducciones, se optó por el sistema *gettext*, muy usado en todos los proyectos de GNU, y muy popular en toda la comunidad de programadores de *software* libre. Gracias a este sistema, se pudo traducir el programa a varios idiomas, incluido el propio castellano, ya que originalmente se escribió en inglés.

6.2. Otras herramientas

Además del programa en sí, se tuvo que escribir la documentación y construir paquetes fáciles de instalar. Las herramientas usadas para estos pasos fueron:

1. Herramientas de creación de paquetes, como *dpkg-buildpackage* o *rpm*. También se usó *debmake*. No se tenía ningún tipo de experiencia previa ni con los formatos de paquetes ni con las herramientas que ayudaban a crearlos.

2. Para formatear la documentación, se usó el sistema teTEX de Linux, una versión del sistema TEX de Donald Knuth. La documentación en sí se escribió en LATEX, con una clase de documento inventada para tal efecto, basada en las clases *book* y *article*. También, para escribir las versiones iniciales de dos de los anexos, se usó el programa LYX.

3. Para la vista preliminar de la documentación se usó el programa *xdvi*.

4. Para la elaboración de los diagrama de la memoria se usó el programa *dia*.

5. Para el retoque y conversión entre formatos de las imágenes se utilizó el programa Gimp.

7 Conclusiones

Durante todos estos intensos meses de trabajo, se han conseguido cumplir todos los objetivos marcados, además de otros que al principio no se habían planteado.

Se ha estudiado el formato RTF en profundidad, lo que ha desembocado también en aprender a hacer ingeniería inversa de programas para descubrir todos los aspectos del formato que no estaban claros en la especificación. Para intentar paliar esta situación, se ha escrito un documento alternativo de descripción del formato RTF.

También, debido a la experiencia conseguida con la construcción del programa en sí, y por la consulta de varios libros de ese tema, se ha desarrollado la capacidad de analizar y diseñar programas orientados a objetos. Esto, por añadidura, se ha conseguido en un entorno de programación de sistemas, en el que es menos común aplicar este tipo de técnicas.

Siguiendo con los estudios sobre la comunidad del *software* libre, se ha aprendido a construir paquetes de programas, a utilizar herramientas tan importantes como *autoconf*, *automake* y *gettext*, y las costumbres de publicación de programas libres. Por ello, en breve se anunciará, tal y como se aconseja en [ESR99], el programa en varios grupos de noticias y páginas, y se pondrá en funcionamiento una página en Internet, donde se podrá copiar el programa, en fuentes o en paquetes fáciles de instalar, y se tendrá acceso a preguntas frecuentes y a las futuras actualizacines que se hagan.

Resumiendo, se ha conseguido:

- Un conocimiento bastante extenso del formato RTF, a pesar de quedar todavía muchas lagunas, dada su inmensa extensión y complejidad.
- Habilidad para investigar sobre formatos de documentos mal documentados, a base de prueba y error, y análisis de ejemplos elegidos.
- Desarrollar extraordinariamente la capacidad de análisis y sobre todo diseño orientado a objetos.
- Crear un conjunto de clases muy útiles para uso ajeno, tanto desde el punto de vista del usuario, como del programador e incluso del investigador.
- Obtener o desarrollar conocimientos sobre el uso de muchas herramientas de programación, como `flex++`, `automake` o `autoconf`.
- Conocimientos para crear sencillos paquetes de programas, para facilitar el uso y la difusión de éstos.
- Habilidad en la programación de sistemas con el lenguaje C++.
- Aprender a utilizar la extensa STL, el conjunto de plantillas básicas de clases de la nueva norma del lenguaje C++.
- Una colección de referencias y documentos muy útiles para introducirse en el mundo del *software* libre.
- Una especificación alternativa mucho más clara, aunque incompleta, del formato RTF.

7.1. Trabajos futuros

Tal y como se explicó en el apartado 5.1.2, se puede modificar de muchas maneras el programa. En los siguientes subapartados se presentan los trabajos más relevantes que se pueden hacer a partir de este proyecto, según la clasificación del apartado 5.1.2.

7.1.1. Mejoras al conversor

Las extensiones posibles del conversor son variadas, interesantes y, hasta cierto punto, sencillas. Entre las más interesantes se encuentran:

1. Añadir al programa más funcionalidad, por ejemplo haciendo que entienda las partes del RTF que todavía no reconoce (ver apartado 5.10).
2. Modificar la manera en la que se genera el HTML, usando hojas de estilo para acercar más el resultado final al documento inicial.
3. Adaptarlo a nuevas normas, como el XHTML.
4. Añadir nuevas variables para sustituir, y así poder personalizar aún más la salida.
5. Diseñar nuevos ficheros de configuración, con ayuda de lenguajes de programación como Python o Scheme, para dar más flexibilidad al usuario a la hora de elegir las traducciones.
6. Mejorar más aún la extensibilidad del programa, añadiendo la capacidad de cargar módulos dinámicamente (escritos preferentemente en un lenguaje interpretado, como los nombrados anteriormente).

7.1.2. Adaptaciones y mejoras a *libFreeRTF*

Aunque *libFreeRTF* es fácilmente adaptable a nuevas normas RTF y a diferentes entornos, siempre se puede mejorar lo actual, y se le pueden añadir nuevas características o mejorar su eficiencia. Entre otras, se pueden llevar a cabo las siguientes modificaciones:

1. Mejorar la eficiencia de *libFreeRTF*. Un ejemplo claro es implementar la clase `RTFTextChunk`, que actualmente es un sinónimo de la clase `string`, de manera eficiente.
2. Añadir nuevas clases para manejar los conceptos que puedan aparecer en futuras revisiones del formato RTF.
3. Facilitar un poco el uso de *libFreeRTF*, modificando algunas de las clases, para que tengan una interfaz más rápida de manejar desde fuera.
4. Estudiar el otro formato nativo de Word, que lleva su mismo nombre, comprobar si es equivalente al RTF, y convertir *libFreeRTF* en *libWord*.

7.2. Usos futuros de *libFreeRTF*

libFreeRTF permite usos casi ilimitados, porque no pone restricciones en cuanto su utilización. Así, se proponen los siguientes ejercicios para probar la versatilidad de *libFreeRTF*:

1. Un conversor de RTF a XML, que es actualmente la norma en Internet, no sólo para documentos de texto, sino para todo tipo de datos, tan dispares como puedan ser transferencias bancarias, diagramas o grafos de Gantt.
2. Construir un pequeño redactor de textos con *libFreeRTF*, para comprobar la fiabilidad y la adaptabilidad de sus clases a circunstancias «dinámicas».
3. Crear un conversor de RTF a LaTeX, que es lo más parecido que hay en UNIX a un formato universal para intercambio de documentos de texto.
4. Diseñar un «embellecedor» de documentos RTF, ya que estos traen mucha información redundante que agranda innecesariamente el fichero.
5. Escribir un creador automático de índices de documentos, quizás con propósitos de gestión de una gran colección de documentos RTF.

8 Referencias

Esta es una lista con la documentación utilizada para tomar las decisiones sobre las herramientas a utilizar y sobre los formatos de documentos considerados. Incluye tanto libros como páginas de Internet relacionadas con el proyecto.

En los siguientes apartados se resaltarán las entradas más importantes de los diferentes temas tratados.

8.1. *Software* libre

- [GNUWeb] La página oficial del proyecto GNU, en varios idiomas, con mucha información, la lista de programas de GNU, novedades, una revista electrónica, humor... Completamente imprescindible.
- [DebianWeb] La página oficial de la distribución Debian, la distribución más «auténtica», en cuanto a *software* libre se refiere.
- [ESR97] El artículo «The Cathedral and the Bazaar», de Eric S. Raymond. Probablemente es el artículo más influyente en la historia reciente del *software* libre. Entre otras «hazañas», consiguió que Netscape liberara su navegador bajo una licencia considerada libre.
- [ESR99] El «Software Release Practice HOWTO», un documento que explica diversos aspectos técnicos sobre el *software* libre. De obligada lectura para principiantes en este mundillo, o para cualquiera que desee conocer las costumbres de la comunidad.
- [JAC96] El documento «Packaging manual», que describe cómo construir paquetes Debian.
- [JAC98] El documento «Debian Policy», que describe la política de Debian en todo lo que respecta a crear paquetes de *software*.
- [BAR97] El documento «RPM HOWTO», que describe la historia del formato RPM, cómo usar el programa que lo gestiona, y cómo construir paquetes. Muy completo.
- [FMWeb] `http://freshmeat.net`, una página donde hay diariamente del orden de 40 anuncios de nuevos proyectos o actualizaciones de programas. De obligada visita para mantenerse al día con todos los proyectos libres existentes.
- [CSWeb] `http://www.cosource.com`, un proyecto de colaboración entre programadores de *software* libre y organizaciones que necesiten programas. Hay algunos proyectos similares, pero éste es de los más veteranos.

8.2. Formatos de ficheros

- [MIC97] La especificación oficial de Microsoft del formato RTF. Tal y como describe esta memoria con asiduidad, es bastante incompleta, pero es imprescindible para empezar a trabajar con el formato.
- [W3CWeb] La página oficial del *World Wide Web Consortium*, donde se pueden encontrar todas las especificaciones oficiales de los formatos y lenguajes que tienen relación con la *web*, como el HTML, el ECMAscript, las hojas de estilo, el XML, etc.
- [WOTWeb] Una página en Internet con enlaces a especificaciones de formatos de fichero de lo más variopinto, desde vídeo y animación hasta formatos de texto, como el RTF.

8.3. Programación

- [BOO95] Un libro muy adecuado para empezar a estudiar UML, el lenguaje gráfico de descripción de diseños orientados a objetos. Es muy fácil de leer y sencillo.
- [STR95] El libro definitivo para estudiar C++. El clásico de Stroustrup.
- [MUS97] Un libro muy completo y muy bien escrito sobre STL.

Bibliografía

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

[OSSWeb] `http://www.opensource.org`

[DebianWeb] `http://www.debian.org`

[JAC96] «Packaging manual» Ian Jackson. 1996.

[JAC98] «Debian Policy» Ian Jackson, Christian Schwarz, David A. Morris. 1998.

[BAR97] «RPM HOWTO» Donnie Barnes. 1997.

[SAN99] «Creación de paquetes Debian» Javier Fernández-Sanguino Peña. 1999.

[DSWeb] `http://packages.debian.org/devscripts`

[GNUstepWeb] `http://www.gnustep.org`

[SGIOSSWeb] `http://oss.sgi.com`

[ESR97] `http://www.tuxedo.org/ esr/writings/cathedral-bazaar/`

[ESR98] `http://www.tuxedo.org/ esr/writings/homesteading/`

[ESR99] `http://metalab.unc.edu/LDP/HOWTO/Software-Release-Practice.html`

[FMWeb] `http://freshmeat.net`

[CSWeb] `http://www.cosource.com`

[GTInfo] Páginas de ayuda *info* del sistema de internacionalización *gettext*.

[AMInfo] Páginas de ayuda *info* del programa *automake*

[ACInfo] Páginas de ayuda *info* del programa *autoconf*

[GAM95] «Design Patterns. Elements of Reusable Object-Oriented Software» Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Addison-Wesley, 1995.

[BOO95] «The Unified Modeling Language User Guide» Grady Booch, James Rumbaugh, Ivar Jacobson. Addison-Wesley, 1998

[MIC97] «Rich Text Format (RTF) Specification», Microsoft Corp.

[W3CWeb] `http://www.w3c.org`

[GOO98] «Dynamic HTML: The Definitive Reference», Danny Goodman. O'Reilly & Associates, 1998.

[HOL92] «Programación en PostScript», David A. Holzgang. Addison Wesley Longman, 1990.

[WOTWeb] `http://www.wotsit.org`

[STR95] «The C++ Programming Language», Bjarne Stroustrup. Addison Wesley Longman, 1997.

[OUA95] «Practical C++ Programming» Steve Oualline. O'Reilly & associates, 1995.

[MUS97] «STL Tutorial and Reference Guide», David R. Musser y Atul Saini. Addison Wesley Longman, 1995.

[SNO92] «TEX for the Beginner» Wynter Snow. Addison Wesley, 1992.

[BAU98] «Una descripción de LATEX2ε» Tomás Bautista, Tobias Oetiker, Hubert Partl, Irene Hyna y Elisabeth Schlegl, 1998.

Anexos

A Manual de referencia de *libFreeRTF*

A.1. Introducción

La clase `RTFTree` y el resto del paquete *libFreeRTF* nace por la necesidad de construir un analizador para RTF que fuera, en la medida de lo posible, (1) fácil de entender (para documentar con un programa libre el formato RTF) y sobre todo (2) reutilizable. Por ello, y a pesar de que se construyó para implementar un conversor, un objeto de la clase `RTFTree` sólo contiene información sobre el fichero original y nada sobre un hipotético lenguaje destino.

A.1.1. Clases

libFreeRTF lo componen varias clases relacionadas:

`RTFTree` Esta clase es la principal, y se utiliza para guardar toda la representación del documento RTF.

`RTFContents` Es una clase abstracta (virtual) cuyas clases herederas son los nodos del árbol RTFTree. Los herederos más importantes son:

`RTFText` Guarda un trozo de texto (no tiene hijos)

`RTFSection` Guarda los contenidos de una sección (tiene hijos)

`RTFCtrl` Guarda una palabra de control (no tiene hijos)

`RTFEnviron` Guarda una palabra de control que se aplica en un entorno (tiene hijos)

`RTFRow` Guarda una fila de una tabla (tiene hijos)

Las operaciones comunes a todas estas clases, declaradas en `RTFContents` son:

`RTFContents *getFather()` Devuelve el padre del nodo actual

`void addNode(RTFContents *)` Añade un nodo al actual, si es posible

`bool isFertile()` Devuelve `true` si la clase puede tener hijos o `false` en caso contrario

`string infoString()` Devuelve un objeto de clase `string` con información sobre el nodo

`RTFParent` Clase de la cual heredan todos los herederos de `RTFContents` que tengan hijos.

`RTFParser` Es una clase que construye objetos de tipo `RTFTree`, a partir de flujos de información con documentos RTF. Se puede decir que es un analizador sintáctico de RTF (de ahí su nombre), y se apoya en un analizador léxico.

`RTFLexer` Es un analizador léxico de RTF. Recibe un flujo de información y devuelve todos los símbolos RTF que se va encontrando. Esta construido con la herramienta *flex++*.

`RTFVisitor` Es la clase base para construir «visitantes» de árboles RTF. Cada clase heredada de `RTFVisitor` realiza una tarea determinada, como recorrer un árbol dado para imprimir información de depuración, traducir el contenido del árbol a un nuevo fichero RTF, traducir el contenido del árbol a uno o más ficheros en otro formato, como HTML, contar el número de párrafos del documento, procesar de alguna forma todas las notas al pie, etc.

Símbolo	Significado
CTRLW	Palabra de control RTF (genérica)
STAR	Asterisco (*)
CTRLS	Símbolo de control RTF
SYMBOL	Símbolo (\'hh)
TEXT	Texto sin formato
OCURLY	Llave abierta
CCURLY	Llave cerrada
SECTD	Palabra de control \sectd
PARD	Palabra de control \pard
PLAIN	Palabra de control \plain
PAR	Palabra de control \par
SECT	Palabra de control \sect
PAGE	Palabra de control \page
COLUMN	Palabra de control \column
LINE	Palabra de control \line
DESTCHANGE	Cambio de destino
TAB	Palabra de control \tab
EMDASH	Palabra de control \emdash
ENDASH	Palabra de control \endash
LQUOTE	Palabra de control \lquote
RQUOTE	Palabra de control \rquote
LDBLQUOTE	Palabra de control \ldblquote
RDBLQUOTE	Palabra de control \rdblquote
TILDE	Símbolo de control \~
DASH	Símbolo de control \-
UNDERSCORE	Símbolo de control _

Cuadro A.1: Símbolos léxicos definidos en *libFreeRTF*

A.2. Símbolos léxicos

libFreeRTF define[1] y utiliza 26 símbolos léxicos, que están explicados en la tabla A.1. En general, se ha intentado crear un símbolo léxico para cada tipo de símbolo existente en RTF, y además otro por cada caso particular relevante y frecuente.

