novática

Revista de la Asociación de Técnicos de Informática

MARZO - ABRIL 1997

126

Novática, revista fundada en 1974, es el órgano oficial y de formación continua de la Asociación de Técnicos de Informática (ATI)

ATI tiene un acuerdo de colaboración con ACM; como miembro de FESI colabora con IFIP y CEPIS, y tiene asimismo acuerdos de vinculación o colaboración con Ada-Spain y con ASTIC.

El servidor Web de ATI contiene información sobre Novática en la siguiente dirección:
http://www.ati.es/PUBLICACIONES/novatica

Coordinación Editorial, Redacción Central y Publicidad (ATI Madrid) Padilla 66, 3º, dcha. 28006 Madrid; (91)4029391; fax.3093685 / *novatica@ati.es*

Composición, Edición y Redacción ATI Valencia
Palomino 14, 2ª, 46003 Valencia
voz/fax (96)3918531 / *secreval@ati.es*

Administración y Redacción ATI Cataluña
Via Laietana 41, 1º, 1ª, 08003 Barcelona
(93)4125235; fax 4127713 / *secregen@ati.es*

Redacción ATI Andalucía
Av. República Argentina 25, 4ª, 41011 Sevilla
voz/fax (95)4273057; / *secreand@ati.es*

Redacción ATI Aragón
Lagasca 9, 3-B, 50006 Zaragoza
voz/fax (976)235181

Imprenta: CIMAGOTIPO, S.L.
Pallars, 161, 08005 Barcelona
Depósito Legal: B 15.154-1975
ISBN: 0211-2124; CODEN NOVAEC

Portada: R.Roldán; Variaciones sobre un icono de Randy Ralph

SUMARIO

Notas editoriales

ACM97: 50 aniversario de ACM

La primera semana de Marzo ATI y Novática estuvieron presentes en ACM97, congreso extraordinario de la *ACM (Association for Computing Machinery)* que ha tenido lugar en San José (California) para celebrar su 50 aniversario. Como es bien sabido, ACM es la mayor asociación profesional informática del mundo con 75.000 miembros, de los cuales 60.000 en los EE.UU., y desde 1991 tiene un acuerdo de doble afiliación con ATI. Esta presencia se enmarcaba dentro de la estrategia decidida por la Junta Directiva General de impulsar de forma autónoma las relaciones internacionales de ATI y durante la misma se han realizado contactos y gestiones que esperamos sean provechosos tanto para la ATI como para Novática.

Por una parte, se celebró una reunión de carácter protocolario con Charles House, Presidente del Council de ACM, al que nuestro Presidente, Pedro Gómez Grau, ha enviado una carta felicitándole por el 50 aniversario de dicha asociación.

En el plano asociativo, se mantuvieron varias reuniones con Lillian Israel, responsable de Programas para Socios, con el director de Publicaciones de ACM, Marc Mandelbaun, y con Nora Cortés, responsable de ACM Press Books, en las que se han tomado varios acuerdos preliminares para ampliar las relaciones entre ambas entidades. De uno concreto ya tienen noticia nuestros socios a través del boletín mensual *ATI Informa*: la oferta a los socios de ATI del libro del 50 aniversario de ACM (titulado *Beyond Calculation: The next fifty years*, con artículos de autores como Edsger Dijstra, Gordon Bell y Vinton Cerf) al precio especial que tuvo para socios de ACM durante el Congreso.

Respecto a cuestiones relacionadas con Novática, se mantuvieron contactos con la directora de *Communications of the ACM*, Diane Crawford, a partir de los cuales, entre otras cosas, se han iniciado gestiones para conseguir permiso para publicar en Novática artículos de dicha revista.

En lo que se refiere al congreso en sí era de carácter extraordinario y estaba organizado a base únicamente de conferencias magistrales de alto nivel medio a cargo de relevantes personalidades del mundo de las Tecnologías de la Información y las Comunicaciones, entre ellas Vinton Cerf, Gordon Bell, Murray Gell-Mann, Carven Mead, Brenda Laurel, Fernando Flores, etc. El lema del Congreso era *The next 50 years of Computing* y ponía por tanto el acento sobre la perspectiva del año 2047, en el que ACM cumplirá un siglo de vida. La asistencia fue de unas 1.200 personas.

Junto al congreso en sí ACM organizó también una atractiva exposición en la que estaban presente un buen número de las principales empresas, organismos públicos y centros de investigación universitaria (casi todos ellos norteamericanos) relacionados con las TIC. La exposición fue visitada por más de 30.000 personas.

Otras dos actividades paralelas merecen ser resaltadas también, sobre todo cara a la presencia de jóvenes en la vida asociativa. Se trata, en primer lugar, de la fase final del Concurso Internacional de Programación para Estudiantes, fase a la que llegaron 50 equipos de centros universitarios de todo el mundo (al parecer ningún centro español ha participado en este concurso). Cada equipo estaba formado por cuatro estudiantes, lo que hizo que la presencia de jóvenes en el congreso fuese muy significativa. Es interesante resaltar que el concurso estaba patrocinado por Microsoft, que ha invertido en él 28 millones de dólares, es decir, unos 4 millardos de pesetas. Novática comenzará a publicar, muy probablemente a partir del próximo número, los problemas planteados a los finalistas.

La otra actividad dirigida a los jóvenes fue el Concurso de Ensayos para estudiantes de Enseñanza Secundaria, que fue patrocinado por la revista *Popular Science*. Durante el congreso se anunciaron y entregaron los premios a los dos vencedores. Se ha solicitado permiso para publicar los dos ensayos en Novática.

Nota importante: Más información sobre ACM97 puede encontrarse en *www.acm.org/acm97*

Rafael Fernández Calvo; Coordinación Editorial de Novática

ATInet despega

El servicio ATInet ha pasado de su fase de instalación y rodaje a un funcionamiento correcto y rápido —tal como esperábamos— con usuarios reales. Los más de 500 socios de ATI que solicitaron han solicitado el alta en las primeras semanas tras el lanzamiento del servicio han recibido ya la confirmación del alta y las instrucciones para la conexión. La Secretaría General de ATI y los administradores del sistema están haciendo lo posible por contestar cuanto antes a todas las peticiones y consultas. Se puede empezar a utilizar ATInet en cuanto se reciben los datos del alta.

Recordamos que la conexión a ATInet se puede realizar desde cualquier punto de la red Internet. Basta con configurar el lector de correo electrónico o el navegador de WWW hacia el servidor de ATI. Y que para que todos podamos conectarnos económicamente a Internet, se ofrece el acceso mediante una llamada de teléfono local, la red Infovía de Telefónica y un proveedor de Internet. Con el alta, se está enviando un manual para configurar la conexión en el sistema Windows 95. Los usuarios con otros sistemas pueden encontrar las instrucciones necesarias en el WWW de ATI (*www.ati.es*), sección Novedades, o directamente en la páginas *http://web.ati.es/ATInet*. Ahí se pueden consultar también las instrucciones para configurar los programas más comunes de correo y noticias, así como un apartado de Preguntas Más Frecuentes (FAQs) sobre ATInet.

Los Capítulos Territoriales, conscientes de que ATI no posee los recursos necesarios para ofrecer la asistencia técnica individualizada para la instalación y solución de problemas en el ordenador del usuario o en la conexión a través de Infovía, están organizando seminarios y conferencias-coloquio para explicar los rasgos más relevantes del servicio, las pautas a seguir para la conexión y utilización, y la atención de todas la dudas, aclaraciones y sugerencias respecto a ATInet.

También se está en contacto con empresas del sector para que ayuden a resolver este problema a quienes se vean incapaces de conectarse (contactad con Secretaria General, por ejemplo, quienes necesitéis disquetes con software de Windows 3.11). De todas formas, recomendamos vivamente a quienes tengan algún problema instalando la conexión que, en vez de pedir ayuda en las Secretarías de ATI (que no están preparadas en absoluto para poder darla), la pidan en su lugar a algún compañero que sepa de estos asuntos... Y mejor si se trata de alguien que sea también socio de ATI, pues entonces no sólo será probable que conozca ya cómo hacer esta conexión específica a ATInet, ¡sino que además iremos de esa forma construyendo entre todos una ATI con relaciones más fuertes entre sus socios!

Nacho Navarro; Responsable Técnico de ATInet

Software Libre

Presentación

Jesús M. González Barahona, Pedro de las Heras Quirós
Grupo de Sistemas y Comunicaciones, Dpto. de Informática, Universidad Carlos III de Madrid

{jgb,pheras}@gsyc.inf.uc3m.es

En el mes de septiembre de 1995 se celebraron en la Universidad Carlos III unas jornadas sobre Software Libre. En aquel momento surgió la idea de realizar un número monográfico para Novática dedicado a este tema. Después de año y medio aquí está el resultado de aquella idea.

Pero antes de continuar conviene que definamos qué entendemos por software libre; no es un asunto sencillo. Precisamente, uno de los artículos incluidos estudia los diversos tipos de software "más o menos libre" y las diferentes licencias bajo las que se distribuyen. Por ahora, podríamos incluir bajo este nombre al software cuyo autor decide (quizá bajo ciertas condiciones) que:

- Cualquiera puede copiar y redistribuir tanto el código fuente como el binario que resulta de compilarlo.
- Cualquiera puede realizar modificaciones al fuente original, copiarlas y redistribuirlas.

Para algunos, estas condiciones pueden parecer sorprendentes, pues coinciden casi exactamente con las opuestas a las que hemos considerado habituales desde que empezamos a movernos en este mundo de la informática. La mayoría de las licencias del software "comercial" o mejor, "propietario" (dado que, como se verá a lo largo de este número, el software libre también puede ser comercial) prohiben explícitamente la copia y desde luego no nos proporcionan el más mínimo acceso a los fuentes de la aplicación. Y sin embargo hubo un tiempo (y no hace tanto —nuestro campo es aún relativamente joven) en el que todo el software era libre, habitualmente se distribuían los fuentes de los programas y cualquier programador se sentía orgulloso de enseñar su código a otro programador y de motivarle lo suficiente como para que este otro lo mejorase de alguna manera. Posteriormente esta situación cambió cuando algunos pensaron que podían conseguir más recursos impidiendo el acceso al código fuente y aplicando las leyes de *copyright* a los programas con el fin de evitar su copia.

Pero ni siquiera en esas épocas el software libre se vio totalmente aniquilado. En algunos reductos (fundamentalmente en Universidades estadounidenses, alrededor de la comunidad Unix) se mantuvieron varios grupos donde, siempre que era posible, el software seguía distribuyéndose libremente.

A comienzos de los años ochenta dos interesantes fenómenos empezaron a influir notablemente en el software libre: la organización de sus desarrolladores y usuarios alrededor de grupos como GNU o BSD y la utilización de Internet. Los primeros han propiciado el intercambio de ideas, la distribución de código, el establecimiento de una infraestructura legal, etc. Internet facilitó no sólo la comunicación entre los interesados por el software libre, sino su distribución a un coste prácticamente nulo.

Como fruto de estos dos factores, hacia finales de los ochenta y principios de los noventa ya se podía encontrar una gran cantidad de software libre en la Red. Sin embargo, dos nuevos fenómenos que tuvieron lugar a principios de los noventa están marcando, junto con los ya mencionados, una nueva etapa de expansión: la aparición de empresas dedicadas al mantenimiento y desarrollo de software libre y la aparición de sistemas operativos libres. Las empresas están demostrando que es posible sobrevivir (y prosperar) en el negocio de la informática con un modelo de negocio nuevo, basado en el soporte a usuarios. En este monográfico contamos con artículos invitados de varios presidentes de este tipo de empresas. La aparición de sistemas operativos libres, junto con una gran cantidad de software acompañándoles, ha permitido que un ordenador funcione sólo con software libre. Asimismo, han permitido que un gran número de usuarios pueda utilizar software completamente libre y de gran calidad. Dicho todo esto, ¿qué ventajas (si hay alguna) proporciona el software libre frente a otros enfoques más "tradicionales"? Podemos destacar las siguientes:

- Se pone el énfasis en el mantenimiento y soporte a usuarios, dado que no se obtienen ingresos por la distribución.
- Cualquiera puede hacer mejoras a los programas y estará incentivado para enviarlas a los desarrolladores "principales", pues así se incluirán en futuras versiones.
- Al distribuirse a unos costes muy bajos, o prácticamente nulos, el nivel de "penetración" en el mercado es muy amplio: muchos usuarios estarán dispuestos a probarlo y si les gusta lo seguirán utilizando. Es así más fácil que estos productos se coloquen como líderes en algún sector del mercado. Un ejemplo en el mundo del software propietario es el navegador **Netscape**.

Pero, si no se obtienen ingresos por la venta de copias del programa, ¿de dónde se obtienen si no? O, dicho de otro modo, ¿es posible un modelo comercial que garantice el desarrollo de software libre? Algunos de los artículos que se presentan en este número pretenden responder precisamente a esta pregunta. Dejemos que el lector los lea y saque sus propias conclusiones.

Los artículos que componen este número provienen de dos fuentes: por un lado se hizo una petición pública de artículos, entre los que se seleccionaron los siete que hemos incluido en este número. Los revisores son los siguientes: Joaquín Seoane (UPM, Madrid), Javier Miranda (ULPGC, Las Palmas), Ricardo Baeza Yates (UCHILE, Santiago de

Chile), José Miguel Alonso (UPV/EHU, San Sebastián), Pedro Álvarez Tabío (UNICAN, Santander).

También contamos con cinco artículos invitados. Esperamos que todos ellos sirvan para acercar al lector al mundo del software libre.

Artículos invitados

Comenzamos el monográfico con los artículos invitados. El artículo de **Richard Stallman** -"Por qué el software libre no debería tener propietarios"- rebate pormenorizadamente el sistema actual de distribución de software basado en propietarios y explica las ventajas del modelo de distribución alternativo basado en software libre. En el siguiente artículo **Peter Deutsch** ofrece un análisis detallado de las diversas licencias que se usan en la actualidad para distribuir software libre. **Robert Young** y **Michael Tiemann** ofrecen en sendos artículos la visión de dos empresas que desarrollan su actividad en el mundo del software libre: *Red Hat* y *Cygnus Solutions,* respectivamente. **Edmond Schonberg** describe el software que desarrolló con su equipo de la Universidad de Nueva York y que hoy en día se distribuye como software libre, soportado comercialmente por la compañía que ellos mismos crearon, *Ada Core Technologies.* **José Ramón Portillo** y **Antonio González** concluyen la sección de artículos invitados introduciéndonos en el mundo de TEX y LATEX, el software de tratamiento de textos, distribuido como software libre, con el que se han editado la mayor parte de los contenidos de este monográfico, así como, entre otros, la mayor parte de las publicaciones científicas de física y matemáticas, publicaciones de informática y otras ingenierías, o los libros de la editorial *Addison Wesley.*

Artículos revisados

Los primeros tres artículos revisados describen experiencias de uso del software libre en diversos ámbitos: el primero de ellos describe la experiencia de los autores en **Red EuroSur**, un proveedor de información y de acceso a Internet que funciona íntegramente con software libre. En "Utilización de Software de Dominio Público en la Administración del Estado" se analizan diversos aspectos legales relativos al uso de software libre en las administraciones públicas y se incluyen algunas experiencias específicas del autor en este entorno. En "SoftwareLibre en la Investigación: Alternativa de Calidad", se detalla la experiencia del autor en el uso de software libre en el mundo de la astrofísica.

Los últimos cuatro artículos describen software específico, en algunos casos realizado por los propios autores: "COES: Herramienta Lingüística de Libre Distribución para la Lengua Española" describe una herramienta para la corrección de textos escritos en lengua castellana, distribuida bajo la licencia GPL y desarrollada por los autores del artículo. En "Software de Dominio Público Orientado a la Seguridad" se expone una extensa panorámica del software libre que se utiliza en la actualidad en el campo de la seguridad informática. "Experiencias en el desarrollo de software de dominio público para uso docente en las enseñanzas universitarias" describe una herramienta desarrollada por los autores que se utiliza como soporte a la docencia en la Universidad Politécnica de Valencia. "Simuladores de Robo-Fútbol y otro Software Libre en la Inteligencia Artificial" utiliza como ejemplo los simuladores de libre distribución de las competiciones de robots autónomos inteligentes, para analizar el impacto que ejerce el software libre en el desarrollo de la investigación en el campo de la inteligencia artificial y de las ciencias en general.

Versión WWW

El contenido íntegro de este monográfico, así como otro material suplementario que no ha podido editarse por razones de espacio, incluidos algunos artículos revisados, puede consultarse también en Internet, en las URLs *http://www.gsyc.inf.uc3m.es/sobre/novatica* y, de forma mucho más limitada, en *http://www.ati.es.* Aprovechando las posibilidades de comunicación bidireccional invitamos al lector a que aporte sus comentarios, utilizando los mecanismos habilitados en la versión **WWW**. Así podrán ser consultados por lectores y autores. Esperamos que se establezca un foro de debate que dé continuidad a este monográfico. Invitamos asimismo a los lectores a debatir los asuntos tratados en estas páginas a través de las listas de correo electrónico de los socios de ATI (*ati@ati.es*) y de pdsoft (*pdsoft@eunet.es*).

Para terminar nos gustaría manifestar nuestro agradecimiento a todos los que han ayudado a que aparezca este monográfico. Entre ellos podemos destacar a Rafael Fernández Calvo, coordinador editorial de Novática, que desde el principio apoyó la idea, a nuestros compañeros del GSyC, que nos han ayudado de muchas maneras, al grupo de revisores que ha hecho posible que la calidad de los artículos publicados sea más que razonable, a todos los que respondieron a la petición de artículos y por supuesto a todos los autores invitados. Pero, fundamentalmente, queremos hacer llegar nuestro agradecimiento a las personas que desde hace mucho tiempo han escrito, difundido y mantenido software libre. Sin ellos este monográfico no tendría razón de ser. Gracias a ellos un nuevo mundo de expectativas y realidades se abre ante la comunidad informática y ante la sociedad en general.

Breves biografías de los autores de artículos invitados

Richard Stallman: Richard Stallman es presidente de la Free Software Foundation y fundador del proyecto GNU, iniciado en 1984 para desarrollar un sistema operativo libre de tipo Unix. Stallman desarrolló el primer editor Emacs en 1975 y es el autor principal de GNU Emacs, del compilador GNU C, de GDB y de otros programas GNU.

Peter Deutsch: Peter Deutsch se doctoró en informática por la Universidad de California en Berkeley en 1973. Su carrera se ha desarrollado trabajando en sistemas operativos en Berkeley (Sistema SDS 940 de tiempo compartido) y en entornos integrados de programación (Interlisp, Cedar Mesa y Smalltalk-80) en Xerox PARC y ParcPlace Systems. Desde 1986 el doctor Deutsch es presidente de Aladdin Enterprises, una compañia consultora que ofrece, bajo licencia, una implementación portable de alto rendimiento del lenguaje

PostScript, desarrollada por eldoctor Deutsch y que también está disponible bajo una licencia libre a través de la Internet con el nombre de Ghostscript. El doctor Deutsch fue cogalardonado con el premio ACM Software System en 1992, por su trabajo en Interlisp y en1993 fue nombrado Alumno Distinguido del programa de informática de la Universidad de California en Berkeley. El doctor Deutsch es miembro de ACM, IEEE, CPSR (Computer Professionals for Social Responsibility) y de la LPF (League for Programming Freedom).

Robert Young: Robert Young es presidente de Red Hat Software Inc, empresa líder en el campo del suministro a empresas de software distribuible libremente. Ha trabajado en la venta de ordenadores y servicios informáticos durante 20 años, comenzando con miniordenadores Digital en 1977. Está convencido de que Linux es el siguiente paso en el progreso del modelo de Sistemas Informáticos Abiertos.

Michael Tiemann: Michael Tiemann se ha sentido fascinado por los ordenadores desde los 12 años. Aprendió a programar en BASIC, FORTRAN, PL/1, Pascal y C leyendo el código fuente que se publicaba en revistas y modificando estos programas para que funcionasen en el ordenador personal de su padre. Mientras estudiaba en la Universidad de Pennsylvania recibió la primera beca de investigación para no graduados por su trabajo en la construcción de generadores de compiladores basado en gramáticas de atributos. Se graduó en Ciencias, Informática e Ingeniería en 1986 y trabajó como investigador de plantilla en Microelectronic and Computer Technology Corporation (MCC, Austin, Texas, EEUU), en L'Institute Nationale pour Recherche en Informatique et en Automatique (INRIA, París, Francia) y en Sun Microsystems (Moutain View, California, EEUU). Fue cofundador de Cygnus Solutions en 1989 y su presidente hasta 1995. Actualmente es Director de Marketing Técnico en Cygnus Solutions y su página Web es *http://www.cygnus.com/~tiemann/*

Edmond Schonberg: Edmond Schonberg es profesor de informática en la Universidad de Nueva York. Ha participado en el diseño, definición e implementación del lenguaje Ada desde sus comienzos. Codirige con Robert Dewar el equipo que ha construido GNAT, el primer compilador completo para Ada95 y la empresa de consultoría ACT (Ada Core Technologies), cuyo objetivo es el soporte y mantenimiento de GNAT. El doctor Schonberg obtuvo un doctorado en física por la Universidad de Chicago y el grado de bachiller en música por el Conservatorio Nacional de Lima, Perú.

José Ramón Portillo: José R. Portillo Fernández nace en Cádiz (España) el 13 de Septiembre de 1960. Es licenciado en Ciencias Matemáticas por la Universidad de Sevilla y profesor titular del Dpto. de Matemática Aplicada 1 en su Escuela Técnica Superior de Arquitectura. Su actividad investigadora se centra actualmente en Matemática Discreta, concretamente en Teoría de Ramsey y Problemas Extremales de Teoría de Grafos. Coordina el grupo de usuarios hispanoparlantes de TEX desde 1994, fecha de su creación y mantiene las páginas web de dicho grupo, una de las mejores fuentes de información sobre TEX en español.

Información en la red

Organizaciones en la red relacionadas con el SL

- FSF-GNU: *http://www.fsf.org, news:gnu.announce*
- Freeware Central: *http://www.ptf.com/free/*
- Free Software Business Archives: *http://www.apocalypse.org/pub/fsb*
- Free Software Union (FSU): *http://www.jaguNET.com/~braddock/fslu/org*
- Usenix: Asociación Técnica y Profesional de Usuarios de Unix.
- League for Programming Freedom: *http://www.lpf.org*
- Linux: *http://www.linux.org, news:comp.os.linux*
- CervanTeX: Grupo de Usuarios de TeX Hispanoparlantes: *http://gordo.us.es/Actividades/GUTH*
- TeX Users Group: *http://www.tug.org*
- Spanish GNU: Traducciones de Software GNU en España: *http://slug.ctv.es/~emelero*

Compañías que apoyan el desarrollo de software libre

- Ada Core Technologies: *http://www.gnat.com*
- Cyclic Software: *http://www.cyclic.com*
- Cygnus Solutions: *http://www.cygnus.com*
- Signum Support AB: *http://www.signum.se*

Software: Sistemas Operativos

- Linux: *http://www.linux.org*
- Net/BSD: *http://www.netbsd.org*
- FreeBSD: *http://www.freebsd.org*
- Gnu Hurd: *http://www.cs.pdx.edu/~trent/gnu/hurd/*
- Mach4: *http://www.cs.utah.edu/projects/flux/mach4/html/Mach4-proj.html*
- Otros sistemas operativos libres: *http://funnelweb.utcc.utk.edu/~williams/freeos/*

Software: Herramientas y Aplicaciones

- 120.000 títulos de Software Libre y Shareware: *http://www.shareware.com/*
- Archivo de Software de Walnut Creek: *http://www.cdrom.com/archive/index.htm*
- GUAVAC: Compilador de Java: *http://http.cs.berkeley.edu/~engberg/guavac*
- Sistema de ventanas X Window: *http://www.x.org*
- Apache: Servidor de WWW: *http://www.apache.org*
- Postgres: Base de Datos: *http://www.postgresql.org*
- Programación de interfaces de usuario: TclTk: *http://www.sunlabs.com/research/tcl/*
- Programación de interfaces de usuario: GNUStep: *http://www.gnustep.org*
- Concurrent Versions System (CVS): *http://www.cyclic.com/cyclic-pages/CVS-sheet.html*
- Povray, 3D: *http://www.povray.org*
- Samba: servidor compatible con Microsoft Lanman: *http://lake.canberra.edu.au/pub/samba/*
- Herramientas tipo Unix para Windows: *http://www.nentug.org/unix-to-nt/*
- TeX: *http://www.tug.org*
- PERL: *http://www.perl.org*
- Compilador GCC (C, C++, Objective C, Modula, Pascal, Fortran): *http://www.fsf.org/software/software.html*
- Compilador de Ada Gnat: *http://www.gnat.com*
- Bison, Emacs, DJGPP, Guile y el resto del software GNU: *http://www.fsf.org/software/software.html*

Eventos: Conferencias, Talleres

3rd Annual Linux Expo. Abril 1997. *http://www.linuxexpo.org*

4th Tokyo GNU Seminar. Marzo 1997. *http://csrl.ge.aoyama.ac.jp/FSFseminar/index.html*

Second Conference on Freely Redistributable Software. Febrero de 1996. Organizado por la Free Software Foundation. *http://www.fsf.org/conferences/97san-fran/97san-fran.html*

First Conference on Freely Redistributable Software. Febrero de 1996. Organizado por la Free Software Foundation. *http://www.fsf.org/conferences/96cambridge/96cambridge.html*

Taller sobre el Software de Libre Distribución. Septiembre de 1995. Organizado por el Grupo de Sistemas y Comunicaciones del Departamento de Informática de la Universidad Carlos III de Madrid. *http://www.gsyc.inf.uc3m.es/~odwfs*

Software Libre

Richard Stallman

Por qué el Software no debería tener propietarios

La tecnología de la información digital contribuye a la sociedad haciendo que sea más fácil copiar y modificar la información. Las computadoras prometen hacer que esto sea más fácil para todos nosotros.

Pero no todo el mundo quiere que sea más fácil. El sistema de *copyright* asigna "propietarios" a los programas software y muchos de estos "propietarios" pretenden negar el beneficio potencial del software al resto del público. Les gustaría ser los únicos que pueden copiar y modificar el software que utilizamos.

El sistema de *copyright* se desarrolló con la imprenta -una tecnología para producir copias en masa. El *copyright* funcionaba bien con esta tecnología porque sólo restringía a los productores masivos de copias. No coartaba la libertad de los lectores de libros. Un lector corriente, que no poseyese una imprenta, sólo podía copiar libros a mano, con pluma y tinta y pocos lectores fueron demandados por ello.

La tecnología digital es más flexible que la imprenta: cuando la información tiene un formato digital, ésta se puede copiar fácilmente para compartirla con los demás. Es esta flexibilidad la que no encaja con un sistema como el del *copyright*. Esta es la causa de las cada vez más repugnantes y draconianas medidas que se utilizan ahora para hacer respetar el *copyright* del software. Ténganse en cuenta estas cuatro prácticas de la Asociación de Editores de Software **(SPA: Software Publishers Association)**:

- Propaganda masiva diciendo que no se debe desobedecer a los propietarios para ayudar a los amigos.
- Ofertas a soplones para que informen acerca de sus compañeros y colegas.
- Redadas en oficinas y escuelas (con la ayuda de la policía), en las que se conmina a la gente a que pruebe que son inocentes, que no han realizado copias ilegales.
- Persecución (por parte del gobierno de los EE.UU., a petición de la SPA) de gente como David LaMacchia, del **MIT (Massachusetts Institute of Technology)**, por el simple hecho de no vigilar los servicios de realización de copias y no censurar su uso.

Estas cuatro prácticas se asemejan a las utilizadas en la antigua Unión Soviética, donde toda máquina copiadora estaba vigilada por un guarda para evitar copias prohibidas y donde los individuos tenían que copiar la información secretamente y pasarla de mano en mano como *samizdat* (**N. del T.:** *samizdat* es un vocablo ruso que significa "publicado por uno mismo"). Existe por supuesto una diferencia: el motivo por el que se controlaba la información en la Unión Soviética era político; en los EE.UU. el motivo es el lucro.

Pero son las acciones las que nos afectan, no los motivos. Cualquier intento de bloquear la compartición de la información, sin importar el por qué, conduce a los mismos métodos y a la misma severidad.

Los propietarios esgrimen diversos argumentos para otorgarse el poder de controlar cómo usamos la información. Veámoslo.

Insultos

Los propietarios utilizan calumnias como "piratería" y "robo", así como terminología pericial como "propiedad intelectual" y "perjuicio", para sugerir una línea de pensamiento al público -una analogía simplista entre programas y objetos físicos. Nuestras ideas e intuiciones acerca de la propiedad de los objetos materiales se centran en si es correcto o no quitarle un objeto a alguien. No se aplican directamente a la copia de los objetos. Pero los propietarios quieren que lo apliquemos también a la copia.

Exageración

Los propietarios dicen que sufren "daños" o "pérdidas económicas" cuando son los usuarios los que copian los programas. Pero la copia no causa un efecto directo en el propietario y no daña a nadie. El propietario sólo puede perder si la persona que hizo la copia hubiese pagado por una proporcionada por el propietario. Reflexionando un poco, la mayor parte de esas personas no habrían comprado copias. Sin embargo, los propietarios calculan sus "pérdidas" como si todos y cada uno de los usuarios hubiesen comprado una copia. Esto es una exageración (por decirlo suavemente).

La ley

Los propietarios muestran a menudo la situación legal actual y las severas penas con las que nos pueden amenazar. En este enfoque está implícita la sugerencia de que la ley refleja una moralidad incuestionable, aunque al mismo tiempo somos instados a considerar estas penas como algo natural que no puede ser imputado a nadie. Esta forma de persuasión no está diseñada para ser rebatida por un pensamiento crítico, sino que tiene la intención de reforzar una manera habitual de pensar.

Es elemental que las leyes no deciden entre lo que está bien y lo que está mal. Todo estadounidense debería saber que hace cuarenta años era ilegal, en muchos Estados, que una persona de raza negra se sentase en la parte delantera de un autobús, aunque sólo los racistas dirían que sentarse allí era algo malo.

Derechos innatos

Los autores reivindican a menudo una relación especial con los programas que escriben y afirman que por lo tanto sus deseos e intereses con respecto a los programas son mayores que los de cualquier otro. Mayores incluso que los de todos los demás (normalmente, las compañías, y no los autores, son quienes poseen los derechos de autoría del software, pero se espera que pasemos por alto esta discrepancia).

A aquellos que proponen esto como un axioma ético ("el autor es más importante que tú") sólo puedo decirles que yo, un notable autor de software, digo que es una bobada.

Pero hay únicamente **dos razones** por las que el público en general podría sentir alguna simpatía por la reivindicación de derechos innatos:

- Una razón es la **analogía**, cogida por los pelos, con los objetos materiales. Cuando cocino espagueti, pongo objeciones a que alguien se los coma, porque entonces no me los puedo comer yo. Su acción me perjudica tanto a mí como a él le beneficia. Sólo uno de los dos puede comerse los espagueti, por lo que la pregunta es ¿quién? La más mínima diferencia entre nosotros es suficiente para inclinar la balanza ética en uno u otro sentido.

Pero el hecho de que utilices o cambies un programa que yo escribí, te afecta directamente a ti y a mí sólo me afecta indirectamente. Si le das una copia a tu amigo, os afecta a ti y a tu amigo mucho más de lo que me afecta a mí. Yo no debería tener el poder de decirte que no hagas esas cosas. Nadie debería tenerlo.

- La **segunda razón** es que se le ha dicho a la gente que los derechos innatos de los autores son una tradición incuestionable en nuestra sociedad.

Históricamente, es justamente al contrario. La idea de los derechos innatos de los autores se propuso y se rechazó contundentemente cuando se redactó la Constitución de los EE.UU. Por ello la Constitución permite un sistema de *copyright*, pero no requiere uno. Por eso dice que el *copyright* debe ser temporal. También dice que el propósito del *copyright* no es recompensar a los autores, sino promover el progreso. El *copyright* recompensa en alguna medida a los autores y mucho más a los editores, pero como una medida para modificar su comportamiento.

La auténtica tradición establecida en nuestra sociedad es que el *copyright* coarta los derechos innatos del público y que esto sólo se puede justificar si es en beneficio de la sociedad.

Aspecto económico

La última razón argüida a favor de los propietarios de software es que así se consigue producir más software.

A diferencia de los otros argumentos, éste al menos adopta un enfoque legítimo sobre el asunto. Se basa en una meta defendible, para satisfacer a los usuarios de software. Y es demostrable empíricamente que si se remunera adecuadamente la producción de un bien, se producirá más cantidad del mismo.

Pero el razonamiento económico tiene un defecto: se basa en la suposición de que la diferencia solamente radica en cuánto dinero tenemos que pagar. Asume que lo que se desea es la "producción de software", sin importar si éste tiene o no propietarios.

La gente acepta fácilmente esta suposición porque concuerda con nuestra experiencia con los objetos materiales. Consideremos por ejemplo un bocadillo. Supongamos que el mismo bocadillo puede ser obtenido, bien de manera gratuita, o bien pagando. En este caso, la única diferencia entre ambos bocadillos es la cantidad que se paga por cada uno de ellos. El bocadillo tendrá el mismo sabor y el mismo valor nutritivo, sea comprado o no y en ambos casos el bocadillo sólo podrá ser ingerido una vez. El hecho de que sea el propietario quien nos proporciona el bocadillo, no afecta directamente más que a la cantidad de dinero que acabaremos teniendo al final.

Esto es cierto para cualquier objeto material: el hecho de que tenga o no un propietario no afecta directamente a lo que es, o a lo que se puede hacer con él si se adquiere.