Así, en el primer conjunto de símbolos nos encontramos las llaves, para los grupos RTF, las palabras de control, los símbolos de control, y el texto. En el segundo, están los símbolos (SYMBOL), que son los símbolos de control que representan caracteres a los que el documento se refiere por su índice en la tabla de caracteres usada; el símbolo STAR, que representa otro caso particular de símbolo de control; PARD, PLAIN, COLUMN, etc., que son palabras de control de uso común y trato muy específico. DESTCHANGE es un caso particular de las palabras de control, y aúna todas las palabras de control que se consideran cambios de destino hasta el RTF 97.

A.3. Clases de apoyo

Hay varias pequeñas clases definidas en *libFreeRTF* que, aunque no sirven de nada por sí solas, son necesarias para el funcionamiento de todo lo demás. Estas clases son abstracciones de conceptos existentes en RTF, como colores, estilos, tipos de letra, o palabras de control. Todas estas se caracterizan por ser clases pasivas, que simplemente contienen datos y tiene métodos para acceder a ellos. Estas clases son:

`SectionNumber` Guarda un número de apartado, como `1.1.3` o `2.5`.

`RTFRefTable` Es una clase que guarda una tabla de referencias cruzadas en el texto, asociando una etiqueta RTF (de clase `string`) a un número de apartado (de clase `SectionNumber`).

[1] En el fichero `tokens.h`

`RTFData` Almacena toda la información necesaria para representar cualquier palabra de control RTF, es decir, su contenido (la palabra en sí), su parámetro numérico, que no siempre es necesario, y una marca para saber si realmente hay parámetro.

`RTFColor` Representa un color RTF, que básicamente es una terna RGB.

`RTFFont` Guarda un tipo de letra RTF, con todas sus características.

`RTFStyle` Abstrae un estilo RTF, con todas sus propiedades.

En los siguientes subapartados se describirán mejor todas estas clases.

A.3.1. `SectionNumber`

Esta clase se utiliza para llevar la cuenta del apartado por el que vamos. Es útil tanto para tenerlo de referencia (por ejemplo, para crear unos hipotéticos nombres de ficheros de salida según los apartados) como para poder saber a qué apartados apuntan las referencias cruzadas, o donde encontrar los destinos.

La tabla de referencias cruzadas necesita asociar una etiqueta con el apartado en el que aparece, para poder de alguna forma saber dónde se encuentra el destino. Por ejemplo, para un hipotético conversor que parta la salida en varios ficheros según el apartado, es completamente necesario saber donde está cada etiqueta referenciable, para poder especificar el fichero en el que caerá el destino de la referencia.

Sólo hay tres métodos definidos para la clase `SectionNumber`: `nextSection()`, `reset()` y `operator<<()`. El primero hace avanzar el apartado en uno, y recibe un parámetro que especifica qué «profundidad» hay que hacer avanzar[2]; el segundo no recibe parámetros, y deja al objeto en el apartado *0*; el último es una sobrecarga del operador `<<`, para poder mostrar por pantalla objetos de este tipo.

A.3.2. `RTFRefTable`

Esta clase es un índice de referencias cruzadas, donde se guardan todas las correspondencias de etiquetas con apartados en los que éstas aparecen. La necesidad de esta tabla se explicó en el apartado A.3.1.

Los métodos definidos en esta clase son cuatro: `addTag()`, `existsTag()`, `getSection()` y `operator[]`. `addTag()` recibe como parámetros una etiqueta (un objeto `string`) y el apartado en el que aparece (un objeto `SectionNumber`), y asocia la una al otro; `existsTag()` recibe una etiqueta como parámetro, y devuelve verdadero si la etiqueta ya tiene un apartado asociado en la tabla de referencias; `getSection()` retorna el apartado asociado a la etiqueta que se recibe como parámetro; el operador corchetes es un sinónimo más legible del método `getSection()`.

A.3.3. `RTFData`

Esta clase almacena la información asociada a los símbolos RTF. Es decir, que podemos definir completamente cualquier elemento léxico RTF con el tipo de símbolo y un objeto de tipo `RTFData`. Esta clase se utiliza profusamente por todo *libFreeRTF*. Tanto es así, que está definido en el fichero `rtfdefs.h`.

Los atributos de esta clase son `content` y `paramVal`, ambos de tipo `string`, y `hasParam`, de tipo `bool`. Tiene seis métodos para acceder como lectura y escritura a los tres campos, un método `str()`, que devuelve un objeto `string` con la representación del objeto `RTFData` actual, y tiene sobrecargado el operador `<<`, para poder mostrar por pantalla cómodamente el objeto.

Además del constructor por defecto, tiene un constructor explícito que recibe dos objetos `string` y uno `bool`, para rellenar todos los datos del objeto desde el momento de su construcción.

A.3.4. `RTFColor`

Esta clase abstrae un color RTF, que es un color guardado como componentes RGB. Como se puede comprobar en el fichero `headerdefs.h`, la definición de `RTFColor` es bastante simple: se compone de tres atributos enteros, a los que se puede acceder tanto para leer como para escribir, como de costumbre, con seis métodos.

Como comentarios adicionales, es destacable que hay definido un constructor que recibe los tres componentes RGB, que el negro, color por defecto, tiene una constante definida llamada BLACK, y que para guardar la lista de colores RTF hay una definición de tipo llamado `RTFColorList`, que no es más que un vector STL de objetos `RTFColor`.

[2]Por ejemplo, si actualmente un objeto de tipo `SectionNumber` está en el apartado 2.4.3, y se llama al método `nextSection(1)`, el objeto quedará en el apartado 2.5.

A.3.5. `RTFFont`

Esta clase almacena un tipo de letra RTF. Actualmente sólo guarda las dos propiedades más importante: el nombre de la letra y su familia, que es muy útil para sustituir un tipo de letra por otro, cuando no se tiene el tipo exacto disponible.

Como era de esperar, tiene definidos dos atributos tipo `string`, llamados `family` y `name`, y cuatro métodos para acceder a ellos. Por otro lado, también hay una definición de un tipo `RTFFontList`, que es una correspondencia STL entre objetos `string` (los identificadores de los tipos de letra) y objetos `RTFFont`.

A.3.6. `RTFStyle`

Esta clase guarda los atributos de un estilo RTF, ya sea de carácter, de párrafo o de apartado. Aquí la cantidad y variedad de datos es mucho mayor que en los anteriores casos.

Los atributos definidos actualmente son:

- `name`, un objeto `string` que guarda el nombre del estilo.
- `additive`, un atributo `bool` que indica si el estilo es aditivo o no.
- `def`, una lista de palabras de control que comprenden la definición del estilo.
- `basedon`, un objeto `string` que indica en qué estilo se basa el actual.
- `nextStyle`, que indica que estilo es el siguiente al actual, cuando se está escribiendo. Es decir, que no vale para nada a no ser que se pretenda diseñar un redactor de textos o un programa similar.

Como de costumbre, hay definidos dos métodos por atributo para acceder a ellos. También para `RTFStyle` hay definido un tipo `RTFStyleList`, que no es más que una correspondencia de enteros con objetos `RTFStyle`.

A.4. Tipos de datos adicionales

Para hacer al programa más legible y fácil de mantener, se han definido varios tipos de datos nuevos, con la primitiva de C++ `typedef`. La mayoría de estos están en el fichero `typedefs.h`, aunque los que no son tan generales se encuentran definidos en otros ficheros. A continuación pasan a describirse los existentes:

- `RTFCtrlWordSet`. Representa un conjunto de palabras de control RTF. Se define como un conjunto (STL) de objetos `string`, es decir, un `set<string>`.
- `RTFTextChunk`. Es un trozo de texto. Se define simplemente como la clase `string`. El hecho de usar una clase «intermedia» `RTFTextChunk` permite en un futuro optimizar *libFreeRTF* creando una nueva clase que guarde la referencia del trozo de texto (algo como su desplazamiento inicial y longitud) en el fichero RTF original, y así pasar por parámetro muchos menos datos. Este aumento de eficiencia es directamente proporcional al tamaño de los objetos pasados.
- `RTFCtrlWordList`. Abstrae una lista de palabras de control. Esto se utiliza para trabajos más «serios» que los conjuntos, por lo que se define como un vector STL de objetos `RTFData`, y no objetos `string`, o sea, `vector<RTFData>`.
- `RTFWordListMap`. Este tipo de dato es una correspondencia de listas de palabras, y se utiliza para conseguir una lista de palabras de control indizando por un objeto `string`. Se define como una correspondencia STL de objetos `string` a `RTFCtrlWordList`, o sea, `map<string,RTFCtrlWordList>`.
- `RTFDestination`. Representa un cambio de destino RTF. Se define como una simple clase `string`.

A.5. La clase `RTFContents` y sus herederas

La clase `RTFContents` es la clase más importante de *libFreeRTF*. Define los métodos básicos de todos los nodos de un árbol `RTFTree` y por tanto la arquitectura de la representación.

Clase	Significado
RTFText	Texto
RTFEnviron	Grupo RTF (símbolos entre llaves)
RTFSection	Apartado («Título 1», «Título 2», ...)
RTFRow	Fila de una tabla
RTFTag	Etiqueta a la que hacer referencias
RTFEmbedded	Objeto incrustado
RTFDestChange	Cambio de destino (notas al pie, encabezados, ...)
RTFAbstractCtrl	Clase abstracta, común a todos los tipos de palabras de control
RTFCtrl	Palabra de control
RTFSymbol	Símbolo de control
RTFPlain	Palabra de control \plain
RTFPard	Palabra de control \pard
RTFSectd	Palabra de control \sectd
RTFPar	Palabra de control \par
RTFSect	Palabra de control \sect
RTFPage	Palabra de control \page
RTFColumn	Palabra de control \column
RTFLine	Palabra de control \line
RTFStar	Palabra de control *
RTFSpecialChar	Carácter especial
RTFAbstractStyle	Clase abstracta, común a todos los estilos
RTFCharStyle	Estilo de carácter
RTFParStyle	Estilo de párrafo
RTFSectStyle	Estilo de apartado
RTFShape	Forma dibujada

Cuadro A.2: Herederos de RTFContents

A.5.1. Clases herederas

Cada clase heredera de RTFContents representa un ente de RTF, como un trozo de texto, un símbolo especial, un cambio de párrafo, una fila de una tabla, un cambio de destino, etc. Todos comparten una serie de métodos descritos en el apartado A.1.1. Seguidamente se pasa a describir todas y cada una de las clases herederas definidas por defecto.

RTFText Esta clase almacena un trozo de texto (un símbolo TEXT devuelto por el analizador léxico). Sus métodos son getText() y setText(), para obtener y establecer, respectivamente, el texto del nodo, mediante objetos RTFTextChunk.

RTFEnviron Esta clase representa un grupo RTF. Sus hijos son el contenido del grupo. No tiene métodos ni atributos adicionales.

RTFSection Esta clase es la representación de un apartado RTF. En teoría, estos nodos sólo pueden aparecer como hijos inmediatos del nodo raíz, si aparecen en absoluto. Sus hijos son el contenido del apartado. Sus nuevos métodos son getName(), que devuelve un puntero al árbol que representa el nombre, y getDepth() y setDepth(), para tratar con la profundidad del apartado.

RTFRow Esta clase guarda una fila de una tabla RTF. Es importante destacar que las tablas, en sí mismas, *no* existen en RTF. Por tanto, no se puede hablar de tablas, sino de conjuntos de filas consecutivas. Las filas tiene como hijos el contenido de éstas. Las diferentes celdas se distinguen por terminar con la palabra de control \cell. Además de los hijos, los nodos RTFRow tienen varios métodos adicionales:

getRowDefs() Devuelve la definición de la fila.

getCellDefs() Devuelve el vector con las definiciones de las celdas de la fila.

getCellDef() Devuelve, dado un índice como parámetro, una definición de celda.

getNCells() Devuelve el número de celdas que hay en la fila.

addRowDef() Añade una palabra de control a la definición de la fila.

addCellDef() Añade una *definición completa* de celda (una lista de palabras de control) al vector de definiciones de celdas.

`RTFTag` Esta clase representa una etiqueta a la que hacer referencias. Estas etiquetas pueden marcar el principio o el final del texto referenciado. Tiene métodos para manejar la etiqueta del nodo (un objeto de tipo `string`) y para comprobar o establecer si la etiqueta es de cierre o no, llamados `getTag()`, `setTag()`, `getEndTag()` y `setEndTag()`.

`RTFEmbedded` Esta clase almacena un objeto incrustado. Es simplemente un objeto que tiene como hijos a todos los hijos de un grupo que ha empezado por `\object`.

`RTFDestChange` Esta clase es la representación de un cambio de destino RTF. Tiene dos métodos para tratar con la marca del destino (un objeto de tipo `RTFDestination`), llamados `getDestination()` y `setDestination()`.

`RTFAbstractCtrl` Esta clase es una abstracción de todas las palabras de control. No añade ningún método ni atributo nuevo, es una abstracción simplemente conceptual.

`RTFCtrl` Esta clase guarda una palabra de control, como `\i`, `\widctlpar` o `\keepn`. Añade los métodos `getContent()`, `getHasParam()` y `getParamVal()` para acceder al contenido, a la existencia de parámetro y a su valor, y `setContent()`, `setHasParam()` y `setParamVal()` para establecer sus valores.

`RTFSymbol` Esta clase almacena un símbolo RTF, ya sea un símbolo de control como `\'f3`, que representa a la vocal «o» con tilde, o uno como `\~`, que representa un espacio en blanco que no se puede «partir» (es decir, que no se puede sustituir por un cambio de línea). Al ser parecida a `RTFCtrl`, añade métodos para consultar y establecer el contenido (es decir, unos métodos `getContent()` y `setContent()`, que se comportan igual que sus homónimos de la clase `RTFCtrl`), aunque carecen de los métodos relativos al parámetro optativo, ya que no tiene sentido en los símbolos de control.

`RTFPlain` Esta clase es una abstracción de la palabra de control `\plain`. Obviamente, no tiene ningún método ni atributo adicional, ya que representa un caso muy particular de palabra de control: ya está establecido el contenido, y estas palabras no pueden tener parámetro.

`RTFPard` Esta clase es una abstracción de la palabra de control `\pard`. Ésta no puede tener parámetro numérico, así que la clase no añade ningún método ni atributo.

`RTFSectd` Esta clase es una abstracción de la palabra de control `\sectd`. Ésta no puede tener parámetro numérico, así que la clase no añade ningún método ni atributo.

`RTFPar` Esta clase es una abstracción de la palabra de control `\par`. Ésta no puede tener parámetro numérico, así que la clase no añade ningún método ni atributo.

`RTFSect` Esta clase es una abstracción de la palabra de control `\sect`. Ésta no puede tener parámetro numérico, así que la clase no añade ningún método ni atributo.

`RTFPage` Esta clase es una abstracción de la palabra de control `\page`. Ésta no puede tener parámetro numérico, así que la clase no añade ningún método ni atributo.

`RTFColumn` Esta clase es una abstracción de la palabra de control `\column`. Ésta no puede tener parámetro numérico, así que la clase no añade ningún método ni atributo.

`RTFLine` Esta clase es una abstracción de la palabra de control `\line`. Ésta no puede tener parámetro numérico, así que la clase no añade ningún método ni atributo.

`RTFStar` Esta clase es una abstracción del símbolo de control `\*`. Ésta no puede tener parámetro numérico, así que la clase no añade ningún método ni atributo.

`RTFSpecialChar` Esta clase almacena palabras o símbolos de control que representen un carácter especial, como `\tab`, `\emdash`, `\endash`, `\lquote`, `\rquote`, `\ldblquote`, `\rdblquote`, `\~`, `\-`, o `\_`. Tan solo añade los métodos `{get,set}Content()` y el atributo `rtfCtrl` para consultar y establecer el contenido exacto del nodo, como de costumbre.

`RTFAbstractStyle` Esta clase es virtual, y simplemente es una unión conceptual de todas las clases de estilos de RTF. Añade el atributo `name` y los métodos `getName()` y `setName()`, para acceder a él.

`RTFCharStyle` Esta clase guarda una referencia a un estilo de carácter. No añade métodos ni atributos adicionales.

`RTFParStyle` Esta clase es una abstracción de una referencia a un estilo de párrafo. No añade métodos ni atributos adicionales.

`RTFSectStyle` Esta clase almacena una referencia a un estilo de apartado. No añade métodos ni atributos adicionales.

`RTFShape` Esta clase guarda un dibujo vectorial guardado a base de primitivas. Es simplemente un nodo de paso que apadrina los entornos que empiezan con las palabras de control \shppict, \noshppict y \shp. Por tanto, estos objetos pueden ignorarse fácilmente, y también puede establecerse un modo especial para recorrer las figuras.

A.5.2. Adición de nuevas clases

En el caso de añadir algún nodo a esta jerarquía, se tendría que añadir un nuevo método a *todas* las clases visitantes que quisieran recorrer un árbol con el nuevo nodo. Tendría que tenerse en cuenta la posible paternidad del nuevo tipo de nodo. En caso afirmativo, habría que hacerlo heredar no sólo de `RTFContents` o una de sus clases derivadas, sino también de `RTFParent`.

A.6. La clase `RTFVisitor` y sus herederas

La clase `RTFVisitor` es la clase base de todos los visitantes de los objetos de tipo `RTFTree`. Define los métodos que tiene que haber definidos para todas las clases de visitantes, y el atributo `tree`, de tipo `RTFTree*`, para guardar el árbol a visitar.

Los métodos definidos, como se dijo anteriormente, son uno por cada clase heredada de `RTFContents`, que visita a un nodo de la clase en cuestión. La forma general es

```
virtual void visitRTFClass (RTFClass *);
```

donde `RTFClass` es alguna de las clases de la jerarquía `RTFContents`. Estos métodos son los responsables de procesar el nodo que se le pasa como parámetro, actualizar el estado interno del objeto visitante si es necesario, y de *recorrer los hijos* del nodo si procede, con llamadas sucesivas al método `visit()`. Poner la responsabilidad en manos de los visitantes tiene la ventaja de que éstos pueden decidir si procesar a sus hijos en su totalidad, en parte, o nada en absoluto. Con este esquema se tiene la ventaja adicional de reducir el recorrido del árbol a una sola llamada al método `visit` del nodo raíz, ya que el resto se visitará recursivamente.

A.6.1. Nuevos visitantes

Crear nuevos visitantes es trivial: sólo tenemos que crear una clase heredada de `RTFVisitor` (o cualquier otra clase de la jerarquía), y rellenar los métodos que procesan los diferentes nodos. Los posibles atributos internos para guardar el estado interno del visitante los podemos declarar como protegidos de la nueva clase. Si hay nodos que queremos ignorar, podemos simplemente no redefinir los métodos que se encargan de ellos. De hecho, todos los métodos de la clase `RTFVisitor` están definidos vacíos justamente por esta razón. La única excepción a esta regla son los métodos que visitan nodos con hijos, que por defecto están definidos para que los visiten.

A.6.2. Nuevos nodos en `RTFContents`

Añadir nuevos tipos de nodo a la jerarquía `RTFContents` hace que tengamos que modificar los visitantes, justamente para que éstos sean capaces de reconocer los nodos y procesarlos adecuadamente. En teoría, tendremos que redefinir y modificar todos los visitantes. Como puede imaginarse por estas afirmaciones, el propio «Design Patterns» daba como restricciones para un uso eficiente que no cambiara con frecuencia la estructura del árbol a recorrer. Por tanto, debe comprenderse que la operación es en cierta manera inesperada y engorrosa. Dicho lo dicho, hay una técnica que consigue minimizar estas modificaciones, que es dar una acción por defecto al nuevo método al añadirlo a la clase `RTFVisitor`. Las acciones por defecto recomendadas son ignorar el nodo o visitar sus hijos, que son las acciones menos susceptibles de cambiar entre aplicaciones. En algunos casos, la acción por defecto será suficiente y no tendremos que hacer posteriores modificaciones.

A.7. Iterador STL

Para facilitar el recorrido de los documentos RTF a los programadores potenciales de C++, primero se diseñó e implementó un iterador tipo STL. La idea detrás de este iterador era recorrer la estructura linealmente, mediante un «puntero», e ir aplicando a cada nodo encontrado su función de visita, mediante el objeto visitante. Después de algunas pruebas y diversos problemas, se llegó a la conclusión de que no era viable mantener un recorrido de ese modo, porque:

- La propia estructura en la que se guardan los documentos RTF es un árbol, que es una estructura recursiva, no lineal.
- El cambio de profundidad en el recorrido era harto difícil y engorroso de comunicar y almacenar.
- La posibilidad de saltar subárboles enteros era una tarea posible, pero tremendamente incómoda y producía programas ilegibles.

Por todo ello, se descartó después de asegurarnos de que la otra solución era mejor en todos los sentidos, y encima hacía los visitantes incompatibles entre las dos soluciones.

La clase del iterador era `RTFTree::iterator`. Ésta da los servicios de un iterador STL de recorrido hacia adelante (*forward iterator*), es decir, que tiene sobrecargados los operadores `==`, `!=`, `++` (postfijo) y `*`.

Por su parte, la clase `RTFTree` necesitaba tener disponibles unos métodos `begin()` y `end()`, que devolvieran iteradores del tipo `RTFTree::iterator` que apuntaran al principio y al final del árbol, respectivamente, y se pudieran utilizar como los iteradores STL.

Para hacer su trabajo, el iterador `RTFTree::iterator` se apoyaba en la suposición de que los objetos tienen métodos llamados `begin()` y `end()`, que devuelven iteradores del tipo `RTFChildren::iterator`. Estos iteradores sirven para recorrer los hijos de una clase heredera de `RTFContents`, y pueden redefinirse para cualquier nuevo heredero de ésta, para hacer posible el recorrido del iterador `RTFTree::iterator` por los hijos de los nodos de cualquier hipotético nuevo tipo de nodo que se cree.

Para añadir nuevos tipos de nodos al árbol `RTFTree` que pudieran recorrerse sin problemas con el `RTFTree::iterator`, tanto las nuevas clases de nodo como los iteradores de los hijos de éstas tenían que cumplir ciertas condiciones:

- Tener un método `addNode` que añadiera un nuevo hijo a la colección del nodo.
- Es recomendable, aunque no necesario, sobrecargar el método `infoString` de tal forma que se pueda saber fácilmente a qué tipo pertenecen los objetos nuevos.

A.8. La clase `RTFParser`

La clase `RTFParser` es un «constructor» de árboles RTF. Es un analizador sintáctico, que se apoya en `RTFLexer`, un analizador léxico hecho con *flex++*. Su función es, dado un flujo de datos de entrada a un documento en formato RTF, construir un árbol que lo represente, y devolver un puntero al citado árbol. Su único método público es `newTree`, que está declarado de la siguiente manera:

```
RTFTree *newTree (istream *stm);
```

El análisis sintáctico de los documentos RTF se puede dividir en dos partes, el análisis de la cabecera y el análisis del texto del documento. Esta división viene por la condición relativamente estricta de los elementos de la cabecera. Para esta parte del documento puede decirse que hay una cierta gramática, aunque, como siempre, es recomendable afrontar en análisis previendo cambios en las convenciones y símbolos inesperados. La mejor opción es estos casos es simplemente ignorar lo que nos convenga e intentar seguir a toda costa.

La organización interna de la clase `RTFParser` es como sigue: por un lado, están los métodos de lectura de la cabecera; aquí están `readHead ()`, `readFonttbl ()`, `readFiletbl ()`, `readColortbl ()`, `readStylesheet ()`, `readListtable ()`, `readRevtbl ()`, `readInfo ()` y `readDocfmt ()`. Por otro, están los métodos para modularizar la lectura del texto del documento, que se lleva a cabo desde el propio `newTree()`; todos estos métodos tienen como nombre `dispatch`*`SIMB`*`()`, donde *`SIMB`* es uno de los símbolos devueltos por el analizador léxico, como `TEXT`, `CTRLW`, `CCURLY` o `PARD`. En la Figura A.1 se muestra un esquema de las llamadas hechas para analizar un documento RTF y construir su árbol correspondiente.

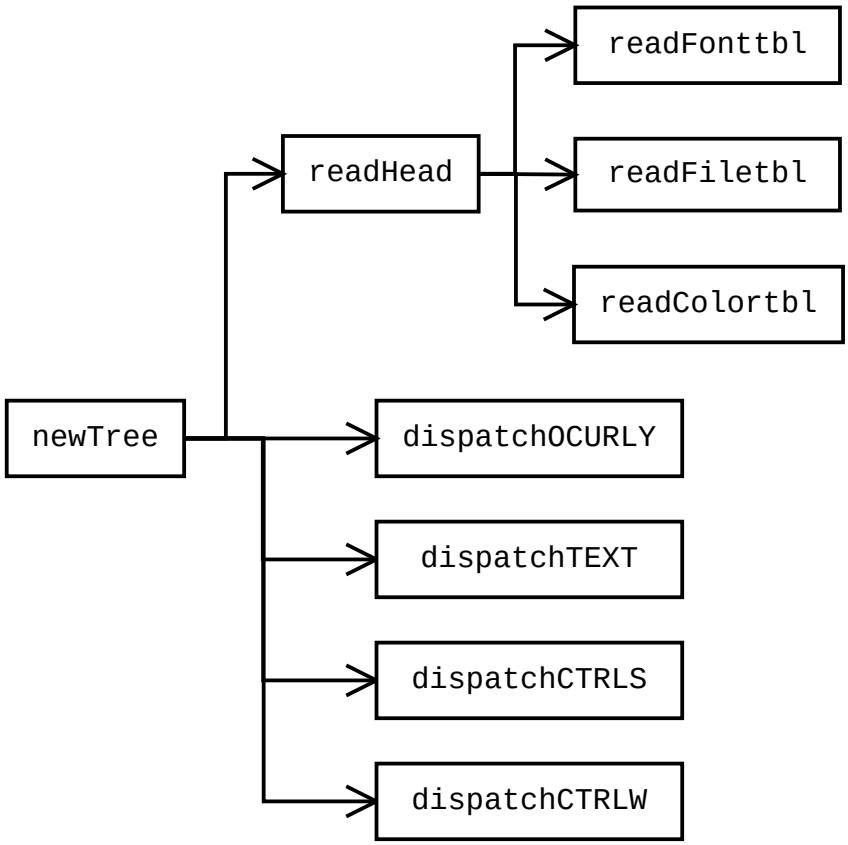


Figura A.1: Esquema de llamadas del método `newTree`

A.8.1. La clase `RTFLexer`

Los símbolos son reconocidos y devueltos por un analizador léxico construido con el *flex++*. Este analizador es una clase llamada `RTFLexer`, cuyo método principal es `yylex()`. Éste va devolviendo los símbolos reconocidos al llamador. Para consultar el valor de los símbolos, en caso de haberlo, se puede llamar al método `tokenValue()`, que devuelve un objeto de tipo `RTFData`[3]. Además, tiene otro método público, `getLineno()`, que devuelve el número de línea actual. Esta información es muy útil cuando se produce un error, para localizarlo rápidamente.

A.9. Fichero de configuración

La clase `RTFParser` tiene un fichero de configuración, llamado por convención `rtf.words`, donde se especifican los tipos a los que pertenecen las diferentes palabras de control. Se busca en el directorio raíz del usuario, bajo el nombre `.rtf.words`, y en los directorios `/usr/local/share/rthc`, `/usr/share/rthc` y `/etc` bajo su nombre normal. El tipo de palabra de control puede ser:

1. Palabras de control independientes.
2. Atributos de carácter.
3. Atributos de párrafo.
4. Atributos de apartado.

En este caso, el fichero de configuración es de formato libre, es decir, que los espacios sólo sirven para separar los diferentes símbolos. Los saltos de línea (su cantidad y posición) no es determinante, simplemente se considera a éstos como espacio en blanco. El único sentido especial que tienen es marcar dónde terminan los comentarios, que son los típicos de UNIX, que empiezan por el caracter almohadilla (`#`), y siguen hasta el final de la línea.

[3]Recordemos que, como se dijo en el apartado A.3.3, toda la información asociada a un elemento léxico RTF puede representarse en un objeto de tipo `RTFData`.

A.9.1. Análisis léxico y sintáctico

A pesar de que se puede decir que conceptualmente hay símbolos diferentes, la discriminación entre éstos realmente se hace en un analizador sintáctico. Sin embargo, todo es transparente para el cliente de la clase `RTFWordsLexer`, ya que, aunque el método `yylex()` simplemente se dedica a ignorar los comentarios y a devolver palabras[4], el analizador sintáctico está empotrado en la misma clase. Tanto es así, que el único método público de la clase es `fillTree()`, que se encarga de llenar un árbol `RTFTree` dado como parámetro con los datos encontrados. Por tanto, a pesar de que el nombre de la clase es `RTFWordsLexer`, no se puede considerar que sea solamente un analizador léxico.

Dicho lo dicho, la descripción léxica del fichero de configuración es:

RTFCTRL Palabra que indica que, a partir de ahora, las palabras recibidas deberán ser interpretadas como nuevas palabras de control independientes.

RTFCHARATTRIBUTE Palabra que indica el comienzo (o continuación) de la lista de palabras de control de atributos de carácter.

RTFPARATTRIBUTE Palabra que indica el comienzo de la lista de atributos de párrafo.

RTFSECTATTRIBUTE Ídem de apartado.

CONTENT Cualquier otra cosa

Una vez obtenidos los símbolos *virtuales*, la gramática del fichero de configuración puede resumirse en que es válida cualquier combinación de símbolos es válida. Entonces, sólo queda describir el fichero a nivel semántico: por defecto (aunque no se recomienda usar esta característica), el analizador está en modo *palabra de control independiente*. Es decir, toda palabra encontrada (símbolo CONTENT) se interpretará como nuevas palabras de control independientes para añadir al árbol. Cuando se encuentra un símbolo como RTFCHARATTRIBUTE, se cambia el modo del analizador, y, a partir de ese mismo momento, todas las palabras que se encuentren se interpretarán como nuevos atributos de carácter, de párrafo, de apartado, u otra vez palabra de control independiente, si aparece el símbolo RTFCTRL.

Como unas reglas de sangrado adecuadas, se puede conseguir un fichero de configuración bastante legible, y el analizador es tremendamente simple de implementar. Como esos eran los únicos requisitos del fichero de configuración, se implementó de esta manera.

A.10. Guía de modificación

A la hora de modificar *libFreeRTF* ayudará tener en cuenta ciertos detalles que se explicarán en este apartado, como las convenciones en los nombres de las variables o al escribir el programa, o la estructura de directorios.