Pero el que un programa tenga o no propietarios afecta a lo que es y a lo que se puede hacer con una copia si se compra. La diferencia no es sólo una cuestión de dinero. El sistema de propietarios de software alienta a estos a producir algo, pero no a producir aquello que necesita realmente la sociedad. Y esto provoca una polución ética intangible que nos afecta a todos.

¿Qué necesita la sociedad? Necesita información que esté realmente disponible para sus ciudadanos. Por ejemplo, programas que la gente pueda leer, corregir, adaptar y mejorar, no sólo utilizar. Pero lo que normalmente distribuyen los propietarios es una caja negra que no podemos estudiar o modificar.

La sociedad también necesita libertad. Cuando un programa tiene propietarios, los usuarios pierden la libertad de controlar parte de sus propias vidas.

Y por encima de todo, la sociedad necesita alentar el espíritu de cooperación voluntaria entre sus ciudadanos. Cuando los propietarios de software nos dicen que la ayuda a nuestros vecinos es una forma de "piratería", están corrompiendo el espíritu cívico de nuestra sociedad.

Por ello decimos que el software libre se refiere a las libertades y no a la gratuidad (**N. del T.:** en inglés, el vocablo *free* es polisémico, pudiéndose entender *free software* como software gratuito o como software libre. De ahí la aclaración que hace el autor).

El argumento económico que esgrimen los propietarios es erróneo, pero el problema económico general es real. Hay gente que escribe software de utilidad por el placer de

escribirlo o por admiración y amor. Pero si queremos tener más software que el que esta gente escribe, necesitamos conseguir fondos para ello.

Hace ya diez años que los desarrolladores de software libre vienen utilizando varios métodos para buscar financiación, habiendo conseguido algunos éxitos. No es necesario hacer rico a nadie; el ingreso anual medio de una familia estadounidense (alrededor de los 35.000$) parece ser suficiente incentivo para muchos trabajos que son menos satisfactorios que la programación.

Durante varios años yo viví de realizar mejoras a medida del software libre que había escrito, hasta que una beca lo hizo innecesario. Cada mejora se añadía a la versión estándar que se distribuía y acababa estando disponible para el público en general. Los clientes me pagaban para que realizase las mejoras que ellos querían, en lugar de las que yo habría considerado como más prioritarias.

La **Free Software Foundation**, una fundación exenta de impuestos para el desarrollo de software libre, obtiene sus ingresos mediante la venta de CD-ROM, camisetas y manuales (todos los cuales pueden ser copiados y alterados libremente por los usuarios) y mediante las donaciones que recibe. Actualmente tiene una plantilla de cinco programadores, más tres empleados que gestionan las peticiones por correo.

Algunos desarrolladores de software libre obtienen sus ingresos de la venta de servicios de soporte. **Cygnus Solutions**, con unos 50 empleados, estima que alrededor del 15 por ciento de la actividad de su plantilla se dedica a la realización y mejora de software libre -un porcentaje respetable para una compañía de software.

Varias compañías, incluyendo **Intel**, **Motorola**, **Texas Instruments** y **Analog Devices**, se han aliado para financiar el mantenimiento continuado del compilador libre de **GNU** para el lenguaje C. Mientras tanto, el Ejército del Aire de los EE.UU. está financiando el compilador de GNU para el lenguaje **Ada**, por pensar que esta es la forma más económica de obtener un compilador de calidad. (La financiación terminó hace algún tiempo; el compilador de Ada de GNU está actualmente funcionando y su mantenimiento se financia comercialmente).

Todos estos son pequeños ejemplos. El movimiento del software libre es aún reducido y joven. Pero el ejemplo de las cadenas de radio mantenidas por los oyentes de este país (EE.UU.) muestra que es posible mantener una gran actividad sin forzar a que cada usuario pague.

Como usuario actual de computadoras, puede que estés utilizando un programa propietario. Si tu amigo te pidiese una copia, estaría mal que te negases a hacérsela. La cooperación es más importante que el *copyright*. Pero la cooperación clandestina, encubierta, no contribuye a formar una buena sociedad. Cualquier persona debería aspirar a vivir abiertamente, erguido, con orgullo y esto significa decir "no" al software propietario.

Mereces poder cooperar abierta y libremente con otras personas que utilizan software. Mereces poder aprender cómo funciona el software y enseñar a tus estudiantes con él. Mereces poder contratar a tu programador favorito para arreglarlo cuando falle.

Te mereces el software libre.

Oferta equilibrada.

Una de las claves de Informat 97 será la distribución inteligente del espacio.

Pabellón EXPERT

Estará dirigido a los profesionales de la distribución y a los responsables de los temas informáticos de empresa.

s e c t o r e s :

- Informática Corporativa
- Telecomunicaciones
- Networking
- Intranet
- CAD
- Distribución y Retail

Pabellón OPEN

Albergará lo último en equipos, programas y servicios dirigidos a usuarios de informática personal y doméstica en general.

s e c t o r e s :

- Informática Personal
- Color e Imagen
- Multimedia y edición en CD
- Internet: Servicios y Contenidos
- Ofimática y Reprografía
- Consumibles y Accesorios

Informat 97

SALÓN INTERNACIONAL DE LAS TECNOLOGÍAS DE LA INFORMACIÓN

Barcelona, del 22 al 26 de Abril

EN MONTJUÏC-2

Fira de Barcelona

INFORMACIÓN: Avda. Reina María Cristina, s/n 08004 Barcelona (España) Tel.(93) 233 20 00 Fax (93) 233 23 19

IBERIA LINEAS AEREAS DE ESPAÑA

tve

Software Libre

Tipos de licencias para software redistribuible libremente *

L. Peter Deutsch
Aladdin Enterprises

lpd@aladdin.com

(* Este artículo es una traducción adaptada de un artículo aparecido en las actas de la "First Conference on Freely Redistributable Software", Cambridge, Massachusetts, 3-5 Febrero, 1996)

Resumen: *Los autores de software redistribuible libremente* ***(SRL)*** *han decidido distribuir su trabajo bajo varias licencias diferentes. Esta variedad conduce a la pregunta de qué significa "distribuible libremente". En este artículo se consideran licencias que, como mínimo, permiten el uso personal sin restricciones ni pagos, y la redistribución no comercial.*

Diferentes propósitos filosóficos y/o económicos han conducido a diferentes licencias de redistribución libre ***(LRL)****, variando en función de las actividades que requieren, permiten o prohiben respecto al uso y distribución. Cada una de las diferentes licencias conduce a diferentes patrones de uso y distribución del software, y a diferentes beneficios para las partes involucradas (autores,* ***OEMs****, otros redistribuidores y usuarios).*

El análisis de las LRLs existentes ofrece a los autores de SRL una imagen más nítida de las consecuencias que se derivan de la elección de una u otra. También se especula sobre porqué las LRL orientadas a código fuente tienden a asociarse con software orientado a desarrolladores, y las LRL menos "libres", con software orientado a usuarios finales.

1. Introducción

En los análisis económicos convencionales, dado un mercado libre con competencia en la oferta y sin factores que compliquen el modelo, el precio de los bienes cae según pasa el tiempo hasta llegar a un valor marginalmente superior al coste de producción. Sin embargo, esto claramente no es aplicable directamente a bienes cuyo coste marginal de producción es nulo, como el software. Si multiplicamos un precio nulo por cualquier cantidad, resultan beneficios nulos, y eso no cubre los costes de desarrollo. Sin embargo sólo desarrollando nuevo software los productores existentes pueden permitirse seguir en el negocio.

Los economistas discrepan radicalmente sobre la forma más apropiada de cubrir los costes de desarrollo de los bienes con costes de producción despreciables, e incluso más radicalmente sobre cómo proporcionar incentivos apropiados a los desarrolladores. En la industria del software la respuesta convencional ha sido ponerle precio al software basándose en lo que pueda aguantar el mercado, en lugar de basarlos en los costes de producción. Para ello se apoyan en barreras ajenas al mercado, tanto legales (como los derechos de autor y la protección de los secretos industriales) como técnicas (como la compatibilidad y la interoperabilidad) para prevenir la copia libre. Sin embargo, un pequeño pero significativo número de autores de software ha elegido permitir la copia y redistribución libres de sus creaciones bajo alguna **LRL**, utilizando algún otro método que no sea el pago por copia para recuperar los costes de desarrollo y proporcionar incentivos económicos para continuar el desarrollo.

El software distribuido bajo una LRL, tal y como se usa el término en este artículo, no prevé una compensación directa para el autor por cada copia. Por lo tanto, la pregunta de cómo puede ser recompensado el autor (y otros participantes en el proceso del software, como los distribuidores y los proveedores de soporte), es de una importancia crucial a la hora de entender las diferencias entre las LRLs. Todas las LRLs que se han considerado aquí cumplen las siguientes condiciones:

- Permiten la copia y la distribución del software sin pagar al autor.
- Permiten el uso del software sin pagar (salvo algunas restricciones en el caso del *shareware*).
- Permiten cobrar por bienes (documentación impresa o soporte físico) y servicios (consultoría, soporte o distribución) que no son el propio software sujeto a la LRL.

Así las cosas, los autores son los únicos participantes que no reciben recompensa por su trabajo según el número de copias o de servicios realizados. Por otra parte, también son los únicos participantes cuyo trabajo no está en relación con el número de copias o de servicios, por lo que la situación es consistente con la observación anterior relativa a los costes marginales. Volveremos sobre este punto en secciones posteriores, en las que discutiremos la base lógica de las diferentes formas de LRL. Dada la gran cantidad de software redistribuible libremente (SRL) que se ha creado y diseminado, este artículo propone:

- Examinar y clasificar las diversas LRLs que utilizan los desarrolladores para controlar las condiciones bajo las que se puede redistribuir de hecho su **SRL**.
- Identificar los tipos de SRL distribuidos con cada LRL.
- Explorar los puntos de vista de los desarrolladores acerca de los beneficios y recompensas tanto filosóficos como económicos que parecen motivar los distintos tipos de LRLs.
- Arrojar algo de luz sobre los beneficios y recompensas que pueden obtenerse usando las diferentes LRLs.
- Y finalmente, hacer algunas sugerencias a los autores de software sobre qué formas de LRL (si alguna) son las más apropiadas, dependiendo de sus metas individuales.

2. Familias de LRLs

Todas las LRLs tienen en común el requisito de mantener, entre otros, los anuncios de *copyright* adosados al software, incluyendo la propia LRL, que debe distribuirse en todas las copias, junto con un repudio de garantía. Por otro lado, las LRLs difieren en:

- Las condiciones impuestas a la redistribución. Por ejemplo, el requisito de la **Licencia GNU** de mantener disponible el código fuente, o la prohibición de la "distribución comercial" de las licencias limitadoras del ánimo de lucro.
- Las circunstancias bajo las cuales se exige pago (por ejemplo, después de un período, o si se hace un uso comercial, como en la **licencia de PKZIP**), se sugiere (como en una licencia típica de *shareware*), o incluso se prohíbe (como en las licencias **GNU** y **Aladdin**).
- Las circunstancias bajo las que se puede o se debe redistribuir el código fuente.

Teniendo en cuenta estas características se han identificado **cuatro grandes grupos de LRLs** que se utilizan actualmente, y que pasamos a examinar por separado.

2.1. Licencias sin restricciones

Además de los requisitos genéricos de las LRLs, las licencias sin restricciones sólo suelen requerir un reconocimiento del autor. El SRL distribuido con licencias de este tipo incluye muchos paquetes de uso extendido en Internet, como **XWindow**, el intérprete **Tcl** y la biblioteca **Tk**, y las bibliotecas **IJG JPEG**, **PNG** y **zlib**.

- Condiciones de redistribución: a veces ninguna, a veces sólo un reconocimiento de autoría. Se permite incorporarlos en productos comerciales.
- Pago: ninguno.
- Código fuente: normalmente disponible. Se permite su redistribución, pero no se exige.

Los autores parecen querer preservar únicamente la integridad del software y asegurarse de que se les reconoce su trabajo. No buscan una compensación directa por el SRL que han desarrollado.

2.2. Licencias tipo *shareware*

El término *shareware* se refiere al software que es accesible por cualquiera, pero por el que sugiere o exige pago después de usarlo. El *shareware* es el único tipo de SRL que casi nunca incluye código fuente y que impone condiciones al uso. Muchos paquetes disponibles a través de **BBSs** utilizan licencias *shareware*. Sin embargo no es usual en Internet.

- Condiciones de redistribución: el pago suele estar prohibido. Sin embargo, muchas compañías venden recopilaciones de SRL que incluyen shareware en disquetes o en CD-ROM.
- Pago: teóricamente se exige en caso de uso comercial, o después de un plazo de tiempo. El pago suele dar acceso a funcionalidad adicional, documentación, o servicios (por ejemplo, notificación de actualizaciones). Normalmente es esto lo que suele llevar a los usuarios a pagar.
- Código fuente: rara vez disponible. En el caso de que lo esté, se permite la redistribución pero no se exige.

Los autores parecen querer mantener el secreto industrial sobre el código fuente y controlar la explotación comercial. La remuneración viene a través de las tasas del *shareware*, la licencia del código fuente y quizá la licencia de redistribución comercial.

2.3. Licencias GNU

Las licencias **GNU** comprenden la ***GNU General Public License (GPL)*** y la ***GNU Library General Public License (LGPL)***. En nuestra opinión, son las LRLs más interesantes, por ser las únicas que se basan en una concepción explícita de la economía del software. Las licencias GNU se aplican a muchas herramientas de desarrolladores (GNU emacs, gcc, gdb, ...), algunas bibliotecas, y algunos paquetes de usuario final, todos disponibles en Internet. Las licencias GNU están íntimamente identificadas con la ***Free Software Foundation (FSF)***, pero algunos autores no afiliados a ella también han elegido distribuir su software bajo estas licencias.

Por razones de espacio aquí sólo resumimos la GPL. Brevemente, la filosofía GNU es que cualquiera que reciba software debe tener el derecho y la posibilidad de:

- Utilizar el programa.
- Modificar el programa.
- Redistribuir el programa.
- Mejorar el programa y contribuir a la comunidad.

Aunque pueden parecer requisitos similares a los de todas las LRL, tienen algunas consecuencias interesantes. La posibilidad de modificar y mejorar obliga a la disponibilidad del código fuente, por lo que la GPL requiere explícitamente que cualquier distribución incluya el código fuente o bien una oferta escrita de proporcionarlo a cambio, únicamente, del coste de reproducción. Respecto a la redistribución, la GPL va un paso más allá que las otras LRLs al afirmar que se aplica automáticamente a la totalidad de cualquier programa que incorpore código protegido por ella. Esto es para apoyar una filosofía más profunda: los autores no deben recibir remuneraciones por cada copia distribuida porque (como decíamos en la introducción) el software es esencialmente un bien público. (**N. del T.**: El texto de la GPL está disponible en castellano en la versión WWW de este monográfico).

La **FSF** se dedica a crear un entorno de software tipo *UNIX* completo que sólo contenga código bajo licencia GPL. Sin embargo, ha tenido que ceder ante factores económicos y sociales en dos áreas:

- Originalmente, **libgcc** (la biblioteca de soporte de ejecución asociada con gcc, el compilador GNU C) se distribuía bajo GPL. Como resultado, cualquier programa enlazado con libgcc (básicamente cualquier programa compilado

con gcc) estaba sujeto a la GPL. Esto no era del agrado de los desarrolladores de software, y habría reducido drásticamente la utilización de gcc. Como resultado, se diseñó la LGPL, una variante de la GPL que requiere esencialmente que si un programa se enlaza con una biblioteca bajo LGPL, sólo se aplica la GPL a la biblioteca, y no al resto del programa que la utiliza.
- Los anuncios de software de la FSF incluyen el siguiente aviso: "Adquiriendo su software GNU a través de la FSF nos ayuda a continuar desarrollando más software libre. Los beneficios del soporte son nuestra principal fuente de ingresos. Las donaciones a la FSF son deducibles de impuestos en EE.UU.".

Creemos que esto muestra la debilidad de la creencia de la FSF de que los desarrolladores de software recibirán la compensación suficiente por crear SRL a través de los servicios y las operaciones ordinarias en el mercado: esencialmente la FSF está apelando a las donaciones en la forma de pedidos de compras (aunque resulta mucho más barato obtener el software a través de Internet).

Pueden resumirse las condiciones de la GPL como sigue:

- Condiciones de redistribución: ya resumidas más arriba.
- Pago: no se permite, salvo en el caso de que el código fuente se proporcione por demanda. En este caso el precio de este se limitará a los costes de reproducción.
- Código fuente: debe estar disponible, ya sea en la propia distribución, o mediante una oferta escrita.

La remuneración de los autores se limita a los servicios, como ocurre con el SRL en general.

2.4. LRL sin ánimo de lucro (LSAL)

Las **LSAL** más comunes son licencias que, en general, no restringen nada, salvo que prohíben el pago. Por lo tanto son similares a las licencias *shareware*. El software sujeto a licencias LSAL incluye algunos paquetes como **zip/unzip** y un conjunto de fuentes de tipos de letras desarrollados en Rusia. También incluye las versiones más recientes del popular programa de utilidad **Ghostscript**, cuya LSAL es una adaptación de la GPL.

- Condiciones de redistribución: se prohíbe el pago.
- Pago: ninguno.
- Código fuente: generalmente disponible.

La remuneración a los autores puede venir de la licencia comercial del mismo software bajo términos de no redistribución libre, como con el *shareware*. La ***Aladdin Ghostscript Free Public License (AGFPL)*** intenta combinar la filosofía de compartición de la GPL con la restricción del lucro. Actualmente la AGFPL sólo se usa con Ghostscript y uno de sus interfaces de usuario, **GSview** para Windows y OS/2. El objetivo de la AGFPL es evitar cierto tipo de abusos que se dan con la GPL, especialmente por parte de compañías que integran software bajo licencia GPL con aplicaciones comerciales. De esta forma, si bien se respeta la letra de la GPL, el resultado es una nueva aplicación.

Las condiciones de este tipo de licencias son:

- Condiciones de redistribución: el pago esta prohibido, incluyendo la adición del SDL en cualquier transacción que lleve consigo el pago (excepto para la distribución en línea al público en general, y los CD-ROMs en los que toda la información sea redistribuible sin cargo alguno para propósitos no comerciales). Las otras condiciones son similares a la GPL.
- Pago: ninguno.
- Código fuente: debe incluirse en cualquier distribución. Aquí se diferencia de la GPL, que permite incluir sólo una referencia al código fuente y la oferta de proporcionarlo pagando.

3. Debate

Son necesarios un gran número de pasos para que el usuario potencial pueda obtener algún beneficio del software (**Tabla 1**). Las LRLs suelen cubrir los ítems (3a), (3c1) y posiblemente (3b) y (3c2). Las LRLs no presentan ventajas particulares respecto al ítem (1). El uso de una LRL hace (2) mucho más fácil: el no requerir pago implica que no se necesita un control *a priori* sobre las copias individuales, por lo que cualquiera puede facilitar el software a través de medios públicos como servicios en línea y métodos de publicación al mayor como los CD-ROMs. Para (3c2), los costes de producción de los documentos limitan en la práctica la redistribución libre incluso cuando la licencia lo permite. Los servicios (4) y (5) representan fuentes de compensación disponibles para los trabajadores del software relacionados con el SRL. Estas fuentes son únicas en el caso de las licencias no restrictivas (como GPL, AGFPL) y una fuente posible (además de los pagos por el registro o actualización del ejecutable) en el caso del *shareware*.

3.1. Punto de vista del autor

Cualquier LRL proporciona grandes ventajas para la distribución (2), que pueden llevar a proporcionar más oportunidades de conseguir servicio (4), adaptaciones y mejoras (5). Las licencias irrestringidas y las licencias GNU, sin embargo, eliminan el ejecutable (3) como fuente de ingresos. Por ello el tiempo gastado en crear el software no proporciona beneficios al autor, salvo una recompensa moral intangible y la posibilidad de un trabajo de servicio futuro. En cuanto

	Actividad	Resultado
1	Conocimiento de la existencia del software	documento
2	Localización del software	documento
3	Obtención de:	
3a	El software ejecutable	software,
3b	El código fuente	software
3c1	Documentación en línea	documento
3c2	Documentación en papel	documento
4	Asistencia a la instalación y uso servicio	
5	Adaptaciones y mejoras	servicio y software

Tabla 1

al *shareware*, la LRL proporciona más oportunidades de proporcionar (3) a cargo de copias registradas (por las que se paga). En cuanto al software LSAL, las condiciones de distribución (2) proporcionan un gran acceso al mercado potencial de licencias comerciales (3). Por ello las LRL del *shareware* y LSAL proporcionan al autor el mayor y más rápido beneficio directo dentro de las LRLs, y en particular recompensan más allá del pago por hora por el trabajo futuro.

Los autores están en una situación curiosa con respecto a (4) y (5). Hasta cierto punto, distribuir software de menor calidad va en favor del autor de SRL, pues le permite llegar antes al mercado, y también crea una mayor demanda de documentación (4) y posiblemente adaptaciones y mejoras (5). De hecho muchas compañías de software no libre parecen estar siguiendo también esta línea de pensamiento. De forma similar, si se supone que el autor desea estar en el negocio del servicio, juega a su favor el distribuir código fuente poco comentado (3b) o, como sucede típicamente con el *shareware*, no distribuirlo, pues así es más difícil que terceros compitan con él proporcionando (5) y en alguna medida (4).

El uso de diferentes LRLs refleja a menudo la actitud subyacente del autor hacia la industria del software. Muchos, al escoger licencias LRL de tipos distintos al *shareware*, recuerdan, o al menos se inspiran, en la excitación ante el descubrimiento y la recompensa de compartir el trabajo que caracterizó los primeros días del software en los años 60 y 70. Su experiencia les deja claro que si todos contribuyen libremente todos se benefician: al menos si "todos" son gente como ellos mismos. Las licencias irrestringidas incorporan este punto de vista en su forma más pura. La GPL adopta una visión menos confiada, al impeler a los participantes a que compartan su trabajo. La AGFPL adopta una visión ligeramente diferente: aquellos que quieran compartir se beneficiarán, mientras que aquellos cuyas actividades se gobiernan por leyes comerciales deben seguir las mismas para obtener los beneficios del software que se distribuye libremente para otros. Esta visión está también implícita en otras LSAL. Las licencias *shareware* conllevan una confianza más limitada en el usuario. Sin embargo, todas las LRL necesitan cierto grado de idealismo, o al menos confianza, por parte del autor.

3.2. Punto de vista del redistribuidor

Desde el punto de vista del redistribuidor comercial, el SRL no es un negocio especialmente lucrativo. En particular, el coste marginal del ejecutable (3) a través de un servicio en línea, permitido por todas las LRL, es tan bajo que los editores de CD-ROM de SRL no pueden ser competitivos a largo plazo. Nosotros esperamos que incluso (1) se verá limitado severamente como nicho para redistribuidores por la disponibilidad de herramientas de búsqueda en WWW cada vez más sofisticadas.

3.3. El punto de vista de las terceras partes proveedoras de servicio

Como todas las LRLs, excepto las *shareware*, requieren o animan a la distribución de código fuente, aquellas crean oportunidades de proporcionar servicio (4 y 5) que no existen con las licencias no-LRL. De hecho, esta es esencialmente la única fuente de compensación para los trabajadores del software bajo GPL. Sin embargo, la industria de soporte de SRL es actualmente minúscula: el autor sólo conoce una compañía dedicada únicamente al soporte y mejora de SRL (***Cygnus Support***, que tiene unos beneficios anuales de más de un millón de dólares) aunque el software de GNU ha sido ampliamente usado durante más de una década. El autor cree que existen varias razones que explican esto:

- Por razones que se discutirán en la próxima sección, el SRL que no es *shareware* tiende a estar muy orientado a desarrolladores y a usuarios con un perfil tecnológico, en lugar de a usuarios finales que simplemente quieren una herramienta que les resuelva algún problema. Esto reduce el mercado para ese SRL y hace mucho más habitual que los propios usuarios de este tipo de software estén dispuestos a investigar los problemas. Por ejemplo, la disponibilidad de código fuente expande el mercado para terceras partes dedicadas al soporte (4) y facilita que los usuarios se las arreglen por su cuenta.

- Si el SRL es de buena calidad y está bien documentado, la necesidad de soporte es mínima. Si el SRL es de mala calidad y/o pobremente documentado es más difícil que a una tercera parte le compense invertir el esfuerzo necesario para soportarlo. Por lo tanto, sería de esperar que hubiese más gente extendiendo SRL que proporcionando soporte para él, y que sean los autores principalmente los que hagan cualquiera de las dos cosas. El autor cree que esta es la situación actualmente.

- El uso extendido de la tecnología de enlazado dinámico que se hace actualmente, popularizado (pero no inventado) por las bibliotecas de enlace dinámico de Microsoft Windows (**DLLs**), ha guiado a la industria siguiendo una tendencia hacia los sistemas "semiabiertos" (aplicaciones que se distribuyen sólo en forma binaria con licencias no-LRL, pero proporcionando **APIs** que permiten que terceras partes escriban extensiones en forma de módulos enlazados dinámicamente, que de nuevo pueden ser distribuidos sólo como binarios con licencias no-LRL). Esto permite que terceras partes participen en las adaptaciones o mejoras (5) incluso con licencias no-LRL.

Resumiendo, las LRL crean más oportunidades para los servicios de terceras partes, pero probablemente no en una gran escala.

3.4. Punto de vista del usuario

Se puede clasificar a los usuarios de software atendiendo a si tienen una relación o interés en la tecnología informática, o si simplemente quieren un software que realice alguna tarea, como análisis financiero o autoedición. Estos grupos se solapan, pero tienen necesidades y expectativas muy diferentes con respecto al software que usan. El grupo relacionado con la tecnología quiere software que sea funcionalmente completo, y que esté adecuadamente docu-

mentado y sea fiable. Se valora la disponibilidad del código fuente y del soporte, y en alguna medida se intenta alcanzar un compromiso entre ambos. Este grupo incluye la mayoría de los desarrolladores tradicionales (los que no usan 4GL). Habitualmente usan estaciones de trabajo u ordenadores personales de gama alta.

El grupo de usuarios orientados a las aplicaciones quiere software que esté presentado atractivamente, bien documentado, que sea fácil de instalar y configurar, diseñado para la usabilidad, de rica funcionalidad y al mismo tiempo fácil de utilizar para tareas sencillas, confiable y soportado. Este grupo incluye a la gran mayoría de usuarios de ordenadores que no son desarrolladores: todos menos los que son estudiantes o usuarios de grandes ordenadores en el trabajo. Normalmente utilizan PCs y Macintoshes, tanto en entornos de trabajo como personales. La competencia por este mercado es enorme en todas las categorías enumeradas. Especialmente en entornos de trabajo, la documentación, la facilidad de instalación, la confiabilidad y el soporte son esenciales. La usabilidad y el precio son también importantes pero secundarios. La disponibilidad del código fuente sólo es apreciada como una garantía en caso de que el proveedor de software deje el negocio o deje de soportar el producto.

Hay poco SRL no *shareware* para usuarios interesados en aplicaciones. Creemos que la razón para esto es que casi todos los autores de software que están motivados por el deseo de crear y compartir tecnología (siendo estos los principales creadores de SRL) no tienen ni interés, ni capacidad, o quizá paciencia para tratar los aspectos de diseño más orientados a los usuarios de aplicaciones finales. La excepción es el *shareware*, que permite a un autor distribuir libremente versiones de software con la esperanza de que una importante fracción de estas versiones se convierta en ventas. Así estos autores que no comparten la ética del SRL pueden utilizar licencias *shareware* como una herramienta de trabajo. Lo mismo ocurre hasta cierto punto con las LSALs.

3.5. Algunos comentarios legales

Nótese que el autor de este artículo no es abogado. La siguiente discusión de aspectos legales no debe tomarse como consejo legal, ni siquiera como fundamentada sólidamente desde una perspectiva legal (**N. del T.**: Y por supuesto, lo aquí dicho está enfocado según la legislación de EE.UU. Por lo tanto su aplicación en otros países es algo más que dudosa). Las LRL, como todas las licencias de software, se apoyan en los derechos de autor. Al igual que las licencias de tipo *shrink wrap*, se apoyan en la teoría de que las licencias entran en vigor automáticamente al realizar ciertas acciones (uso de un elemento software, ruptura física de un sobre, etc.) son defendibles. Sin embargo, ya que los únicos beneficios que aporta una LRL al que detenta los derechos de autor son intangibles, podría ser difícil que un tribunal haga cumplir una LRL, sobre todo si (como sucede con las licencias no restrictivas) no hay en la propia LRL ninguna medida prevista que tome efecto cuando no se respeta aquella. Sería además muy difícil que un tribunal reconociese ningún daño producido al que detenta el derecho de autor.

Por otra parte, el autor conoce varias situaciones en las que los redistribuidores que infringen las LRL han subsanado voluntariamente su infracción, aconsejados probablemente por sus propios abogados.

4. Conclusiones

Las LRL cubren un amplio espectro, suficiente para brindar a los usuarios los beneficios del SRL a la vez que sirven al menos a tres tipos diferentes de autores de software:

- Los autores cuya única motivación es la satisfacción y el orgullo de la creatividad y el bien social, o que encuentran satisfacción al estar en un negocio de servicio, pueden utilizar una licencia no restrictiva o la LRL GNU.
- Los autores que quieran tener unos ingresos modestos mientras que mantienen un control más estrecho sobre su tecnología pueden utilizar la vía del *shareware*.
- Los autores que quieran proporcionar los beneficios del SRL a los usuarios, y aprovecharse a la vez de los beneficios de las licencias comerciales de software no libre, pueden utilizar las LSAL.

La recompensa disponible para los autores que utilizan LRL puras y la naturaleza de las expectativas de los usuarios finales, hacen poco probable que se cree software con una funcionalidad rica para usuarios finales, salvo en el caso del *shareware*. Sin embargo, el modelo del SRL puede funcionar bien para aplicaciones más orientadas a desarrolladores. Será interesante ver cómo compite el software libremente redistribuible y el no libremente redistribuible en el campo emergente de los componentes reutilizables.

Agradecimientos

Nuestro agradecimiento a **Richard M. Stallman** por la creación de la GPL y la LGPL, y por las discusiones sobre la filosofía del software libre.

Gracias a **Stephen J. Turnbull**, del Instituto de Planificación Socioeconómica de la Universidad de Tsukuba, en Japón, por sus comentarios sobre el SRL desde un punto de vista económico.

Por supuesto, el autor asume toda la responsabilidad de los comentarios que se han reflejado en este artículo.

Software Libre

Robert F. Young *
Red Hat Software, Inc.

¿Por qué Linux es mejor que NT y otros sistemas operativos comerciales?

Cuando nos topamos por primera vez con el sistema operativo ***Linux*** estábamos seguros de que no tendría éxito. A lo sumo tendría un éxito fugaz. Se iría olvidando según fuesen madurando las alternativas comerciales: **NT**, **OS/2**, **Neststep**, **Interactive**, etc.

Hoy en día la distribución Linux de Red Hat cuenta con decenas de miles, si no millones, de usuarios entusiastas, y prevemos un éxito a largo plazo.

1. ¿Qué es Linux?

Linux, para aquellos pocos que aún no lo conocen, es el sistema operativo avanzado, multiusuario y multitarea, construido originalmente para ordenadores PC compatibles con Intel. Es el resultado de un desarrollo a escala mundial, realizado a través de Internet. Puede utilizarse como una estación de trabajo **Unix**, o como un servidor para tareas que van desde servidores *web* fiables para Internet, hasta estaciones de bajo coste para redes cliente-servidor.

Los factores que asegurarán el éxito a largo plazo de *Linux* tienen poco que ver con el repertorio de funcionalidades ofrecidas, si bien este es impresionante. Por ejemplo, soporta una gama de hardware PC más amplia que cualquier otro sistema operativo no-Microsoft. Microsoft cuenta con la ventaja injusta de tener a su disposición al fabricante de hardware Intel para fabricar productos diseñados específicamente para correr DOS y Windows. Sin embargo, a diferencia de los sistemas operativos de Microsoft, *Linux* funciona también en plataformas **Sun SPARC**, **Apple PowerMac** y **Digital Alpha**.

2 ¿Cómo se distribuye Linux?

Conviene enfatizar que *Linux* se distribuye "libremente", sujeto a los términos de la licencia **GPL** (***GNU Public License***). En pocas palabras, esta licencia permite a cualquiera utilizar *Linux*, siempre que, si lo modifica y decide distribuir sus cambios, lo haga en las mismas condiciones: bajo licencia GPL. Esto significa que mientras que *Linux* se vende de muy diversas formas, también puede ser obtenido mediante copia sin ningún coste o restricción adicional.

La GPL no es la única licencia bajo la que se distribuye el "software libre". El propio *Linux* se beneficia de software distribuido bajo otras licencias. Los cuatro tipos principales son: **Software de Dominio Público**, ***Shareware***, ***copyright*** tipo **BSD**, y la **GPL**. Cada una de ellas presenta ventajas e inconvenientes, pero nosotros preferimos la GPL por razones que ilustramos a continuación.

La GPL proporciona al usuario la misma libertad que las otras licencias, con un beneficio añadido, no para el usuario ni para el autor, sino para el propio proyecto: al requerir que cualquier cambio o mejora que se distribuya de un código GPL, tenga que hacerse bajo esta misma licencia, se asegura que ningún desarrollador obtendrá ventajas adicionales a costa de los demás. No obstante, el software que funciona encima de *Linux* o de cualquier software distribuido bajo licencia GPL, puede distribuirse con cualquier tipo de licencia.