Lo más importante de todo es que tanto los nombres de las variables, métodos, funciones y clases, además de los comentarios, están en inglés, por ser éste el idioma más extendido entre la comunidad de programadores, y ser uno de los objetivos del proyecto aportar a la comunidad de programadores un paquete de clases fácil de utilizar. En castellano, todo el trabajo habría sido en balde, excepto para unos pocos (comparativamente) programadores españoles interesados en *libFreeRTF*.

A.10.1. Formato de los ficheros

Para empezar, todos los ficheros del programa están escritos con tabuladores de 8 espacios, usando el estilo de sangrado de Kernighan & Ritchie, es decir, poniendo todas las llaves de comienzo de bloque en la misma línea que el bloque que comienzan, excepto las funciones y las clases. Un ejemplo de este estilo es:

```
int main ()
{
        int     foo;
        cin >> foo;
        if (foo > 5) {
                foo += 4;
                cout << "La variable foo ha quedado como " << foo << endl;
```

[4] En este contexto, definimos «palabras» como caracteres distintos de blanco, separados por éstos.

```
        } else
                cout << "Se ha elegido un número menor o igual a 5\n";

        return 0;
}
```

En cuanto al formato del texto se refiere, está escrito todo en UNIX, así que los saltos de línea son caracteres *linefeed* únicos, y no *carriagereturn-linefeed*. Esto puede traer algunos problemas abriendo los ficheros en programas como el Bloc de notas de Microsoft, que supone que todos los ficheros tienen los saltos de línea al estilo de MS-DOS/Windows, y no lo interpreta bien (lo representa como un fichero de una sola e inmensa línea, con cuadrados negros en los finales de línea).

A.10.2. Variables

Para nombrar las variables, por otro lado, se ha seguido la convención del lenguaje Objective-C, por no haber ninguna convención clara en C++, y ser la del primer lenguaje bastante legible y clara. Esta convención consiste en nombrar todas las variables, funciones y métodos con minúsculas, resaltando las iniciales de la segunda palabra y sucesivas con letra mayúscula, de esta manera:

```
myVariable = infoString();
```

Por su parte, los nombres de clases y otros tipos y estructuras resaltan *todas* las iniciales con letra mayúscula. Un ejemplo de la convención, por tanto, es:

```
class SectionNumber;
```

Además, por ser los formatos RTF y HTML el centro de *libFreeRTF* y el proyecto RTHC, se han dejado sus nombres siempre en mayúsculas. Así nos encontramos clases como `RTFHTMLVisitor`, `RTFTree` o `RTFText`.

En lo que respecta a las palabras usadas para los nombres, vale la pena comentar que se ha seguido por todo el programa una convención más en el nombre de las funciones: siempre que se han escrito nuevos métodos para acceder a ciertos datos internos de los objetos, el nombre de éstos siempre han empezado con los tres caracteres `get` (como en `getParent`); siempre que los métodos han servido para establecer el valor de algún atributo interno del objeto, se han usado nombres que comenzaban con `set` (como en `setParent`).

Por último, para evitar nombres de variables excesivamente largos, por todo el programa se abrevia la palabra `stream` (flujo de datos) a `stm`.

A.10.3. Ficheros

Los nombres de los ficheros, por otro lado, se ponen siempre en minúsculas, siguiendo la convención de UNIX, excepto cuando los nombres de éstos coinciden con nombres de clases, en cuyo caso el nombre del fichero coincide en uso de mayúsculas y minúsculas con el nombre de la clase. Así, tenemos ficheros llamados `rtfdefs.h`, `RTFTree.cc`, `rtf.ll`, `RTFPlainTextVisitor.h`, `RTFHTMLVisitor.cc` y `tokens.h`.

Los ficheros están repartidos de directorios de la siguiente manera:

- En el directorio `RTFTree` están los ficheros principales pertenecientes a *libFreeRTF*. El subdirectorio `util` contiene un pequeño programa para modificar los valores compilados por defecto en la aplicación.
- En el directorio `RTFPlainTextVisitor` están los fuentes del visitante conversor a texto sin formato.
- En `RTFDebugVisitor` está la clase visitante del mismo nombre, que sirve para depurar árboles RTF.
- En `RTFHTMLVisitor` está el visitante del mismo nombre, y el resto de la aplicación RTHC, excepto el fichero principal (`main.cc`, en el raíz). En el subdirectorio `util` hay un programa para modificar los valores por defecto compilados en el programa.
- En `GetOpt` está la modificación de la clase de `libg++`, que lee opciones largas.
- En `PathOpen` está la clase del mismo nombre, que permite buscar ficheros en una lista dada de directorios.

B Manual de usuario de RTHC

RTHC es un conversor de documentos en formato RTF a páginas HTML. Las siglas significan «RTF To HTML Converter», o sea, «Conversor de RTF a HTML». Su objetivo es ser extensible, fácilmente modificable y flexible hasta cierto punto. Sus prestaciones son:

- Convierte por defecto todos los caracteres especiales RTF en sus equivalentes según la tabla de caracteres utilizada, aunque se puede personalizar mediante un sencillo fichero de configuración.
- Reconoce la estructura lógica del documento, si la tiene, y la conserva en las páginas HTML resultantes.
- Parte el documento original en varios ficheros, para facilitar el manejo del HTML producido.
- Permite modificar al gusto del usuario el aspecto de las páginas finales mediante sencillas plantillas.
- Las plantillas no limitan al usuario a simples páginas HTML: permiten construir programas en Perl, páginas PHP, y cualquier cosa que se le ocurra al usuario, al estar basado en simples sustituciones.

Dado que el programa RTHC es principalmente una implementación básica de un analizador RTF, que no intenta manejarlo todo desde la primera versión, algunas de las características menos comunes de RTF se ignoran o no se tratan completamente. Entre las entidades RTF que RTHC comprende están:

- Texto simple
- Caracteres especiales
- Caracteres de control
- Palabras de control independientes
- Atributos de carácter, párrafo y apartado
- Tablas simples
- Referencias cruzadas

Entre las entidades RTF todavía no tratadas por RTHC están:

- Tablas complicadas
- Imágenes (mapas de bits)
- Dibujos (diagramas vectoriales)
- Objetos incrustados

B.1. Opciones

El formato de llamada a `rthc` es:

`rthc [`*`opciones`*`] [doc.rtf [raíz-nombres-salida]]`

El significado de las opciones de línea de órdenes son:

`--help` Muestra en pantalla la ayuda.

`--with-index` Crea un fichero de índices en un fichero. El nombre de éste se haya sumando a la raíz la terminación `-toc.html`. El índice incluye las hojas de estilo especificadas por la opción `--use-css`.

`--with-jsindex` Crea un índice flotante en un fichero, para que se cargue automáticamente al cargar cualquiera de los ficheros de contenido. El nombre de éste se haya sumando a la raíz la terminación `-jsindex.html`. El índice incluye las hojas de estilo especificadas por la opción `--use-css`.

`--use-css=1.css,2.css` Usa las hojas de estilo especificadas en las páginas HTML finales.

`--cut-depth=d` Profundidad de corte del documento. Por defecto, vale 0, que significa que el documento final queda en un solo fichero.

`--with-template=t` Plantilla de salida usada para producir las páginas HTML correspondientes.

`--config-file=c` Especifica explícitamente qué fichero de configuración usar.

`--use-stdout` Escribe todo el HTML producido en el flujo básico de salida (`stdout`), en vez de en uno o más ficheros. Esta opción es incompatible con `--cutDepth` y `--with-index`. Si se especifican estas opciones, se ignorarán y se dará preferencia a `--use-stdout`.

El nombre del documento RTF puede especificarse en cualquier lugar, aunque se recomienda que se sigan las buenas costumbres de UNIX, y se especifique después de las opciones, si hay alguna. Si no se especifica el nombre, por defecto se intenta traducir el fichero `test.rtf` en el directorio actual. Si se especifica, también se puede fijar qué raíz se utilizará para construir los nombres de las páginas de salida. Estos nombres se componen de la raíz, un guión, y el apartado cuyo texto contienen. Si ésta no se escribe explícitamente, se utiliza como raíz la serie de caracteres «`index`».

B.2. Ficheros de configuración

El fichero de configuración de `rthc` se llama `rthcrc`. En él se pueden personalizar las traducciones de los caracteres especiales encontrados, y las traducciones a HTML de las diferentes palabras de control, ya sean éstas independientes o de atributo. Por si fuera poco, al depender `rthc` de *libFreeRTF*, también se puede utilizar el fichero `rtf.words`, de configuración de la clase `RTFParser`, para adaptar más aún la salida producida por el conversor. El fichero se busca en los directorios `/usr/local/share/rthc`, `/usr/share/rthc` y `/etc` bajo su nombre normal, y en el directorio raíz del usuario bajo el nombre `.rthcrc`.

B.2.1. Formato

Las reglas para interpretar el fichero de configuración son las siguientes:

- Las almohadillas y el resto de los caracteres que la siguen, hasta el fin de la línea, se consideran comentarios y por tanto se ignoran.
- Cada línea se considera una orden de traducción diferente.
- Las traducciones de caracteres especiales deben ser dos dígitos hexadecimales, juntos, algo de espacio para separar, y contenido HTML.
- Las traducciones de palabras de control deben ser la palabra en sí, sin parámetro numérico, algo de espacio para separar y contenido HTML. Optativamente, se puede añadir al final más espacio en blanco y más contenido HTML.

Hay que tener en cuenta que el «contenido HTML» puede ser una serie de caracteres que no incluya espacios en blanco, o una serie de caracteres entre comillas dobles. En este caso, se pueden incluir comillas dobles literales con los caracteres \"
, y barras invertidas literales con los caracteres \\. Para verlo todo más claro, se adjunta un ejemplo de fichero de configuración `rthcrc`:

```
aa      &mientidad;
ef      "Caracter normal"
f3      |       # Comentario. Me ignorarán
\nowidctlpar    "Paso por un \\nowidctlpar"
\i      <i>     </i>
\b      <b>     # Omitimos, por obvio
\img    "<img src=\"foo.jpeg\">"    # Siempre la misma imagen
```

B.2.2. Interpretación

Para interpretar el contenido del fichero de configuración, se utilizan las siguientes reglas:

- En las traducciones de caracteres especiales, el contenido HTML se la traducción de éstos. Cada vez que se encuentre en el texto un caracter especial, se escribirá en la salida la traducción dada en el fichero. Por ejemplo, con el fichero anterior, cada vez que se encuentre en el documento original los caracteres «\'aa», en la página HTML de salida se escribirá «&mientidad;»
- En las traducciones de palabras de control podemos distinguir dos casos:
  1. Si la palabra se considera un atributo, el primer (y obligatorio) contenido HTML será la traducción a poner al principio del «entorno de validez» de la palabra. El segundo (y optativo) contenido HTML será la traducción del final del entorno de validez. En caso de que no se especifique el segundo, el conversor intentará inducirlo del primero.
  2. Si la palabra se considera independiente, el primer contenido HTML será la traducción a producir cada vez que aparezca la palabra. El segundo, si existe, se ignorará.

que será interpretado como el final del «entorno de validez» de la palabra (esto, naturalmente, sólo tiene sentido si la palabra de control se considera un atributo y no una palabra independiente).

B.3. Plantillas de salida

Aparte de los ficheros de configuración que permiten modificar el comportamiento del programa en sí, se pueden usar «plantillas de salida», que no son más que documentos HTML con referencias a «variables» RTHC. De esta manera se consigue personalizar completamente el aspecto de las páginas HTML resultantes, ya que lo único que se fuerza es cómo se traduce el texto en sí. Por otro lado, esto también puede controlarse hasta cierto punto con los ficheros comentados anteriormente.

Estas variables tienen el mismo aspecto que las de Perl o PHP, es decir, un signo dólar y el nombre de la variable. Para no confundir con posibles variables definidas por el usuario, por ser la plantilla una página PHP o incluso un programa en Perl, todos los nombres de las que utiliza RTHC empiezan por los caracteres «`rthc_`».

La lista de variables que se pueden utilizar es:

`$rthc_contents` El texto del fichero. Si el documento original se va a partir en varios ficheros, el texto que le corresponda a cada página.

`$rthc_css` Las hojas de estilo a usar. Produce una línea de la forma «`<link href="foo.cssrel="stylesheet»`» por cada hoja.

`$rthc_title` El título del documento.

`$rthc_author` El nombre del autor del documento.

`$rthc_subject` El tema del documento.

`$rthc_operator` El nombre de la persona que escribió el documento.

`$rthc_prev_file` El nombre del anterior fichero.

`$rthc_curr_file` El nombre del fichero actual.

`$rthc_next_file` El nombre del siguiente fichero.

`$rthc_index_file` El nombre del fichero de índice, o el primero, si no hay.

Además de estas, se puede hacer referencias a cualquier otra: cualquier variable de la forma `$rthc_`*`algo`* será interpretada como una variable a sustituir por RTHC. Si no se conoce, se buscará en la hoja de información, y se sustituirá la variable por el contenido de la entrada *`algo`*. Es decir, si se encuentra una variable `$rthc_createim`, ésta se sustituirá por el contenido de la entrada `createim` de la hoja de información.

C Guía de referencia del formato RTF

El «Formato de Texto Enriquecido» de Microsoft («Rich Text Format», o RTF) está pensado para intercambiar documentos entre programas de diferentes sistemas operativos y máquinas. Aunque es así en teoría, en la práctica el RTF es bastante difícil de entender, en parte por su confusa especificación. Por tanto, este documento se ha escrito con el objetivo de servir de guía para construir analizadores de RTF 95 y 97, arrojando algo de luz sobre los puntos más oscuros y conflictivos de este lenguaje. No se pretende reemplazar a la especificación oficial de Microsoft, que *siempre es la autoridad definitiva*, sino ser un apoyo más para el programador que tenga que vérselas con este formato.

C.1. Metodología de trabajo

Dada la naturaleza y el objetivo de esta documentación, se ha partido de la documentación oficial de Microsoft y de ficheros de prueba. Teniendo en cuenta que la pretensión es formar un puente entre los formatos cerrados y los abiertos, ya que aquéllos son los más ampliamente usados en los omnipresentes sistemas cerrados, los ficheros de prueba utilizados se crearon principalmente con el Microsoft Word, el programa de redacción de textos más utilizado, responsable de la mayor parte de los documentos RTF existentes. Como es natural, cuando se tuvo oportunidad de crear documentos con otros programas, se hizo, para asegurar la robustez del reconocimiento.

Los orígenes de parte de esta documentación son empíricos, por rellenar muchas de las lagunas encontradas en la documentación. En estos casos, la única fuente de referencia fue la creación de documentos específicos de prueba y la inspección de los resultados, para especular sobre el significado, la posición y la necesidad de la existencia del contenido encontrado.

C.2. Conceptos generales

Un documento en formato RTF se compone de dos partes: cabecera y cuerpo.

En la cabecera se incluyen todos los datos sobre el documento en conjunto, como el nombre del autor, la lista de tipos de letra usados, la hoja de estilo, etc. Tiene muchos apartados, cada uno con su propio formato. Eso dificulta la construcción de reglas sintácticas que analicen la cabecera, porque se necesitan varias reglas por uno de estos apartados.

En el cuerpo va todo el texto y el resto de los objetos que compongan el contenido del documento. Tiene un formato bastante regular, por lo que se puede analizar con unas pocas reglas. La dificultad aquí estriba en interpretar todo el contenido que nos podemos encontrar, es decir, en escribir las acciones a ejecutar al encontrarnos con cada elemento.

C.3. Elementos léxicos

Léxicamente, el lenguaje RTF tiene los siguientes símbolos:

Llaves («{}») marcan el principio y final de los llamados *grupos*.

Palabras de control símbolos que nos dan información sobre el formato del contenido del fichero RTF. Empiezan con la barra invertida («\») a la que siguen una o más letras. Después, optativamente, puede haber un parámetro numérico, que consiste en un menos («-») optativo y una o más cifras.

Símbolos de control símbolos que por lo general representan caracteres especiales, no representables de otra forma. Son una barra invertida («\») seguida de un carácter *no* alfanumérico.

Contenido caracteres alfanuméricos, espacios, etc. Es decir, cualquier carácter que no sea un salto de línea o retorno de carro, llave (abierta o cerrada) o una barra invertida.

Contenido binario cualquier otro carácter, incluso no imprimible. Puede formar parte de un fichero RTF por describir imágenes u otros objetos incrustados.

Es bastante fácil hacer un analizador léxico con una herramienta con Lex, aunque en el programa de ejemplo de Microsoft se haga todo a mano.

Es importante resaltar una escandalosa omisión por parte de Microsoft en la documentación oficial, que consiste en no hacer notar que los retornos de carro y saltos de línea no significan *absolutamente nada*, es decir, que *ni siquiera separan palabras*. Por tanto, cada vez que un analizador léxico los encuentre, tiene que seguir como si no hubiese encontrado ningún caracter.

C.4. Estructura sintáctica (gramática del RTF)

La descripción sintáctica del formato RTF es lo menos claro de toda la especificación. Tanto es así, que la mentalidad con la que hay que analizar un documento de texto enriquecido es recorrerlo preparados para ignorarlo todo, pero «recogiendo las perlas» que veamos, y no estar preparados para leer un documento RTF e ignorar lo que no entendamos.

Además, esta gramática está concebida para recorrerla linealmente, sin una estructura muy elaborada. En ciertas partes de la cabecera, por ejemplo, se necesitan dos símbolos adelantados para analizarse, así que el Yacc (o, a esos efectos, el Bison) no nos pueden resolver el problema bien. Si aún así se quiere utilizar un analizador que sólo lea un símbolo por adelantado, como los creados por Yacc o Bison, podemos fácilmente cambiar algunos de ellos para convertir la gramática en una que sólo necesite un símbolo por adelantado, o hacer una gramática ultra simple, que no distinga entre palabras de control, y, efectivamente, analizar sintácticamente el documento a mano.

Estos cambios consisten en asignar símbolos especiales a las palabras de control `\colortbl`, `\filetbl`, `\stylesheet`, `\listtable`, `\revtbl` e `\info`, y añadir al principio de éstas una llave abierta, es decir, convertirlas en `{\colortbl`, `{\filetbl`, `{\stylesheet`, `{\listtable`, `{\revtbl` e `{\info`.

Dicho lo dicho, esto serviría para leer ficheros RTF bien estructurados, que puede no ser suficiente. Esto puede dar problemas si encontramos documentos que no estén perfectamente bien estructurados, o documentos de una revisión futura de RTF, que añadan campos a la cabecera, etc. Por tanto, la estrategia propuesta es tener métodos que entiendan todas las partes de la cabecera, y otro «principal» que los active cuando se encuentre una marca de que empieza una de esas partes.

C.5. Lectura de la cabecera

La cabecera es una parte importante del fichero, porque contiene información que nos ayudará a interpretar correctamente el texto del documento. La información guardada en la cabecera comprende:

- Lista de tipos de letra a utilizar. Asocia un número de tipo de letra a una familia y un nombre, entre otros datos.
- Lista de ficheros externos utilizados.
- Lista de colores utilizados en el documento. Es una lista de colores definidos en RGB, que se suponen numerados empezando por 0.
- Hoja de estilo del documento. Refleja los estilos (de carácter, párrafo o sección) definidos para el documento, junto con una descripción consistente en el tipo de letra utilizado, el nombre y tipo del estilo y otros datos.
- Tabla de listas. Indica los tipos de listas (series de elementos numerados o no) definidos para el documento.
- Tabla de revisiones. Contiene información sobre las revisiones del documento.

Es importante tener en cuenta que, exceptuando la lista de tipos de letra, estos apartados de la cabecera son todos optativos, con lo que ésta podría consistir sólo en una lista de tipos de letra. Además, y teniendo en cuenta los cambios en la cabecera entre RTF 95 y RTF 97, uno debería estar preparado para que en versiones posteriores del formato RTF se añadieran nuevos apartados a la cabecera, y *no necesariamente después de todos estos*. Es algo sorprendente, ya que, aunque son optativos, uno espera que los apartados aparezcan en un orden determinado y las adiciones sean al final. Como aviso adicional se aconseja que, a pesar de que en la descripción de la gramática del formato RTF se especifica el orden en el que deben aparecer los apartados, cualquier analizador esté preparado para leer documentos que hagan caso omiso de esta indicación, de tal forma que esperen las distintas partes de la cabecera en un orden completamente arbitrario.

Otro detalle importante que no se resuelve en la documentación oficial es, dado que no hay una marca especial para indicarlo, cómo saber que se ha terminado la cabecera: debido a que casi todos los apartados son optativos, no podemos asegurar cuándo ha terminado exactamente, es decir, deberíamos leer varios símbolos por adelantado para saber si lo siguiente es una parte que sabemos analizar. Pero aún resuelto este problema, todavía queda la incertidumbre de si lo siguiente, si no lo reconocemos como parte de la cabecera, es una extensión de ésta o si es el principio del contenido. Se ha encontrado un caso en el que lo primero del texto es un grupo, así que no podemos, estrictamente hablando, pensar que la cabecera son un conjunto de grupos sucesivos. Aún así, éste parece el comportamiento más lógico y más conservador. Otro, más complicado, sería analizar el contenido del grupo, y en caso de reconocerlo como contenido, empezar el reconocimiento del cuerpo, y, en caso contrario, tomarlo como una extensión de la cabecera. Con este comportamiento, si lo primero fuera una extensión del *cuerpo*, poco importaría, porque tendríamos que ignorarla y seguir de todas maneras.

C.6. Hoja de información

Entre la cabecera y el cuerpo del documento están la hoja de información y un conjunto de palabras reservadas (incluyendo ciertos grupos) que describen las propiedades del documento.

La hoja de información contiene algunos datos generales sobre el documentos, como la fecha de última modificación, autor, título del documento, tema, redactor, compañía, etc. Algo importante a la hora de analizar la hoja de información es que tanto Word 6 como Word 97 actúan de forma (al menos teóricamente) contraria a la especificación oficial. En ésta se dice que la hoja de información es un conjunto de palabras reservadas y texto rodeados por llaves y palabras sueltas. Exactamente define `<info>` como:

```
<info> '{' <title>? & <subject>? & <author>? & ...
       ... & <comment>? & \version? & <doccomm>? & \vern? & ...
       ... \nofchars? & \id? '}'
```

donde los símbolos no terminales `<title>`, `<subject>`, `<author>`, etc. están definidos *siempre* como un par de llaves encerrando a una palabra reservada seguida de texto o una descripción de tiempo, que se hace mediante un conjunto de palabras reservadas especiales.

En realidad, Word encierra *toda* la información (incluidas las palabras reservadas sueltas como `\version`) entre llaves. No se descarta que en RTF 2000 empiece a cumplirse esta especificación, pero las versiones anteriores de Word actúan contra ésta.

C.7. Caracteres especiales

Los caracteres especiales de RTF se usan para representar caracteres no ASCII, es decir, no representables con 7 bits. Son de la forma \'*hh*, donde *hh* son dos cifras hexadecimales que determinan el carácter correspondiente.

Una vez tenida la posición del carácter que queremos, debemos comprobar qué tabla de caracteres se está usando. Dependiendo de ésta, la posición representará un carácter u otro. Esto conlleva que, efectivamente, cualquier analizador de RTF tenga que conocer todas las tablas de caracteres de las diferentes plataformas para asegurarse de que sabe interpretar correctamente todos los documentos RTF posibles.

C.8. Símbolos de control

Son parecidos a las palabras de control, sólo que, después de la barra invertida, sólo puede aparecer un carácter no alfanumérico. Por lo general, indican acciones especiales. Los símbolos de control más comunes son:

* Indica que la palabra de control que sigue es una extensión. Si aparece al principio de un grupo y no se reconoce la extensión, hay que ignorar *el grupo entero*.

\ Indica un espacio en blanco que no se puede partir.

\- Indica un guión optativo.

_ Indica un guión que no se puede partir.

Al encontrar uno de estos símbolos, podremos tener la necesidad de producir algún tipo de salida, actualizar el estado interno del analizador, o hacer alguna de estas cosas tras comprobar cierta condición. Por ello, lo más seguro es asociar a los símbolos de control en general una acción a ejecutar, y no simplemente una serie de caracteres de representación.

C.9. Estilos

Al igual que en el formato nativo de Word, en RTF hay dos tipos de estilos: de carácter y de párrafo o bloque. Éstos se aplican a *bloques* de texto completos, es decir, forman un párrafo por sí mismos, mientras que aquéllos se aplican a conjunto de caracteres dentro de un párrafo. Por lo general, los estilos de bloque se utilizan para títulos, apartados, etc. (al acabar la aplicación del estilo, comienza obligatoriamente un nuevo párrafo), y los de carácter, para resaltar de alguna manera ciertos caracteres dentro de un párrafo, como por ejemplo para poner letra de terminal y similares.

C.9.1. Declaración

En la hoja de estilo (el entorno que comienza con `\stylesheet`) se especifican todos los estilos, tanto los de bloque como los de carácter, pero de manera diferente: mientras que a los de bloque (los más usuales) se les asigna un identificador de la forma `\s#`, donde `#` es un número, y a los de carácter se les asigna uno de la forma `\cs#`. Es de notar que los estilos de carácter se tratan como una extensión, por lo que lo primero que aparece en el entorno que define a cada uno de éstos es el símbolo de control especial `\*`.

C.9.2. Uso

A la hora de referirse a ellos en el contenido del fichero, tanto en un tipo de estilo como en el otro se incluye tanto el identificador del estilo como una copia de *todos* los atributos que componen el estilo. El final de la aplicación de un estilo de bloque se determina de dos maneras: si hay cambios de estilo (o, en Word 6, siempre), el contenido afectado por el estilo se encierra entre llaves. Si no, la siguiente aparición de la palabra de control `\par` nos marca el final de la aplicación del estilo. Los estilos de carácter siempre se aplican dentro de un grupo (esto es, entre llaves), por lo que se terminan de aplicar al cerrar las llaves correspondientes. Por tanto, para encontrar el texto al que aplicar el estilo *de párrafo* tenemos que hacer las siguientes comprobaciones: si se encuentra primero una llave que una marca de nuevo párrafo, hay que llegar hasta la llave que cierra la que hemos encontrado (que no es necesariamente la siguiente); si se encuentra primero una marca de nuevo párrafo, terminamos ahí la aplicación del estilo de párrafo.

Esta forma de aplicar los estilos produce un gasto tremendo de espacio en disco (y probablemente en memoria) y añade muchas más palabras de control de las necesarias, además de hacer que el identificador de estilo sea superfluo, ya que uno podría saltárselo y el resultado sería el mismo. En la documentación oficial dice que la razón de hacer esto es la compatibilidad con lectores de RTF anteriores. En realidad esta reiteración en la definición es necesaria: dejando la marca del estilo (`\sn`), pero quitando la definición, el propio Microsoft Word 97 reconoció el estilo lógico, pero quitó todo rastro de estilo físico.

Un dato importante es que los estilos que se utilizan por convención para definir la estructura del documento, es decir, «Título 1», «Título 2», etc., se dejan *sin traducir* en el fichero RTF, como es lógico,

y sus nombres son «heading 1», «heading 2», ... Tristemente, algunas implementaciones, como la de StarOffice, no sigue esta convención, y traduce los nombres de estos estilos, haciendo casi imposible la identificación correcta de los apartados del documento.

C.10. Marcadores

Los marcadores RTF son etiquetas que se aplican a una parte del texto, que lo hacen *referenciable*. Estos marcadores tienen un nombre para poder referirse a ellos, compuesto por texto y caracteres especiales (para dar la posibilidad de incluir tildes y otros caracteres no ASCII). Por tanto, en principio los nombres no pueden manejarse como simples series de caracteres. En realidad, uno puede por simplificación guardar el texto y caracteres especiales RTF en un objeto `string`, dado que la mayoría de las veces sólo se utilizará para búsquedas y comparaciones. El inconveniente de esta solución es que para mostrarlo a una persona tendrá que sufrir ciertas modificaciones.

Las palabras de control de los marcadores son dos, `\bkmkstart` y `\bkmkend`. Siempre vienen en la forma

```
{\*\bkmkstart texto}Texto referenciable{\*\bkmkend texto}
```

lo cual sugiere un aspecto algo incómodo de los marcadores, y es que se pueden solapar. Esto complica sobremanera la generación de HTML, entre otros formatos, por no permitir solapamiento de etiquetas. Es decir, que podemos encontrarnos algo como:

```
{\*\bkmkstart etiq1}Texto{\*\bkmkstart etiq2} referenciable
   {\*\bkmkend etiq1}y más texto{\*\bkmkend etiq2}
```

En este caso, el texto referenciable por la etiqueta `etiq1` sería `Texto referenciable`, y el referenciable con la etiqueta `etiq2` sería `referenciable y más texto`. Esto, por supuesto, es perfectamente legal en RTF, aunque no tenga traducción directa a otros formatos de documentos más «ordenados».

C.11. Referencias cruzadas y campos

En realidad, las referencias cruzadas son un caso particular de los «campos». Los campos son trozos del documento que no se escriben directamente, sino que se producen automáticamente de acuerdo a ciertas reglas. Los documentos RTF, aparte de estas reglas para hallar el trozo de documento, guardan la versión más recientemente hallada del campo en cuestión. Así, éstos se dividen en dos partes: las instrucciones y el último resultado calculado. Su estructura es:

```
{\field{\*\fldinst {...}}{\fldrslt ...}}
```

Lo que acompaña a la palabra `\fldrslt` es un grupo RTF (contenido entre llaves) que representa el RTF que habría que mostrar como resultado del campo. Puede estar vacio, en cuyo caso *tampoco* estarán las llaves.

Lo que acompaña a la palabra `\fldinst` son las instrucciones para calcular el valor del campo. Estas instrucciones *no* están documentadas en la especificación oficial, pero se puede hacer una lista probando todas las posibilidades en el Microsoft Word. Básicamente se compone de una texto normal, que indica la «orden» a ejecutar y los parámetros. También aparecen a veces caracteres de escape, pero todavía no se ha descubierto su función.

Las referencias cruzadas se indican mediante la orden `REF`, y recibe como parámetro el nombre del marcador al cual se refiere. Hay que tener en cuenta que las referencias son *siempre* a un marcador. Si uno, en Microsoft Word, hace una referencia a un título, el programa crea automáticamente un marcador, y hace la referencia a éste. Un ejemplo de referencia puede ser:

```
{\field{\*\fldinst {\lang1034  REF mi_marcador \\h }}
{\fldrslt {\lang1034 contenido referenciado}}}
```

C.12. Listas

Las listas en RTF no tienen una representación como tales. Al igual que las tablas, además, no hay marca de principio y fin, sino que cada párrafo que pertenece a una «lista» se marca de una manera especial. En cambio, se tratan como párrafos numerados, a los que se cambia el sangrado inicial. Aún así, existe de alguna manera el concepto de «profundidad», ya que hay una palabra de control (`\ilvl`) que la indica. No se ha descubierto completamente su uso, por lo que no se describe más precisamente.