Este mecanismo solventa los problemas que ocurrieron en el proyecto *Unix* original. Al principio del proyecto **AT&T** poseía el *copyright* de *Unix*, pero muchos grupos, incluyendo la Universidad de California en Berkeley, ayudaron a construirlo. Cuando estuvo terminado AT&T pudo ejercitar su derecho de autoría, restringiendo el uso de una tecnología que había sido desarrollada de manera cooperativa.

La licencia GPL permite retener el derecho de autoría a los promotores originales de *Linux* (entre ellos ***Linus Torvalds***), pero ni ellos ni nadie puede restringir el uso y modificación de *Linux* en manera alguna.

3. ¿Cuáles son las fuentes de financiación de Linux?

¿Cómo puede competir con las compañías de software más ricas del mundo un sistema operativo "libre", que aparentemente no tiene respaldo económico? Las apariencias engañan. Existe mucho dinero detrás del desarrollo de *Linux*. Compañías comerciales como ***Red Hat Software*** invierten directamente en *Linux* mediante código que escriben y distribuyen bajo licencia GPL, y mediante dinero que donan a equipos de desarrollo de *Linux*, como la ***Free Software Foundation***, el proyecto GNU, el ***Linux Documentation Project*** y **XFree86**.

El papel jugado por *Red Hat Software* en el desarrollo de *Linux* tiene que ver con el desinterés de los desarrolladores de *Linux* por aspectos de facilidad de uso del sistema operativo. Estos suelen pensar que, si posees la capacidad de ayudar a desarrollar el código del sistema operativo, no necesitas de menús, cajas de diálogo y herramientas de configuración para instalar y administrar el sistema operativo. Sin embargo, la mayor parte de los usuarios sí necesitan este tipo de herramientas para utilizar productivamente el sistema operativo.

La historia del *Linux* de *Red Hat* avala este argumento. En 1994, Marc Ewing era un estudiante egresado del famoso departamento de informática de la Universidad Carnegie Mellon, de EE.UU. En aquellas fechas trabajaba en IBM. En su tiempo libre pretendía construir una herramienta de desarrollo avanzada. Sus ordenadores eran un conjunto de estaciones de trabajo *Linux*. La versión de *Linux* que utili-

zaba era la distribuida por **SLS**, debiendo solventar numerosos defectos constantemente. Cuando notó que no estaba avanzando en su proyecto, pensó en que estaba derrochando el tiempo. Descubrió que estaba gastando más tiempo en el mantenimiento de las estaciones de trabajo *Linux* que en el trabajo real de su proyecto. En ese momento decidió que lo que la comunidad necesitaba no era otra herramienta de desarrollo, sino una buena distribución de *Linux*.

Por ello comenzó el proyecto *Red Hat Linux*, para eliminar las limitaciones que él mismo había sufrido. Un ejemplo es **RPM**, posiblemente el sistema de gestión de paquetes software más avanzado y sofisticado de la industria. RPM permite a un usuario de *Linux* instalar paquetes nuevos o actualizar otros ya instalados, sin el tedioso proceso de prueba y error consistente en colocar los paquetes en el lugar adecuado del sistema de ficheros, actualizar las variables de entorno, realizar los enlaces adecuados y, en el caso de las actualizaciones, borrar los ficheros y enlaces viejos. RPM hace todo esto en lugar del usuario, incluyendo avisos cuando otros programas y ficheros que el nuevo paquete pueda necesitar no se encuentran en el sistema. Se le ahorran así al administrador del sistema horas de trabajo tedioso.

Mediante la aportación de estas características de facilidad de uso al sistema operativo *Linux*, y al hacerlo bajo licencia GPL, *Red Hat* facilita el uso de *Linux* por parte de un mayor número de usuarios de computadoras, contribuyendo así a la expansión de la comunidad *Linux*. Como contrapartida, *Red Hat* puede mantenerse como negocio, vendiendo discos compactos con el software *Linux*, o vendiendo libros y aplicaciones, expandiendo asimismo el mercado de *Linux*. Sin embargo, las contribuciones económicas de compañías como *Red Hat Software* representan sólo una mínima parte de los recursos aportados al desarrollo de *Linux*.

4. *Linux* se construye por y para los usuarios

El grueso del desarrollo de *Linux* está financiado por los propios desarrolladores que utilizan *Linux* en sus propios proyectos o aplicaciones. Desde proyectos de supercomputación de la NASA hasta el desarrollo de software en *Empress SW* (ver más abajo), el trabajo está financiado por los usuarios, en su propio beneficio. El hecho de que este trabajo también beneficie a la comunidad de usuarios de *Linux* es fortuito (por casual como por afortunado).

La historia del manejador del dispositivo de almacenamiento **Iomega Zip**, incorporado en *Linux*, es un buen ejemplo. Grant Guenther, el principal desarrollador de la empresa de bases de datos ***Empress Software Inc.*** (*www.empress.com*), estaba promoviendo el uso de Linux como sistema operativo de desarrollo en *Empress*. Proporcionó a sus colegas acceso a estaciones de trabajo de bajo coste, tanto en el trabajo como en sus domicilios. Cuando *Empress* escogió unidades de disco Zip como estándar de la compañía para la transferencia de datos, Grant descubrió que no existían manejadores en *Linux* para ellas. Tenía dos alternativas: abandonar *Linux* y comprar un número considerable de costosas licencias de algún sistema operativo comercial, distribuido sin compiladores ni código fuente (valiosos recursos disponibles en *Linux* para los desarrolladores), o emplear algún tiempo investigando y escribiendo su propio manejador de las unidades Zip en *Linux*. Esta última fue su decisión. Como desarrollador principal de una compañía de bases de datos, Grant estaba capacitado para hacer un buen trabajo. Una vez desarrollado el manejador, lo anunció a través de Internet, reclamando ayuda para probarlo y mejorarlo. Pronto se convirtió en parte del sistema operativo *Linux*, pudiendo ser utilizado hoy en día por cualquiera que disponga de unidades de almacenamiento **Zip Iomega**.

5. Desarrollos en curso de *Linux*

En caso de que alguien hubiera estado interesado en el manejador para *Linux* desarrollado por *Empress Software*, esta compañía habría cobrado miles de dólares por los servicios de Grant. Al utilizar el modelo *Linux/GNU*, el incentivo no fue el beneficio puntual proporcionado por el desarrollo del manejador, sino el beneficio indirecto y continuado que Grant y *Empress* reciben del uso de *Linux*: los nuevos desarrollos aportados a *Linux* son utilizados a diario por *Empress*. Pero *Empress* y *Red Hat* son sólo dos de los miles de organizaciones relacionadas con *Linux*, entre las que se encuentran, literalmente, todos los organismos de investigación de la administración estadounidense, la mayor parte de las Universidades, y la mayor parte de equipos de desarrollo de software de compañías comerciales.

El enorme y creciente número de esfuerzos de desarrollo que respaldan *Linux* provocarán que esta tecnología esté por delante de cualquier sistema operativo comercial. Un ejemplo ilustrativo es la seguridad. Dada la naturaleza abierta de *Linux*, por disponerse del código fuente del mismo, los aspectos de seguridad pueden ser identificados, discutidos y reparados en tiempo real. El problema se discute abiertamente, los parches son probados por un gran número de desarrolladores, y el problema se va solucionando hasta que todos están de acuerdo en la solución obtenida. Al airear estos problemas de seguridad en público, los usuarios de sistemas operativos tradicionales pueden pensar que *Linux* padece un mayor número de problemas de seguridad que los sistemas operativos comerciales. Nada más lejos de la realidad: todos los sistemas operativos tienen problemas de seguridad, mas en el caso de *Linux* estos se identifican y solucionan inmediatamente, gracias precisamente a la discusión abierta que sobre los mismos se mantiene habitualmente en Internet.

Cuando nos cruzamos de brazos y nos preguntamos cómo han podido triunfar compañías como Red Hat, que tienen que competir con gigantes como Microsoft e IBM en el negocio de los sistemas operativos, pensamos en gente como Grant Guenther, los desarrolladores de la Universidad de Helsinki, la NASA, los equipos de investigación académicos y comerciales que trabajan con *Linux*. De pronto, la pregunta se convierte en ¿cómo va a competir Microsoft con *Linux*?

Nota

* Bob Young es presidente de Red Hat Software, Inc., Durham, Carolina del Norte, EE.UU. El presente artículo provocará más preguntas que respuestas proporciona. Se pueden encontrar respuestas adicionales en: *www.redhat.com*, *www.li.org*, *www.ssc.com*, y en otros muchos lugares con información sobre Linux, que pueden alcanzarse desde estos.

Software Libre

Sourceware Solutions

Michael Tiemann
Cygnus Support

1. ¿Cómo resolver la paradoja del software abierto?

Hay que cambiarlo todo para que todo siga igual. El cambio más significativo producido por la revolución de los Sistemas Abiertos fue que los usuarios comenzaron a pedir soluciones que no restringiesen su libertad. Lo que sigue inamovible es que las compañías del mercado de los Sistemas Abiertos aún tienen que preservar su ventaja competitiva y su escalabilidad utilizando un modelo de negocios que puedan controlar. El modelo de negocio basado en sistemas propietarios ha llevado a la siguiente paradoja: cuanta más libertad se proporciona, menos control se conserva, debilitándose así el negocio. Al mismo tiempo, el mercado dice que a menor libertad proporcionada, menor demanda existirá para los productos. La solución a esta paradoja, que requiere un modelo nuevo de hacer negocios, se puede encontrar al analizar las tendencias del mercado libre que tienen éxito en la actualidad.

2. Cambio de paradigma económico en la industria del software

La industria del software de los años ochenta era atractiva desde el punto de vista económico: los márgenes eran elevados, era más fácil innovar, y la demanda del mercado era tal que cualquier software que hiciese algo era un producto comercial viable. Al madurar la industria los precios se han nivelado, los márgenes han desaparecido, y tan sólo un número limitado de vendedores distribuye productos que se pueden considerar "estándar". Compañías que en su tiempo fueron boyantes líderes tecnológicos (**Ashton-Tate, Borland, Software Publishing, SCO** y **Symantec**, por nombrar sólo algunos) luchan por sobrevivir en los años noventa.

Según van disminuyendo los precios de los productos que se venden bajo licencias "de rompe y rasga" las compañías tienen que ir repensando sus supuestos. Muchas están dándose cuenta de que no es viable económicamente ofrecer servicios técnico y de mantenimiento de manera gratuita.

Esta tendencia se manifiesta incluso en el mundo de los ordenadores personales, en el que compañías como Borland, Microsoft y Word Perfect se están desprendiendo de sus servicios de mantenimiento para proporcionar a los clientes productos software de bajo coste y con opciones de mantenimiento alternativas. Todos ellos están viendo en los servicios de mantenimiento una oportunidad de crecimiento. En los últimos tres años han aparecido muchas compañías de nueva creación cuyo único objetivo es proporcionar servicios de mantenimiento técnico, y muchas compañías conocidas, como Arthur Andersen, Microsoft y Oracle, han creado divisiones con este propósito. Ya sea ofreciendo el mantenimiento como un producto, u ofreciendo el software como un servicio, estos vendedores han encontrado una forma de convertir el software estándar no especializado en productos adaptados a las necesidades particulares.

Al amoldarse al enfoque de los servicios, la industria está reconociendo que los usuarios no están interesados únicamente en la adquisición de bits en un disco flexible (software), sino en la adquisición de soluciones (parte de las cuales pueden residir en los propios bits). Este cambio, del modelo de "software como producto", al modelo de "software como servicio", tendrá un impacto dramático en la forma en que se distribuye y se mantiene el software, y en cómo evolucionará para satisfacer las necesidades de los usuarios.

En el modelo que trata el "software como producto" los vendedores de software se apoyan en las licencias para controlar el precio de cada unidad vendida. El mantenimiento técnico generalizado del software se ve limitado porque lo esencial del producto, el código fuente, permanece restringido. El desarrollo de nuevos productos está bloqueado porque los desarrolladores deben anticipar las necesidades de sus clientes basándose en fuentes de información limitadas. En el modelo que trata el "software como servicio" los vendedores de mantenimiento deben suministrar valor añadido a sus clientes viendo que pueden incorporar al software y no que pueden llevarse de él. Llegados a este punto, el software necesita un mantenimiento constante, pues la mejora continuada es la que hace que se aprecie ese valor añadido.

3. El alto coste de los modelos basados en propietarios

Los productos de información son interesantes económicamente porque pueden distribuirse, o como mercancías de bajo coste, con amplios márgenes, o como servicios flexibles de alto valor. Hoy en día, el principal atractivo del software reside en la combinación entre la facilidad de uso, la independencia de plataformas hardware específicas, y la promesa de que para cuando los usuarios comiencen a dominar el producto, este no se habrá quedado obsoleto.

Si bien el software propietario es un negocio atractivo, hay varios aspectos que resultan problemáticos para los usuarios. Y lo que es más importante, el software propietario hace prácticamente imposible que los usuarios intervengan directamente en su evolución. En lugar de eso, las demandas de los usuarios deben filtrarse a través de los canales de

marketing, luego han de aprobarse por los gestores del negocio y por último, en su caso, el departamento técnico realizará los cambios. Mediante este proceso, problema nimios del software pueden tardar años en repararse.

El modelo de edición tradicional se basa en la premisa de que son mucho más costosas la duplicación y la distribución que la autoría. Pero en el caso del software, que puede duplicarse a un coste prácticamente nulo, y distribuirse mundialmente a través de redes a un coste nominal, ocurre lo contrario. La reducción de los precios del software vendido como paquetes cerrados y la demanda creciente de soporte técnico de alta calidad son signos de que el mercado ha de solventar estas inconsistencias.

Las mejoras de la tecnología hardware han producido un incremento constante del rendimiento, mientras sus costes se han reducido continuamente. Esta tendencia reduce el riesgo de las inversiones en hardware. Mediante su combinación con la interoperabilidad de los sistemas abiertos, los usuarios pueden crear soluciones globalmente óptimas a partir de componentes optimizados localmente. En el mundo del software, la situación dista mucho de ser óptima: pocos paquetes interoperan incluso con productos del mismo vendedor, y mucho menos con los de otros vendedores. Debido a esto, cada compra de software debe analizarse con sumo cuidado, si no se quiere amenzar la infraestructura existente.

Actualmente, los usuarios exigen para el software las mismas mejoras y niveles de seguridad que han visto en el hardware: productos cuyos precios reflejen el coste real y que operen de manera fiable con los demás. Estas mejoras no podrán alcanzarse mientras que los modelos basados en sistemas propietarios sigan dividiendo en lugar de coordinar a los diversos actores. Al final, los usuarios conseguirán la libertad de elección, y sólo sobrevivirán aquellos vendedores que puedan ofrecer y suministrar lo que los usuarios demanden.

4. La alternativa del *Sourceware*

Existe un denominador común a todo el software: el código fuente en el que este está escrito. Para algunos, el código fuente es de la familia de las joyas, para otros es la clave para alcanzar la libertad. Pero en un número creciente de áreas el software puede ser beneficioso tanto para las organizaciones que lo mejoran, lo modifican o lo distribuyen como para aquellos que lo utilizan.

El *sourceware* es software con mantenimiento, gestionado de manera centralizada, y cuyo código fuente está disponible para cualquiera. Esto reduce las barreras del coste de desarrollo, lo que permite realizar un código robusto y dinámico. El *sourceware* acelera el ciclo de evolución: aquellos que precisan mejoras son libres de hacerlo, y todos los usuarios se benefician de las contribuciones de los demás. El *sourceware* incorpora todos los beneficios del software: confiable, adaptable, potente, e incrementa la velocidad de las revisiones. Por lo tanto, el *sourceware* soluciona los problemas económicos del desarrollo comercial de software.

El *sourceware* se distribuye bajo una licencia que proporciona a los usuarios la libertad de utilizar, distribuir y modificar el código fuente. También proporciona un entorno para definir versiones preliminares de estándares, realizaciones de referencia, y un proceso de mantenimiento y mejora continuados, que satisfacen las necesidades del mercado comercial. El *sourceware* es una solución genérica que proporciona los beneficios asociados a los sistemas abiertos, sin sobrecargas o gastos innecesarios.

Un ejemplo de este enfoque es el sistema *X Window*. *X* es el estándar *de facto* para sistemas de ventanas multiplataforma. El sistema *X Window*, creado y mejorado regularmente por un grupo del **MIT** (***Massachusetts Institute of Technology***), juega un papel fundamental en el mercado *Unix*. Incluso *Sun Microsystems* abandonó el desarrollo de su sistema *NeWS*, favoreciendo así una versión estándar de *X*. El sistema *X Window* se distribuye gratuitamente. Se promueve que los usuarios modifiquen y adapten el software a sus propias necesidades. La versión actual es la 6, y está funcionando en prácticamente cualquier combinación de procesador y sistema operativo, gracias a la comunidad de vendedores de sistemas, y a muchos usuarios que realizaron los ajustes oportunos al código original. Entre otras plataformas está disponible para **Sun**, **HP**, **IBM**, **DEC** y *Silicon Graphics*. Estas compañías se han unido para soportar el estándar **COSE**, basado en *X*. Los usuarios de **DOS**, *Windows* y **Mac** también disponen de una amplia gama de realizaciones del servidor *X*, pudiendo así acceder a datos y aplicaciones, conforme al espíritu de los sistemas abiertos.

Otro ejemplo (soportado por *Cygnus* como *sourceware*) es el software de **GNU**. GNU es el estándar *de facto* en compiladores y depuradores multiplataforma de los lenguajes C y C++. En entornos de desarrollo y universitarios, las herramientas de GNU son aceptadas como una alternativa potente a las basadas en sistemas propietarios. Si bien las herramientas de GNU soportan muchas plataformas hoy en día, por distribuirse como *sourceware*, también pueden ser adaptadas a nuevas plataformas que vayan apareciendo, resguardando así las inversiones de los clientes. Estos pueden elegir cómo invertir sus recursos, contratando a terceros el desarrollo y el mantenimiento cuando sea necesario. Incluso las compañías tradicionales de software encuentran un valor añadido en el *sourceware*. La compañía Novell anunció recientemente su intención de proporcionar a los desarrolladores de su producto *Netware* las herramientas de desarrollo de GNU para el entorno *NetWare* basado en *Unix*. Las herramientas permiten que los desarrolladores de *Netware* proporcionen servicios de red independientes de las plataformas hardware, aprovechando al tiempo los beneficios del entorno de desarrollo *Unix*. Utilizando estas herramientas *sourceware*, la productividad de los desarrolladores se incrementará, al dedicar sus esfuerzos al desarrollo, en lugar de invertir su tiempo en aspectos de adaptación del software a diversas plataformas.

Por último, el *sourceware* proporciona los prometidos sistemas abiertos, aclamados durante años como la solución a los problemas de la industria. El modelo del *sourceware* permite a los clientes separar los estándares de los vendedores,

porque es una solución global. Al estar disponible el código fuente, cualquiera puede realizar las mejoras oportunas del software para adaptarlo a sus necesidades. El *sourceware* es una solución completamente abierta, que no depende de ningún sistema propietario. El *sourceware* preserva las inversiones en tecnología de los clientes, y permite que el mantenimiento sea una parte fundamental de la realización del software.

5. Mirando al futuro

Al principio, los vendedores de sistemas propietarios bloqueban a sus clientes para imposibilitar la competencia. Según fue destacándose la importancia de la interoperabilidad, estas barreras han acabado afectando al cliente mucho más que a la competencia. Los sistemas hardware abiertos proporcionan beneficios porque solucionaron este problema, al basar su negocio en las economías de escala.

El *sourceware* es el siguiente paso en la evolución de los sistemas abiertos: preserva las inversiones en tecnología de los clientes y permite que el mantenimiento se convierta en parte integral de la realización del software. En último término, el *sourceware* permitirá que los usuarios gasten de manera más eficiente los recursos dedicadas al desarrollo y al mantenimiento, mejorando así su ventaja competitiva. El modelo de **software como servicio** creará también, con el tiempo, oportunidades económicas para las compañías de software que se adapten a las nuevas condiciones del mercado.

Software Libre

Edmond Schonberg
Universidad de Nueva York

Orígenes e historia del compilador GNAT

1. Introducción

El proyecto **GNAT**, realizado en la Universidad de Nueva York, estuvo dedicado a la construcción de un compilador para **Ada95**. Como GNAT (acrómino de GNU-NYU *Ada Translator*) usa el generador de código reorientable de GCC, es relativamente fácil de transportar, y ya está disponible para la mayoría de las arquitectura actuales, desde estaciones **RISC** hasta PCs basados en **procesadores ix86**. El proyecto terminó en Junio de 1995. GNAT ha sido desarrollado en colaboración con, y siguiendo las directrices de la *Free Software Foundation* y es distribuido libremente, con fuentes, en Internet y en varios medios físicos. Se creó una organización independiente, *Ada Core Technologies* (**ACT**), para asegurar el mantenimiento de GNAT a partir de la terminación del proyecto. ACT ha validado formalmente GNAT en varias arquitecturas, y continúa produciendo nuevas versiones mejoradas del compilador. ACT se ha comprometido a continuar dejando estas versiones como software libre.

2. Primeras actividades relacionadas con Ada en la Universidad de Nueva York

El equipo GNAT de la Universidad de Nueva York ha participado en la definición e implementación de Ada durante los últimos 15 años, desde la aparición de la versión preliminar del manual de referencia para Ada83. El proyecto **NYUADA**, como se llamó entonces, estaba al principio centrado en las técnicas de optimización para lenguajes de alto nivel. Pronto fue evidente que el manual de referencia, tal y como estaba, sólo proporcionaba una definición muy parcial de la semántica del lenguaje, y que la presencia de concurrencia afectaba críticamente al significado potencial de todas las operaciones. En un esfuerzo de proporcionar una semántica operacional del lenguaje que pudiera servir como marco para estudios de optimización, Robert Dewar escribió un intérprete para Ada centrado en los aspectos más novedosos del lenguaje: la interacción entre el flujo de control, las tareas y las excepciones. Para hacer que esta definición operacional fuese lo más sutil posible, el intérprete fue escrito en **SETL**, un lenguaje de muy alto nivel desarrollado en la Universidad de Nueva York, cuyos elementos básicos son los de la teoría de conjuntos finitos.

El intérprete estaba estructurado como una definición denotacional: sus estructuras de datos básicas eran representaciones de la continuación y del almacén, y la semántica de cada instrucción primitiva se describía en términos de modificaciones explícitas de estos. Las tareas se describían operacionalmente asignando un almacén y una continuación propios para cada tarea. La granularidad de las primitivas del intérprete proporcionaban una descripción coherente de la interacción entre el flujo secuencial de control, la comunicación entre tareas, y la elevación de excepciones.

3. Ada/Ed

Este intérprete inicial tenía menos de 2000 líneas, y tomaba como entrada un árbol de análisis gramatical escrito a mano para Ada. Para proporcionar también una definición de la semántica estática se decidió completar el interprete con un frontal que produjese un árbol sintáctico abstracto analizado semánticamente. Gerry Fisher (ahora en IBM) usó SETL para construir un analizador **LALR** para Ada y para desarrollar un serie de nuevos algoritmos para recuperación de errores. Edmond Schonberg escribió el analizador semántico, también en SETL. El sistema resultante fue un traductor completo para el lenguaje, y fue usado por el equipo ACVC, dirigido por John Goodenough, en el desarrollo del juego de validación. El sistema era notablemente lento según cualquier estándar (10 líneas/s. de velocidad de compilación, 1 línea/s. de velocidad de ejecución en un VAX/780) pero en 1982/83 era el único traductor disponible para Ada, y fue usado como herramienta docente por varias universidades e industrias. El sistema, llamado **Ada/Ed** para enfatizar su papel docente, fue el primer traductor validado para Ada. Su rendimiento produjo muchos chistes ("Ada/Ed se usa para la simulación en tiempo real de cálculos con lápiz y papel") pero durante un tiempo fue un recurso educativo útil, y cumplió su propósito original de proporcionar una definición operacional informal del lenguaje, que fuese complementaria a la del Manual de Referencia y al juego de validación de compiladores Ada (ACVC).

Ada/Ed era también un prototipo completo de un traductor, y después de 1984 las actividades del proyecto se dirigieron hacia un experimento a gran escala en prototipado de software. El código SETL de Ada/Ed fue usado como documento de diseño para un compilador convencional, escrito en C, y cuyo rendimiento era comparable al de un intérprete convencional de BASIC. Esta tarea fue realizada en dos etapas: primero se transformó el intérprete denotacional en un intérprete que operaba sobre un flujo lineal de código, y luego se añadió un generador de código genuino para producir el flujo. Este nuevo componente fue también escrito en SETL. Por último el sistema SETL fue traducido a C.

El resultado de este esfuerzo fue Ada/Ed-C, que también fue validado con el ACVC. Dave Shields (ahora en IBM) dirigió el esfuerzo de la traducción de SETL a C, escribiendo la implementación de varias primitivas de conjuntos y asentando las directrices que permitieron que la traducción de SETL a C fuese casi mecánica, sin precisar información detallada sobre los algoritmos de Ada/Ed. El diseño del generador de código y de la biblioteca de tiempo de ejecución fue realizado por Philippe Kruchten (ahora en Rational) y Jean-Pierre Rosen (ahora en Adalog en París). La versión en C fue completada por Bernard Banner y Gail Schenker, que fueron más tarde miembros senior del proyecto GNAT. Ada/Ed-C fue mejorado por Michael Feldman y sus estudiantes

en la Universidad George Washington. Ellos le añadieron un entorno amigable, similar al de Turbo-Pascal, que hizo el sistema mucho más accesible a los estudiantes principiantes. El resultado, llamado GWU-Ada/Ed debe su éxito a la combinación de facilidad de uso, relativa completitud y disponibilidad libre.

Cuando comenzó la revisión del lenguaje, el venerable sistema Ada/Ed encontró un nuevo uso como herramienta de prototipado. Una de las principales reglas de base en el diseño de Ada95 ha sido minimizar la cantidad de sorpresas que tuviesen que afrontar los implementadores al hacer la transición desde compiladores Ada83 a compiladores Ada95. Merece la pena señalar que a pesar de que los diseñadores de Ada83 consideraban que su diseño era conservador, Ada rozaba los límites de la tecnología de compiladores en la década anterior, y los compiladores robustos aparecieron en el mercado mucho más tarde de lo esperado. Para evitar una situación similar con los compiladores de Ada95, la oficina del proyecto Ada9X decidió financiar varios esfuerzos de prototipado para evaluar la complicación inherente de las construcciones que se proponían para el nuevo lenguaje.

El equipo de la Universidad de Nueva York se centró en las construcciones de orientación a objetos (tipos etiquetados, extensiones de tipo, despacho dinámico) y en los nuevos mecanismos de genericidad, que están fuertemente relacionados. Estas características (tal como eran en 1991/92) fueron implementadas en SETL, dentro del viejo sistema Ada/Ed, y probaron ser relativamente simples de construir, confirmando las afirmaciones del Equipo de Revisión que "las partes difíciles de Ada9X son las características de Ada83".

4. Hacia Ada9X

A mediados de 1992 el equipo de la **NYU** obtuvo un contrato de la Oficina del Proyecto Ada9X y de las Fuerzas Aéreas de los EE.UU., para construir un compilador libre para Ada9X, usando la tecnología de GCC para la fase de generación de código. El propósito del sistema era poner en las manos de los usuarios de Ada una herramienta con la que pudieran explorar el nuevo lenguaje, lo antes posible. Se puso el énfasis en la pronta disponibilidad y no en la completitud, y se dio por hecho que el sistema resultante no sería validado y que la implementación temprana de las características de orientación a objetos era más importante que disponer de todas las comprobaciones semánticas. Además, la elección de GCC significaba que adoptaríamos las políticas de *copyright* de la *Free Software Foundation*, consiguiendo de esa forma que el sistema estuviese disponible para el mayor número de usuarios potenciales.

Además de las ventajas de distribución, y de nuestra simpatía con los objetivos de la *Free Software Foundation*, había dos razones importantes para realizar esta elección. La primera era técnica: la calidad del código generado por el sistema de compilación GCC y su destacable reorientabilidad. La segunda era que el mantenedor principal de GCC, Richard Kenner, era un miembro de la plantilla de otro proyecto de la Universidad de New York y estaba interesado en extender GCC a otros lenguajes, otras plataformas, y otras áreas de aplicación.

Nuestras discusiones técnicas iniciales con Richard Stallman, fundador y espíritu guía de la *Free Software Foundation* nos llevó a tomar otra decisión técnica importante: el modelo de compilación de GNAT debería ser lo más parecido posible a los de otros lenguajes. Esto significaría que la relación con otros lenguajes sería mucho más simple. También significaría que el modelo convencional de Ada, una biblioteca de programa monolítica alrededor de la cual trabajan todas las compilaciones, debería ser alterado sustancialmente.

Gracias a la astuta insistencia de Stallman encontramos una forma de seguir el modelo de "una compilación, un fichero objeto" y al mismo tiempo preservar la semántica de las reglas de compilación de Ada, que resultaron no necesitar un biblioteca centralizada. El resultado es un sistema mucho más cómodo para los usuarios no habituados a Ada y que funciona bien en un entorno de programación con mezcla de lenguajes.

5. El desarrollo de GNAT

Otra decisión técnica (y política) temprana fue utilizar Ada para escribir GNAT. Aunque el resto de GCC está escrito en C (unas 500K líneas entre los varios frontales, el dorsal común, las utilidades y los ficheros de descripción de máquina) no era adecuado usar C para el subsistema Ada. Comenzamos escribiendo un compilador para un pequeño subconjunto de Ada83, y usamos un compilador comercial para compilarlo. En junio de 1993 el subconjunto era suficientemente completo para compilarse a sí mismo, y el compilador se autoarrancó a tiempo para la conferencia de Ada-Europa. El uso de las características de Ada9X en el propio compilador ha crecido sustancialmente, según la implementaron se ha ido haciendo más completa, y actualmente GNAT sólo puede ser compilado con el mismo. El compilador usa bibliotecas hijas muy frecuentemente y en menor medida tipos etiquetados. Es muy poco el código que usa el despacho dinámico.

El uso de Ada para el frontal establece una división clara entre las dos mitades del sistema y nos permite programar en nuestro lenguaje favorito. Sin embargo, hace necesaria la construcción de una nueva interfaz entre un frontal para un lenguaje específico y un generador de código independiente del lenguaje.

Ha resultado poco práctico generar directamente el árbol de sintáxis abstracta usado por otros frontales de GCC, debido fundamentalmente a la diferencia semántica entre Ada y C (el lenguaje para el cual se diseñó la parte interna de GCC). En su lugar elegimos construir el frontal alrededor de un árbol más apropiado para Ada, y diseñar una interfaz puramente funcional que recorra el árbol Ada y genere al vuelo el árbol GCC que dirige la generación de código. Esta fase de traducción de árboles fue llamada **Gigi** (GNAT to GNU) y fue diseñada originalmente por Franco Gasperoni, actualmente en la Escuela Superior de Telecomunicación de París. El diseño posterior y la implementación de Gigi se llevaron a cabo por Brett Porter, Richard Kenner, Cyrille Comar y otros miembros del equipo GNAT.

6. Desarrollo de software en una comunidad global

La **Free Software Foundation** es una organización sin ánimo de lucro fundada por Richard Stallman con el propósito de promover el desarrollo de software libre. Los principales productos de la FSF son: el editor extensible **EMACS**,

el **sistema GCC** de compiladores reorientables, el **depurador GDB** y un sistema operativo en desarrollo, originalmente llamado **GNU** (que es un acrónimo recursivo que significa "GNU No es Unix"). El modelo de distribución de software libre promovido por la FSF es diametralmente opuesto a la existencia de patentes de software. La FSF distribuye sus productos bajo unas reglas que están dirigidas a promover su difusión y a prevenir que se dirijan reclamaciones de propiedad contra ellos. Estas reglas se encuentran en un anuncio de *copyright* que se añade a cada producto, llamado **Licencia Pública GNU** (**GPL** también conocida como *copyleft*). La GPL proporciona derechos ilimitados de copia a cualquier usuario del software y obliga a cualquier usuario que modifique el software y lo redistribuya a distribuir también los fuentes modificados.

La *Free Software Foundation*, por su propia naturaleza, depende fundamentalmente del trabajo de voluntarios. El mantenimiento de GCC y sus varios frontales, y los transportes del sistema a nuevas máquinas y nuevos sistemas operativos han dependido de los esfuerzos y la buena voluntad de muchos. El desarrollo de GNAT se ha visto beneficiado del mismo entusiasmo manifestado sobre Internet, y GNAT hubiera tenido un pequeño impacto en la comunidad Ada si no hubiese sido por los esfuerzos constantes de pacientes usuarios. El modelo de distribución de GCC es obviamente un tremendo activo: los usuarios pueden examinar los fuentes, delimitar errores, sugerir mejoras, colaborar en la realización de mejores implementaciones, etc. Los usuarios experimentados en GCC pueden transportar el sistema a nuevas plataformas apoyándose en la facilidad con la que GCC puede ser configurado como compilador cruzado. Gracias a sus esfuerzos han aparecido ya versiones de GNAT para *Linux*, *FreeBSD*, *NetBSD*, *NextStep*, *AmigaDOS* y *Solaris*, y un compilador cruzado para el 1750A. Debemos dar las gracias a todos ellos y a los que han ayudado con sus sugerencias a que GNAT continúe mejorando constantemente su calidad.