Básicamente, se puede decir que las listas numeradas son un conjunto de párrafos que van seguidos, que tienen al principio una indicación del texto generado automáticamente para numerarlos, junto con variadas y numerosas especificaciones sobre los tamaños de los márgenes y otros estilos físicos. El susodicho texto generado automáticamente se embebe en un grupo que empieza con la palabra de control `\listtext`. En gran parte de los casos querremos ignorar el grupo entero, aunque su existencia nos revelará la inminente aparición de un nuevo elemento de la (una) lista.

Las listas no numeradas son similares, sólo que en vez de usar la palabra de control `\listtext` se usa la palabra `\pntext`. En Word 6 siempre se usa la palabra `\pntext`, sea cual sea el tipo de lista, lo que provoca que sea más difícil analizar correctamente documentos de RTF genéricos.

C.13. Tablas

Las tablas en RTF no existen. Lo más parecido que hay son conjuntos de filas seguidas, compuestas a su vez por una o más columnas. Es decir, que es *imposible* poner dos tablas seguidas, porque se convierten en una. El formato de las filas es como sigue:

```
<row>       <tabledef> <cell>+ \row

<cell>      <textpar>+ \cell
```

donde `<tabledef>` es la definición de *toda* la fila, esto es, el conjunto de todas las propiedades de la fila y de sus columnas. Empieza con la palabra de control `\trowd`, aunque se han encontrado algunas filas, que nunca eran las primeras de la tabla, comenzadas por `\intbl`. Las definiciones de las filas son palabras de control que empiezan por los caracteres `\tr`, y las de las celdas son las que empiezan por `\cl`. Una vez empiezan las propiedades de las celdas, que se separan con `\cellx`, no puede haber ninguna propiedad de fila, aunque cualquier reconocedor debería estar preparado para encontrar cualquier cosa en cualquier orden, como siempre. El principio del texto de la fila se marca con la palabra de control `\pard`[1].

Los bordes se definen con las palabras de control siguientes:

`\trbrdrl` Especifica el borde izquierdo de la fila.

`\trbrdrr` Especifica el borde derecho de la fila.

`\trbrdrt` Especifica el borde superior de la fila.

`\trbrdrb` Especifica el borde inferior de la fila.

`\trbrdrh` Especifica el borde horizontal (interno) de la fila.

`\trbrdrv` Especifica el borde vertical (interno) de la fila.

`\clbrdrl` Especifica el borde izquierdo de la columna.

`\clbrdrr` Especifica el borde derecho de la columna.

`\clbrdrt` Especifica el borde superior de la columna.

`\clbrdrb` Especifica el borde inferior de la columna.

[1] Esta proposición es, en parte, empírica, así que no se debe tomar como un principio absoluto

Hay que tener en cuenta que las definiciones de bordes son conjuntos de, al menos, dos palabras reservadas. Como éstos se definen en otras entidades RTF, las palabras que los definen comienzan por `\br` y no por `\tr` o `\cl`. Esto significa que al encontrar una palabra de control de especificación de borde de fila o columna, *habrá que leer varios símbolos más*, que serán el resto de ésta. Por ejemplo, una definición de borde (superior de una celda) podría ser:

```
\clbrdrt\brdrs\brdrw1\brdrcf0\brsp55
```

A pesar de que hay una descripción oficial para la definición de bordes, la recomendación, como siempre, es relajar la especificación de Microsoft y es tomar el enfoque más «liberal» posible, en este caso añadir todas las palabras de control que empiecen por `\br`.

Por otro lado, es destacable el hecho de que, a pesar de que la especificación no dice nada, *no* se pueden anidar las tablas. Esto, desde el punto de vista del análisis de los documentos, es francamente de agradecer.

Como último comentario, un error más en la especificación oficial de Microsoft del formato RTF: al mezclar celdas, Word simplemente «ignora» que se han mezclado, y crea una celda tan ancha como sea necesario. Esta descripción tan física de las tablas dificulta muchísimo el manejo de éstas y la conversión a otros formatos. En la especificación oficial, las únicas referencias a celdas mezcladas (*merged cells*) son cuatro palabras de control: dos para la mezcla horizontal y dos para la vertical. Las dos de mezcla horizontal, `\clmgf` y `\clmrg`, no se utilizan, al menos en Microsoft Word 97.

C.14. Objetos incrustados

Como último apartado, se comentan por encima los objetos incrustados. Éstos son entidades de otras aplicaciones metidas en un documento RTF, como puedan ser imágenes, ecuaciones, diagramas hechos con programas externos, etc. Debido, entre otras razones, a las infinitas posibilidades que se pueden producir, los objetos incrustados se dejaron fuera del estudio del formato RTF.

El único comentario que se hará es que todos los objetos incrustados tienen la forma:

```
...{\object ...} ...
```

Así, si uno pretende ignorarlos, lo único que debe hacer es ignorar el resto del texto del grupo, hasta encontrar la llave que cierre a la abierta antes de la palabra de control `\object`.

D Creación de paquetes Debian y RPM

En cualquier distribución moderna de Linux se utilizan, desde hace años, los «paquetes» de programas. Estos paquetes no son más que ficheros que contienen programas, o partes de ellos, junto con instrucciones sobre cómo instalarlos, dependencias de otros paquetes, etc. Esto facilita sobremanera la instalación y, especialmente, la desintalación, ya que los programas que los manejan hacen todas las comprobaciones necesarias para que todo vaya bien, no se instalen programas antes de desintalar sus dependencias, no se desintalen funciones utilizadas por programas que siguen instalados, etc.

Además, dado que, como casi siempre, los paquetes y los programas que los manejan están hechos para ser extensibles y tratables desde otros programas, se han hecho muchas interfaces gráficas para gestionar el *software* instalado en una máquina de manera sencilla.

Por tanto, la existencia de paquetes fácilmente manejables abre mucho las posibilidades de popularidad de un programa. En este caso, sin embargo, esta necesidad no era tan crítica, porque gran parte del trabajo de fin de carrera estaba pensado para otros programadores, que no son tan amigos de los paquetes precompilados, dado que, para que funcionen en todas las máquinas, no pueden aprovechar algunas características modernas de los últimos procesadores.

D.1. Dos formatos diferentes

El primer sistema y formato de paquetes que salió en Linux fue el RPM, de la empresa Red Hat. Rápidamente se popularizó, por las ventajas que acarreaba su uso, y porque Red Hat abrió completamente las especificaciones del formato, e incluso liberó por la GPL el programa que manejaba los paquetes (el `rpm`). Algunos meses después, los voluntarios de la distribución Debian idearon otro sistema y formato de paquetes, que lleva el mismo nombre de la distribución.

A pesar de que el sistema de Red Hat era bueno, y funcionaba perfectamente, Debian decidió no aprovechar la oportunidad de unificar el formato de paquetes entre todas las distribuciones, y creó el suyo propio. ¿Por qué? Básicamente, se puede decir que Debian siguió las antiguas costumbres de la comunidad UNIX. En particular, intentó a toda costa no inventar un formato de paquetes nuevo, dado que con las herramientas existentes en UNIX podía resolverse el problema.

Así, en vez de adoptar el nuevo formato de Red Hat (que, aunque funciona y es abierto, sigue siendo algo «nuevo», que hay que estudiar), utilizó el `ar` y el `tar`, ambas herramientas básicas de cualquier sistema UNIX, y el `gzip` de GNU, ampliamente extendido por entornos propietarios, para construir un «nuevo» formato de paquetes. De esta manera, y siguiendo las normas de simplicidad máxima de UNIX, se guardó la información en pequeños ficheros de texto, y se escribieron varias pequeñas utilidades para ayudar al creador de paquetes con la ardua tarea que eso conlleva. Por lo tanto, los paquetes Debian son más fáciles de manejar, en lo que se refiere a crear nuevos programas que interactúen con ellos. Esta solución, a pesar de ser peor a corto plazo, ya que hacía perder bastante tiempo creando las especificaciones de un nuevo formato, cuando ya había uno funcionando desde hacía un tiempo, a la larga está demostrando ser mejor, ya que, tal y como se ha hecho en UNIX siempre, las soluciones son a largo plazo: ahora, es más fácil crear programas que ayuden en la construcción de paquetes Debian que en la creación de paquetes RPM. Esto también conlleva que sea más fácil adoptar paquetes Debian en otras distribuciones que al revés.

D.2. Paquetes Debian

Hay muchas herramientas y documentación para crear paquetes Debian. Con la propia distribución vienen varios paquetes con programas y páginas HTML describiendo el proceso y las herramientas disponibles.

Los dos documentos de referencia oficiales más importantes son los dos de Ian Jackson, [JAC96] y [JAC98]. El primero es más técnico, y está más centrado en el proceso en sí de construir los paquetes. El segundo es una referencia con las convenciones, normas de estilo y reglas para ponerles nombre, para elegir nombres para «paquetes virtuales» (paquetes que realmente no existen, pero que los puede proveer uno o más paquetes «reales», como «navegador de Internet» o «visor de documentos PostScript») y para explicar cómo determinar si debe considerarse que un paquete es necesario, optativo, a qué apartado pertenece, etc.

La herramienta más útil para construir paquetes Debian desde cero es `deb-make`. Este programa casi se considera obsoleto, porque se ha escrito todo un conjunto de herramientas para sustituirlo, pero sigue sirviendo, al menos para pequeños proyectos. Su cometido es preparar un árbol de directorios, que contiene los *fuentes* de un programa, para poder crear luego paquetes Debian de todas las versiones que se hagan. Su funcionamiento es muy sencillo. Tanto, que lo único que tenemos que hacer es ejecutarlo en el directorio raíz donde tengamos nuestro proyecto (en formato original, en *fuentes*), y el programa se encargará de crear el directorio `debian`, donde además pondrá todos los ficheros necesarios para la creación del paquete Debian. Después de esto, sólo tendremos que rellenar algunos campos vacíos en el fichero `control`, como los comentarios explicando el contenido del paquete, y quizás retocar algo que haya hecho mal, como poner la arquitectura a *cualquiera*, cuando lo más probable es que el programa sólo funcione, por ser compilado, en arquitectura *i386* (o cualquier otra).

Después de tener preparado el árbol de directorios, ya sea por haberlo hecho a mano o por ayudarnos de una herramienta como `deb-make`, ya podemos crear el paquete en sí. Esto se lleva a cabo con el programa `dpkg-buildpackage`. Es necesario hacer notar que, para poder ejecutar este programa, hay que tener ciertos objetivos cubiertos en el fichero de creación del proyecto (el `Makefile`). También vale la pena destacar que, si utilizamos las herramientas `autoconf` y `automake` de GNU, como en este proyecto, estos objetivos ya estarán automáticamente cubiertos, y no hará falta tener ninguna consideración especial. En resumidas cuentas, si nuestro proyecto está construido con `autoconf` y `automake`, sólo tendremos que teclear:

```
$ deb-make
...
$ dpkg-buildpackage
```

Y tendremos automágicamente un paquete Debian de nuestro proyecto en el directorio inmediatamente inferior, con sus dependencias halladas y todo.

Un conjunto de herramientas interesante es el paquete *debhelper*. Contiene varios programas, cuyos nombres empiezan siempre por `dh_`, por convención, que ayudan en la creación de paquetes Debian. Uno de ellos es `dh_make`, que es una de las alternativas al programa `deb-make`. Otro conjunto de herramientas, más moderno, son los *devscripts* [DSWeb], que contienen programas como `debchange`, `debclean`, `release`, `build`, etc. que sirven para el mismo cometido.

D.2.1. El fichero de `control`

El fichero principal de configuración de un paquete Debian es el fichero `control`, que especifica la información general, como el nombre del paquete, el encargado de mantenerlo, versión, arquitectura, dependencias, etc. Éste, al igual que otros, son ficheros de control, que comparten una serie de campos posibles. A continuación se da una descripción de todos estos campos y su sintaxis.

En general, un fichero se compone de uno o más párrafos de campos. Los párrafos se separan con líneas en blanco. Cada párrafo es una serie de campos y valores, de la forma *`nombre_campo: valor`*, que ocupan una línea entera. Por normal general, se deja un espacio en blanco después del caracter `:`. Algunos valores de campos puede extenderse por varias líneas. En ese caso, cada línea adicional de contenido tiene que empezar necesariamente por un espacio o caracter tabulador. Cualquier cantidad de espacio en blanco encontrada al principio de las líneas se ignorará completamente.

Excepto en raras ocasiones, sólo se permite una línea de datos. No debe haber espacio en blanco dentro de los nombres (de paquetes, arquitecturas, ficheros, o lo que sea), números de versiones o entre los caracteres de una versión.

Los campos definidos por Debian son:

Package Nombre del paquete binario. Los únicos caracteres permitidos son los alfanuméricos, el «+», el «-» y el «.». El nombre debe empezar por un caracter alfanumérico, y debe tener como mínimo dos caracteres.

Version La versión del paquete. Debian tiene ciertas reglas para representar los números de versión, descritas en [JAC96].

Architecture Es una sola palabra para la arquitectura de la CPU. El valor especial «`all`» indica que el paquete es independiente de la arquitectura (ficheros de datos, documentación, programas interpretados, etc.)

Maintainer El nombre y la dirección de correo electrónico del encargado del paquete.

Source Identifica el nombre del paquete fuente. Puede ser solamente el nombre, o el nombre y el número de versión entre paréntesis.

Campos de relaciones entre paquetes Estos campos son *Depends*, *Pre-Depends*, *Recommends*, *Suggests*, *Conflicts*, Provides y *Replaces*. Su sintaxis y semántica está descrita en [JAC96].

Description Una descripción del contenido del paquete.

Essential Es un campo lógico. Si vale `yes`, programas como `dpkg` rehusarán desinstalarlo, aunque sí podrán actualizarlo o reemplazarlo.

Section Incluye al paquete en una clasificación hecha por Debian.

Priority Representa cómo es un paquete de importante.

Binary Es una lista de paquetes binarios, separados por comas, que puede producir un paquete fuente. Los paquetes binarios no tienen que producirse necesariamente para todas las arquitecturas.

Installed-Size Da la cantidad total de espacio en disco requerido para instalar el citado paquete. Se representa en kilobytes, con un simple número decimal.

Files Es una lista de ficheros, con información de cada uno. Dependiendo de en qué fichero aparezca, tendrá una sintaxis y semántica diferentes. De nuevo, véase [JAC96] para más información.

Standards-Version La versión más reciente de las normas (los manuales de dpkg y de política de Debian, y los documentos asociados) que sigue el paquete.

Distribution Para qué tipo de distribución está preparado el paquete. Los valores posibles son `stable`, `unstable`, `contrib`, `non-free`, `experimental`, `frozen`

Urgency Una medida de lo importante que es actualizar este paquete desde versiones antiguas. Es una sola palabra, y normalmente es una entre `LOW`, `MEDIUM` y `HIGH`. Se puede añadir al final un comentario, generalmente entre paréntesis.

Date Una fecha, de última modificación, de última construcción del paquete...

Format Este campo aparece en el fichero `.changes`, y especifica el revisión del formato del fichero.

Changes Los cambios realizados sobre diferentes versiones de un programa, para que las lea una persona.

Filename El nombre de un fichero. Si hay que especificar varios, se separan por espacios.

MSDOS-Filename El nombre de un fichero en formato MS-DOS.

Size El tamaño en bytes, en decimal.

MD5sum La suma de control del/de los fichero(s) que conforman un paquete binario en la distribución.

Status Este campo es para que `dpkg` lleve la cuenta de los paquetes que el usuario quiere instalar, desinstalar o dejar como están.

Config-Version Si un paquete no está instalado o configurado, este campo indica la última versión del paquete que se configuró correctamente.

Conffiles Información sobre los ficheros de configuración gestionados automáticamente.

Campos obsoletos Son campos que todavía reconoce `dpkg`, pero que no deberían aparecer más:

Revision, Package-Revision, Package_Revision La revisión del paquete, que antes estaba en un campo aparte, cuyo nombre fue cambiando.

Recommended El nombre antiguo de *Recommends*.

Optional El nombre antiguo de *Suggests*.

Class El nombre antiguo de *Priority*.

D.2.2. Actualización de los paquetes Debian

Hacer el primer paquete Debian es una tarea trivial: llamamos a la utilidad *deb-make* para que nos prepare los fuentes, y a *dpkg-buildpackage* para que construya el paquete en sí. Pero ¿qué pasa cuando queremos actualizar un paquete o construir una versión nueva? ¿de dónde saca *dpkg-buildpackage* la información sobre la versión del programa y del paquete? La respuesta es muy sencilla: del fichero de cambios, el fichero `changelog`.

Esto tiene dos ventajas. Por un lado, la sencillez: sólo tenemos que escribir el nombre del nuevo paquete, y nos obliga a escribir los cambios entre versiones. En el nombre del nuevo paquete podemos discriminar entre la versión del programa y la versión del paquete para una versión dada del programa, lo que da más precisión sobre la naturaleza de los cambios, con sólo mirar el nombre del paquete.

La única desventaja es que es difícil de automatizar, dado que los cambios no los puede decir una máquina, normalmente, y tampoco es posible, de manera fácil, que la máquina se dé cuenta de qué cambios son al paquete y cuáles al programa en sí.

D.3. Paquetes RPM

La compañía Red Hat fue la pionera en utilizar paquetes de programas en las distribuciones de Linux[1]. Antes, los programas estaban simplemente comprimidos, y, aunque su instalación no era especialmente dificultosa, estos «paquetes» carecían de información sobre dependencias de otros, y podía llegar a resultar realmente complicado llevar la cuenta de qué paquetes se habían actualizado, cuáles estaban obsoletos, etc. Además, al actualizar programas, podían quedar restos de las anteriores versiones. Y la desinstalación, por añadidura, dejaba bastante que desear.

Los paquetes RPM fueron un gran avance para las distribuciones de Linux, ya que simplificaron sobremanera la gestión de los programas instalados en una máquina. El enfoque que dio Red Hat al formato de paquetes al diseñarlo fue que el usuario debían tener acceso a los fuentes originales disponibles, además de los fuentes preparados para compilar en la máquina sin problemas y a los binarios finales. La forma de resolver esto fue tener dos tipos de paquetes, los binarios y los de fuentes. En éstos estarían los fuentes originales, un parche para hacerlo compilar perfectamente y las instrucciones para compilarlos; en aquéllos estarían solamente los programas finales a ejecutar y posiblemente la documentación, es decir, lo que la mayoría de los usuarios quiere verdaderamente.

En cuanto a su creación se refiere, los paquetes RPM también gozan de herramientas para ayudar en esa tarea, además de en su gestión y depuración. La propia orden `rpm` puede ayudar en todas estas tareas, aunque hay otras herramientas más adecuadas para la gestión, como `glint` o `gno-rpm` (para GNOME). Para crear paquetes, hay otras herramientas, pero hechas por personas ajenas a Red Hat, que con frecuencia duplican innecesariamente esfuerzos o resultan ser de baja calidad. Entre estas herramientas «no oficiales» podemos mencionar `yasop`, `rpmbuilder` o `rpmproc`.

D.3.1. Construcción de paquetes RPM

Para construir un paquete RPM, hay dos ficheros importantes: uno es el `/etc/rpmrc`, que es común para todo el sistema, y el otro es el fichero *spec*, que es particular para cada paquete. A grandes rasgos, los pasos que hay que dar para crear un paquete RPM son:

1. Asegurarse de que el fichero `/etc/rpmrc` está perfectamente configurado para el sistema.
2. Hacer que los fuentes del programa para el que se va a construir el paquete compilen en la máquina.
3. Hacer un parche con los cambios hechos en el paso anterior para que los fuentes compilaran correctamente.
4. Escribir un fichero *spec* para el paquete.

[1] La idea en sí ya existía en versiones comerciales de UNIX, por ejemplo en la versión de Hewlett-Packard, HP-UX.

5. Asegurarse de que todo está donde debe estar.

6. Construir el paquete utilizando la orden `rpm`.

El fichero `rpmrc`

En este fichero se guardan los datos que, al menos en principio, son comunes a todos los paquetes que se vayan a construir en la misma máquina. Estos datos son cosas como la distribución para la que están preparados, el nombre y una dirección de correo electrónico del responsable de construir los paquetes, el directorio a partir del cual se construyen los paquetes, etc.

Para ver los contenidos del fichero, se puede utilizar la orden del sistema:

```
rpm --showrc
```

Un ejemplo de la salida producida por esta orden es:

```
ARCHITECTURE AND OS:
build arch            : i386
compatible build archs: i586 i486 i386 noarch
build os              : Linux
compatible build os's : Linux
install arch          : i586
install os            : Linux
compatible archs      : i586 i486 i386 noarch
compatible os's       : Linux
RPMRC VALUES:
builddir              : /usr/src/redhat/BUILD
...
optflags              : -O2
rpmdir                : /usr/src/redhat/RPMS
rpmfilename           : %{ARCH}/%{NAME}-%{VERSION}-%{RELEASE}.%{ARCH}.rpm
signature             : none
sourcedir             : /usr/src/redhat/SOURCES
specdir               : /usr/src/redhat/SPECS
srcrpmdir             : /usr/src/redhat/SRPMS
timecheck             : (not set)
tmppath               : /var/tmp
topdir                : /usr/src/redhat
vendor                : (not set)
```

El fichero *spec*

Los ficheros *spec* mantienen información específica de cada paquete. Son una descripción de su contenido, junto con instrucciones sobre cómo compilarlo, instalarlo, y una lista de los ficheros de los que se compondrá el paquete final. Por convención, los ficheros *spec* tienen un nombre como *nombre_paquete-versión-edición*-spec

Un pequeño ejemplo de fichero *spec* podría ser[2]:

```
Summary: ejects ejectable media and controls auto ejection
Name: eject
Version: 1.4
Release: 3
Copyright: GPL
Group: Utilities/System
Source: sunsite.unc.edu:/pub/Linux/utils/disk-management/eject-1.4.tar.gz
Patch: eject-1.4-make.patch
Patch1: eject-1.4-jaz.patch
%description
This program allows the user to eject media that is autoejecting like
CD-ROMs, Jaz and Zip drives, and floppy drives on SPARC machines.

%prep
```

[2]De hecho, este fichero es un ejemplo real, del programa *eject*

```
%setup
%patch -p1
%patch1 -p1

%build
make RPM_OPT_FLAGS="$RPM_OPT_FLAGS"

%install
install -s -m 755 -o 0 -g 0 eject /usr/bin/eject
install -m 644 -o 0 -g 0 eject.1 /usr/man/man1

%files
%doc README COPYING ChangeLog

/usr/bin/eject
/usr/man/man1/eject.1
```

Los ficheros *spec* se dividen en varios apartados bien diferenciados, que son:

Header Es la cabecera del fichero, donde se da información general sobre el paquete y su contenido.

Prep Donde se prepara el fichero para su compilación.

Build Donde se dan las órdenes precisas para compilar el programa.

Install Donde se instala el programa.

Files La lista de los ficheros que compondrán el paquete final.

Todos estos apartados se verán con más detenimiento enseguida.

Header

La cabecera tiene unos campos fijos que hay que rellenar. La lista de estos campos es la siguiente:

Summary Una descripción de los contenidos del paquete RPM, en una sola línea.

Name Nombre del paquete. Debe coincidir con el nombre inducible del nombre del fichero del paquete.

Version Versión del programa. Debe coincidir, de nuevo, con el indicado por el nombre del fichero del paquete.

Release Edición del paquete. Debe coincidir con el indicado por el nombre del fichero del paquete.

Icon El nombre del icono del paquete. Este icono puede ser utilizado por programas gráficos, como «glint» (de la propia Red Hat), para representar al paquete. Debe estar en formato GIF, y residir en el directorio SOURCES.

Source Esta línea apunta al origen de los fuentes originales. Se utiliza para conseguir de nuevo los fuentes sin modificar, y para buscar nuevas versiones del programa. Se puede especificar más de una dirección, poniendo `Source0`, `Source1`, etc., en líneas consecutivas.

Patch Indica dónde puede encontrarse el parche con las diferencias respecto a los fuentes originales, por si se quiere obtener de nuevo. Al igual que con el campo anterior, se pueden especificar varios parches, con `Patch0`, `Patch1`, etc. en líneas consecutivas. Estos ficheros deben ir en el directorio SOURCES.

Copyright Especifica la licencia bajo la cual se distribuye el contenido del paquete, como pueda ser GPL, BSD, MIT, dominio público, distribuible, o comercial.

BuildRoot Esta línea permite especificar un directorio, a partir del cual se compilará y se instalará el nuevo paquete. Se puede utilizar como directorio de prueba, antes de instalar realmente el paquete en la máquina.

Group Indica el grupo de programas al que pertenece el contenido del paquete, dentro de una jerarquía de tipos creada por Red Hat para tal fin. Algunos ejemplos de grupos de programas son Comunicaciones, Hojas de cálculo, Redes, Correo, Noticias, Matemáticas, Maquetación/TEX, Utilidades/Compresión, Utilidades/Terminal, Utilidades/Texto, X11/Aplicaciones/Gráficos, Servicios, Documentación, X11/Juegos/Estrategia, X11/Gestores de ventanas, Programación/Lenguajes/Tcl, Intérpretes de órdenes...

%description Realmente no es parte de la cabecera, pero tiene cierta relación. Se necesita una serie de caracteres para describir cada paquete o subpaquete. Este campo es multilínea, y se utiliza para dar una descripción (para que la lea una persona) de los contenidos del paquete.

Prep

Es el segundo apartado del fichero *spec*. Se utiliza para preparar los fuentes para ser compilados. Aquí debe especificarse todo lo necesario para que los fuentes de compilen con un simple `make`.

Debe hacerse notar que cada uno de los siguientes apartados no es más que un hueco para escribir órdenes que ejecutará un intérprete normal. Es decir, que podemos poner cualquier orden que deseemos, siempre que estemos seguros de que va a estar disponible en la máquina que va a instalar el paquete. Sin embargo, en algunos de los apartados se definen algunas macros para facilitar la escritura de éstos.

Una de estas macros, la más útil de este apartado, es `%setup`. Se encarga de desempaquetar los fuentes y cambiar a su directorio raíz. Además, puede recibir algunas opciones, todas documentadas en [BAR97].

Otra de las disponibles es la macro `%patch`, que se encarga de automatizar el proceso de aplicación de parches a los fuentes. Acepta varias opciones, de nuevo documentadas en el «RPM HOWTO».

A pesar de que hay más macros definidas, estas son las más comunes, y no deberían ser necesarias más, excepto en paquetes muy complicados.

Build

Realmente no hay ninguna macro definida para este apartado. Aquí simplemente se ponen órdenes que se interpretarán directamente, a ejecutar después de haber desempaquetados los fuentes, entrado en su directorio principal y haber aplicado los parches necesarios. Hay que tener en cuenta que el directorio actual de trabajo vuelve al inicial al principio de cada apartado, aunque se puede cambiar dentro de cualquiera de ellos.

Install

Tampoco hay macros para este apartado, debido a su simplicidad. Lo único que hay que especificar aquí es la orden, u órdenes, para instalar el resultado de compilar los fuentes.

Si se tiene un objetivo `install` en el fichero de proyecto que funcione bien, el contenido del apartado *install* puede ser tan simple como la línea

```
make install
```

Si no, puede parchearse el fichero de proyecto convenientemente y ejecutar un `make install`, o bien copiar los ficheros necesarios «a mano», con las órdenes del sistema.

Además de las órdenes de instalación, pueden especificarse mediante cuatro macros las acciones a ejecutar antes y después de instalar y desintalar. Estas macros son:

`%pre` Las órdenes a ejecutar antes de instalar el programa.

`%post` Las órdenes a ejecutar después de instalar el programa.

`%preun` Las órdenes a ejecutar antes de desinstalar el programa.

`%postun` Las órdenes a ejecutar después de desinstalar el programa.

Como siempre, el contenido de estas macros son órdenes que se pasarán sin modificación al intérprete `/bin/sh`.

Files

El último apartado indica la lista de ficheros que formarán parte del paquete final, dado que RPM no tiene ninguna forma de saber qué ficheros hay que instalar, guiándose por el resultado del `make install`. Realmente es una pena que tenga que darse explícitamente esta lista, ya que Debian tiene una forma de hallar esta lista automáticamente, lo que es más cómodo para el que construye los paquetes, y además evita potenciales equivocaciones u omisiones en la elaboración de ésta.

En este apartado sí que hay algunas macros definidas, que ayudan a especificar la naturaleza exacta de cada fichero:

`%doc` Especifica que el fichero siguiente es documentación. Este tipo de ficheros se instalarán en el directorio `/usr/doc/$NAME-$VERSION-$RELEASE`.

`%config` Marca los ficheros de configuración de un paquete. Es muy útil a la hora de conservarlos cuando se actualice el paquete. Además, al desinstalar el paquete, se conservarán los ficheros que hayan cambiado respecto a la versión original del paquete, con la extensión `.rpmsave`.

`%dir` Indica que el directorio siguiente es el único que forma parte del paquete, y no su contenido. Si no se especifica esta macro, cada directorio especificado en la lista se interpretará como un directorio que se tiene que incluir «en peso» en la lista, es decir, junto con su contenido.

`%files -f` *`fichero`* Especifica un fichero del que leer parte de la lista. Es muy útil cuando los programas producen listas totales o parciales de los ficheros que los componen, y queremos aprovecharnos de esta característica.

Como referencia, se adjunta el fichero *spec* final que quedó para la aplicación RTHC:

```
%define name rthc
%define version 0.1.0
%define release 1
%define prefix /usr
%define _tmppath /tmp

Summary: RTF To HTML Converter
Name: %{name}
Version: %{version}
Release: %{release}
Source0: %{name}-%{version}.tar.bz2
Copyright: GPL
Group: Utilities/Text
BuildRoot: %{_tmppath}/%{name}-buildroot
Prefix: %{prefix}

%description
RTHC stands for "RTF To HTML Converter". Is a converter build up from a
software library called libFreeRTF.

%prep
%setup

%build
./configure --prefix=%{prefix}
make

%install
make prefix=$RPM_BUILD_ROOT/%{prefix} install-strip


%clean
rm -rf $RPM_BUILD_ROOT

%files
%attr(755,root,root) %{prefix}/bin/rthc
```

```
%changelog
* Sat Aug 19 2000 Esteban Manchado <a2092@dis.ulpgc.es> 1.0-1
- First spec file.

# end of file
```

D.3.2. Actualización de paquetes RPM

Una vez se ha construido el primer paquete, la creación de actualizaciones es trivial. Lo único que hay que cambiar es el número de versión en el fichero *spec*. Por lo general, el enfoque seguido es crear automáticamente el fichero *spec* a partir de uno *spec.in*. La ventaja es que no tenemos que hacer absolutamente nada para actualizar los paquetes, aparte de llamar a la orden *rpm* para crear el nuevo. La desventaja es que no nos obliga a mantener una lista de cambios entre versiones: si queremos hacer cambios, tendremos que cambiar a mano el fichero *spec*, para añadir al final los cambios hechos. Además, tendremos que hacer coincidir a mano el número de versión del paquete con el que pongamos a la hora de describir los cambios, con lo que se pueden producir incoherencias entre ambos.