7. Larga vida para GNAT

El sistema GNAT ha proporcionado a la comunidad Ada una oportunidad de examinar Ada95 incluso antes de que el lenguaje consiguiera su normalización por **ISO**, y mucho antes de que el propio GNAT estuviera completo. La existencia de un compilador parcial ha sido beneficiosa para la comunidad de usuarios y recíprocamente, el interés de esta comunidad ha facilitado el desarrollo del propio compilador. Este interesante modelo de desarrollo exploratorio debe ahora dejar paso a la práctica industrial. El uso serio de cualquier compilador por una organización industrial requiere ciertas garantías sobre el soporte disponible para él. En el caso de Ada la validación formal se requiere antes de que los proyectos de misión crítica puedan usar un compilador dado, y se requieren nuevas validaciones según el juego de pruebas ACVC evoluciona. El proyecto GNAT de la Universidad de Nueva York no está en la posición de validar GNAT ni de proporcionar el mantenimiento a largo plazo que los usuarios necesitarán. En su lugar hemos decidido crear una organización privada cuyo propósito será mantener el compilador y asegurarse de que permanecerá libremente disponible bajo el modelo GPL.

Esta organización, *Ada Core Technologies*, ofrece contratos de mantenimiento a los usuarios industriales de GNAT que lo requieran. Las mejoras y la corrección de erratas que surgen de esta actividad de mantenimiento son incorporadas en el sistema, que se continúa distribuyendo libremente sobre Internet. Nos beneficiamos de la misma sinergia entre desarrolladores y usuarios de GNAT que nos han acompañado en los comienzos del proyecto, y vemos con satisfacción que GNAT ayuda a difundir el uso del mejor lenguaje de programación de hoy en día. La versión 3.09 de GNAT acaba de ser distribuida al público. Además de la difusión a través de Internet, esta versión está incluida en el CD que será distribuido a todos los asistentes a la próxima conferencia ACM, un público bastante más amplio que la sola comunidad Ada.

Los años venideros indicarán si el modelo mixto de desarrollo y mantenimiento que describimos es viable a largo plazo. Citando una vez más a Richard Stallman: "la palabra libre en software libre tiene el mismo sentido que en expresión libre y no que en cerveza gratis[1]" . La actividad de mantenimiento de un sistema de software complejo no puede realizarse simplemente a través del trabajo voluntario: la calidad del sistema solo puede asegurarse gracias a un equipo dedicado a tiempo completo a esta tarea, y remunerado en forma adecuada. Al mismo tiempo, la distribución libre del sistema trae ventajas evidentes a la comunidad de usuarios, principalmente porque la mejora del sistema se vuelve en cierta medida una actividad colectiva. La disponibilidad del sistema permite la evolución independiente de otras herramientas de programación que complementan al compilador.

Por ejemplo, una colaboración entre Sergey Rybin, de la Universidad de Moscú, y Alfred Strohmeyer, del Politécnico de Lausane, está desarrollando una implementación de **ASIS** ***(Ada Semantic Interface Specification)*** basada en el frontal de GNAT. También viene al caso mencionar el sistema de programación distribuida, que implementa el modelo descrito en el anexo E del manual de referencia. Este modelo, que tiene una funcionalidad parecida a la de CORBA pero con un modelo semántico mucho más sencillo, fue el objeto de investigaciones independientes por grupos en Francia (Yvon Kermarrec y Laurent Pautet) y en Tejas. El equipo de Yvon Kermarrec y sus alumnos modificó partes importantes de GNAT para generar el código de comunicación entre las particiones de un programa distribuido. Laurent Pautet y sus colaboradores construyeron varias implementaciones de la capa de comunicación del sistema, basada en mecanismos estándar en Unix. El resultado es ahora un sistema robusto llamado **GLADE** ***(GNAT Libraries for Ada Distributed Execution)*** cuyo mantenimiento está a cargo de la sucursal Europea de ACT, localizada en París. El mismo modelo mixto de desarrollo se aplica ahora a GLADE: es distribuido libremente, pero **ACT-E** ofrece servicios de mantenimiento y adaptación a industriales que lo requieran. La adopción de GNAT y GLADE por grupos industriales importantes en los Estados Unidos y en Europa confirma nuestro optimismo.

Nota

[1] El término inglés "free" puede traducirse por libre o por gratis. En castellano no existe esa ambigüedad, pero en inglés es necesario aclararla.

Software Libre

TEX, LATEX y CervanTEX (Grupo de Usuarios de TEX Hispanoparlantes)

José R. Portillo, Antonio González
Dptos de Matemática Aplicada I y Física Aplicada, Universidad de Sevilla (Spain)

josera@obelix.cica.es
gonfer@obelix.cica.es

Resumen: *Se pretende hacer una breve introducción a TEX, el procesador de textos de alto nivel diseñado por Donald E. Knuth, incluyendo su dialecto más extendido, LATEX. Ambos son productos de software gratuito con una importante red internaciónal de grupos de usuarios. Entre ellos, el grupo de reciente formación CervanTEX, para usuarios hispanoparlantes, intenta organizar la adaptación de estos programas a las características peculiares de nuestro idioma.*

1. ¿Qué es TEX? ¿Y LATEX?

1.1. Introducción

TEX (pronúnciese "tek") es el más potente de los procesadores de textos encaminados a documentos científicos, si bien puede usarse para cualquier tipo de documento. LATEX (y su sucesor LATEX 2ε) es un lenguaje estructurado construido a partir de TEX para la elaboración de documentos tales como artículos, libros, boletines de problemas, boletines de prácticas... Casi todas las revistas de Física y Matemática que se publican en el mundo actualmente, los libros de la editorial Addison-Wesley, etc. se preprocesan utilizando TEX. Una lista no exhaustiva de revistas científicas que aceptan (o exigen) originales escritos en TEX puede encontrarse en la página WWW: *http://www.tex.ac.uk/texarchive/info/biblio/texjourn.bib*

La diferencia esencial entre TEX y otros programas, como WordPerfect, es que TEX no es un procesador WYSIWYG (*What you see is what you get*). Ello quiere decir que un documento en TEX debe ser escrito en forma de fichero fuente y posteriormente compilado. Esto, que en principio puede parecer un inconveniente, puede tornarse en una de sus grandes ventajas, como veremos más tarde. En este aspecto, es similar al lenguaje HTML en que están escritas las páginas WWW de INTERNET, aunque más complejo.

Una de las razones de la difusión del LATEX es que es gratis. Tal como ocurre con el Linux, el LATEX es *freeware*. Esto quiere decir que puede conseguirse libre y legalmente haciendo un ftp anónimo a las direcciones apropiadas (que indicamos más tarde). Esto permite disponer siempre de las últimas versiones (ya que aparece una versión nueva cada seis meses), y es lo que ha facilitado que su desarrollo haya sido realizado por personas a todo lo ancho del mundo.

1.2. Un poco de historia

La primera versión de TEX, obra de Donald E. Knuth (**DEK**), nació a principios de los años ochenta. En su nivel más básico, estaba constituido por un conjunto de órdenes (o primitivas) del tipo "centrar texto", "dejar un espacio vertical", etc. El uso de un lenguaje de tan bajo nivel, si bien muy potente, era muy difícil, por lo que el mismo DEK desarrolló la primera ampliación de TEX, el ***Plain TEX***.

La primera gran ventaja de TEX es que es extremadamente fácil definir macros a partir de las primitivas (o de otras macros). A un conjunto estructurado de macros se le denomina un formato. El Plain TEX fue el primero de ellos. Introducía gran número de órdenes de alto nivel, pero seguía estando más orientado a aspectos tipográficos que a la estructura de los documentos como un todo.

Por ello, en 1985, Leslie Lamport desarrolló un lenguaje construido sobre el *Plain TEX* (el cual lo estaba sobre el TEX), que recibió el nombre de LATEX (el nombre, propicio a chistes de dudoso gusto, se forma uniendo La (de Lamport) y TEX). Consiste en una serie de macros y formatos dirigidos a confeccionar documentos completos (artículos, informes (*reports*) o libros) y así presta especial atención a aspectos tales como estructura en secciones, subsecciones, etc., control de numeración de ecuaciones y referencias cruzadas, citas bibliográficas, tablas y figuras, índices,... conservando toda la potencia a la hora de escribir ecuaciones de gran calidad.

Desde su aparición, LATEX ha conocido una gran difusión en el ámbito científico, siendo hoy día el procesador más usado por matemáticos y físicos y gran número de ingenieros. Es también utilizado y existen formatos preparados para Química Orgánica (incluyendo dibujo de moléculas), Biología, Lingüística, Fonética, etc. Hoy día hay multitud de libros y revistas científicas escritas en LATEX. LATEX es tan programable como TEX y por ello ha estado sujeto a desarrollos ulteriores. Estos desarrollos no siempre han sido compatibles entre sí, ya que fueron realizados independientemente por individuos en todo el mundo. Para armonizar estos avances se creó el equipo LATEX3 con el objeto de desarrollar una versión de LATEX que englobara a las existentes y permitiera un crecimiento uniforme en el futuro. El primer resultado de sus trabajos es el llamado LATEX 2ε, nacido en junio de 1994 y aceptado actualmente como el nuevo estándar mundial. LATEX 2ε contiene a todo lo anterior y experimenta un crecimiento continuo, publicándose una versión cada seis meses.

1.3. Algunos ejemplos

1.3.1. Un documento simple

Para ver cómo funciona TEX, consideremos el documento `hola.tex` siguiente:

```
\documentclass{article}
\begin{document}
Hola, mundo.
\end{document}
```

Este fichero contiene, en primer lugar un encabezamiento, que lo define como un documento en LATEX 2ε en formato `article` (que es el más simple). Después vendría el preámbulo (que aquí está vacío) que contiene cosas tales como el tamaño de la página. Sigue la línea `\begindocument`, que marca el principio del texto. Luego el texto en sí y luego el fin del documento. Todo lo que haya más allá es ignorado.

Este ficherohola.`tex` se compila, generándose un fichero `hola.dvi`, que puede ser visto por pantalla (mediante un previsualizador) o mandado a la impresora, si se dispone de los *drivers* correspondientes.

1.3.2 Un ejemplo de ecuación con referencia

Veamos un ejemplo más complicado, que muestra algunas de las características del LATEX:

```
\documentstyle[11pt]{article}
\begin{document}
La f\'ormula de Cauchy establece que
\begin{equation}
f(z)=\frac{1}{2\pi i}
\oint\frac{f(\zeta)}{z-\zeta}d\zeta \label{cauchy}
\end{equation}
La f\'ormula (\ref{cauchy}) es aplicable a cual-
quier funci\'on anal{'\'\i}tica.
\enddocument
```

Una vez procesado este fichero, el resultado es el siguiente:
La fórmula de Cauchy establece que

$$f(z_0) = \frac{1}{2\pi i}\oint \frac{f(z)}{z - z_0}$$

La **fórmula 1** es aplicable a cualquier función analítica.

Es evidente que la calidad de la ecuación escrita con LATEX es superior a la de otro procesador, pero el fichero que la produce resulta un tanto misterioso. Este fichero es una muestra de las características típicas del LATEX. Las ecuaciones no se escriben en forma gráfica, simplemente se indica dónde comienzan y dónde acaban. A su vez, el contenido de la ecuación está formado por una serie de comandos, p.ej. \pi y \zeta producen letras griegas y \oint produce el signo de integral curvilínea. Algunos de estos comandos admiten argumentos, así \frac{ }{ } produce una fracción, siendo el primer argumento el numerador y el segundo el denominador.

El ejemplo ilustra otra propiedad interesante: las referencias cruzadas. La ecuación anterior contiene una etiqueta (fijada por el comando \label), de forma que cuando posteriormente uno cita a la misma, sólo debe indicar su nombre. El procesador se encarga de asignar el número y modificarlo si es necesario (porque en una redacción posterior se haya insertado o suprimido una ecuación anterior).

Otra propiedad es que el significado de los comandos es dependiente del formato elegido. Supongamos que inicialmente se pensaba escribir un artículo, pero posteriormente se ha decidido transformarlo en un libro. Simplemente hay que cambiar la palabra `article` de la cabecera por `book`. El procesador se encarga de cambiar la numeración de (1) a (3.1) (suponiendo que la misma aparezca en el capítulo 3.)

2. ¿Dónde conseguir TEX?

Por ser gratis (*freeware*) siempre puede pedirse a alguien que ya lo tenga pero si se pretende una versión completa y actualizada, lo mejor es realizar un ftp anónimo al sitio adecuado o conseguir una de las distribuciones en CD-ROM.

Los archivos de TEX y LATEX se encuentran en la llamada ***CTAN* (Comprehensive TeX Archive Network)**, que es un conjunto de ordenadores repartidos por todo el mundo. De estos, dos son los nodos principales (el nodo norteamericano ha desaparecido): *ftp.dante.de* (Alemania); *ftp.tex.ac.uk* (Reino Unido)

El primero de ellos es el nodo local correspondiente a la zona de España y es el más recomendable por su alta velocidad. Existen multitud de *mirrors* (espejos) de estos servidores, donde se puede encontrar copia de la información, si bien es posible que no contengan la última versión. En España hay una copia en *ftp://ftp.rediris.es/*. El grupo CervanTEX (v.infra) está preparando un espejo de CTAN en México y un `ftp`site en Chile, donde podrán obtenerse versiones de TEX preparadas para nuestro idioma.

La estructura de un nodo CTAN es siempre la misma. Existe un directorio `tex-archive` del cual cuelga un árbol de directorios. En particular, existe un directorio `systems`, del que penden las distintas implementaciones. Existen versiones para los siguientes sitemas operativos: **Amiga**, **Apple Macintosh**, **Atari**, **MS-DOS**, **OS/2**, **UNIX**, **VMS** y **Windows NT**.

Para PC compatibles existe una versión bastante extendida que es el emTEX (la "em" deriva de Eberhard Mattes, el autor, que se dedica a tiempo completo a este procesador). emTEX contiene, aparte de las últimas versiones de LATEX 2ε, un conjunto de *drivers* de alta calidad para multitud de impresoras (matriciales, de chorro de tinta o láser). emTEX posee el inconveniente de que no incluye un editor adecuado. Por ello, es conveniente agenciarse (también en CTAN) el llamado *TEXshell*, diseñado para emTEX. Este es un *shell* muy parecido al de **Borland C** para **DOS**, que permite editar, compilar, hacer un *preview* o imprimir. Usa asimismo diferentes colores para marcar los textos según se trate de comandos, texto normal, comentarios, etc.

Entre las implementaciones anteriores hay una ausencia notable: Windows (3.1 ó 95). Existen versiones para Windows, pero no se encuentran en *CTAN* pues son de pago. Hay una gratis, pero no está adaptada al uso de caracteres acentuados. Sí existen *shells*, como *dviwin* que permiten usar un editor diferente y simultanear la edición con el previsualizado del resultado.

Aunque TEX es gratis, algunas implementaciones para diversas máquinas son comerciales. Suelen ser baratas, pues es una de las condiciones que puso Donald E. Knuth para permitir la venta y sólo se cobran las "ventajas añadidas" (mejores editores, previsualizadores, etc.). Por ejemplo, para Macintosh existen dos implementaciones de TEX: ***Textures***, que es comercial y ***OzTEX***, que es gratuita. La primera presenta una interfaz mucho más cómoda, editor propio y previsualizador integrado, y eso es lo que se paga. Una implementación de TEX se incluye también en la distribución *slackware* de *Linux* (UNIX).

3. Ventajas de TEX frente a otros procesadores

En este apartado se indican las más importantes.

3.1. Alta calidad en la edición de ecuaciones

Esta es siempre la razón última por la que un usuario científico se inclina hacia LATEX. Este procesador ajusta los tamaños de paréntesis, integrales, subíndices y superíndices, alinea los elementos de las matrices, construye cajas, etc.

3.2. Permite (obliga a) redactar documentos estructurados

A través de los llamados formatos, LATEX posibilita escribir textos dividiéndolos en capítulos, secciones, subsecciones, controlando en todo momento la numeración y las referencias cruzadas. Construye índices de contenidos, tablas o figuras. Ajusta los tamaños y tipos de letras según la parte del documento en que se hallen.

3.3. Facilidad en la construcción de macros y comandos

A poco de comenzar a usar este procesador, el usuario se encuentra definiendo o redefiniendo comandos para que estos se ajusten a sus preferencias personales. Por ejemplo, es posible que una determinada expresión aparezca repetidas veces. Nada más fácil que definir un comando que reemplace a todo un bloque. O bien, es posible que no guste la forma en que LATEX numera las páginas. Una redefinición al principio del documento permite cambiar esto.

3.4. Se escribe en ASCII

Esto, que al principio puede parecer un inconveniente (ya que implica teclear mucho más) se torna en ventaja al cabo del tiempo. Por un lado permite incrementar la velocidad de escritura (pues no hay que andar tirando de ratón o menús), por otro facilita el uso de cualquier editor de texto (no contiene caracteres de control) y permite su transmisión por correo electrónico (puede escribirse en ASCII de 7 bits). Esto hace que ya muchas revistas científicas admitan artículos escritos en LATEX, enviados por *e-mail*. Ellos lo procesan en el lugar de destino, hacen los cambios necesarios y lo imprimen.

3.5. Es compatible

Un fichero de TEX creado en cualquier ordenador y/o sistema operativo produce la misma salida impresa en cualquier otro ordenador o sistema operativo que use TEX. Ningún otro programa hace lo mismo y ningún otro tiene tantas versiones para diferentes máquinas y sistemas.

3.6. Es gratis

Sin comentarios.

4. Inconvenientes de TEX

¿Tiene TEX inconvenientes? Por supuesto. Si no fuera así, hoy tendríamos LATEX en cada ordenador en cada casa. Veamos algunas de las críticas más habituales.

4.1. Es muy difícil

A diferencia de los procesadores visuales, que pueden usarse casi desde el primer día con resultados aceptables, LATEX requiere un periodo de aprendizaje antes de conseguir los primeros frutos. Incluso cuando ya se es un usuario medio o avanzado, siempre es conveniente tener cerca un manual o un LATEXperto, al que preguntarle. Este proceso de iniciación consiste principalmente en aprenderse los comandos esenciales. Después le siguen comandos secundarios.Más tarde, puede uno aprender a programar en este lenguaje. Llegado ese punto se da el siguiente salto, atacar el *Plain TEX* o incluso el TEX, para introducir comandos de bajo nivel. Estas dos últimas fases sólo son precisas si uno está interesado en los intríngulis del sistema.

4.2. Es oscuro

O dicho de otro modo: uno no ve los resultados hasta que compila el fichero. Una de las decepciones que sufre el usuario novato es que no ve los frutos de su trabajo hasta que compila. Como suelen aparecer errores de compilación, esto suele ser frustrante (aparte de conllevar una pérdida de tiempo). La única solución es armarse de paciencia. Con el tiempo, los errores disminuyen y el resultado final compensa la dificultad y oscuridad de TEX.

De todas formas, LATEX no está especialmente dirigido a los aspectos estéticos, sino a los estructurales. Si uno está interesado en la estética más que en el fondo (por ejemplo, porque trabaja en la autoedición gráfica), LATEX quizás no sea el procesador adecuado. Existen, no obstante, soluciones intermedias entre el LATEX y los procesadores WYSIWYG. Así, *Textures* para *Macintosh* o *dviwin* para *Wintel* permiten mantener en pantalla el fichero que se está editando y el último previsualizado, lo cual facilita la corrección de los documentos. *Textures* incluso actualiza automáticamente el visualizado del fichero compilado al modificar el fuente.

4.3. No permite introducir imágenes

Esta acusación habitual no es del todo cierta. Poderse, se puede, pero no es algo trivial en LATEX. Resulta más fácil dejar el espacio y luego recortar y pegar. Estos problemas se solucionan usando los *drivers* para salidas *postscript* y figuras del mismo tipo. O soluciones particulares para cada máquina, pero ello rompe la compatibilidad de TEX.

4.4. No tiene variedad de fuentes

Al igual que la afirmación anterior, esta crítica no es del todo correcta. Cuando DEK desarrolló el TEX, lo acompañó de un conjunto completo de fuentes de alta calidad. Estas fuentes eran independientes de la impresora utilizada (ya que eran tratadas como gráficos) lo que producía la misma calidad en cualquier sitio. Con el tiempo, estas fuentes (las *Computer Modern* o **CM**) se convirtieron en la "firma" del procesador, que permitía identificar un documento a primera vista. A medida que el LATEX se difundía han aumentado las demandas de más fuentes para usos diversos. El crecimiento ha sido lento, pero el número de fuentes disponibles actualmente es muy elevado. Además, en las versiones para salidas *postscript* sí es posible el uso de cualquier fuente.

4.5. Ocupa mucha RAM y mucho disco duro

Correcto. Hasta hace unos años. Las disponibilidades de RAM y disco duro son hoy mucho mayores, incluso para el usuario doméstico. Y además, las exigencias de sistema de todos los procesadores son hoy mucho mayores. Una instalación completa de TEX gasta seguramente menos recursos que Word 7.0, por ejemplo.

5. ¿Dónde puedo conseguir más información?

5.1. Libros y revistas

Debido a la difusión limitada de este procesador, existe una escasa bibliografía en castellano. Sí hay en inglés, e incluso los ficheros fuente (escritos en TEX, por supuesto) son accesibles en la Web. Algunos títulos al respecto son:

- *A Guide to LATEX 2ε*, H. Kopka and P.W.Daly, Addison-Wesley (1995). Este es, en opinión de muchos, el mejor manual existente sobre LATEX. Contiene una guía completa de comandos, abundantes ejemplos e información adicional. (Incluye las dos versiones en uso del LATEX, la 2ε y la más antigua, 2.09).
- *The LATEX Companion*, M. Goosens, F. Mittelbach and A. Samarin, Addison-Wesley (1994). Este manual sirve de ampliación del anterior. Es una recopilación e información sobre los llamados *packages*, conjuntos de macros que distintos autores han puesto a disposición de la comunidad.
- *LATEX-A Document preparation system* L. Lamport Addison-Wesley (1985 y 1994). Durante mucho tiempo este fue "el libro" del LATEX. Escrito por el mismo autor del programa, contiene todo lo esencial para introducirse en este procesador, si bien resulta un tanto insuficiente para usuarios avanzados. La primera edición corresponde a la versión antigua del programa (hoy obsoleta) y la segunda al LATEX 2ε.
- *The TEXbook*, D.E, Knuth, Addison-Wesley (1984). Este es el libro para los que quieren conocer las entrañas del LATEX. Contiene todo sobre el lenguaje TEX, escrito por el mismo autor del programa. Es un libro completo y ameno pero absolutamente incomprensible para los novatos. El fichero fuente en TEX de este libro está disponible para aquellos que desean conocer los trucos que el autor empleó a la hora de escribirlo (algunos muy complejos).
- *LATEX primeros pasos*, F. Ortegón Gallego, Masson (1992). Un buen manual de introducción que además está en castellano.
- *Una descripción de LATEX 2ε* , Tomás Bautista (1996). Basado inicialmente en LaTeX-Kurzbeschreibung de Hubert Partl, Elisabeth Schlegl e Irene Hyna es un buen manual de introducción a LATEX 2ε en español y muy actualizado. Además, se encuentra disponible gratuitamente a través de INTERNET, en: *ftp://ftp.cma.ulpgc.es/pub/users/bautista/ldesc2e/ldesc2e.ps.gz* (versión PostScript).
- *Composició de textos científics amb LaTeX*. G. Valiente. Edicions UPC, Barcelona, 1996. En catalán. Se espera la traducción al español para 1997.

Existe bibliografía adicional en cada uno de estos libros, así como en los nodos de la **WWW** dedicados al TEX. También se incluyen más referencias al final de este artículo.

Existe además una revista dedicada exclusivamente al TEX, el llamado *TUG-boat*, publicado por el grupo de usuarios de TEX (TUG). TUG edita también un boletín de noticias y novedades sobre TEX y LATEX: *TEX and TUG news*. Ambos son accesibles en CTAN.

5.2. Internet

El gran impulso del desarrollo de TEX en el mundo ha sido la red de ordenadores INTERNET. El trabajo activo de un grupo de voluntarios extendidos a lo ancho del planeta y la organización mediante los puntos de referencia de CTAN han conseguido hacer de TEX un producto ejemplar. Las nuevas versiones de TEX, los *bugs* de estas, los problemas y soluciones eran puestos en la red gratuita instantáneamente. Por ello no debe extrañarnos que INTERNET sea una de las mejores fuentes de información para todo lo relacionado con este programa.

Para empezar, existen unas **FAQ's** (lista de preguntas frecuentes o *Frequent Asked Questions* sobre TEX y LATEX, escritas y mantenidas por el TUG inglés donde se encuentra información de lo más completa y variopinta. Es *searchable*, esto es, puede consultarse un término concreto. La dirección es *http://www.chem.emory.edu/latex.faq*

También existe una guía muy completa en francés: (*http://curia.ucc.ie/info/TeX/menu.html*). Contienen información sobre comandos, documentación, FAQ's, etc. aparte de las típicas cuestiones técnicas.

Existe un nodo dedicado específicamente al TEX. Está mantenido por el grupo de usuarios alemán y se encuentra en *http://www.dante.de*. El único (y no pequeño) inconveniente es que está en alemán (algunas páginas también en inglés). En él aparecen datos, bibliografía, direcciones diversas.

Un nodo en inglés con abundante información es el de la universidad de Cambridge: *http://www.tex.ac.uk* . También se puede conseguir información en otro nodo: *http://www.shsu.edu*. Pero este es un nodo general y hay que explorar un poco hasta llegar a la información buscada.

También están las conexiones ftp ya indicadas antes, y un nodo gopher: *gopher.dante.de*.

Para información sobre TEX en español en la red, la referencia principal es la página del grupo de usuarios de TEX hispanoparlantes: CervanTEX. Su dirección es: *http://gordo.us.es/Actividades/GUTH* y contiene numerosos enlaces a otros puntos de información. La FAQ sobre TEX en español está también accesible mediante ftp en *ftp://ftp.unex.es/pub/tex/faq/*

Además de la información pasiva disponible en red, existen foros de discusión sobre estos procesadores. En ellos los novatos suelen plantear sus dudas y los "latexpertos" (a veces el mismo DEK) pueden dignarse a contestarlas. También se discuten las ventajas y defectos de las diferentes versiones, implementaciones, fuentes, etc. El grupo principal es *comp.text.tex* en el que el idioma oficial es el inglés. Aparte existe un grupo de habla hispana (aunque muy poco utilizado): *es.comp.lenguajes.tex* y numerosos grupos locales.

5.3. Asociaciones de usuarios: CervanTEX

Existen asociaciones de usuarios de TEX (los *TUGs* o *TEX Users Groups*) distribuidas por el mundo, en las cuales los miembros se dedican a desarrollar proyectos conjuntos para la difusión y mejora de este procesador (sin una empresa que lo respalde, depende exclusivamente de los usuarios).

El nombre de TUG suele reservarse para el grupo internaciónal, con más de seis mil miembros individuales y unos cien miembros institucionales, entre los que se encuentran instituciones de elevado prestigio como el CERN, Hong Kong University, Los Alamos National Laboratory, Princenton University, Smithsonian Astrophisical Observatory, Uppsala University, Yale University, etc.

Simultáneamente a este gran grupo internacional, los usuarios de TEX tienden a agruparse con aquellos otros usuarios que encuentran las mismas dificultades. Por ello, los grupos locales de TEX acogen a personas que utilizan el mismo idioma (*Dante* para el alemán, *GUTemberg* para el francés...) o una familia de idiomas con construcciones similares (Hay un único grupo para los países escandinavos).

Posiblemente debido a la escasa tradición de asociacionismo que hay entre latinos, el grupo de usuarios hispanoparlantes, denominado inevitablemente *CervanTEX* es de los últimos en formarse. Como es frecuente entre los usuarios de TEX, la Internet es fundamental en la formación de este grupo y prácticamente la totalidad de sus miembros son usuarios activos de la red.

Como antecedente de este grupo es forzoso citar a la lista de correo electrónico *spanish-tex* (*spanish-tex@eunet.es*. Para suscribirse, mandar un electrocorreo a *listserv@eunet.es* que incluya en el cuerpo (*body*) del mensaje la siguiente línea: **subscribe spanish-tex Nombre**. Para desuscribirse (borrarse) de la lista, el mensaje (a la misma dirección) debe incluir (en el cuerpo): **signoff spanish-tex**)

Aunque CervanTEX es un grupo nuevo, aún en fase de organización, ya son bastantes las actividades que mantiene. Podrían destacarse:
- Elaboración de un boletín (para el que se ha propuesto el nombre de QuijoTEX)
- Unificación de los patrones de guionado (partición de palabras), pues estos son diferentes para cada idioma y existían varias versiones para el castellano.
- Implementaciones sobre diversas plataformas de las distribuciones preparadas para el español de *Plain TEX*, LATEX 2ε, babel y spanish.sty.babel es una extensión de TEX multilingüe.
- Preparación y mantenimiento de diccionarios, verificadores ortográficos y otras herramientas TEXnicas.
- Localización de formatos y estilos en español.
- Creación y mantenimiento de un servidor WWW y espejos del mismo en Chile y en México.
- Creación y mantenimiento de un servidor ftp para TEX en castellano.
- Mantenimiento de la FAQ de TEX en español.

La coordinación del grupo la efectúa actualmente uno de los coautores de este artículo, José R. Portillo. Las diversas actividades se hallan repartidas entre grupos de trabajo cuyos miembros viven en diferentes ciudades o países y trabajan usando la red INTERNET.

6. Referencias

[1] Michel Goossens, Frank Mittelbach and Alexander Samarin. *The LATEX Companion*. Addison-Wesley, Reading, Massachusetts, 1994, ISBN 0-201-54199-8.

[2] Tomás Bautista. *Una descripción de LATEX 2ε, ftp://ftp.cma.ulpgc.es-/pub/users/bautista/ldesc2e/ldesc2e.ps.gz,* 1996.

[3] D. P. Carlisle. *Packages in the 'graphics' bundle.* Se incluye en el conjunto 'graphics' como grfguide.tex, disponible en el mismo sitio de donde se ha tomado la distribución de LATEX.

[4] H. Kopka and P.W.Daly, *A Guide to LATEX 2ε*, Addison-Wesley, Reading, Massachusetts, 1995.

[5] Donald E. Knuth. *The TEXbook*, Tomo A de *Computers and Typesetting*, Addison-Wesley Publishing Company (1984), ISBN 0-201-13448-9.

[6] LATEX3 Project Team. *LATEX 2ε for authors.* Viene con la distribución de LATEX 2ε como usrguide.tex.

[7] LATEX3 Project Team. *LATEX 2ε for Class and Package writers.* Viene con la distribución de LATEX 2ε como clsguide.tex.

[8] LATEX3 Project Team. *LATEX 2ε Font selection.* Se incluye en la distribución de LATEX 2ε como fntguide.tex.

[9] Leslie Lamport. LATEX: *A Document Preparation System.* Addison-Wesley, Reading, Massachusetts, segunda edición, 1994, ISBN 0-201-52983-1.

[10] Cada instalación de LATEX debería proporcionar la llamada *Guía Local de LATEX*, que explica las cosas que son particulares del sistema local. Debería residir en un fichero llamado local.tex. Por desgracia, en algunos sitios no se halla dicha guía. En este caso, pídale ayuda a un experto de LATEX.

[11] F. Ortegón Gallego. *LATEX primeros pasos*, Masson 1992.

[12] G. Valiente. *Composició de textos científics amb LaTeX.* Edicions UPC, Barcelona, 1996.

Software Libre

Experiencia con programas libres en Red EuroSur

Javier Simó*, Miguel Morales*, Joaquín Seoane**
* *Red EuroSur*
** *Dpto. de Ing. Telemática, E.T.S.I. Telecomunicación*

jsimo@eurosur.org
miguel@eurosur.org
joaquin@dit.upm.es

Resumen: *Se expone la experiencia de los autores en Red EuroSur [1], servicio proveedor de información y de acceso a Internet que funciona íntegramente con programas disponibles en fuente y de distribución libre. En primer lugar, se destacan las características, calidad y prestaciones del servicio ofrecido. Seguidamente, se describen las opciones elegidas como sistemas operativos, programas servidores y clientes. Para justificar el uso de dichos programas, se analiza alguno de los utilizados, en comparación con programas comerciales de características similares. Finalmente, se establecen las conclusiones de lo aprendido, tanto en sus aspectos positivos como negativos.*

Palabras clave: *Programas libres, Internet, servicios de información, bases de datos, www, wais, linux.*

1. Introducción

Entendemos como programas libres aquellos que se obtienen fácilmente, sin costo apreciable, y que están disponibles en fuente, de modo que se pueden modificar con facilidad para ajustarlos a las necesidades y para mejorarlos en beneficio de la comunidad de usuarios. En este artículo describimos nuestra experiencia con este tipo de programas en Red EuroSur, un proyecto que integra tres dimensiones:

- Servicio de comunicaciones, en tanto que proveedor de acceso a Internet, especialmente dedicado al mundo de las **ONGs** y organizaciones sociales y educativas.
- Servicio de información, que se ofrece a la comunidad de usuarios de **Internet** e **InfoVia** con contenidos muy completos sobre temas como la cooperación internacional, desarrollo, medio ambiente, derechos humanos, etc., en lengua castellana.
- Servicio de formación e información interno, que permite el acceso desde puestos de trabajo y aulas locales a los dos recursos anteriores y otros propios de la red interna.

Cuando **Red EuroSur** nació, se juntaron una serie de condicionantes que hicieron optar desde el principio por programas libres: no había recursos suficientes para adquirir productos y servicios comerciales y, sin embargo, había que buscar soluciones para ofrecer un sistema fiable, sin fallos constantes; si los había, además, teníamos que ser capaces de resolverlos sin ayuda ténica externa.

Para un servicio de estas características se requiere elegir cuidadosamente los sistemas operativos de los servidores (sistemas que van a estar en explotación las 24 horas del día con una carga considerable), y gran cantidad de programas que ofrezcan los servicios críticos: servidores de correo, *web*, conferencias electrónicas, transferencia de ficheros, autentificación de usuarios, buscadores de información, herramientas de automatización de tareas, programas de análisis y creación de estadísticas, etc., así como multitud de herramientas menos críticas pero indispensables para dar un servicio integrado de calidad. Nosotros elegimos en todos los casos programas libres para los servidores. Después de un año funcionando, podemos evaluar los resultados y comparar con otros productos comerciales que hemos conocido entre tanto.

2. El sistema operativo

El sistema operativo, en singular, porque desde el principio consideramos que era preferible tener el mismo en todos los servidores, salvo que alguna necesidad específica exigiera un sistema diferente. Esta decisión ha posibilitado una administración más homogénea y centralizada de las diferentes máquinas, además de disminuir la lógica dispersión que produce el seguimiento de los sistemas para incorporar nuevas versiones, cuando las nuevas características o la corrección de errores justifican el cambio.

Como plataforma física elegimos la arquitectura PC, por su bajo costo y por la mayor variedad de tarjetas y de sistemas operativos disponibles. También consideramos la posible necesidad de transferir tecnología a países en desarrollo, donde la disponibilidad de otras arquitecturas es más problemática que aquí. Como contrapartida, las máquinas instaladas son más voluminosas y menos fiables que arquitecturas más cerradas, con un mantenimiento más artesanal. Por ello nos vimos forzados a adquirir demasiados conocimientos acerca de su estructura física y aprender a elegir componentes apropiados y de calidad suficiente, puesto que no son pocos los problemas que nos podemos encontrar derivados de ellos: hay mucha electrónica de PC que es adecuada para un uso personal y no continuado, pero es desastrosa cuando se le exige el esfuerzo de un funcionamiento continuo. A pesar de esas excepciones, con un poco de cuidado, las máquinas pueden mantenerse durante meses sin incidencias.

Empezamos sin descartar ninguna posibilidad, pero muy orientados al mundo *Unix*, muy potente y probado en esta clase de servicios. Consideramos productos tanto de pago y de distribución libre, evaluando alternativas como **SCO**[2], **BSD/OS**[3], **Linux**[4], **FreeBSD**[5] y **NetBSD**[6], seleccionando finalmente *Linux*. SCO se descartó a pesar de las fuertes presiones a favor de su uso, ya que se disponía de algunos programas costosos y controladoras que sólo funcionaban en dicho entorno. Carente de las herramientas que consideráramos necesarias, opaco y difícil de configurar, fue abandonado en favor de algún sistema que lo emulara. Los restantes

sistemas evaluados cumplían el requisito, disponiendo además de los fuentes del núcleo y las herramientas. De ellos sólo **BSD/OS** es un sistema de pago, soportado por **BSDI** vía correo electrónico. La elección de un sistema *Linux*, o más exactamente, de un sistema *GNU7* basado en *Linux*, no se basó tanto en la calidad percibida como en la cantidad de personas que lo mantenían, conocían y utilizaban, especialmente en el entorno más próximo.

Linux es un núcleo de sistema operativo variante de *Unix* y desarrollado por un pequeño equipo de personas en torno a su diseñador, Linus Torvalds. Bastante estable, si se sabe elegir la versión adecuada, tiene un soporte muy bueno de gran cantidad de dispositivos. Varias distribuciones gratuitas empaquetan un número ingente de programas libres precompilados, lo que en la mayoría de los casos ahorra un trabajo considerable. Los programas básicos de estos sistemas son siempre los del proyecto **GNU**, por lo que es más justo llamarlos sistemas GNU basados en *Linux*, como antes mencionábamos.

No todo son ventajas, desde luego. Hay que mantenerse a la escucha de la evolución del sistema, que es bastante vertiginosa. Salen constantemente parches para el núcleo y siempre se está renovando para incorporar nuevas prestaciones y soportar la mayor cantidad posible de dispositivos que salen para PC. Por otro lado, el avance del hardware es más vertiginoso todavía y hay que tener cuidado con el que se pretende instalar el sistema. Los fabricantes habitualmente sacan sus productos al mercado con discos que contienen los controladores necesarios para **MS/DOS**, **Windows** (3.1, 3.11, 95 o NT) y a veces **SCO** y otros *Unix* comerciales, por lo que los demás sistemas tienen que lograr el soporte por su cuenta.

En cuanto a funcionamiento... bueno, quien está acostumbrado a sistemas operativos para servidores del estilo de **NT** puede quedarse muy sorprendido de lo bien que gestiona los recursos y de la rapidez con que va un *Linux* con el mismo hardware. Nuestra experiencia es de un estupendo rendimiento y un funcionamiento muy estable, una vez se ha logrado el núcleo adecuado al hardware que se tiene. Un PC bien dimensionado en memoria, procesador, etc., con *Linux* no tiene nada que envidiar a otras arquitecturas con *Unix* propietarios, y en muchos casos funciona mejor.

Toda nuestra red de servidores está constituida por máquinas *Linux* de capacidad muy diversa que se mantienen coherentes mediante el uso intensivo de **rdist**, un programa para distribuir ficheros. Disponemos de máquinas Pentium con mucha memoria y *bus* SCSI para los servicios proporcianados al exterior, pero también tenemos un viejo 386SX como servidor de varias impresoras y varios 486 como servidores de terminales, servidores de ficheros NFS y SMB, encaminamiento entre subredes, etc.

La adopción de *Linux* como sistema operativo por los proveedores de Internet está bastante generalizada en España y en el mundo[8] . Gran cantidad de estos lo usan para las máquinas más críticas en sus servicios. No disponemos de estadísticas exactas, pero *Linux* está entre los dos o tres sistemas más extensamente utilizados como nodos de proveedores de servicios Internet, junto con *Solaris*, *Irix* y *Windows NT*.

3. Programas de sistema

Inicialmente el proyecto se planteó como un servicio fuera de línea, ya que no existían posibilidades de conexión a Internet. En esta fase jugaron un papel crucial el paquete **uucp** de Taylor[9] y el **sistema cnews**[10] de Spencer. El primero es a nuestro juicio el mejor paquete uucp existente, al menos por su versatilidad en el soporte de modems y de una gama muy amplia de protocolos de transferencia, algunos muy eficientes. En la disponibilidad de esta gran variedad de protocolos jugó un papel crucial la aportación de colaboradores desinteresados en la Red. Ni comparación con el primer uucp utilizado en el sistema SCO, donde era necesario parchear los binarios para cambiar parámetros de los protocolos.

Creemos importante mencionar que los agentes de correo (*sendmail* [11]) y noticias (*cnews* primero, **inn**[12] después, con la conexión a Internet), así como otros muchos programas vitales (servidor de nombres, dominios de encaminamiento, etc.) son programas libres que prácticamente no han encontrado sustituto comercial, aunque en muchos casos se distribuyan como parte de un sistema comercial cerrado, a veces utilizando versiones obsoletas modificadas de las versiones públicas.

Cuando hubo que plantearse la elección de un servidor de *Web* nos encontramos de nuevo con que no tenía demasiado sentido enfrentarse a un producto comercial. Disponíamos de productos como *Apache*[13] , que son de altísima calidad y que tienen un soporte notable. Frente a los programas comerciales como los de *Netscape* o Microsoft, el concepto de soporte en los programas de libre distribución, no se refiere a una garantía por parte de una empresa productora de resolver problemas a sus usuarios, sino que tiene más que ver con la colaboración de los usuarios entre sí a través de conferencias electrónicas y listas de correo. El caso es que ese soporte, junto con la documentación, resulta suficiente en la mayoría de los casos.

Nosotros nos enamoramos del proyecto *Apache*. Al ser un servidor de *Web* modular de libre distribución y con una interfaz de programación bien definida, ha dado lugar a la creación de gran cantidad de módulos por parte de usuarios del producto. Algunos de éstos, de suficiente calidad y adecuadamente depurados y documentados, se incorporan a versiones posteriores de la distribución oficial. Esto significa que *Apache* es un servidor de calidad, adaptable a las necesidades de cada uno gracias a la modularidad, y que tiene todas las características que uno puede razonablemente pedir precisamente gracias a la colaboración de mucha gente en el desarrollo del producto. Además, hasta la posibilidad de poner en funcionamiento un servidor seguro nos la solucionó *Apache-SSL* que es también de distribución libre.

Otra cuestión relacionada con el servidor de *Web* y el sistema de información era la elección de uno o varios

programas que permitieran la búsqueda eficiente de documentos y las consultas a bases de datos. Para búsqueda de documentos por contenido, encontramos que la gran parte de lo que hay de calidad es de libre distribución; hay productos comerciales, pero de nuevo los libres ofrecen suficientes prestaciones. Por diversas razones optamos por *freeWAIS-sf*[14] y *SFgate*[15] .

Aquí le sacamos partido a otra de las características de los programas de libre distribución: la disponibilidad de los códigos fuentes de los programas. *SFgate* es una pasarela entre *freeWAIS-sf* y el servidor de *Web*, pero cuando llegó a nosotros la interfaz de usuario estaba sólo en inglés, francés y alemán; hicimos la traducción al castellano y la pusimos a disposición del resto de la comunidad de usuarios, incorporándose en la distribución oficial a partir de la versión 4.0.13. En cuanto a *freeWAIS-sf*, hicimos importantes modificaciones en el código en C para adaptarlo a un motor de búsquedas de bases de datos CDS/ISIS, basándonos en código de otra distribución también libre. Esto permitió disponer de unos productos muy adaptados a nuestras necesidades.

Cuando nos empezaron a interesar las bases de datos relacionales, encontramos productos de libre distribución, pero aquí nos inclinamos por uno que no lo era del todo. Por muchas razones (integración de la administración de usuarios con *Apache*, velocidad, sencillez,...) optamos por **Minerva SQL**[16], que es gratuito para nosotros como organización sin ánimo de lucro, pero de pago para las empresas. **MSQL** empezó siendo de *shareware*, pero dada la poca disposición de las empresas a aportar dinero al proyecto (3 de 2000 licencias aportaron algo, al decir del Sr. Hughes, su autor) pasaron a esta última formula.

Otros productos que utilizamos, como los programas de estadísticas de uso, pasarelas, cortafuegos, etc, son también programas libres, por lo que hemos podido adaptarlos a nuestras necesidades en muchos de los casos. En otros casos, son grandes compañías las que han hecho modificaciones a programas libres, como es el caso de Telefónica con **Radius**[17], que usamos en nuestro servicio y es imprescindible para el funcionamiento de **InfoVia**. No hay que olvidar los procesadores de lenguajes necesarios, como el excelente compilador de C de **GNU**, el intérprete de órdenes *bash*, también de GNU, o el ubicuo lenguaje de administración y prototipado **Perl**[18] .

La experiencia con nuestros sistemas es de calidad, estabilidad y dinamismo. Empezamos no viendo la necesidad de embarcarnos en productos comerciales y seguimos en la misma línea. La contrapartida ética de esta decisión es estar dispuestos a compartir cuantas mejoras podamos hacer en los programas y apoyar en las conferencias electrónicas a usuarios que se encuentran con problemas que hemos resuelto.

4. Programas de usuario

Si en un ámbito más puramente técnico, donde las máquinas están directamente controladas por los administradores, los programas de libre distribución son más que satisfactorios, el panorama cambia de manera radical al encontrarnos con los programas que el usuario va a usar directamente en su trabajo. El motivo es la imposición por parte de los usuarios de entornos para los que no abundan los programas libres.

Salvo excepciones, el usuario final siempre quiere trabajar con Windows (en sus distintas versiones), MAC/OS o MS/DOS, fundamentalmente porque ya utiliza el entorno, necesita herramientas que sólo funcionan en él, le resulta más fácil de usar, desconoce otros sistemas, tiene con quien intercambiar experiencias y dónde buscar ayuda, o ha sido convencido de su bondad por la propaganda o sus colaboradores o amigos. Cualquier otra elección no es en absoluto bienvenida, aunque se le mencionen ventajas como la mayor eficiencia o una menor inversión en hardware.

En el mundo Windows no existen muchos programas libres, al contrario que en MS/DOS y sobretodo en el mundo *Unix*, por lo que poco podemos hacer por fomentar su uso. No hay que confundir los programas libres con el denominado *shareware* (término a veces traducido como programas de coste compartido), que son programas comerciales, generalmente muy económicos, que se distribuyen dejando copias accesibles en diversos medios y pidiendo una contribución económica si el programa gusta y se va a usar. Estos programas casi nunca se distribuyen en fuente y existen en gran cantidad para MS/DOS, Windows o MAC/OS.

A pesar de todo esto, dada la fuerte apuesta por los programas libres, optamos por adoptar varios de ellos en nuestra red interna. Por ejemplo, desde un principo se han utilizado los clientes de *Telnet* (*telbin*) y *FTP* (*ftpbin*) de **NCSA**[19] para MS/DOS, dada su sencillez, eficiencia y flexibilidad, así como por su capacidad para funcionar en el ordenador más humilde. Estos programas se pueden utilizar en un entorno de red local gracias a otro hito en la programación libre: la especificación del *packet driver* [20] por FTP Software y la construcción de muchos manejadores (libres o no) de tarjetas *ethernet* acordes con dicha especificación. Junto con demultiplexores de paquetes, también libres, se pueden hacer coexistir varias torres de protocolos, permitiendo el uso simultáneo de dichos clientes con un sistema de ficheros remoto (**NFS**). Por último cabe destacar que gracias a un manejador de PPP de la Universidad de Michigan (*etherppp*[21]), nuestros usuarios pueden conectarse al servicio a través de InfoVía con una máquina de poca capacidad y *telbin*, y utilizar clientes en modo texto en el otro extremo (*pine*[22] para correo y noticias, *lynx*[23] para hipertexto WWW). En honor de estos usuarios, hemos insertado entre la sesión de terminal remoto y la aplicación un traductor entre cualquier juego de caracteres e **ISO-Latin1**, nuestro juego oficial. En dicho filtro juega un papel crucial el programa libre *recode* de GNU.

Para correo y conferencias electrónicas hemos apostado muy fuerte desde un principio por Pine, de la Universidad de Washington. Este es de los pocos programas de distribución liberal con versiones para Unix, MS/DOS, OS/2 y los diversos Windows. Derivado de *elm*, ofrece una interfaz sencilla y coherente al correo electrónico y a las conferencias

o noticias. Desde el principio soportó **MIME** y por tanto alfabetos internacionales, requisito casi imprescindible para nuestro entorno.

Ya que disponíamos de los fuentes del programa, hicimos la traducción completa al castellano de la versión 3.91, así como algunas mejoras, como autentificación en las conferencias y la corrección de pequeños fallos. El trabajo se hizo para todas las arquitecturas, pero sólo maduró en el entorno *Unix*. Ello se debió a que fuimos incapaces de compilarlo para MS/DOS y Windows, desconocíamos el compilador usado y los desarrolladores ignoraron las peticiones que se lanzaron pidiendo esos datos. En este sentido, el apoyo mutuo característico de la programación libre no pudo llevarse a cabo: no eramos los únicos que teníamos esos problemas y nadie más supo o quiso responder. Si eso ocurre, las horas de trabajo crecen exponencialmente y puede ser necesario abandonar la modificación de programas de este tipo, en favor de versiones comerciales.

A pesar de estos problemas, Pine (o su versión castellana, Pino) se sigue usando en nuestra red interna y por parte de varios usuarios de la externa que lo encuentran más cómodo y potente que los programas para DOS o Windows. Y está disponible para quien lo quiera en nuestro FTP anónimo[24] . No obstante no consideramos esta una versión definitiva y estamos trabajando en la internacionalización de Pine 3.95, de modo que pueda usarse en cualquier lengua. Para ello utilizamos el paquete *gettext* de GNU para utilizar catálogos de mensajes en diversos idiomas.

Finalmente cabe mencionar que la necesidad de dar soporte a las redes Microsoft, especialmente con Windows-95, no nos ocasionó demasiados problemas, ya que están perfectamente soportadas por programas libres. Microsoft utiliza un protocolo llamado **SMB** para compartir ficheros y otros recursos. Dicho protocolo puede funcionar sobre una torre TCP/IP. Un servidor libre llamado *samba*[25] ofrece servicios SMB en cualquier plataforma *Unix*. Además el núcleo de *Linux* soporta SMB como cliente. Por tanto podemos compartir recursos de Unix desde máquinas Windows y recursos de Windows desde máquinas Linux.

5. Conclusiones

Para nosotros la experiencia con los programas libres es muy positiva, más que por su gratuidad, por la facilidad de acceso a los mismos, especialmente por medio de Internet, y también por recopilaciones en discos ópticos compactos, pero sobre todo por la libertad de modificación y adaptación a las necesidades.

Creemos que el entorno colaborador de la gente que desarrolla estos programas, en muchos casos, da lugar a resultados más flexibles y más depurados que sus contrapartes comerciales.

Las dificultades encontradas se han debido casi siempre a la quizá saludable falta de normalización en el mundo *Unix*, y pocas veces debidas a errores en los programas. Hoy día en el mundo académico es inconcebible el trabajo sin este tipo de programas, y cada vez más la misma filosofía está penetrando en el mundo comercial, muchas veces sin querer admitirlo.

Reconocemos que en el mundo de las aplicaciones de usuario, los programas libres llevan un gran retraso y que es preciso encontrar fórmulas para que los desarrolladores puedan vivir de su trabajo. Mientras tanto no debemos olvidar que el que usa programas libres se hace responsable de mejorarlo y distribuir esa mejora por los canales adecuados en cada caso.

Finalmente no hay que olvidar la presión de los usuarios, a los que hay que servir. Cuanto más cerca de ellos, más difícil es trabajar con programas no comerciales. Las empresas y organizaciones grandes y pequeñas suelen ser incondicionales de NT, Novell y cosas así. Por ejemplo, la búsqueda de soluciones para usuarios corporativos nos ha obligado a considerar seriamente la necesidad de un servidor NT en nuestra red. Pero esto es ya otra historia.

Notas

1 *http://www.eurosur.org*
2 *http://www.sco.com*
3 *http://www.bsdi.com*
4 *http://www.linux.org*
5 *http://www.freebsd.org*
6 *http://www.netbsd.org*
7 *http://www.gnu.org*
8 *http://www.anime.net/linuxisp/*
9 *http://www.cygnus.com/»ian/uucp.html*
10 *ftp://ftp.zoo.toronto.edu/pub/c-news*
11 *http://www.sendmail.org*
12 *http://www.isc.org/inn.html*
13 *http://www.apache.org)*
14 *http://ls6.informatik.uni-dortmund.de/ir/projects/freeWAIS-sf/*
15 *http://ls6.informatik.uni-dortmund.de/ir/projects/SFgate/*
16 *http://Hughes.com.au*
17 *http://www.merit.com/radius*
18 *http://www.perl.com*
19 *http://www.ncsa.uiuc.edu/SDG/Software/PCTelnet*
20 *http://www.crynwr.com*
21 *ftp://ftp.merit.edu/internet.tools/ppp/dos/*
22 *http://www.cac.washington.edu/pine*
23 *http://lynx.browser.org*
24 *ftp://ftp.eurosur.org*
25 *http://samba.canberra.edu.au/pub/samba*

Software Libre

Juan Jesús Muñoz
Funcionario del Cuerpo Superior de Sistemas y Tecnologías de la Información de la Administración del Estado

Utilización de Software de Libre Distribución en la Administración Española

Resumen: *La utilización de software de libre distribución en la Administración es un tema novedoso, desconocido y que causa recelos. Por sus ventajas operativas en la práctica, empiezan a aparecer algunas experiencias ciertamente positivas que plantean la conveniencia de que los organismos públicos puedan beneficiarse del mismo dentro de los cauces legales.*

0. Introducción

La utilización de programas en la Administración debe cumplir unas garantías legales para la Administración y los particulares. Por otro lado su implantación es una tarea técnica a desarrollar por personal concreto, propio o externo, que se prolonga más allá de la configuración inicial con un mantenimiento que garantice la operatividad.

El conocimiento de las ventajas que el uso de software de libre distribución puede ofrecer (económicas, funcionales y de dinamismo en su actualización sin generar dependencias con empresas) no está muy extendido entre los directivos de la Administración. No hay mucha experiencia en su uso ni en el procedimiento para su obtención, o los distintos modelos (dominio público, gratuito, licencias GNU y *shareware*).

Actualmente, existen iniciativas en cuanto a la utilización de este tipo de programas en organismos oficiales, y no sólo para su utilización por y para técnicos (compresores, utilidades gráficas, editores HTML...), sino a nivel corporativo, confiándoles funciones que pueden llegar a paralizar la unidad, como es el software de sistemas operativos y comunicaciones, aspecto en el que se centra este artículo.

1. Aspectos legales y de procedimiento

1.1. Competencia

La **ley 13/1995 de 18 de mayo**, de Contratos de las Administraciones Públicas, especifica que los contratos de suministros y servicios deben cumplir ciertas prescripciones que además de adecuar la legislación a la pertenencia de España a la CE "aseguren, de manera unitaria y en condiciones de igualdad, los intereses generales de todos los españoles".

Pero difícilmente puede competir una empresa nacional que desarrolla programas con otra que distribuye copias de software gratuito al módico precio de lo que cuesta el soporte y el tiempo de reproducción. Ciudadanos cuyo negocio es la producción de programas verían cómo la Administración, uno de los principales contratistas, desvía las inversiones hacia empresas que no tienen costes de desarrollo y muy bajos de mantenimiento (los detalles de configuración, optimización y depuración en muchas ocasiones los recogen gratuitamente en Internet). Aunque por la naturaleza de este tipo de programas no son una amenaza para desarrolladores de programas a medida, sí lo podrían llegar a ser para los distribuidores de las multinacionales del sector.

1.2. Garantía

El **Real Decreto 263/1996 de 16 de febrero**, por el que se regula la utilización de técnicas electrónicas, informáticas y telemáticas por la Administración General del Estado, desarrolla aspectos de la **Ley 30/1992 de 26 de noviembre**, de Régimen Jurídico de las Administraciones Públicas y del Procedimiento Administrativo Común, en especial el reconocimiento formal de su validez. Distingue cuatro extremos, uno de los cuales es "Programas y aplicaciones utilizados para el ejercicio de potestades": "Los programas y aplicaciones ... cuyo resultado sea utilizado para el ejercicio por los órganos y entidades del ámbito de la Administración General del Estado ... deberán ser objeto de aprobación y difusión pública...". Salvo si se usan con carácter meramente instrumental, es decir, que no determinen directamente el contenido de las decisiones administrativas. En el caso de que proceda la difusión, deberá atenderse "a la protección de los derechos de propiedad industrial e intelectual".

Los programas de libre distribución en general son instrumentos (almacenar, trasladar o cambiar el formato de la información) que no toman decisiones (esto exige aplicaciones a medida). Si al fallar llegan a paralizar el funcionamiento de una unidad, la responsabilidad se limita al gestor, por lo que aunque no se exija que sean "aprobados mediante resolución del órgano administrativo que tenga atribuida la competencia para resolver el procedimiento", su utilización en la práctica dependerá de la confianza que el gestor tenga en estos programas y su mantenimiento, con independencia de sus bondades técnicas.

1.3. Obtención

En el caso del software gratuito, su utilización no implica efecto económico alguno que exija reserva de crédito ni expediente de gasto a justificar ante la Intervención. La Administración no es propietaria de tales programas, y no es necesario expediente de donación (entre otras cosas porque no hay ningún particular que lo inicie por su voluntad). Simplemente la organización se beneficia de su existencia. Pero en caso de existir condiciones previas a su uso, habría que analizar su aplicabilidad para la Administración española dentro del respeto a la propiedad intelectual y el reconocimiento del autor.

La regularización de software cuya utilización sí esté condicionada al pago de una cantidad, cualquiera que sea, se ajusta con dificultad a los procedimientos habituales. El pago de cantidades (y siempre que sean de menor cuantía) a un particular de otro país es muy complejo, y la validez de la factura emitida dudosa. La ausencia de garantía al no controlar la Administración el producto, ni tener ninguna capacidad de actuación frente al suministrador y carecer de soporte explícito (aunque en la práctica sea mejor que muchos del canal comercial), dificulta aún más las cosas. Ajustarse a la ley de contratos sería más fácil si se pudiese romper el modelo de distribución directa, basado en la confianza de que el usuario satisfecho pagará a *posteriori* tras el uso de un producto en depósito, introduciendo una empresa intermediaria con capacidad de contratación en el extranjero y con la Administración. Para ello debería poder revender el producto y en este caso habría que analizar cómo se promueve la competencia y la concurrencia de ofertas.

2. Aspectos operativos: la entrada de software libre en la Administración

El primer paso es la detección de una necesitad y la obtención, instalación y configuración el producto que la resuelve. No hay una labor comercial, así que hay que contar con un mecanismo para localizar esos programas concretos. No todos los organismos tienen acceso a Internet o BBSs. Tampoco suele haber suscripciones a revistas que regalen CD-ROMs.

La principal fuente de este software es "lo que el funcionario interesado quiera traer de su casa". La voluntad de las personas concretas con cualificación técnica y con la iniciativa suficiente es el factor más importante a la hora de plantear esta alternativa. No debe verse como un aspecto negativo de dependencia de la organización frente a sus empleados. No depender de empresas externas que quieran realizar las instalaciones en el mínimo tiempo posible y desligarse después del problema, sino de personal propio, es un modelo viable cuando la necesidad no es temporal y se necesita gran confianza. Por fortuna la incorporación a los nuevos cuerpos de informática de la Administración de personal generalmente joven y con buena formación de base, que con frecuencia compaginan trabajo y estudio, puede ser aprovechada para resolver problemas concretos de la Organización salvando las restricciones presupuestarias.

Ya no es extraña la utilización de programas que el funcionario instala en el ordenador con el que trabaja. Pero no es este aspecto el referido, ya que con frecuencia no va acompañado de la licencia correspondiente sino de algún virus. Una cosa es que existan copias de uso restringido y otra que una unidad de un departamento utilice sistemáticamente y de forma institucional un determinado producto para sus fines.

2.1. Mantenimiento

El uso de un programa no se limita a la configuración inicial. Por si surgen problemas, es necesario contar con un soporte técnico que se encargue del mantenimiento para garantizar la operatividad. En general el modelo ha sido la asistencia técnica externa: disponer de una determinada cantidad en el capítulo adecuado de la aplicación presupuestaria que permita contratar a una empresa (tras fomentar la concurrencia, seleccionar la oferta más ventajosa y culminar el procedimiento administrativo) para que instale y/o mantenga una determinada aplicación.

La dependencia generada por la informática es enorme. Los gestores buscan la seguridad de contar con una empresa clasificada y con prestigio que responda ante los imprevistos. Sería aceptable contratar empresas que mantengan software del que legalmente puedan disponer, con independencia de su origen. Si el software no genera mucha confianza, al menos, que la empresa disponga de los medios y personal cualificado para resolver los problemas, y no se limite a probar parches al azar que circulen por Internet.

2.2. Casos concretos

La renovación de equipos se hace imprescindible a medida que surgen nuevas aplicaciones ofimáticas más complejas, amigables y que permiten en general obtener documentos de mejor calidad (quizá esta sea la utilidad más visible de estas inversiones, y no la agilización de las gestiones, que se produce más por aplicativos a medida y aspectos organizativos). Esto comienza a generar un *stock* de ordenadores personales obsoletos, inoperativos para la mayoría de los trabajos ofimáticos y en red, y que quedan almacenados a la espera de la incoación de su exoneración.

2.3. Por la gama baja

En esta situación, el aprovechamiento de estos PCs puede "ahorrar" ciertos gastos. Poner una impresora en red puede ser caro (tarjeta especial, boca del concentrador ethernet, cableado, roseta...). Conectarla a un puesto de trabajo y que el resto de ordenadores impriman por ella es muy barato, pero incómodo para el usuario de ese ordenador. Conectarla al puerto paralelo de un PC de gama baja dedicado, al que se le acopla una tarjeta Ethernet de bajo coste, es una solución intermedia para obtener un (lento) *unspooler*.

Un 286 con DOS y dos tarjetas de red baratas constituyen, con el software adecuado, un económico *router* que además de aislar tráfico y realizar un filtrado básico, puede llegar a ser un cortafuegos con traducción de direcciones. La obsolescencia también afecta a los 386. Pero a diferencia de los anteriores, éstos dan más juego porque pueden convertirse en máquinas *UNIX*. En este sentido la utilización de *Linux* como encaminador (y cortafuegos) es sólo un primer aspecto que se enriquece con la utilización de líneas Ibercom y modems de alta velocidad para enlazar mediante PPP diferentes redes de área local, constituyendo un método rápido y económico para la implantación de Intranets que abarquen múltiples edificios de una misma ciudad. Y si se utiliza RDSI en lugar de un modem, las posibilidades son aún mayores.

2.4. En competencia directa

Linux (o algún *BSD*) no es sólo un buen *router*. Ofrece

múltiples servicios adicionales que se pueden explotar con un ordenador personal de gama alta que tenga un disco rápido y mucha memoria (y velocidad de proceso similar a la de una estación de trabajo). Puede ser estafeta de correo electrónico, servidor *Web*, servidor de disco de red, gestor de colas de impresión, servidor *proxy*, etc. El coste de un servidor de red basado en un ordenador personal y *UNIX* de dominio público puede tentar a cualquier gestor que lo compare con estaciones de trabajo, máxime cuando justifica poder tener varios servidores que aporten tolerancia a fallos por el mismo coste.

En todo caso, además de servidor de disco, impresión, y *WWW*, *UNIX* es un buen servidor de aplicaciones. En ese sentido existen versiones de varios gestores de bases de datos de empresas comerciales para *Linux*. En especial mencionar *Multibase*, de la empresa española *Transtool*. Es un gestor de base de datos relacional con herramientas de generación de aplicaciones de cuarta generación.

2.5. Y cuando no hay equivalente comercial

Las ventajas de utilizar *Linux* y tener infinidad de programas disponibles directamente en formato ejecutable en ocasiones no son aplicables en servidores *UN*X* comerciales. Muchos de ellos no traen compilador de *C*, o si lo tienen, compilar la aplicación para ellos puede no ser trivial. Utilidades como *celvis*, la versión de *vi* que permite editar archivos ilimitadamente grandes y con anchos de línea de más de 200 caracteres (mediante *scrolling* lateral) no suelen estar disponibles (ni por defecto ni solicitándolo a la empresa). Compresores como *gzip*, o herramientas de análisis y resolución de problemas de la red como *traceroute* no siempre se encuentran compilados para el *UNIX* comercial de turno. Incluso a nivel de protocolos, *ARP* y *RARP* suelen venir por defecto, pero no así el servidor de *bootp* (que sí se utiliza junto con *tftp* para el arranque de terminales *X* y estaciones sin disco), y tampoco *DHCP*(muy cómodo para la asignación dinámica de direcciones IP a los ordenadores personales). Por último mencionar *Apache* como servidor *Web* con características ausentes en sus equivalentes comerciales. O las últimas versiones de *sendmail* (más seguras y más fácilmente configurables que las comerciales).

3. Conclusiones

Las restricciones presupuestarias y los nuevos cuerpos informáticos de la Administración del Estado, que aportan iniciativa y formación, confluyen en el tiempo en un momento en que está de moda Internet y de su mano el software de libre distribución. La promoción de su uso pasa por lograr la confianza de los directivos, haciéndoles conscientes de sus ventajas sin ocultar que su obtención y mantenimiento no se ajustan demasiado bien al modelo habitual de contratación de bienes y servicios.

Hay múltiples organismos que están evaluando la posibilidad de utilizar este tipo de software para su operativa habitual. En unos años esta tendencia se puede generalizar, pero ya es necesaria una concienciación de los directivos sobre esta realidad para que se pronuncien al respecto. Las ventajas técnicas no pueden suplir la necesidad de cierta garantía si se va a generar dependencia.

El fomento de la competencia puede tener efectos que supongan un ahorro para la Administración, aunque hoy por hoy no se percibe el software de libre distribución como competencia real. La disponibilidad de soluciones abiertas, que en muchas ocasiones ofrecen facilidades no existentes en el mundo comercial, es un factor más atractivo y con más posibilidades.

Software Libre

Antonio Ruiz-Falcó
Instituto de Astrofísica de Andalucía, CSIC

Software libre en la investigación: alternativa de calidad

Resumen: *El uso de software libre no es nuevo en el mundo de la astrofísica. El software de aplicación científica en este campo siempre se ha compartido, tanto en uso como en desarrollo, por los centros de investigación de todo el mundo. Sin embargo, debajo de ese software de uso libre había un sistema muy rígido, totalmente dependiente de un fabricante. En este artículo se describe el proceso de migración de un centro de investigación que, manteniendo (y aumentando) la funcionalidad del software de aplicación, en dos años ha pasado a un entorno totalmente abierto con uso masivo de software libre. Asimismo se apuntan algunas conclusiones y las tendencias de futuro.*

1. Introducción

La investigación astrofísica es un campo con grandes necesidades informáticas: se manipulan cantidades ingentes de datos, se requiere gran capacidad de cálculo, gráficos de alta resolución, etc. En astrofísica observacional, la informática se encuentra en todas las fases del proceso de investigación: control del instrumento con el que se observa, la adquisición del dato y el procesado posterior del mismo.