E Otros anexos

A continuación se presentan algunos documentos de interés, todos concernientes al llamado *software* libre:

1. La guía de Debian para clasificar licencias como libres.
2. La licencia General Public License (GPL), bajo la que se distribuye el programa RTHC.
3. La licencia Lesser General Public License, o Library General Public License (LGPL), especialmente diseñada para aplicarla a programas que formarán parte de otros más grandes, y bajo la que se distribuirán las clases de manejo de RTF, la *libFreeRTF*.
4. Un artículo, en versión original, que describe doce reglas para mejorar un proyecto de *software* libre.
5. Clasificación de las licencias de programas.

E.1. Debian Free Software Guidelines

Aquí se presenta el original, en inglés, de la guía de Debian para calificar como «libre» a una licencia.

The Debian Free Software Guidelines (DFSG) as is our definition of 'free' software.

Free Redistribution The license of a Debian component may not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license may not require a royalty or other fee for such sale.

Source Code The program must include source code, and must allow distribution in source code as well as compiled form.

Derived Works The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

Integrity of The Author's Source Code The license may restrict source-code from being distributed in modified form *only* if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software. (This is a compromise. The Debian group encourages all authors to not restrict any files, source or binary, from being modified.)

No Discrimination Against Persons or Groups The license must not discriminate against any person or group of persons.

No Discrimination Against Fields of Endeavor The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

Distribution of License The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

License Must Not Be Specific to Debian The rights attached to the program must not depend on the program's being part of a Debian system. If the program is extracted from Debian and used or distributed without Debian but otherwise within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the Debian system.

License Must Not Contaminate Other Software The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be free software.

Example Licenses The "GPL," "BSD," and "Artistic" licenses are examples of licenses that we consider *free*.

E.2. General Public License

Aquí se presenta, en su original en inglés, el texto de la licencia bajo la cual se distribuirá el programa RTHC.

GNU GENERAL PUBLIC LICENSE Version 2, June 1991
Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

E.2.1. Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

E.2.2. GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

1. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program"means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as ɀou".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

2. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

3. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

4. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

5. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

6. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

7. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

8. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

9. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

10. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and .ªny later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

11. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

12. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM .ªS IS"WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

13. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

E.2.3. How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the çopyright"line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) 19yy  <name of author>

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.
```

```
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.
```

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than ‘show w’ and ‘show c’; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a çopyright disclaimer"for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

E.3. Lesser General Public License

Aquí se presenta, en su original en inglés, el texto de la licencia LGPL, bajo la que se distribuirá el paquete de clases *libFreeRTF*. La licencia LGPL es más adecuada para paquetes de funciones y otros programas que serán parte de otros más grandes, porque permite que los programas distribuidos por esta licencia se enlacen con otros distribuidos por otra completamente diferente, como programas comerciales. Esta licencia también se conoce como «Library General Public License».

GNU LIBRARY GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright (C) 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. [This is the first released version of the library GPL. It is numbered 2 because it goes with version 2 of the ordinary GPL.]

E.3.1. Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.

This license, the Library General Public License, applies to some specially designated Free Software Foundation software, and to any other libraries whose authors decide to use it. You can use it for your libraries, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library, or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program with the library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.

Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that what they have is not the original version, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies distributing free software will individually obtain patent licenses, thus in effect transforming the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License, which was designed for utility programs. This license, the GNU Library General Public License, applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don't assume that anything in it is the same as in the ordinary license.

The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program. However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.

Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers did not use the libraries. We concluded that weaker conditions might promote sharing better.

However, unrestricted linking of non-free programs would deprive the users of those programs of all benefit from the free status of the libraries themselves. This Library General Public License is intended to permit developers of non-free programs to use free libraries, while preserving your freedom as a user of such programs to change the free libraries that are incorporated in them. (We have not seen how to achieve this as regards changes in header files, but we have achieved it as regards changes in the actual functions of the Library.) The hope is that this will lead to faster development of free libraries.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library.ªnd a "work that uses the library". The former contains code derived from the library, while the latter only works together with the library.

Note that it is possible for a library to be covered by the ordinary General Public License rather than by this special one.

E.3.2. GNU LIBRARY GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

1. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library General Public License (also called "this License"). Each licensee is addressed as zou".

A "library"means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library"means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code"for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

2. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

3. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a) The modified work must itself be a software library.

   b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

   c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

   d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

   (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

4. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

   Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

   This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

5. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

   If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

6. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

   However, linking a "work that uses the Library"with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

   When a "work that uses the Libraryüses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

   If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

   Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

7. As an exception to the Sections above, you may also compile or link a "work that uses the Library"with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

   You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

   a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

   b) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

c) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

d) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library"must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

8. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

   a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

   b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

9. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

11. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

12. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

    If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

    It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to

decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

13. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

14. The Free Software Foundation may publish revised and/or new versions of the Library General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and .ªny later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

15. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

16. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY .ªS IS"WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

17. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

E.3.3. How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the çopyright"line and a pointer to where the full notice is found.

```
<one line to give the library's name and a brief idea of what it does.>
Copyright (C) <year>  <name of author>
```

```
This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Library General Public
License as published by the Free Software Foundation; either
version 2 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
Library General Public License for more details.

You should have received a copy of the GNU Library General Public
License along with this library; if not, write to the Free
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
```

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a çopyright disclaimer"for the library, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the
library 'Frob' (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990
Ty Coon, President of Vice
```

That's all there is to it!

E.4. Twelve Rules For A Better Open Source Project

El siguiente es un artículo de Dan York aparecido en la «Linux magazine» de febrero de 2000, en el apartado «In the trenches».

Open source changes everything. That's what the pundits will tell you. What they mean is that open source isn't just about free or better software. It's a whole new way of thinking about how to solve problems and manage projects. When we launched the project that I'm working on — the Linux Professional Institute — a year ago, we knew that we would be able to leverage this open source methodology to make the project stronger and better, even though we weren't developing software.

To step back, the Linux Professional Institute (LPI) has been around since the fall of 1998. Its goal is to develop a certification program for Linux professionals similar to those available to Microsoft (MCSE) and Novell (CNE) users. We have developed a series of exams that people can take to demonstrate their knowledge of and proficiency with Linux.

Of course Linux is different from NT or NetWare. And one of the challenges we face at the LPI is that no one entity controls the Linux operating system, so no one can dictate the standards for certification. Anyone can create their own Linux certification programs — in fact several have. In this environment, our efforts have been focused on building a consensus and making sure that we are as inclusive as possible — something that the open source model is ideally suited to achieve.

In our first year of operation, there have been a lot of lessons learned. So, without further ado, here are the 12 most important:

1. Democracy is Critical and Inefficient. Most of the LPI's major decisions have been made through public mailing lists where anyone can participate. They have been çonsensus points"that were hammered out in (sometimes lively) discussions on the lists. This process has allowed many different viewpoints to be recognized and debated. The end result has been a much stronger and better-defined product. This democratic process can drag on, however. It has sometimes taken great effort

to ensure that discussions stay on track and do eventually reach a conclusion. 2. Living in a Fishbowl Can Be Challenging. What do you do when all your major discussions occur in a public forum, with competitors and members of the media watching your every move? On the one hand, it's important to be as open as possible. On the other hand, we have been very aware that anything posted to our public lists can very easily turn up in the media — or be used by competing programs. Day by day, this remains one of our toughest challenges.

2. Living in a Fishbowl Can Be Challenging. What do you do when all your major discussions occur in a public forum, with competitors and members of the media watching your every move? On the one hand, it's important to be as open as possible. On the other hand, we have been very aware that anything posted to our public lists can very easily turn up in the media – or be used by competing programs. Day by day, this remains one of our toughest challenges.

3. Building Coalitions is Critical. Early on, we recognized that our initial group of volunteers didn't have all the expertise necessary to truly pull off a world-class certification program. So we built an advisory council of other companies and individuals who could help move us forward. This council has put us in touch with organizations that could help us, and provided the public backing for LPI that has attracted others to our project. This public support also gave us the legitimacy we needed to go out and attract financial investment.

4. Be Inclusive. We use public mailing lists that are open to everyone because we want to hear what others have to say and also because we need the help of many people to make this project a success. We want to see arguments against some part of our project discussed, debated, and resolved before the project goes forward and arguments flare up in the larger public media. We also have sought to include as many different viewpoints and people so that others would not go off to create yet another competing project. Finally, by being inclusive, we have fostered a community of volunteerism. Our project will not succeed without the many long hours put in by volunteers, and the forum where they participate is the public mailing list.

5. Recruiting New Volunteers is Crucial. The fact of the matter is that volunteers come and go. People's lives change, and sometimes they find themselves with less time to volunteer on projects. People burn out too. At LPI we have learned to constantly grow our pool of volunteers — we advertise and promote ways that people can become involved and are constantly working to identify and recruit new people to bring new energy to the project.

6. Give Credit Where Credit is Due. Part of recruiting and retaining volunteers is to make sure we publicly recognize the many contributions that have been made to our project. We have worked to make sure that special Web pages, articles in magazines, and our online newsletters all recognize the people who help out in some way.

7. Delegate. Once volunteers are found, it is extremely important to give them real tasks to do. Being caught up in the management of the project, it is easy for us to just do something ourselves because doing things ourselves often takes less time than explaining them to others. Everyone is guilty of this. But sometimes taking the quick and easy way means losing an opportunity to get someone else involved. And all those quick and easy tasks can add up too, and without delegation you can soon find yourself in a state of complete and utter overwork. 8. Accountability is a Must. With every action delegated or assigned, there must be deadlines and also someone designated to make sure the deadlines are met. We're all volunteers and it's easy for things to fall through the cracks. Within the LPI board, we've implemented a list of weekly action items that outline what each of us has committed to, and each week we are asked the status on each item. We've also worked to make sure there are committee chairpersons responsible for the actions of a committee.

8. Accountability is a Must. With every action delegated or assigned, there must be deadlines and also someone designated to make sure the deadlines are met. We're all volunteers and it's easy for things to fall through the cracks. Within the LPI board, we've implemented a list of weekly action items that outline what each of us has committed to, and each week we are asked the status on each item. We've also worked to make sure there are committee chairpersons responsible for the actions of a committee.

9. Do Not Rush in Response to Market Pressure. Market pressure, of course, is very real. Our project has been in a race to get our program to market as quickly as possible. There is great pressure for us to push our program out as soon as possible, but the challenge has been to balance that pressure against the need for a top-quality program. We will live and die by the quality of our certification exams and how they are perceived by people within the Linux and larger information-technology

worlds. We have focused on making sure our program is of the highest quality and have made a conscious decision to deliver it when it is right to do so.

10. Don't Ignore Market Pressure. Having said that market pressure cannot be the end-all and be-all of your project, what good would it be if we delivered a high-quality program after the Linux and larger IT markets had already crowned someone else the Linux certification program? Market pressures count. We have had to set a timeline for the project that reflects this.

11. Communicate, Communicate, Communicate. If there is one thing we have learned above all, it is that our project must be in constant communication with the larger world. There must be news releases to the media, updates on the Web site, newsletters sent out in e-mail, and meetings with sponsors and supporters. Every little piece helps. We have spoken at conferences, written for magazines, and scheduled BOF (Birds of a Feather) sessions at conferences. The key has been to let people know that we exist, what we are doing, and how they can become involved. Without big-league resources like PR firms, we have to work doubly hard to make sure that our message gets out there. Our supporters and sponsors (from the coalitions we have built) have also been of great assistance in spreading the word.

12. Keep the Faith. It's a challenge at times to remain solid in our belief that the project will become reality. Faced with inevitable roadblocks and challenges, and with naysayers claiming we could never do it, the struggle has been to remain positive and moving forward. We have learned to rely on one another to get through worrying times, and to continue to project this positive image publicly to motivate and inspire others to help us.

It has been a crazy first year for the LPI. We have accomplished an amazing amount in a short time period, and could not have done it without the many volunteers and supporters who have helped us along the way. We have encountered and triumphed over many challenges and learned a great many lessons along the way. In the end, probably the greatest thing we learned was that you must gather as many supporters as possible and move as quickly as possible to bring your project to reality. And in the end, you must simply believe in your project.

E.5. Clasificación de las licencias de programas

En el primero de los famosos, que no populares, «Documentos Halloween»[1] apareció una clasificación bastante acertada de las diferentes licencias existentes en el mundo de la informática.

Aquí se presenta el pequeñísimo fragmento del Documento Halloween I, en el original en inglés (hay una traducción disponible, pero es bastante mala), donde se hace esa acertada clasificación:

[...] The broad categories of licensing include:

E.5.1. Commercial software

Commercial software is classic Microsoft bread-and-butter. It must be purchased, may NOT be redistributed, and is typically only available as binaries to end users.

E.5.2. Limited trial software

Limited trial software are usually functionally limited versions of commercial software which are freely distributed and intend to drive purchase of the commercial code. Examples include 60-day time bombed evaluation products.

[1] Documentos privados de Microsoft, que vieron la luz debido a una filtración, y en los que, entre otras cosas, se desvelaban técnicas que utilizaría la compañía para luchar contra el *software* libre en general y GNU/Linux en particular

E.5.3. Shareware

Shareware products are fully functional and freely redistributable but have a license that mandates eventual purchase by both individuals and corporations. Many internet utilities (like "WinZip") take advantage of shareware as a distribution method.

E.5.4. Non-commercial use

Non-commercial use software is freely available and redistributable by non-profit making entities. Corporations, etc. must purchase the product. An example of this would be Netscape Navigator.

E.5.5. Royalty free binaries

Royalty-free binaries consist of software which may be freely used and distributed in binary form only. Internet Explorer and NetMeeting binaries fit this model.

E.5.6. Royalty free libraries

Royalty-free libraries are software products whose binaries and source code are freely used and distributed but may NOT be modified by the end customer without violating the license. Examples of this include class libraries, header files, etc.

E.5.7. Open Source (BSD-style)

A small, closed team of developers develops BSD-style open source products & allows free use and redistribution of binaries and code. While users are allowed to modify the code, the development team does NOT typically take çheck-ins"from the public.

E.5.8. Open Source (Apache-style)

Apache takes the BSD-style open source model and extends it by allowing check-ins to the core codebase by external parties.

E.5.9. Open Source (CopyLeft, Linux-style)

CopyLeft or GPL (General Public License) based software takes the Open Source license one *critical* step farther. Whereas BSD and Apache style software permits users to "fork"the codebase and apply their own license terms to their modified code (e.g. make it commercial), the GPL license requires that all derivative works in turn must also be GPL code. "You are free to hack this code as long as your derivative is also hackable"

Y el comentario siguiente a estas palabras, de Eric S. Raymond, que intenta aclarar algunos puntos:

It's interesting to note how differently these last three distinctions are framed from the way the open-source community generally views them.

To us, open-source licensing and the rights it grants to users and third parties are primary, and specific development practice varies ad-hoc in a way not especially coupled to our license variations. In this Microsoft taxonomy, on the other hand, the central distinction is who has write access to a privileged central code base.

This reflects a much more centralized view of reality, and reflects a failure of imagination or understanding on the memo-authors's part. He doesn't grok our distributed-development tradition fully. This is hardly surprising...

Que, traducido, viene a ser algo como:

Es interesante recalcar lo diferente que se representan aquí las distinciones de como las ve normalmente la comunidad del *software* libre.

Para nosotros, las licencias libres y los derechos que garantizan a los usuarios y a terceras partes son principales, y las costumbres específicas de programación varían según el momento, sin estar especialmente influidas por variaciones en las licencias. En la clasificación de Microsoft, sin embargo, la diferencia principal es quién puede escribir en un almacén de programas central y privilegiado.

Esto refleja una visión mucho más centralizada de la realidad, y una falta de imaginación y comprensión por parte del autor del artículo. No consigue comprender completamente nuestra tradición de desarrollo distribuido. Claro, que es difícilmente sorprendente...