Dada la especifidad del campo de trabajo, no existe software comercial *ad hoc* al tema por dos motivos: por un lado, el bajo número de clientes potenciales (en España la población de astrónomos es de menos de quinientas personas); por otro lado, es un software muy complejo y costoso de desarrollar (programas como Midas, Yraf o Aips han requerido más de 50 años-persona para su realización). Esto motiva que la investigación astrofísica haya tenido que ser autosuficiente en materia de software de aplicación.

Como hacer software de calidad representa un esfuerzo considerable, es obvio que compartir programas resulta positivo para todos. Por tanto el concepto de software libre en nuestro campo es muy viejo, como viejo es el concepto de calidad en el mismo. La astro-física observacional ha experimentado en la última década un avance notabilísimo, ayudada por los avances tecnológicos en instrumentación, detectores, adquisición de datos y tratamiento posterior de los mismos. Detrás de esto ha habido multitud de proyectos con un nexo común: hacer desarrollos de calidad y compartirlos.

Pero los desarrollos realizados se refieren sólo al software de aplicación, funcionando este sobre sistemas comerciales costosos y rígidos. La aparición de gran cantidad de software libre -de sistema y utilidades anexas-, el imparable avance de Internet apoyando estos desarrollos, y la colaboración de programadores en todo el mundo para este propósito han sido claves para en tan sólo dos años dar un gran vuelco en la explotación informática que antes de empezar hubiera sido impensable.

Esta alternativa, en principio sólo considerada barata y casera, ha puesto el listón de calidad (globalmente entendida como el resultado de la explotación) muy alto, mejorando la eficacia, manteniendo un buen nivel de fiabilidad, permitiendo la integración fácil de nuevas soluciones y dejando la puerta abierta para cuantas necesidades puedan surgir.

2. Antecedentes: la situación en 1994

2.1. Software de aplicación científica

Los programas de aplicación científica en astrofísica se pueden dividir en dos grupos:

- ***Grandes paquetes*** de propósito general: programas mastodónticos (se distinguen, entre otras, cosas en los cientos de Mbytes necesarios para su instalación) para abordar toda clase de problemas, si bien cada uno de ellos está enfocado a un campo específico. Suelen ser modulares, con diversos subprogramas y utilidades. Están respaldados por instituciones que mantienen grupos de trabajo para desarrollo y soporte. Los más relevantes son: **IRAF** ***(Image Reduction Analysis Facility)***, desarrollado en el **NOAO** ***(National Optical Astronomy Observatories)***; **AIPS** ***(Astronomical Image Processing System)***, del **NRAO** ***(National Radio Astronomy Observatory)***; **MIDAS** ***(Munich Image Data Analysis System)*** desarrollado por **ESO** ***(European Southern Observatory)***; y el proyecto **STARLINK**, compuesto por multitud de paquetes y desarrollado en el ***Rutherford and Appleton Laboratory***. De tamaño algo menor, pero destacable por haber sido desarrollado en nuestro Instituto es **SIPL**, para tratamiento de tablas de datos e imágenes.
- ***Paquetes de tamaño pequeño y medio***, que no por ser más pequeños son menos importantes. En muchos casos complementan a los anteriores, pudiendo ser utilizados como *external packages*.

Todos estos paquetes, en su origen, estaban disponibles para VMS, aunque han migrado a *Unix* (en la mayoría de los casos se ha abandonado la línea VMS). En ningún caso existe versión para sistemas operativos *pequeños*.

2.2. El sistema

La historia de la investigación astrofísica ha estado ligada al sistema operativo **VMS** durante muchos años. Hasta la década de los noventa la presencia era mayoritaria en los centros europeos y algo menor en los estadounidenses, donde coexistía (una coexistencia a veces no muy pacífica) con *Unix*.

La masiva presencia de VMS tenía efectos positivos y negativos. Entre los positivos se pueden citar los siguientes:

- ***Homogeneidad de entorno***: los astrofísicos no son espe-

cialmente proclives a cambiar de sistema, y disponían de un entorno común allá donde fueren.
- ***Comunicaciones***: en la era Internet es conveniente recordar otros intentos. **SPAN/HEPnet**, y su rama española, **FAENET**, de la que nuestro centro fue nodo desde 1987.

Y entre los negativos:
- ***Hardware***: fiable y fácil de usar, pero lento y caro.
- No resolvía bien el problema de los ***puestos de trabajo gráficos***, y menos aún el tratamiento de imágenes.
- Sistema muy ***centralizado***.

3. La evolución (revolución)

El proceso de cambio, que se venía fraguando desde algún año antes, se desencadena por una conjunción de factores. Entre ellos hay que destacar tres: dos internos y uno externo. El externo es la explosión de **Internet** y los servidores *ftp*, que se convierten en auténticos autoservicios de software. Los internos son una incipiente demanda de *Unix* y, por absurdo que parezca, la necesidad de periféricos hardware, sobre todo discos y puestos de trabajo gráficos.

Internet es determinante en esta nueva forma de trabajar. En primer lugar aparecen multitud de servidores *ftp* con gran cantidad de software. Se pone en evidencia que desarrollar software sólo es interesante si los resultados pueden ser compartidos, por lo que muchos desarrolladores ponen sus programas a disposición de quien quiera usarlos, para lo que aparecen gran cantidad de servidores *ftp*; y para facilitar más las cosas se dispone de servidores *archie* para buscar lo que se necesite. Por tanto, se dispone del medio en el que buscar programas, poderlos ejecutar y poder analizar y/o modificar el código fuente. En segundo lugar, el correo electrónico nos acerca al desarrollador. Normalmente se puede obtener la información que necesitamos de la fuente más autorizada: el autor.

En ocasiones se dispone de incluso más: una lista de distribución cuyos corresponsales son expertos en el tema a tratar. El resultado es que se dispone de un apoyo real para usar y resolver problemas relacionados con el software libre cuyos resultados compiten y ganan con el soporte contratado y pagado a casas comerciales.

La necesidad de periféricos hardware citada anteriormente también es determinante: los requerimientos de almacenamiento crecen más rápido que el tamaño de los discos y que el descenso de los precios, por lo que aunque el costo por Mbyte es menor, la disponibilidad económica no permite disponer de la capacidad de almacenamiento necesaria.

Para aprovechar el dinero de la mejor forma posible, bastaba darse cuenta que un GByte en discos SCSI era sensiblemente más barato que un Gbyte para el *host... workstation* incluida. De esta forma se aumentaba la capacidad de disco, a la par que el parque de *workstations* empezaba a crecer.

Disponer de un número creciente de *workstations* conlleva la necesidad de muchos programas asociados, tanto en comunicaciones y gráficos como utilidades y herramientas de todo tipo. De esta forma, se empiezan a "colar" algunas utilidades de uso libre.

Simultáneamente, y debido a que algo había empezado a moverse en la astronomía europea -los programas estadounidenses **Iraf** y **Aips** comienzan a ganar la partida a los europeos **Midas** y **Starlink**-, existe en el centro una incipiente demanda de *Unix*. Debido a ello en 1992 se instaló la primera *workstation Unix* en el Centro.

El introducir estas *workstations* cambia radicalmente la forma de trabajar. Se convierten en el puesto de trabajo idóneo para el astrónomo, que buena parte de su trabajo se basa en el tratamiento de imágenes. Y una vez levantada la veda, la demanda de puestos de trabajo *X-windows* es explosiva. Como debido a su precio no se puede atender esta demanda a base de *workstations*, se buscan soluciones alternativas, y la primera son terminales X. No es una buena solución: aunque algo más baratos que las *workstations*, son muy caros. Necesitan bastantes recursos del *host* para el arranque, generan tráfico de red y no son autosuficientes para ejecutar aplicaciones. Excesivos inconvenientes para un ahorro pequeño.

Se intentan otro tipo de soluciones, también fallidas. Dado que el requisito imprescindible del puesto de trabajo es *X-windows*, se prueban servidores X para Microsoft Windows. Tampoco es una buena solución, pues necesita un PC de alta gama, y la funcionalidad final tampoco es comparable a la de una *workstation*.

Entre los terminales X y los servidores X para PC se agotan dos soluciones comerciales, que son fallidas.

Reflexionando un poco, una alternativa a una *workstation* que tenga la misma funcionalidad que una *workstation* es ...una *workstation*. Esta trivialidad es cierta si la "*workstation* alternativa" es *muy* barata, pero ¿cómo se consigue una *workstation* barata? La respuesta es también trivial: construyéndola -uno mismo, si es necesario- con hardware muy barato y software libre. En junio de 1994 realizamos la primera prueba: sobre un clónico 486 DX50 se instaló *Linux* 0.99.12 ¿Será posible hacer funcionar este ordenador, construido prácticamente con placas de desecho? ¿Habrá problemas con los *drivers*? ¿Hará falta mucho trabajo? Todas estas preguntas tuvieron respuesta en unas horas (gracias, entre otras cosas, al soporte prestado por el propio Linus Torvalds): fue perfectamente posible, era factible encontrar *drivers* para toda clase de dispositivos y al día siguiente, la *workstation* estaba funcionando, en *Unix*, con *Xwindows* y *Xview*. Como punto de partida no está nada mal.
Al menos podía usarse como un Xterminal. En una primera fase sirvió para que pudiera usarse como puesto de trabajo, ejecutando las aplicaciones en otro *host*. Una vez comprobado que el sistema funciona, había que dar un paso más: utilizar el software de aplicación científica en modo local. No tardaron en hacerse versiones de los paquetes más usados para *Linux*.

4. El estado actual

En tan sólo dos años el cambio ha sido radical. Se ha adoptado como "puesto de trabajo tipo" un PC con *Linux*. Estos sistemas se definen de la siguiente manera:
- **Hardware:** el puesto de trabajo tipo es un PC clónico, procesador de 64 bits, mínimo 32 Mbytes RAM y 1 Gbyte

de disco, Monitores de alta resolución de calidad, Tarjetas de video aceleradas, etc.

- **Software de base:** sistema operativo *Linux*. Entorno gráfico: *Xwindows*, con *Xview*, *Motif* o el *window manager* deseado. Entorno de red: TCP/IP, usando NIS y NFS para homogeneidad: cualquier usuario puede acceder a cualquier nodo y cualquier disco. Para una mejor gestión de los múltiples *filesystems* accesibles se usa el *automounter* amd.
- **Internet:** todas las herramientas al uso: la gestión de mensajería la realiza sendmail 8.8.5, en la configuración recomendada por el grupo de trabajo de correo electrónico de **RedIris**; agente de correo *pine*, servidor *web* del CERN, servidor ftp de la Washington University, diversos *browsers* Web, etc. Desde el punto de vista de seguridad también se utiliza *Satan*, cops, etc.
- **Software de aplicación científica:** versiones modernas de los paquetes de siempre, migrados a *Unix* (con especial velocidad a *Linux*). A estos paquetes se les han ido incorporando por primera vez paquetes comerciales: **Mathematica** e **IDL**.

Estos puestos de trabajo se complementan con los *hosts* centrales. Se mantiene el *host* VMS (cuyo peso específico disminuye día a día) y existe un servidor central *Unix*, en el que residen una copia de cada aplicación y plataforma, compartidas por todos los ordenadores. Los usuarios ejecutan las aplicaciones en su estación *Linux*, siempre que es posible, lanzando en el servidor los procesos de cálculo más pesado. En cualquier caso, disponen de las mismos programas tanto en el servidor *Unix* como en su *workstation* local.

Gracias al software libre, se pueden ejecutar los mismos programas de aplicación sobre diferentes plataformas, lo que permite una total libertad en la elección de la misma. Esto redunda en un abaratamiento importante de costos, sobre todo si la plataforma elegida también es libre. Pero en cualquier caso se mantiene la libertad de elección, lo que facilita el modelo escogido: *hosts* potentes compartidos para las tareas de cálculo más pesado y estaciones de trabajo de usuario que ejecutan exactamente el mismo software y los mismos ficheros de datos.

5. Las claves

En las siguientes tablas puede apreciarse cómo ha evolucionado la situación en tan sólo dos años. En la **tabla 1** puede verse la evolución del peso específico de los sistemas operativos usados: la importancia de los sistemas operativos comerciales en el aspecto científico ha decrecido notablemente. Se mantiene la de **DEC Unix**, debido al servidor de cálculo y de aplicaciones Unix, pero posiblemente sea pasado a Linux próximamente. En la **tabla 2** puede verse la evolución de las utilidades de comunicaciones: las tres primeras opciones son software comercial y el resto de uso libre (o se usan versiones de uso libre). Como puede verse, la responsabilidad crítica de las comunicaciones, internas y externas, recae hoy en día en aplicaciones de libre distribución. En la **tabla 3** se muestra la evolución de utilidades. En cuanto al software de aplicación científica **(tabla 4)**, se sigue usando masivamente el software libre, con alguna incorporación de software comercial. Pero el software incapaz de migrar a Unix ha desaparecido.

	1994	1996
VMS	+++	-
Unix (Ultrix, DEC Unix, SunOs)[1]	+	+++
Unix (Linux)	-	+++
Netware[2]	+	-
Windows (NT, 95)[3]	-	+
MacOs	+	+

Tabla 1

	1994	1996
DECnet	+++	-
PSI[4]	+++	-
Mensajería EAN	+++	+
sendmail	-	+++
pine[5]	-	+++
NIS	-	+++
NFS	-	+++
wu ftp	-	+++
Browsers WWW[6]	+	+++
Servidor WWW	+	+++
Satan	-	+++
amd	-	+++

1 La importancia se debe a DEC Unix, correspondiente al servidor de cálculo Unix
2 Ha dejado de operar en 1996. Se usaba para tareas de gestión
3 Tareas de gestión y ofimáticas
4 X.25 fue sustituido por una línea punto a punto
5 Se usan además otros agentes de correo
6 Todos de libre disposición

Tabla 2

	1994	1996
Compiladores comerciales	+++	++
Compiladores libres (GNU)	-	+++
Sistema CASE comercial	+++	+
Herramientas de desarrollo libres	-	+++
Procesador de textos TeX	+++	+++

Tabla 3

	1994	1996-VMS	1996-Unix-Linux
IRAF	+	-	+++
MIDAS	++	+	-
AIPS	+++	+	+++
FIGARO	+++	+	++
SIPL	+++	+++	+++
CLASS	++	+	++
IDL	+	++	-[7]
Mathematica	-	-	+

7 Próximamente se cambiará la licencia a Linux

Tabla 4

Para no hacer excesivamente prolija esta lista, se han incluido solamente los paquetes más representativos, a los que se les ha acompañado los dos paquetes comerciales en uso. Puede apreciarse una victoria aplastante del software de aplicación libre, pero además, de aquel que permita que el sistema operativo y las utilidades que haya debajo también lo sean. Esto introduce el concepto de hardware libre: puedo comprar la máquina que desee en función de mis necesidades y del mercado, pues el software está solucionado.

6. Conclusiones y tendencias de futuro

En el debate software libre vs. software comercial se suele apelar a diversos tópicos: no está probado, no hay una casa comercial que se responsabilice, etc. La experiencia habida en nuestro centro ha desmontado una por una estas ideas. Como resumen, extraemos las siguientes conclusiones:

- El uso de software libre representa una alternativa muy válida, no sólo con criterios económicos, sino con criterios de calidad. El software pasa a ser responsabilidad de los desarrolladores y del conjunto de sus usuarios, con lo que la asistencia técnica y la rapidez de respuesta ante los *bugs* es muy alta.

- La fiabilidad es igual o mayor que en el software comercial. Ningún software está exento de *bugs*, pero el tiempo de solución tiende a ser menor que en el software comercial. Como ejemplo, baste citar el conocido *bug* del "ping asesino". Este *bug* afectó, entre otros sistemas operativos, a *Linux*. Pero su solución estaba disponible en la red ¡2 horas y 34 minutos después de ser reportado! Ningún software comercial dispone de una respuesta más eficiente y rápida.
- El desarrollo -en el que participo- del software de vuelo de diversos módulos de la misión *Rosetta* de la **Agencia Espacial Europea** se está llevando a cabo con el compilador gcc. Es decir, herramientas de libre distribución para el desarrollo de aplicaciones extremadamente críticas.

- La asistencia técnica suele ser mejor que en el software comercial. Normalmente es posible contactar con el autor/res , que dan ayuda precisa a los problemas presentados.

- Se puede llegar a la situación ideal de automantenimiento. Con los conocimientos necesarios es posible mantener y actualizar el software, adaptarlo al nuevo hardware, etc.

- El software libre permite una casi total libertad en la elección de plataforma y hardware, pudiendo llegar también al automantenimiento.

- El software libre permite un muy fácil manejo, así como automatizar la mayoría de tareas de gestión de sistemas. Por ejemplo, el proceso de instalación (desde 0) de una nueva *workstation Linux*, instalando el sistema operativo con todas las utilidades, aplicaciones científicas e integración en el entorno de red para su uso por cualquier usuario, representa menos de una hora.

- El software libre no entra en la administración y la ofimática. No por calidad -si el software libre es capaz de resolver problemas asociados a proyectos costosísimos en investigación astronómica, ha de ser capaz de resolver un problema de gestión-, sino más bien por la propia idiosincrasia administrativa.

- El entorno ofimático y administrativo está dominado por los sistemas operativos de Microsoft; por el contrario, *Windows* (NT, 95) no consigue entrar en la investigación astrofísica.

- El tratamiento de textos, usado en el entorno científico, es diferente del correspondiente al entorno administrativo. Este hecho es muy relevante, ya que el usado en el entorno científico es libre, disponible para multitud de plataformas y de calidad contrastada (muchas publicaciones se realizan en base a él).

- El software de uso libre normalmente requiere un mayor nivel de conocimiento por parte de usuarios y administradores de sistemas que el software comercial, pero sus posibilidades de uso son mayores.

- El crear un sistema de aplicaciones y periféricos distribuido obliga a disponer de una red local rápida y fiable. Aparte de ser necesario un cableado de calidad y fiabilidad contrastada, es necesario pensar en nuevas tecnologías para eliminar cuellos de botella (*switching, fast ethernet*, etc).

De cara al futuro se apuntan las siguientes tendencias:

- Desaparición del entorno VMS o, en el mejor de los casos, una presencia meramente testimonial.
- Probablemente, *Linux* acaparará los sistemas de uso científico, con dos posibles excepciones: una posible máquina de cálculo intensivo y los ordenadores de los correligionarios de *MacOs* (en cuyo caso quizá llegue al nivel de coexistencia pero no a reemplazarlo). Para el resto de ordenadores, *Linux* será el sistema operativo independientemente de la plataforma hardware. Por fin se logra un estándar bajo *Unix*: se puede elegir procesador, manteniendo total compatibilidad en el sistema operativo.
- Aunque el uso de software libre de aplicación en el entorno de investigación seguirá siendo masivo, entrarán algunos paquetes comerciales para propósitos concretos. En este sentido, se atisba una tendencia inversa ya que antiguamente se usaba soft libre de aplicación y soft de base comercial. El último tiende a liberalizarse y el primero tendrá que coexistir con algunos paquetes comerciales (de alta calidad).
- Uso de *middleware* y estricta observación de criterios de compatibilidad y migrabilidad de aplicaciones.

7. Referencias

Las mejores referencias son prácticas:

AIPS: *http://aips.nrao.edu*
IRAF: *http://iraf.noao.edu*
MIDAS: *http://www.hq.eso.org/midas-info/midas.html*
STARLINK: *http://star-www.rl.ac.uk*
Linux y utilidades GNU: *http://sunsite.rediris.es*
Satan, amd, wu-ftpd, etc: *ftp://ftp.rediris.es*
PGPLOT: *http://astro.caltech.edu/~tjp/pgplot*
wxWindows: *http://web.ukonline.co.uk/julian.smart/wxwin*

Software Libre

Jesús Carretero, Santiago Rodríguez
Facultad de Informática, Universidad Politécnica de Madrid, España

fjesus@fi.upm.es
srodrig@fi.upm.es

COES: herramienta lingüística de libre distribución para la Lengua Española

Resumen: *La carencia de herramientas software de libre distribución que permitan la corrección de textos escritos en castellano llevó a los autores a la construcción de un diccionario de castellano basado en el programa de libre distribución* ***ispell****. Este artículo presenta la integración del diccionario al entorno de la herramienta* ***ispell****, haciendo especial hincapié en los aspectos de la especificación formal de las reglas de derivación, etiquetado de las palabras que componen el diccionario raíz y la generación final del diccionario.Por último se muestran algunas medidas de evaluación realizadas por los autores. El diccionario se distribuye como una herramienta de libre distribución desde 1994 bajo los términos de la General Public License de Free Software Foundation.*

Palabras Clave: *Lenguaje Natural, Especificación Formal, Software de Libre distribución.*

1. Introducción

La introducción de los computadores en el procesamiento de textos ha demostrado la carencia de herramientas especializadas (correctores ortográficos, correctores gramaticales, etc.) para el castellano. Esta es la tercera lengua más extendida y, sin embargo, la disponibilidad de este tipo de herramientas está muy lejos de otras lenguas mucho menos extendidas, pero con mayor influencia tecnológica que los países de habla hispana.

La importancia potencial del castellano en un futuro próximo llevó a los autores a desarrollar algunas herramientas gramaticales software. Además, para promocionar la utilización de estos programas, se pensó en su distribución totalmente gratuita. Una de estas herramientas es el diccionario de castellano para el corrector ortográfico *ispell* desarrollado por Geoff Kuenning que permite la incorporación de distintos diccionarios (Ispell se puede obtener mediante *anonymous* ftp de *ftp.math.orst.edu en /pub/ispell-3.1/ispell-3.1.20.tar.gz*).

Uno de los principales objetivos impuestos en el desarrollo del diccionario de castellano fue su distribución totalmente gratuita para permitir su utilización al mayor número de usuarios posible. Por otra parte el diccionario debe ser exhaustivo, es decir, debe contener el mayor número posible de palabras aceptadas en castellano, así como la mayor parte de sus derivaciones. Puesto que es una herramienta de libre distribución debe ser fácil de mantener ya que se espera de la colaboración de los usuarios finales para actualizar y mejorar tanto el diccionario raíz como el conjunto de reglas de derivación. Por último, debido a la diversidad de vocabulario del castellano, dependiendo de la zona geográfica del usuario que utilice la herramienta, se utiliza un conjunto de palabras ligeramente distinto del resto. Por tanto el proceso de generación del diccionario debe permitir al usuario seleccionar el conjunto de palabras que mejor se adapte al que se utiliza en su zona geográfica.

El principal problema encontrado en el desarrollo del diccionario fue la adaptación de las reglas gramaticales castellanas a una especificación formal. A diferencia del inglés, el castellano contiene un número muy elevado y complejo de reglas de derivación a partir de una palabra raíz. Las principales tareas que se realizaron fue la formalización de las reglas de derivación gramaticales y la generación de un conjunto de palabras etiquetadas (palabras con su conjunto de reglas a aplicar). El desarrollo de esta herramienta se inició a comienzos de 1994. El primer prototipo estuvo finalizado a mediados de 1994 y se dedicó a su uso interno para detectar errores. Se distribuye de forma gratuita desde finales de 1994 (se puede obtener mediante *anonymous* ftp en *ftp.fi.upm.es en pub/unix/espa~ol.tar.gz* o mediante el URL *http://www.datsi.fi.upm.es/~coes/*).

2. Características Morfológicas del Castellano

El castellano es una lengua que derivó del latín y tiene una gramática compleja. Para llevar a cabo la construcción de cualquier plataforma léxica la primera tarea que se debe llevar a cabo es el estudio de la gramática castellana [2]. Este estudio debe permitir formalizar el conjunto mínimo de reglas necesario que permita extraer el conjunto de palabras reconocidas por la lengua castellana a partir de un conjunto de palabras raíces mínimo. Para alcanzar dicho objetivo se construyó un árbol de derivación a partir de una palabra raíz. Una versión simplificada se muestra en la **figura 1**.

Los principales problemas que se encontraron en la construcción de este conjunto de reglas de derivación vinieron originados por las características del castellano. Los más relevantes se exponen a continuación:

Derivaciones de género y número. Los adjetivos (ADJ) y sustantivos (NOM) tienen género (masculino o femenino) y número (singular y plural). La situación habitual es que un adjetivo o nombre tenga derivación tanto en género como en número. Por ejemplo: *perro® perra* y (*perros, perras*). Otros casos únicamente tienen un género y, por tanto, sólo admiten derivación en número. Es el caso de un sustantivo masculino como *álamo, álamos*, o femenino como *casa, casas*.

Conjugación verbal. Cada una de las tres conjugaciones verbales del castellano tiene 40 derivaciones temporales (P.

ACTIVO, P. PASADO, GER, INF y FORMAS VERBALES). Como es sabido, los verbos regulares tienen un conjunto estricto de reglas de derivación que son idénticas para todos los de una misma conjugación. Los verbos irregulares tienen al menos una derivación diferente que las derivaciones regulares correspondientes a su conjugación. Las derivaciones irregulares se agrupan en alrededor de 100 tipos diferentes de irregularidades [2].

Formas enclíticas. Algunas derivaciones verbales se generan añadiendo una forma pronominal al final de una forma verbal (ENCLIT). En el castellano escrito se pueden encontrar dos formas enclíticas diferentes: los verbos pronominales, cuyas formas enclíticas se generan añadiendo los sufijos *-me*, *-te*, *-se*, *-nos* y *-os* en el infinitivo y en el gerundio (*amar ® amarte*), y los verbos transitivos, cuyas formas pronominales se generan añadiendo las terminaciones *-lo*, *-la*, *-los*, *-las*, *-le* y *-les* (*amar ®amarla*). Ambas formas enclíticas se pueden combinar para formar enclíticos más complejos (*ajustar ® ajustármelo*). Esto genera un conjunto de reglas de complejidad $O(n^2)$. Además, estas formas enclíticas se ven afectadas por las irregularidades que presentan algunos verbos en su gerundio (*vestir ® vistiéndote*), lo que incrementa el grado de complejidad para las formas enclíticas.

Nombres derivados de verbos. Algunos sustantivos son formas derivadas de un verbo como *imaginar® imaginación* o *abatir ® abatimiento* (VERBO ® NOM).

Adverbios derivados de adjetivos. Gran parte de los adverbios modales se generan añadiendo el sufijo *-mente* a un adjetivo (ADJ ® ADV): *tranquilo ® tranquilamente*.

Superlativos y diminutivos. Las formas regulares de superlativos se forman añadiendo el sufijo *-ísimo* a un adjetivo (*grande ® grandísimo*). Los diminutivos se forman añadiendo los sufijos *-ico*, *-ito* y *-illo* a un adjetivo o nombre.

Vocales acentuadas. Hay muchas particularidades relacionadas con las derivaciones en género y número que se han tenido en cuenta al hacer el estudio del modelo. Algunas palabras pierden una vocal acentuada sustituyéndola por su equivalente no acentuada: *gañán*, *gañanes*.

Teniendo en cuenta las características descritas en los párrafos anteriores se ha desarrollado un conjunto de reglas formales que comprende un extenso subconjunto de las que conforman la gramática castellana. Cada una de las entradas del diccionario, que contiene las palabras raíces, tiene una etiqueta que representa una lista de reglas de derivación que se deben aplicar a dicha palabra para obtener sus formas derivadas.

3. Implementación del modelo

Las características gramaticales estudiadas en el apartado anterior han llevado a la realización del modelo ajustándose a las restricciones que imponía la herramienta *ispell*. Estas restricciones se basan en agrupar un conjunto de reglas (clase) al que se asocia una etiqueta que será referenciada en las etiquetas del diccionario raíz. Puesto que el uso del castellano se basa en gran parte en las formas derivadas (un verbo castellano tiene 55 formas derivadas), las reglas de derivación del diccionario incluyen todas las derivaciones de los verbos regulares. Además, se han incorporado reglas adicionales para tener en cuenta la mayor parte de los patrones por los que se rigen las derivaciones de los verbos irregulares. La inclusión de las formas derivadas de los verbos *ser*, *ir*, *haber* y *estar* se han incluido en su totalidad en el diccionario raíz al no adecuarse fácilmente a ninguno de los patrones considerados. En resumen, el conjunto de reglas implantado para especificar la gramática castellana contiene alrededor de 3.500 reglas agrupadas en 57 macroreglas o clases.

Cada macrorregla que se describe a continuación refleja un aspecto particular de la gramática castellana que se ha descrito en la sección anterior. Cada una de las reglas de derivación que componen una clase o macrorregla trata un caso particular [9 , 10 , 1]. La condición que se muestra en la primera columna de cada uno de los ejemplos representa la aceptación de una palabra para ejecutar la acción que se muestra en la segunda columna. Si una palabra termina con

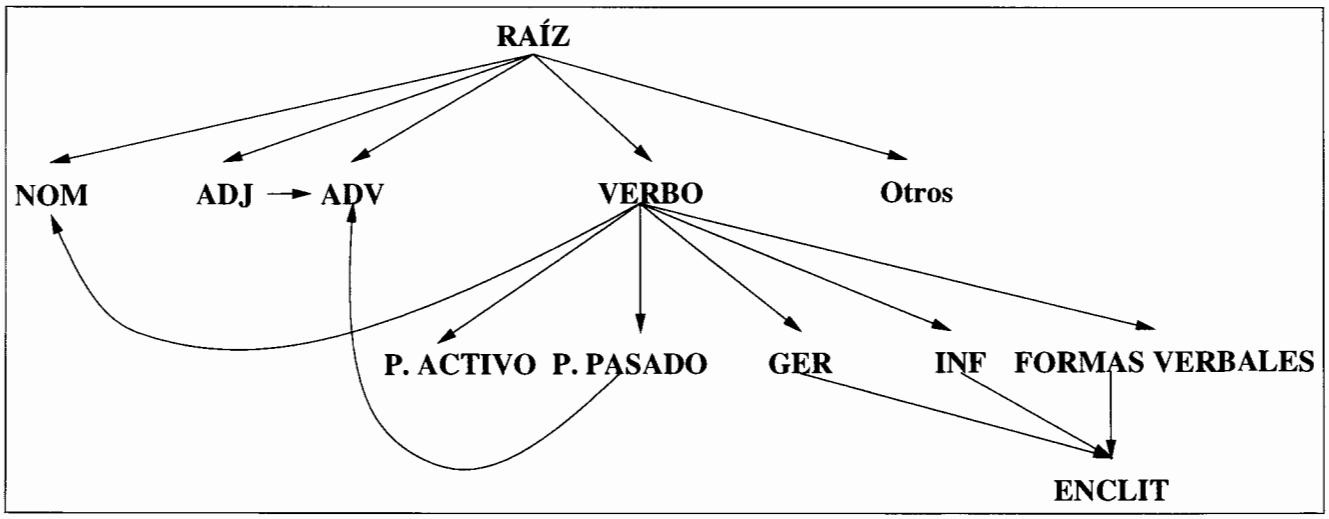


Figura 1: Estructura Morfológica simplificada del castellano

el sufijo especificado, se realiza la acción subsiguiente. Esta acción se basa en sustituir un morfema de la palabra raíz por otro, o simplemente añadir un morfema adicional.

Derivaciones de género y número. Se han incluido dos macrorreglas que realizan estas derivaciones. Las derivaciones en número incluyen 14 reglas. La regla a aplicar depende de la terminación de la palabra raíz sobre la que se aplica la regla. La macrorregla de derivación en género y número se compone de 18 reglas. A continuación se muestran algunos ejemplos de derivación de estas clases:

Derivaciones en número			**Derivaciones en género y número**		
Condición	*Acción*	*Ejemplo*	*Condición*	*Acción*	*Ejemplo*
[AEIOU]	S	vaca vacas	O	-O, A	amigo amiga
Z	-Z, CES	arroz arroces	O	S	amigo amigos
UN	-UN,UNES	atún atunes	[AONS]	ES	pastor pastores

La última regla del ejemplo anterior muestra la generación del plural masculino y femenino para aquellas palabras que no acaban en *a*, *o*, *n* ni *s*.

Conjugación Verbal. Las reglas de derivación que permiten generar las formas verbales se agrupan en cuatro clases: dos de ellas se aplican a verbos regulares y las otras dos a verbos irregulares. Alrededor de 200 reglas componen las dos clases que derivan los verbos regulares, mientras que el conjunto de derivaciones de los verbos irregulares se compone de 2.700 reglas. En este último aspecto es donde se ha dedicado la mayor parte del esfuerzo. Sin embargo su formalización ha sido factible puesto que las formas irregulares del castellano siguen patrones de derivación bien definidos: *-ontar ® -uento*, *-oder ® -uedo*, *-ervir ® -irvo*, etc. Algunas reglas de derivación para verbos regulares e irregulares se muestran a continuación:

Verbos Regulares			**Verbos Irregulares**		
Condición	*Acción*	*Ejemplo*	*Condición*	*Acción*	*Ejemplo*
AR	-AR, O	amar amo	IAR	-IAR, ÍO	enviar envió
CER	-CER, ZO	vencer venzo	OÑAR	-OÑAR, UEÑO	soñar sueño
CIR	-CIR, ZO	zurcir zurzo	SABER	-ABER, É	saber sé

Estas reglas no han tenido en cuenta los verbos *ser*, *estar*, *ir* y *haber* puesto que no existen un conjunto de patrones que permitan derivar todas sus formas verbales a partir del infinitivo. Todas sus formas derivadas se han incluido explícitamente en el diccionario de palabras raíces.

Formas enclíticas. Los verbos regulares incluyen alrededor de 200 reglas de derivación para generar las formas enclíticas, mientras que los verbos irregulares incorporan en torno a 400. Estas reglas representan los enclíticos generados por las derivaciones pronominales, transitivas y combinadas de ambas. Estas reglas únicamente se aplican a las formas del gerundio e infinitivo.

Verbos Regulares			**Verbos Irregulares**		
Condición	*Acción*	*Ejemplo*	*Condición*	*Acción*	*Ejemplo*
[AEI]R	ME	amarme	[AEO]ER	-ER, YÉNDOME	cayéndome
[AEI]R	SE	amarse	[AEO]ER	-ER, YÉNDOSE	cayéndose
[AEI] R	TE	amarte	[AEO]ER	-ER, YÉNDOTE	cayéndote

Condición	**Acción**	**Ejemplo**
[AI] R	-R, MIENTO	esparcimiento
E R	-ER, IMIENTO	aborrecimiento
[AI] R	-R, CIÓN	abolición

Nombres derivados de verbos. Se han tenido en cuenta los nombres acabados en *-miento* y *-ción* que se derivan de verbos a partir de dos macrorreglas.

Adverbios derivados de adjetivos. Se ha considerado una macrorregla (clase) que contiene las dos reglas que se muestran a continuación:

Condición	**Acción**	**Ejemplo**
O	-O, AMENTE	tonto tontamente
[ELNRSZ]	MENTE	virtual virtualmente

Superlativos y diminutivos. Actualmente, únicamente los superlativos regulares se han considerado y constituyen una clase.

Vocales acentuadas. No hay macrorreglas específicas para estas derivaciones. Este tipo de particularidades se han tenido en cuenta en las reglas anteriores.

4. Generación del Diccionario

El léxico para esta plataforma ha sido extraído de un *Corpus de Español* compilado por los autores. Este corpus, que contiene más de 20 millones de palabras, incluye textos extraídos de las siguientes fuentes:
- Textos de periódicos españoles (ABC Cultural, El Mundo, El Periódico, etc.)
- Libros seleccionados, como la Biblia.
- Textos técnicos (informes técnicos, artículos, proyectos de fin de carrera, diccionarios técnicos y libros).
- Corpus oral [7].
- Versión concisa del diccionario Español-Inglés Collins [11].

El *léxico básico* resultante contiene más de 80.000 palabras distintas, 51.000 de las cuales han sido ya etiquetadas de acuerdo a las reglas de derivación mostradas en la sección anterior. Las restantes 29.000 están en proceso de etiquetado. El léxico ya etiquetado contiene aproximadamente 9.000 verbos, 41.000 nombres y adjetivos y 1.000 preposiciones, adverbios, artículos, pronombres, etc. El etiquetado se hace de forma semiautomática mediante una herramienta que extrae los morfemas de cada palabra y propone una o varias *etiquetas tentativas*. Sin embargo, las formas derivadas son comprobadas manualmente para verificar la corrección o no del etiquetador. El *léxico de referencia* usado para desarrollar COES es el diccionario de la *Real Academia Española* (RAE), la institución oficial que vela por la pureza del lenguaje español y admite las nuevas palabras del mismo.

La versión de libre distribución de COES incluye un fichero de afijos y varios ficheros de léxico:
- `español.words` contiene una lista de palabras del diccio-

nario oficial de Español [3].

- `español.comp` contiene una lista de palabras que no aparecen en el diccionario oficial de la Lengua Española, pero de uso habitual en los textos técnicos.
- `antiguas.words` contiene una lista de palabras que aparecen en el diccionario oficial de Español, pero etiquetadas como palabras en desuso.
- `español.nofl` contiene una lista de palabras que no aparecen en el diccionario oficial de Español, pero que han sido frecuentemente encontradas en el corpus.
- `español.propios` contiene una lista de nombres propios.

Cuando se aplican las reglas de derivación al léxico básico usado en COES actualmente, se crea un diccionario que contiene más de 530.000 palabras. Este número de palabras se incrementará en más de 700.000 tan pronto como estén etiquetadas las 29.000 palabras pendientes de etiqueta. El diccionario de español se construye usando la herramienta *ispell*, que aplica las reglas de derivación elaboradas por los autores, siguiendo el formato de esta herramienta, al léxico básico. *Ispell* sigue cuatro pasos básicos para generar el diccionario (**figura 2**):

1. Generación de las reglas de derivación a partir del fichero de reglas.
2. Unificación de las entradas del diccionario para evitar redundancias y formas ilegales.
3. Interpretación de las reglas de derivación para calcular las formas derivadas.
4. Construcción de un árbol indexado para conseguir una búsqueda eficiente en el diccionario.

Existe la posibilidad de que los usuarios puedan generar diccionarios *particularizados* mezclando varios ficheros de léxico cuando se construye el diccionario. Esta opción permite a cada usuario incluir sus propios léxicos (que deberían estar etiquetados). Además, los usuarios pueden particularizar el diccionario eligiendo el formato en que se codificarán los caracteres especiales (ü,ñ y letras acentuadas), que no se encuentran definidos en el conjunto básico de caracteres ASCII de siete bits. Para permitir esta particularización, se proporciona a los usuarios la codificación de estos caracteres en los cuatro formatos más habituales cuando se definen las reglas: Formato por defecto, el usado para el desarrollo del diccionario y el usado por el paquete TEX multilingüe (Babel); formato TeX, el usado en LATEX; formato TeX puro (plainTEX), usado por TEX; formato Latin1, en el cual los caracteres acentuados se codifican segun se especifica en el juego de caracteres estándar iso_8859_1.

5. Evaluación

Se han identificado tres fuentes principales de error en COES: 1. Palabras correctas que, sin embargo, no aparecen en el diccionario; 2. Palabras incorrectas que aparecen en el diccionario; 3. Reglas de derivación incorrectamente aplicadas.

Algunos de estos problemas se pueden paliar mediante la colaboración de los usuarios de la herramienta que detecten errores. Por ello, los autores animan a los usuarios a enviar toda esta información cuando obtienen la copia de COES (informes y errores pueden ser enviados a la siguiente dirección de correo electrónico: *espanol-bugs@datsi.fi.upm.es.*).

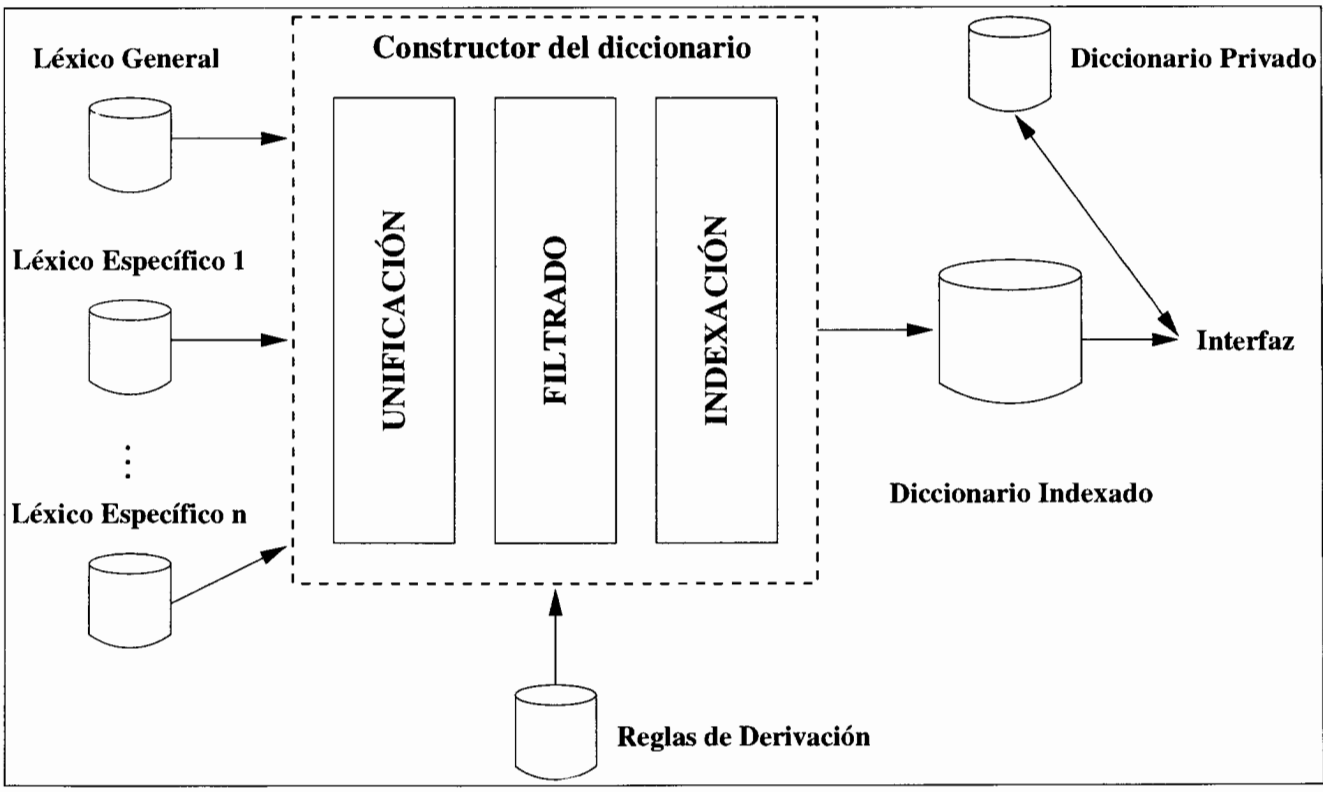


Figura 2: Generación del Diccionario

Corpus	Tamaño del Corpus	Fallos Totales	Tasa Global	Palabras distintas	Fallos distintos	Tasa sobre palabras únicas
Biblia	817.184	3.749	0,4 %	31.616	818	2,5 %
Abc	4.287.676	8.878	0,2 %	160.746	4.467	2,7 %
Textos Técnicos	445.770	399	0,08 %	23.202	64	0,2 %

Tabla 1: Tasas de error (versión 1.5)

Corpus	Tamaño del Corpus	Fallos Totales	Tasa Global	Palabras distintas	Fallos distintos	Tasa sobre palabras únicas
Biblia	817.184	12.399	1,5 %	31.616	1.220	3,9 %
Abc	4.287.676	27.553	0,6 %	160.746	7.573	4,7 %
TextosTécnicos	445.770	467	0,10 %	23.202	130	0,56%

Tabla 2: Tasas de error (versión 1.4)

Para medir la habilidad de COES para corregir textos en español, se han hecho pruebas exhaustivas con distintos conjuntos de textos para tratar de reflejar los diferentes aspectos del lenguaje español: periodismo (ABC cultural), lenguaje técnico (libros y tesis doctorales), lenguaje coloquial (corpus oral) y textos clásicos (La Biblia y El Quijote). Dichos textos han sido filtrados para quitar los acrónimos, nombres propios, extranjerismos, etc. Sobre la parte restante se han ejecutado pruebas para evaluar dos características importantes en COES: cobertura léxica y palabras alternativas propuestas cuando se encuentran errores en el texto. Los resultados que se describen a continuación corresponden a las pruebas ejecutadas sobre una estación de trabajo Sun-20 con la versión 1.5 de COES para tratar de detectar errores del tipo 1 (palabras correctas que no aparecen en el diccionario), midiendo la tasa de error global y la calidad global de COES (**tabla 1**).

En las columnas 2 a 4 de la tabla anterior se muestra la tasa de fallos de COES tomando en cuenta el número total de palabras incluidas en el corpus (y, por tanto, también las repetidas). Las columnas 5 a 7 muestran los mismos parámetros pero calculados teniendo en cuenta solamente las palabras únicas (no repetidas) del corpus. La columna *Palabras distintas* muestra el número de palabras distintas existentes en cada corpus. La columna *Fallos distintos* indica el número de palabras distintas no reconocidas. Es

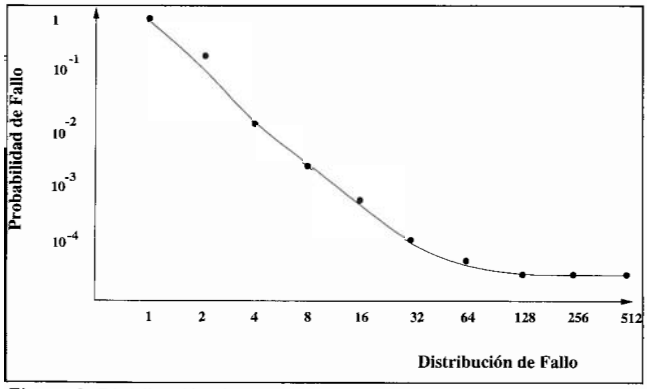


Figura 3: Distribución de errores

importante resaltar que el porcentaje de palabras únicas fallidas es mucho más alto que el de palabras repetidas, lo que indica que las palabras frecuentemente usadas están, en su mayoría, presentes en COES y son reconocidas por él. Otra conclusión que puede extraerse de la tabla anterior es que el comportamiento de COES es mejor para textos técnicos, lo que es razonable habida cuenta de que el banco de textos técnicos usados para depurar la herramienta es bastante mayor que el de otro tipo de textos.

Un estudio comparativo de la primera versión de COES con otros correctores de español ha sido publicado en [4]. Para evaluar el incremento de rendimiento obtenido en la nueva versión de COES (1.5), se han repetido las mismas pruebas que las publicadas, mostrándose los resultados en la tabla 2. Como puede verse, la tasa de error de palabras repetidas, o tasa de error global, ha sido reducida para textos clásicos (La Biblia un 600 %) y periodísticos (ABC cultural un 300 %). La tasa de error de palabras distintas también ha sido reducida, aunque en menor medida debido a que el esfuerzo de mejora del léxico se ha orientado a la inclusión de palabras frecuentemente usadas y a la corrección de las reglas de derivación defectuosas.

No existen criterios estandarizados para comprobar la bondad de un corrector ortográfico. Cada palabra del texto debe ser buscada en el diccionario. Cada palabra no reconocida debe compararse con palabras similares para proponer alternativas. La tasa de *Error Global* se ve fuertemente afectada por las palabras correctas no reconocidas que aparecen frecuentemente en un texto. La calidad del léxico puede ser medida mediante la identificación de las palabras equivocadas que aparecen en él y de la frecuencia de aparición de las mismas. En las evaluaciones de COES llevadas a cabo se ha establecido una distribución de probabilidad para fallos repetitivos (**ver figura 3**). El eje X representa el número de veces que una palabra equivocada aparece en el corpus. El eje Y es la probabilidad de encontrar esta palabra. Como puede observarse en la **figura 3**, la probabilidad de encontrar un fallo en una palabra que aparezca más de 32 veces es menor del 0,1 %. La principal conclusión que puede extraerse de esta figura es que el léxico

de COES incluye las palabras más frecuentemente usadas en español. También puede concluirse que las reglas de filtrado de COES contemplan las derivaciones gramaticales más frecuentes.

La versión actual de COES usa un diccionario que ocupa aproximadamente 2,5 Mbytes. Su rendimiento se ha evaluado por los autores procesando un corpus de tamaño fijo (40.000 palabras) con un número de palabras erróneas variable. El rendimiento de COES se ha comparado con el *ispell* original, que usa un diccionario en inglés que ocupa aproximadamente 750 Kbytes, para evaluar la influencia de la morfología española en las prestaciones finales de COES. La figura 4 muestra el rendimiento de ambas herramientas, en palabras procesadas por segundo, en función del número de palabras erróneas existentes en el corpus. Los mejores resultados se obtienen cuando no hay errores en el corpus: 9.153 palabras/segundo para el inglés y 2.032 palabras/segundo para el español. Esta diferencia es principalmente debida al mayor tamaño del diccionario español. A medida que se incrementa el porcentaje de palabras erróneas, el rendimiento decrece considerablemente para ambas herramientas. Sin embargo, tal como se vio en las evaluaciones anteriores (**ver tabla 1**), la tasa de error debería ser menor del 1 %, por lo que el rendimiento medio de COES en su versión 1.5 debería estar alrededor de las 1.600 palabras/segundo. Este resultado puede ser considerado como muy bueno si se compara con otros correctores citados en la literatura [4].

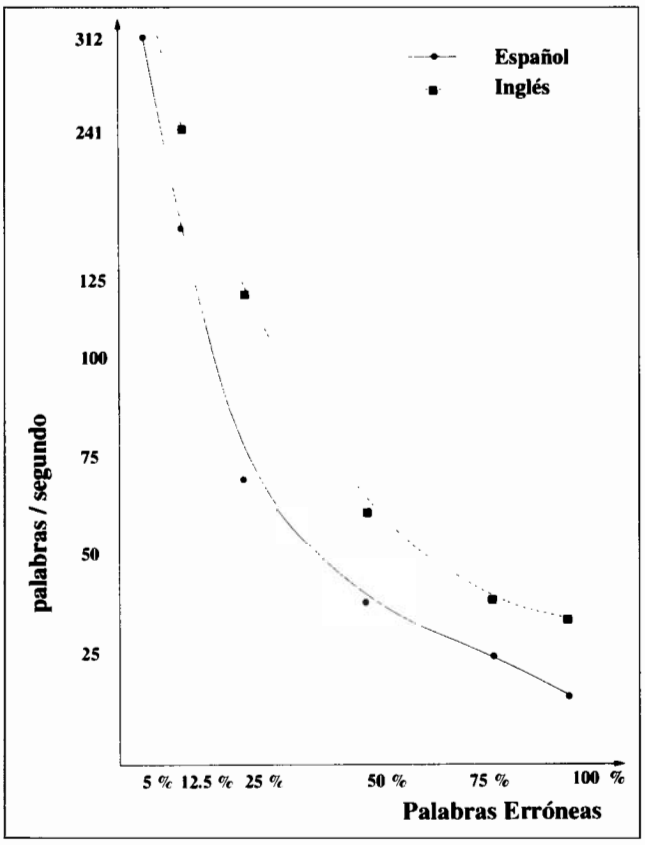


Figura 4: Evaluación de prestaciones

6. Conclusiones y Trabajo Futuro

En este trabajo se ha presentado un diccionario de español desarrollado para la herramienta *ispell*, diccionario que esta siendo usado por una comunidad creciente de usuarios. Las pruebas realizadas nos permiten afirmar que el diccionario trabaja adecuadamente y que es exhaustivo. La tasa de error observada en las evaluaciones es aproximadamente del 0,4 % de las palabras presentes en el *corpus*, siendo del 2,5 % cuando se calcula teniendo en cuenta únicamente las palabras únicas del texto. Esta tasa de error se debe principalmente a prefijos, comparativos y localismos.

La versión actualmente existente de COES puede ser mejorada en los aspectos siguientes: Elaboración de diccionarios locales y temáticos, para dar cabida a palabras usadas en áreas restringidas de la comunidad hispanohablante o en entornos lingüísticos especializados (leyes, medicina, etc.); optimización de reglas; incrementar el léxico básico para reducir la tasa de error de COES y aumentar su eficiencia.

Actualmente están en desarrollo algunas nuevas utilidades para COES. Un tesauro, que usará intensivamente las reglas de derivación, estará disponible pronto. Además se está llevando a cabo un estudio preliminar de las reglas y modelos sintácticos y gramaticales del español [5 , 6, 8, 12] con el propósito de construir un corrector sintáctico en un futuro próximo.

Referencias

[1] J. Carretero, S. Rodríguez. Building lexical tools to manage information written in Spanish. *Journal of Information Science*, 22(5):391-399, Octubre 1996.

[2] Real Academia Española de la Lengua. *Esbozo de una Nueva Gramática de la Lengua Española.* Espasa Calpe, 1991.

[3] Real Academia Española de la Lengua. *Diccionario de la Lengua Española.* Espasa Calpe, 21a edición, 1992.

[4] J. González, J.M. Goñi, A.Nieto. ARIES: a ready for use platform for engineering Spanish-processing tools. *Digest of the Second Language Engineering Convention*, pp. 219-226, Octubre 1995.

[5] J. Hallebeek. A formal approach to Spanish grammar. *Language and Computers: Studies in Practical Linguistics*, 1992.

[6] J.Hallebeek. *Morfología y Sintaxis del Español: Introducción al Análisis Oracional.* Playor, Madrid, España, 1994.

[7] F. Marcos, A. Ballester, C. Santamaría, E. Pertierra, O. Brandeo, P. Díez. Corpus oral de referencia de la lengua española contemporánea. Technical report, Universidad Autónoma de Madrid, 1992.

[8] M. Meya. Morphological analysis of Spanish for retrieval. *Literary & Linguistic Computing*, 2(3):166-170, 1987.

[9] S. Rodríguez, J. Carretero. Building a Spanish speller. *Taller sobre Software de Libre Distribución.* Universidad Carlos III de Madrid, España, 1995.

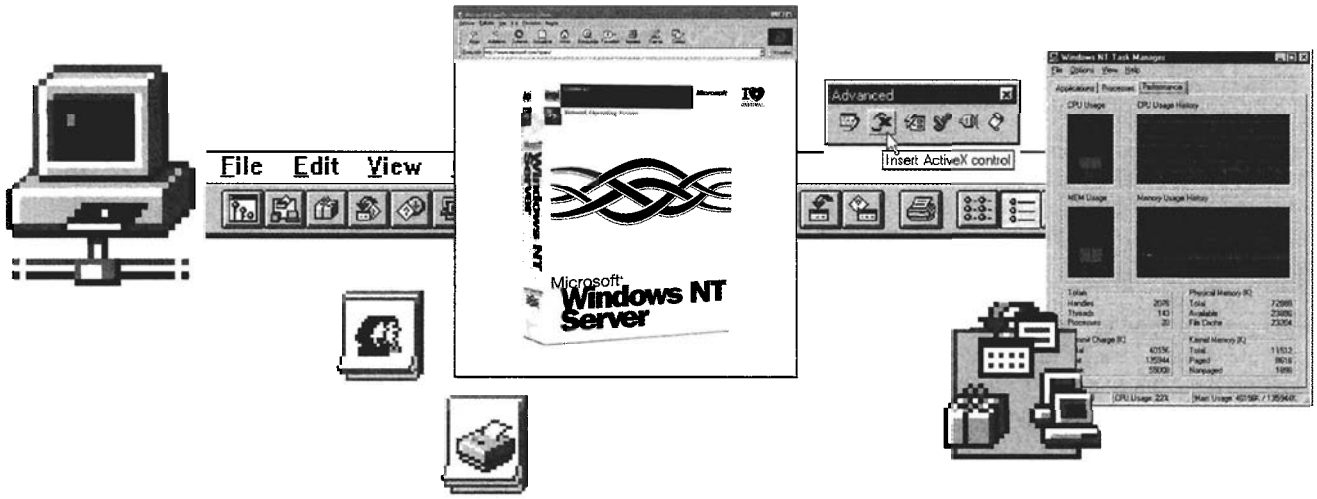
[10] S. Rodríguez, J. Carretero. A formal approach to Spanish morphology: the COES tools. *SEPLN'96 Conference Proceedings*, pp. 118-126. SEPLN, Sevilla, España, 1996.

[11] C. Smith. *Collins English-Spanish Dictionary.* Collins, 1988.

[12] E. Tzoukermann, M. Liberman. A Finite-State Morphological Processor for Spanish. *Proceedings of the 13th International Conference on Computational Linguistics (COLING 90)*, pp. 277-281, 1990.

Microsoft Windows NT Server 4.0

con la sencillez de Windows 95 toda la potencia de Windows NT



1. Microsoft Windows NT Server: El Servidor Polivalente:

- Servidor de Red (ficheros e impresoras).
- Servidor de Comunicaciones (Intranet & Internet).
- Servidor de Aplicaciones (BackOffice, etc.).
- Servidor de Teletrabajo.

2. Un único paquete. Todo de Serie.
Todo lo que necesita, incluido en un único producto:

- Arquitectura Intel, Digital Alpha, MIPS y PowerPC.
- Monoprocesador y Multiprocesador (SMP).
- Soporta clientes MS-DOS, Windows 3.x, Windows 95, Windows NT Workstation, Mac, OS/2, Unix,...
- Soporta todo tipo de protocolos: TCP/IP,IPX/SPX, NetBeui, NetBios, Appletalk,...
- Soporte Teletrabajo (RAS).
- Integrado con sistemas preexistentes (NetWare, Unix,...).
- **Servidor No dedicado.**
- Puede utilizarlo como puesto más con multitarea simétrica.

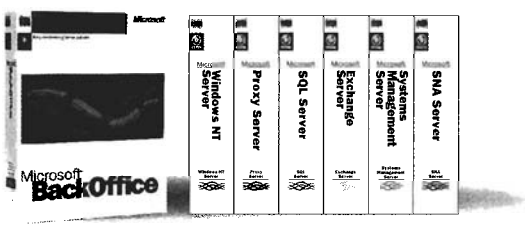
3. Facilidad, Interface Gráfico Windows 95

- Es el sistema Operativo de Servidor más sencillo de utilizar porque aporta a su red la familiaridad y facilidad de uso de Windows.

4. Intranet/Internet al alcance de su mano.

- El único Sistema Operativo que incluye Servidor Internet.
- Microsoft Internet Information Server incluido (www,ftp, Gopher).
- Gestión, control y creación de su entorno profesional Web: Microsoft FrontPage.
- Búsqueda automática sensible al contexto de cualquier documento en su Intranet: Microsoft Index Server.
- Comunicación de aplicaciones distribuidas (Intra/Internet) con tecnología Distributed Component Object Model (DCOM) (antes Network OLE).

5. Listo para trabajar como servidor InfoVía.



Microsoft Windows NT está preparado para sacar el máximo rendimiento a su negocio y es la base para la nueva familia de aplicaciones cliente-servidor, denominada Microsoft BackOffice

Adquiéralo en su Distribuidor Habitual.
Si desea más información
llame al número 902 197 198.

Microsoft SOLUTION PROVIDER

¿Hasta dónde quieres llegar hoy?

Microsoft

www.microsoft.com/spain/

Software Libre

Software de dominio público orientado a la seguridad

Jose Luis González *, Mª Soledad Sánchez *, José Caballero **
* *Universidad de Extremadura;* ** *Universitat Oberta de Catalunya*

jlgs@unex.es
marisol@unex.es
caballero@uoc.es

1. Resumen

La filantropía de los informáticos que han implementado software de dominio público (libre, gratis o *freeware*) poniéndolo a disposición de todo el que desee aprovechar su esfuerzo ha "beneficiado" enormemente, también, a la seguridad informática.

Vamos a presentar algunas de las más significativas herramientas *freeware* para garantizar una mayor seguridad de la información que está depositada en los sistemas informáticos o "viajando" por las redes. Pero urge destacar que la mayor parte del software libre que va a presentarse puede ser considerado como un arma de doble filo. Aunque la mayoría de los desarrollos se realizaron pensando en usuarios dignos de confianza, paradójicamente, algunos pueden convertirse en auténticos misiles dirigidos a la línea de flotación de muchos sistemas o redes que están débilmente administradas, en cuanto a seguridad se refiere, si caen en manos de *hackers* o *crackers.*

En lo referente a la seguridad de la información, podríamos decir que los **Sistemas Informáticos (SI)** presentan debilidades ("agujeros" en la jerga de la profesión) en tres importantes facetas: en los Sistemas Operativos, en las Comunicaciones y en las propias negligencias de los usuarios finales y/o administradores de los mismos SI. Pretendemos realizar la exposición de todos los productos agrupándolos según las debilidades de los SI que intentan conjurar, asumiendo que ésta es una labor complicada puesto que muchos de ellos podrían situarse en varios grupos simultáneamente ya que intentan solventar diferentes agujeros de seguridad. En cualquier caso, la mayor parte de estos productos se centra, primordialmente, en alguno de los tres aspectos de debilidad citados.

Por razones obvias, no vamos a poder analizar todos los productos existentes, ni con la profundidad que desearíamos, pero nos centraremos en aquellos que hemos considerado más útiles, tanto para usuarios finales, como para administradores de Sistemas o Comunicaciones y responsables de seguridad informática.

Destacar que, aunque el software aquí estudiado está pensado para la seguridad informática, no hay que olvidar que el verdadero contexto en que se situa este artículo es el del software libre, por lo que los aspectos de seguridad van a ser tratados colateralmente.

2. Debilidades en los Sistemas Operativos

La mayor parte de las herramientas de dominio público orientadas a la seguridad están pensadas para entornos *Unix* en cualquiera de sus variantes. No obstante, muchas de ellas pueden ser también instaladas en sistemas como *OpenVMS, Macintosh, Linux, Windows NT, MS-Windows, MS-DOS* o *Netware.*

Es sabido que *Unix* ha sido uno de los Sistemas impulsores de Internet desde sus orígenes y el que más servidores de, salvada la redundancia, servicios Internet soporta. *Unix* ha sido, y es a menudo, noticia por la gran variedad de agujeros de seguridad que padece, pero en la actualidad, la mayoría de variantes de *Unix* está certificada a nivel C2 (y superior) de seguridad. A pesar de esto, ocurre que algunos sistemas están aún a nivel C1 o bien, a pesar de ser C2, no están configurados como tal o están descuidados en lo que a seguridad respecta. Queremos destacar con esto, que las debilidades no sólo están en los sistemas. Además, no sólo *Unix* es inseguro, aunque sí el más conocido y extendido en Internet. Para evitar las tentaciones que este popular sistema pueda despertar a algún aventajado se proponen las siguientes soluciones pensadas para reforzar su seguridad.

2.1. COPS *(Computer Oracle and Password System)*

Conjunto de programas para monitorizar el sistema *Unix* y detectar "caballos de Troya" o *backdoors* (puertas traseras), sin necesidad de que el administrador del sistema esté pendiente constantemente. Cuenta con apartados diferentes que se cuidan de agujeros de seguridad concretos realizando las siguientes operaciones:

-Comprueban que los ficheros y directorios "vitales" tengan las protecciones adecuadas.
-Verifica que no haya aparecido, o desaparecido, ningún SETUID, SETGID o algún fichero con esos bits activos haya sido modificado.
-Localizan *passwords* débiles (demasiado sencillas, evidentes o triviales).
-Comprueban los campos de los ficheros */etc/passwd, /etc/group* e *yp.*
-Examinan los ficheros */etc/rc** y */usr/lib/crontab* cuidando que estos no tengan permiso de escritura para todos los usuarios del sistema.
-Verifican que los directorios de trabajo (*home dir*) de los usuarios no tengan concedido el permiso de lectura para el resto de usuarios, y que los ficheros *.profile, .cshrc y .rhosts* no tengan activado el privilegio de escritura para otro usuario que no sea su propietario.
-Comprueban que los sistemas de ficheros no estén corruptos.
-Los resultados de todas estas comprobaciones son enviadas por mail al usuario *root* para alertar al administrador del sistema.

Como norma, es muy conveniente ejecutar este paquete de herramientas de seguridad cada 10 ó 15 días con finalidades preventivas.

2.2. SATAN (*Security Administrator Tool for Analyzing Networks*)

Herramienta de ayuda para los *Systems managers* pensada para encontrar agujeros de seguridad en los siguientes componentes del sistema Unix: *NFS, NIS, TFTP*, versiones antiguas (por debajo de la 8) de *sendmail, Xserver, rsh, ftp anonymous*, etc. **Satan** genera informes como resultado de la "auditoría" realizada. Esta es una peligrosa herramienta en manos oportunistas, ya que permite comprobar de forma inmediata y remota los agujeros por donde colarse en sistemas descuidados o confiados.

2.3. *TripWire*

Este producto puede entenderse como un monitor de integridad para sistemas *Unix*. Detecta cambios, o alteraciones en los ficheros que, previamente, se han seleccionado para ser monitorizados. Controla los cambios en permisos, *links* y tamaños de ficheros y directorios. Cuando se instala en un sistema seguro, detectará los cambios producidos por intrusos o modificaciones realizadas para, por ejemplo, introducir *backdoors*, bombas lógicas y virus en entornos *Unix*.

De forma similar, otro producto llamado ***TIGER*** informa sobre agujeros de seguridad del sistema en que se instala. Comprueba los privilegios de ficheros de usuarios y de ficheros especiales, asi como el estado de los servidores de *ftp anonymous*.

2.4. *Xwatchwin*

Esta es una herramienta pensada con fines docentes. Se usa para observar, desde un X-terminal o estación de trabajo, las ventanas de otro X-terminal. Su sentido inicial es que un profesor pueda impartir clases prácticas en un aula informática usando *xwatchwin* para que en los monitores de sus alumnos pueda visualizarse lo que él está tecleando en su terminal o puesto de trabajo.Pero esta herramienta también puede tener un uso maléfico que es el poder capturar las ventanas de un X-terminal sin el consentimiento, ni conocimiento, de la persona que está siendo observada.

Cuando este software libre es usado con malas intenciones, más que aprovechar debilidades en los Sistemas, lo que hace es sacar partido de configuraciones confiadas de la **GUI (Interfase Gráfica de Usuario) X11**, íntimamente ligada al sistema operativo.

Para utilizarlo no hay más que indicar el nombre de la máquina que desea observarse e, incluso, el nombre o identificador de la ventana que desea capturarse.

Si el entorno gráfico en que estamos trabajando es *X11* puede evitarse el ser fisgado con el comando *xhost*. Con *xhost -*, nadie podrá observar. *xhost +* permite que cualquier nodo de nuestra LAN pueda observarnos. Con un nombre de nodo tras el signo + ó - se especifica que se desea autorizar o desautorizar a ese nodo particular. (Ej. *$xhost +nodo1* permite que el nodo1 nos observe).

En entornos gráficos *Motif* puede evitarse ser observado con la opción *Security* del menú *Options* del *Session Manager* que permite elegir el protocolo (*TCP/IP* ó *DECNET*) y los nombres de nodos a los que se permite observarnos.

Para sistemas *OpenVMS* existe otra potente herramienta, llamada ***SUPERVISOR***, para auditar los terminales asignados a arquitecturas VAX y Alpha. *SUPERVISOR* necesita privilegios para ser ejecutado, por lo que es difícil que pueda ser usado con malos fines a no ser que se obtengan indebidamente los privilegios requeridos. En realidad, es una buenísima herramienta para que los administradores de sistemas puedan auditar actividades sospechosas, registrando las pistas (con la utilidad *PHOTO* de *SUPERVISOR)* y fechorías realizadas. Todo lo ejecutado por un terminal auditado puede ser posteriormente reproducido con la opción *PLAYBACK*.

3. Comunicaciones inseguras

Gran parte de los atentados a la seguridad de los sistemas son realizados a través de las redes. El impulso dado a las Comunicaciones por los exitosos protocolos de Internet es incuestionable. Pero *TCP/IP* y tecnologías de redes como *Ethernet* tienen también importantes debilidades que no han tardado en ser aprovechadas para acceder a la información que transportan usando diversas y arteras técnicas.

Las herramientas que siguen se usan con dos fines principales: por un lado, acceder a la información que viaja por las redes sin proteger y, por otro lado, aprovechar las comunicaciones para acceder a sistemas con las debilidades ya citadas en el apartado 2, o aprovechando los descuidos de los usuarios que comentaremos en el punto 4.

Algunos de estos productos pueden ser usados con malos fines, pero también para detectar inseguridades en las comunicaciones y para conseguir reforzar sus debilidades.

3.1. *Sniffers*

Muy populares en la actualidad, aunque ya en 1994 constituyeron el 80% de los ataques a la seguridad de las redes. Son programas usados por los *crackers* para "escuchar" los paquetes que viajan por las redes (*Ethernet, DECnet,*etc.) filtrando *username* y *password* en conexiones *telnet, rlogin o ftp*. Pero también son capaces de capturar cualquier otro tipo de información empaquetada según los protocolos de comunicaciones más extendidos.

Se usan para visualizar o desviar información privilegiada, aunque su uso mas habitual es averiguar *passwords* de cuentas de usuario (que no viajan encriptadas por la red) para, posteriormente, acceder a ellas.

Suelen usarse desde dentro de las organizaciones en las que se permiten "pinchazos" incontrolados a la red.

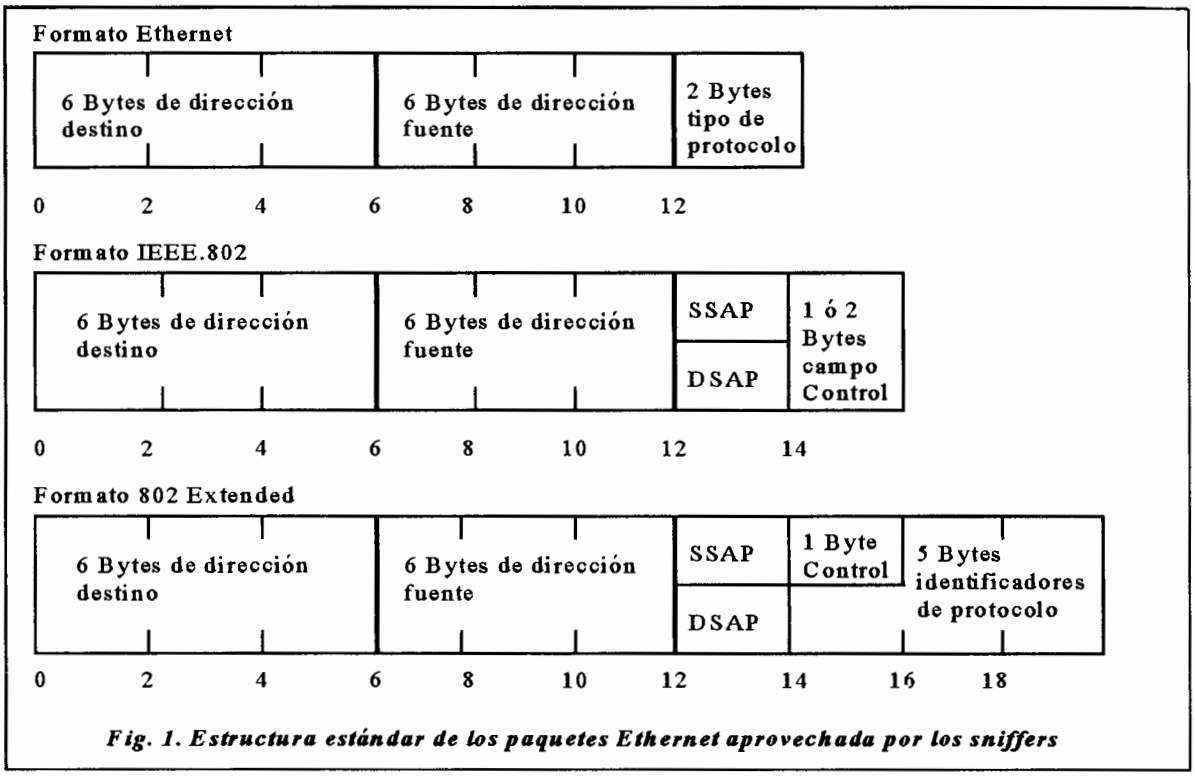


Fig. 1. Estructura estándar de los paquetes Ethernet aprovechada por los sniffers

La mejor forma de detectarlos es manteniendo un control de todas las maquinas conectadas a la red por parte del Administrador de Comunicaciones para evitar su uso desde ordenadores personales. En sistemas multiusuario como *Unix* y *OpenVMS* también es posible su utilización. Aunque en *VMS* se requieren privilegios para su ejecución, en *unix* es más sencillo su uso por lo que el Administrador del Sistema o Comunicaciones debe encargarse de auditar su sistema para detectar fisgones que suelen ser procesos escuchando en los puertos *UDP 891* y *937* y en el port *TCP 3011*.

Etherwatch

Es un programa *sniffer ethernet* que permite monitorizar la actividad de la red y puede ser usado para la identificación y diagnosis de problemas de red. Captura el trafico basado en direcciones Ethernet (fuente, destino o ambas), tipo de protocolo (formato ethernet), identificador de protocolo (*IEEE 802 Extended Format*) o combinación de todos esos modos. Son necesarios privilegios para ejecutarlo en sistemas multiusuario, pero también puede ser usado en ordenadores personales (sin ningún tipo de privilegio, evidentemente) que se hayan "pinchado" a la red de manera estratégica para capturar o filtrar el trafico que les interese.

Etherwatch aprovecha los paquetes estándares de *Ethernet* para capturarlos y mostrarlos según respondan a los intereses de la persona que ha puesto el *sniffer* en marcha.

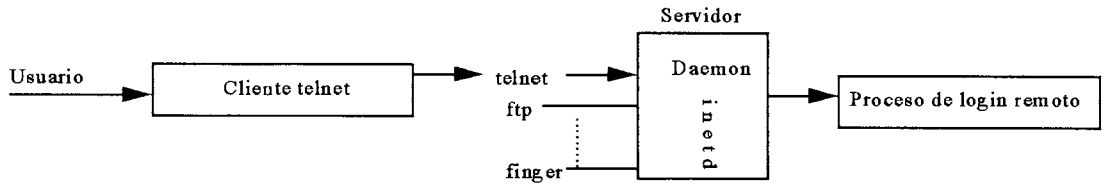
La **Figura 1** presenta los formatos de cabeceras de los paquetes *Ethernet*, para poder observar la situación de las direcciones fuente, destino, campos de tipo de protocolo y otros campos que pueden ser usados para filtrar y manipular la información que está viajando por un tramo de red *Ethernet*. Como puede observarse, cada dirección *Ethernet* tiene una longitud de 48 bits y esta representada por el estándar *ethernet* como 6 pares (bytes) de dígitos hexadecimales separados por guiones (08-00-20-19-4e-cc) para almacenar la dirección de la máquina fuente y destino de cada paquete. *Etherwatch* tiene la posibilidad de indicar parámetros como *FROM* y *TO* para especificar los dos nodos que desean ser "espiados". Sólo se necesita que el nodo que hace de *sniffer* se encuentre en el tramo de red entre el originador y el destinatario de los paquetes, o en la misma subred *broadcast*.

Del mismo modo, *Etherwatch* permite especificar otros parámetros opcionales para filtrar los paquetes en función del tipo de protocolo que se desee, o de los contenidos de cualquiera de los campos de la cabecera de los paquetes.

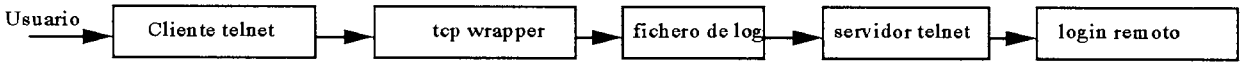
Una vez puesto en marcha el *sniffer*, e indicados los parámetros deseados, sólo queda registrar la información que viaja en el cuerpo de los paquetes *Ethernet* en un fichero de *log* para, posteriormente, analizarlo y visualizar la comunicación establecida entre el nodo fuente y destino. Si, por ejemplo, se estableció la comunicación entre ellos para iniciar una sesión de *telnet* ó *ftp*, en que es necesario indicar la *password* al llegar al nodo destino, ésta habrá quedado al descubierto pues, aunque en el nodo fuente no se visualiza por un simple mecanismo de desactivación del "eco" en pantalla, realmente está viajando por la red en texto plano (legible) para cualquiera que tenga la posibilidad de localizar el paquete en que se ha enviado.

Es realmente espeluznante ver nuestra *password* impresa en pantalla e imaginar cómo puede quedar a disposición de cualquiera y, por tanto, también la cuenta de usuario y toda la información en ella contenida.

Otro ejemplo de *sniffer Ethernet* público es el conocido ***LATWATCH*** que permite monitorizar la actividad de la red y puede usarse para la identificación y diagnosis de problemas en nuestra LAN. Éste procesa únicamente paquetes **LAT** propios del protocolo *DECnet* en sistemas *OpenVMS* que, al menos en este caso, no puede ser utilizado a no ser que el usuario disponga del elevado privilegio *PHY_IO* reservado al Administrador del sistema.



El usuario remoto realiza una petición de conexión telnet desde su cliente. En el servidor, el daemon inetd escucha en sus ports y espera la solicitud externa de servicios. En el ejemplo, atiende una petición de telnet y permite la conexión remota arrancando un proceso servidor de telnet. Mientras tanto, el proceso inetd sigue esperando otras peticiones de servicio externas.



El telnet original del servidor ha sido sustituido por tcpd (tcp wrapper) que obtiene un fichero de log para registrar las peticiones de servicios y consulta los ficheros allow y deny para controlar los acceso. Wrapper arranca luego el telnet original permitiendo los accesos controladamente y sin que el usuario note ninguna diferencia en el filtro realizado.

Fig. 2 Forma de trabajo de telnet sin y con tcp wrapper

3.2. TCP *Wrapper*

Una de las técnicas más empleadas por los *crackers* para acceder a los sistemas es usar el servicio de red *finger* que no requiere ningún tipo de privilegio para ser usado desde una máquina remota. El programa *finger* informa de los usuarios que están presentes en un sistema en ese momento, obteniendo su *username* y otros datos personales. Si un *cracker* desea acceder a un sistema desde otro remoto, lo lógico es esperar a que no se encuentre ningún usuario presente en él, o por lo menos no esté presente *root*. Lo que hará, por tanto, es "bombardear" la máquina que desea atacar con *fingers* continuos hasta que detecte que el sistema está sólo.

Otro uso de *finger* es averiguar nombres de cuenta de usuario (*usernames*) para intentar luego averiguar sus *passwords* con las astutas técnicas que exponemos en el apartado 4. La solución para esto es eliminar de la máquina atacada los *daemons* que atienden los servicios como *finger, telnet, ftp*, etc., lo que acaba afectanto a todos los usuarios del sistema y se convierte en una solución demasiado restrictiva.

TCP Wrapper es una solución muy apropiada ante este tipo de situaciones, lamentablemente, muy habituales. Puede verse como el complemento perfecto para *firewalls hardware* o, incluso, un sustituto, ya que puede considerarse un *firewall software* a coste cero. Sirve para monitorizar y controlar tráfico de entrada a la red, y puede usarse a modo de escudo de sistemas y detección de actividades de *crackers*. Delimita qué nodos (internos o externos) de la red pueden acceder a otros nodos. Además, permite determinar los servicios Internet (*ftp, telnet, rlogin, finger*,etc) que van a autorizarse a los usuarios.

Puede también configurarse para obtener informes de intentos de acceso fallidos y como herramienta para controlar desde qué direcciones **IP** se accede y qué servicios se están usando. Por tanto, es muy útil, junto con el *accounting* de los sistemas, para seguir la pista a los *crackers* o a cualquier actividad sospechosa.

Su instalación no necesita ninguna modificación de software del sistema, tampoco implica ningún tipo de acción ilegal sobre los usuarios del sistema y es muy adecuado para servicios orientados a la conexión (*TCP*) o de datagramas (*UDP*) que se centralizan en un sólo *daemon* como *inetd*. Este mecanismo de seguridad consiste en interponer el *wrapper* antes del *inetd* (**Figura 2**). Para ello se toma el fichero *inetd.conf* y se protegen los servicios deseados (*ftp, telnet, finger*, etc.). Además, para tener operativo el *wrapper* es necesario copiar su ejecutable, *tcpd,* al directorio de *daemons (/usr/sbin)* y crear los ficheros */etc/hosts.allow* y */etc/hosts.deny* como ficheros de control de acceso.

Cuando, desde un cliente, llegue al servidor una petición de acceso, *tcpd* consulta *hosts.allow* para saber si al nodo que está intentando acceder se le debe permitir el paso; si es así, el servidor le ofrecerá el servicio solicitado. Si en el fichero *hosts.allow* no existe ninguna regla *tcpd* leerá el fichero *hosts.deny* para comprobar si se le debe denegar el acceso. Lo más sensato sería configurar el fichero *hosts.deny* con la regla *ALL:ALL* que deniega el acceso a todos los servicios para todos los nodos que no estén autorizados en el fichero *hosts.allow* que será el que realmente centralice los accesos.

Los ficheros *allow* y *deny* contienen reglas para controlar el acceso de los nodos cliente. En ambos ficheros, el primer parámetro de cada línea referencia el tipo de servicio (*finger, telnet, ftp*, etc.) y a continuación, separado con dos puntos (:), se indica el *host o hosts* al que se refiere el servicio concreto. Ambos ficheros pueden contener varias reglas en líneas diferentes que serán leídas secuencialmente por *tcpd* hasta encontrar el fin de fichero. Por ejemplo, la regla *ALL: nodo1.subred.org.es* en el fichero *hosts.allow* indica que al host de nombre *nodo1* se le permite el acceso a todos los servicios que centraliza el *daemon inetd*. Si esta misma regla se coloca en el *hosts.deny* se denegarán todos los servicios al *nodo1*. En ambas partes de la regla pueden incluirse listas de servicios o máquinas separadas por comas. Por ejemplo, la regla *fingerd,ftpd: nodo1.subred.org.es, nodo3. subred.org.es* colocada en el *hosts.deny* indica que a esos dos nodos se les impide el acceso a los servicios *finger* y *ftp*. Las reglas también cubren la posibilidad de indicar subredes en lugar de máquinas únicas, e incluso excepciones de máquinas particulares dentro de una subred.

Otra importante característica de esta herramienta de seguridad es la de poder ser configurada para que, automática-

mente, realice "contraataque" (*finger*) a los nodos que consideramos sospechosos cuando nos están intentando acceder para así poder registrar pistas de actividades sospechosas de presuntos atacantes.

Una limitación de *TCP Wrapper* es la de no ser apropiado para bloquear el acceso a *ports* de servicios *RPC* como *NIS (yellow pages), NFS, portmapper* y *Xserver*, ya que está pensado para los servicios que arrancan un *daemon (telnetd, fingerd, ftpd, tftpd..)* cada vez que son invocados, y no para aquellos como *NIS* ó *NFS* que se arrancan al botar el sistema y que crean subprocesos a medida que van recibiendo peticiones.

Por tanto, para los agujeros de seguridad permitidos por los *daemons* que *TCP Wrapper* no protege (*nfsd, ypserv, portmap...*) pueden usarse bloqueos en los propios *routers*, cortafuegos hardware u otros productos *freeware* como *securelib, rpcbind, portmap* ó *Kernel_Wrap*.

3.3. *Kernel Wrap*

Este es un *wrapper* para los daemons que usan *RPC* como *portmap, ypserv, ypbind, ypupdated, mountd*, etc... Está pensado para *SunOS 4.x* y deben reconstruirse las librerías *shared*. La función que realiza es modificar las llamadas a funciones que usa el *Kernel* para establecer las conexiones *RPC* como *accept(), recvfrom()* ó *recvmsg()*. Todas estas llamadas están en las librerías *shared* por lo que, reconstruyéndolas, no será necesario manipular el *kernel* para poder proteger los daemons RPC.

3.4. SOCKS

Es otra forma de implementar un *TCP Wrapper*. *Socks* no está pensado para hacer más seguro un sistema, sino que su función es más bien la de centralizar (*firewall*) todos los servicios externos de Internet.

3.5. CAP (*Controlled Access Point*)

Es, esencialmente, un *router* con funcionalidades para: Detectar peticiones de conexión externas; Interceptar el proceso de comprobación de usuario para generar alarmas; Utilizar un servidor de comprobación de usuarios. Suministra mecanismos para reducir los riesgos de: *Passwords* débiles de usuarios; Detección de cuentas con *passwords* por defecto; *rlogin* desde nodos inseguros o sospechosos; *passwords* capturados por *sniffers*.

4. Usuarios confiados

Dentro de este grupo realmente podríamos enmarcar todos los productos anteriores pues, en cierto modo, la mayor parte de inseguridades de la información se producen por descuidos de los usuarios finales o de los administradores de los mismos SI que no suelen contar, ni con tiempo, ni con presupuesto para cuidar la seguridad como les gustaría.

No obstante, queremos destacar los siguientes productos de dominio público que, aunque se centran en la debilidad de los sistemas para almacenar sus ficheros de *passwords* y los privilegios de ficheros, enfatizan, sobre todo, la elección de *passwords* complicadas para el acceso a los sistemas y el seguimiento de los ficheros por parte de sus propietarios que han de estar concienciados e implicarse en los problemas de la inseguridad informática.

4.1. *Chkacct*

Podríamos decir que es un *COPS* para que los usuarios generales (no *systems managers*) puedan llevar un control diario del estado de seguridad de sus ficheros más importantes. Genera informes por mail para alertar al usuario de que algunos de sus ficheros, como *.rhosts*, están inseguros.

4.2. *Crack*

Esta es la mejor ayuda que poseen los "Administradores de Sistemas" para averiguar qué usuarios de sus sistemas tienen protegido el acceso a su cuenta de usuario con *passwords* débiles (demasiado evidentes) posibilitando el acceso de una forma fácil.

Paradójicamente, *crack* es también una de las herramientas que más insomnio produce a los *Systems* cuando es usada por piratas informáticos, ya que el fichero */etc/passwd* ha sido una de las debilidades históricas de *unix* por ser legible y copiable por cualquier usuario del sistema. Una vez conseguido este fichero de *passwords*, sólo queda aplicarle *crack* para que comiencen a aparecer todas las cuentas de usuario con *password* débil que, desde ese momento, quedarán a merced de los antojos del violador que alcanza una posición anónima y privilegiada, si la cuenta lo es, desde la que realizar sus fechorías pudiendo, incluso, cargar con fuertes responsabilidades al verdadero propietario de la cuenta pirateada.

Para sistemas *OpenVMS* puede usarse también el producto *guess* con funcionalidades semejantes. Detecta qué usuarios tienen como *password* su propio *username*, o parte de su nombre y apellidos; o bien, si usan como *password* palabras típicas de diccionario, nombres de personas, animales, etc., e incluso aquellas cuentas que están sin *password*.

4.3. *Shadow*

La mejor forma de luchar contra los devastadores efectos de *crack*, es usar la técnica de *shadowing* con este *freeware* que ha sido adoptado por varios sistemas *unix* para certificarlos a nivel C2 de seguridad. *Shadow* es un conjunto de programas que cambian el mecanismo de control de *passwords* para eliminar el *password* encriptado del inseguro fichero */etc/passwd* (legible para todo el mundo, en el más amplio sentido de la palabra mundo si el sistema forma parte de Internet) y colocar la clave encriptada en un fichero que sólo es legible para el programa *shadow*.

Opcionalmente, puede usarse un mecanismo de "envejecimiento" de *passwords* para obligar a los usuarios a cambiar sus *passwords* periodicamente (similar a *OpenVMS*), o imponer el uso de *passwords* de más de seis caracteres, etc.

Shadow puede complementarse con *passwd+* para establecer restricciones en la elección de las *passwords* con el fin de que sean lo menos débiles posible por si son "crakeadas", que es como se dice en la jerga.

5. Criptosistemas ¿freeware?

La criptografía es, nunca mejor dicho, la clave para obtener la ansiada privacidad en la transmisión de información a través de las inseguras redes de comunicaciones. Por tanto, la encriptación de la información que se considere sensible es el mecanismo idóneo ante las debilidades de los SI.

El interrogante sobre la palabra *freeware* que da título a este apartado tiene justificación desde el punto de vista que realmente existen productos - y muy buenos - de dominio público orientados a la criptografía pero con serios problemas legales de uso. Las leyes norteamericanas impiden la exportación de material criptográfico fuera de sus fronteras y es allí donde, precisamente, se ha implementado la mayor parte de los criptosistemas *freeware*.

Seguidamente queremos exponer algunos interesantes productos relacionados con este tema.

5.1. El sistema Kerberos: el cancerbero ideal para redes informáticas seguras

En 1983 el ***MIT (Instituto Tecnológico de Massachusetts)*** junto con IBM y DEC emprenden el proyecto *ATHENA* basado en la filosofía Cliente/Servidor para cuidar la seguridad de la ***MITnet***. Para reducir los peligros de inseguridad implementaron el sistema de certificación ***Kerberos*** para evitar el envío de *passwords* sin encriptar a través de las, siempre vulnerables, redes.

Kerberos es un sistema de autentificación para redes físicamente inseguras (casi todas), basado en un modelo de distribución de claves propuesto por R.M. Needham y M. D. Schoroeder. Posibilita la comunicación a través de red garantizando la identidad de las entidades que participan en la comunicación y previene las indiscreciones y ataques. También garantiza la integridad de datos evitando su detección y manipulación. La privacidad también es garantizada, evitando las lecturas no autorizadas por medio del sistema criptográfico *DES*.

Kerberos suele usarse en protocolos del nivel de "Aplicación" (Nivel 7 del modelo OSI) como *Telnet y FTP* para ofrecer seguridad desde usuario a *host*. Suele usarse también, aunque con menor frecuencia, como sistema de autentificación de datos *(SOCK, STREAM)* o mecanismos *RPC* (Nivel 6 del modelo ISO). Incluso puede usarse a niveles bajos para seguridad de *host* a *host* con protocolos *IP, UDP* ó *TCP* (niveles 3 y 4 de ISO).

Características de la *MITnet:*

- Más de 10.000 ordenadores.
- Backbone de Fibra Óptica con ancho de banda de 100 Mbits/s.
- Más de 100 LAN's Ethernet.
- Más de 1.200 ordenadores forman parte de *Athena.*
- 25.000 estudiantes.

Los ingenieros del MIT eran conscientes de que:

- Las grandes redes son, físicamente, inseguras (cableado difícil de proteger).
- Casi todas las redes son *broadcast* (difusión pública) y todo ordenador conectado a la red tiene acceso a toda la información que viaja por la red.
- Un intruso en la red puede hacer que le lleguen datos que no son para él, y suplantar la personalidad de otro usuario.

Para evitar los problemas anteriores se propuso el sistema *Kerberos*:

- Verifica y certifica los paquetes de datos que viajan por *Athena.*
- Sistema distribuido que usa un mecanismo de intercambio de información cifrada antes de permitir el acceso a los servidores.
- Realiza verificaciones criptográficas para garantizar que los datos transferidos entre estaciones de trabajo y servidores no estén corruptos por accidentes o por accesos no autorizados.
- Utiliza el estándar *DES* para cifrar cada paquete de información mediante una clave que sólo conocen el cliente y el servidor, de forma que la información no tenga ningún valor para un tercero que pueda interceptar el mensaje.
- Si el atacante intenta modificar el contenido del mensaje, el destinatario solicitará el reenvío de los paquetes manipulados, y el interceptor quedará al descubierto.
- Las claves de todos los usuarios y servidores son sólo conocidas por un servidor especial llamado **CDC (Centro Distribuidor de Claves)** que interviene en todas las transacciones usando tickets digitales de certificación. El CDC se encuentra en lo que en el MIT llaman "la mazmorra", inaccesible sin autorización.

En cuanto a la situación legal de *Kerberos,* cabe decir que parece ser que siguen sin quedar claras las competencias de los Departamentos de Estado y Comercio estadounidenses.

El Departamento de Comercio controla la exportación del software y hardware de autentificación, mientras el Departamento de Estado impide la exportación de material criptográfico fuera de USA. El problema, por tanto, está en encontrar la línea divisoria entre lo que realmente es *Kerberos*, un sistema de autentificación como parece evidente, o un criptosistema que usa *DES* considerado no exportable por el Departamento de Estado. Mientras esto queda claro, y para no infringir ninguna ley Federal, lo que puede hacerse para testear esta potente herramienta es usar *Bones* que ofrece únicamente el *API* de *Kerberos* sin usar encriptación ni ofrecer seguridad. *Bones* es una solución parcial al problema legal de exportación de material criptográfico fuera de USA. Mediante la definición de un símbolo en tiempo de compilación se evita usar las llamadas y librerías de encriptación de *DES* para poder transportar los

fuentes entre fronteras de distintos países sin infringir ninguna ley americana.

5.2. PGP (*Pretty Good (tm) Privacy*)

Herramienta *freeware* ejemplo de criptosistema de clave pública que funciona sobre *MS-DOS*, *Windows*, *Mac*, *Unix* y *OpenVMS* entre otros sistemas operativos:

- Puede usarse para criptografía convencional de información contenida en ficheros.
- Muy adecuado para cifrar mensajes de correo electrónico.
- Permite una gestión personal de anillos (ficheros) de claves públicas y privadas.
- Comprime la información, lo que redunda en ahorro de tiempo, ancho de banda en las transferencias y en ocupación de espacio de disco.
- Proporciona, también, autentificación con la posibilidad de usar firmas digitales que pueden ser certificadas por notarios o certificadores de firmas.
- Permite elegir entre módulos de 512, 768 ó 1024 bits en función de la complejidad de clave que desee obtenerse.
- Emplea el algoritmo *RSA* para la generación de claves públicas.
- Destacar que este utilísimo *freeware* está provocando importantes problemas legales a su autor por, entre otras cosas, haberlo puesto al alcance de todo el mundo siendo un material criptográfico.

5.3. STEL (*Secure TELnet*)

Uno de los usos habituales de los *sniffers* es "escuchar" en los tramos adecuados de las redes para conseguir *usernames* y sus *passwords* asociadas. Esto es posible, por ejemplo, en conexiones remotas de terminales usando *telnet* ó *ftp*. Cuando nos conectamos por *telnet* a una máquina remota para hacer *login* se nos solicita el *username* y la *password*. El *username* es legible para nosotros, pero la *password* no, pues -previamente- se desactiva el eco de caracteres a pantalla. Esto nos protege de miradas indiscretas mientras tecleamos la *password*. Pero el problema está en que, aunque la secuencia de caracteres que forman la *password* no es visible, está viajando por la red en texto plano de forma totalmente legible para los *sniffers* de red que no tienen más que esperar a que los paquetes adecuados pasen por el tramo en que se encuentra uno de ellos escuchando el tráfico. Quiere esto decir, que cada vez que hacemos una conexión vía *telnet*, *ftp*, *rsh*, *rlogin* o *set host*, estamos anunciando públicamente nuestra *password*.

La solución propuesta a este problema, como era de esperar, consiste en encriptar la *password* antes de que comience a viajar por la red. Esta es la solución aportada por *STEL*.

6. Localización y obtención del software libre

En la **Tabla 1** se presenta una relación de los productos de dominio público comentados, y otros que pueden ser de interés. La tabla presenta los nombres de los productos, seguidos del autor o autores que los han puesto, gentilmente, al alcance de todos. La tercera columna muestra los servidores y direcciones en que pueden ser localizados.

La mayor parte de localizaciones son servidores de *ftp anonymous* a los que puede accederse con nombre de usuario (*username*) *anonymous* e introduciendo como *password* la dirección de correo electrónico o alguna identificación personal del usuario que establece la conexión *ftp*. Algunas localizaciones indican la dirección de correo electrónico de los autores, mientras otras muestran la URL con la dirección de protocolo *http* ó *ftp* donde conseguir el freeware desde visualizadores de *World Wide Web*. También aparecen referencias a grupos de USENET (*news*).

Destacar que la mayor parte de estos productos están disponibles en el servidor de *ftp anonymous* de **RedIRIS** en España y es recomendable, y más rápido, conectarse antes a este servidor (*ftp.rediris.es*) y consultar el directorio */mirror/coast/tools/unix/** o */pub/soft/security/** para localizar el *freeware* deseado. No obstante, la tabla muestra las direcciones originales y es necesario tener en cuenta que algunos de estos servidores puede dejar de existir, o de ofrecer, el servicio sin previo aviso. Se incluyen las direcciones IP entre paréntesis.

7. Organizaciones públicas para velar por la seguridad mundial de redes y sistemas

Además del software de dominio público puesto amablemente a disposición de todos por sus desinteresados autores, existen también organizaciones que podríamos llamar de dominio o servicio público que, sin ningún afán de lucro, realizan una impagable labor como asesores ante episodios relacionados con vulneraciones a la seguridad de los sistemas informáticos de todo el planeta. Entre ellas:

7.1. CERT (*Computer Emergency Response Team*)

Idea de la Universidad de *Carnegie Mellon*, actúa como asesor ante ataques que afecten a la seguridad. Informa periódicamente sobre todo lo relacionado con problemas de inseguridad (*bugs* en protocolos y sistemas, ataques, estadísticas, etc.). No tiene fuerza legal pero sí autoridad reguladora. Atiende consultas las 24 horas del día (+1 412 268-7090) y mantiene también valiosísimas listas de distribución de correo electrónico como *cert-advisory-request@cert.org*.

7.2. CIAC (Computer Incident Advisory Capability)

Soportado por el Departamento de Equipos de Energía de *Lawrence Livermore National Laboratory*. Desarrolla líneas de acción para responder ante incidentes, e implementa software para responder a eventos relacionados con inseguridad. Informa, forma, alerta y asiste a organizaciones que han sufrido ataques (+1 510 422-8193. *ciac@lln1.gov).*

Otras organizaciones que ofrecen un servicio similar son: ***NIST (National Institute of Standards and Technology)*** ***FIRST (Forum of Incident Response and Security Teams).*** Además, organizaciones españolas como RedIRIS, entre otras, disponen de su propio ***CERT-ES*** *(iris-cert@rediris.es)* ofreciendo los servicios del CERT original en el ámbito de la red académica y de investigación española y de sus organizaciones asociadas.

PRODUCTO	AUTOR(ES)	LOCALIZACIÓN
Audit	Bjorn Satdeva	gjorn@sysadmin.com
CAP	D. Thompson/K. Arndt	thompsond@orvb.saic.com
COPS	D. Farmer	ftp cert.org:/pub/tools/cops (192.88.209.5)
Crack	Alec Muffetf	ftp cert.org:/pub/tools/crack/crack_4_1.tar.Z
CrackLib	Alec Muffet	ftp cert.org:/pub/tools/cracklib
Etherwatch y Latwtach	David B. Sneddon	sneddo@perth.dialix.oz.au
guess	Joe Meadows Jr.	http://www.wku.edu/www/fileserv/fileserv.html
HSC-Gatekeeper	Herve Schauer	Herve.Schauer@hsc-sec.fr
ISS	Chris Klaus	USENET comp.sources.misc y ftp cert.org:/pub/iss
Kerberos y Bones	MIT	ftp athena-dist.mit.edu:/pub/kerberos5 (18.159.0.42)
Kernel_wrap	CIAC, CERT, FIRST	ftp eecs.nwu.edu:/pub/securelib (129.105.5.105)
Miro	miro@cs.cmu.edu	ftp ftp.cs.cmu.edu (Proyecto Miro) (128.2.206.173)
NPasswd	Clyde Hoover	ftp ftp.rediris.es:/pub/soft/security (130.206.1.2)
One-time Password Key-card	cert@cert.org	cert@cert.org
Passwd+	Matt Bishop	ftp ftp.dartmouth.edu:/pub/security/passwd+.tar.Z (129.170.16.79)
PGP	Philip Zimmermann	ftp://idea.sec.dsi.unimi.it (149.132.3.4)
Satan	Wietse Venema/ D. Farner	ftp ftp.caltech.edu:/caltech/security/satan (131.215.48.49)
SecureLib	William LeFevre	ftp eecs.nwu.edu:/pub/securelib.tar
Shadow	John F. Haugh III	comp.sources.misc:/usenet/comp.sources.misc/volume38 y volume39
SOCKS	D. Koblas/M. R. Koblas	s1.gov:/pub/socks.tar.Z (128.15.32.7) y ftp.rediris.es
SPI	Gene Spafford	spaf@cs.purdue.edu o consultar CIAC
STEL	D.Vincenzetti/S.Taino	ftp://idea.sec.dsi.unimi.it
SUPERVISOR	Hunter Goatley	http://www.wku.edu/www/fileserv/fileserv.html
Swatch	CERT,FIRST	ftp://ftp.Stanford.EDU/general/security-tools/swatch
TAMU	D. Safford/D.Schales/D.Hess	ftp net.tamu.edu:/pub/security/TAMU (128.194.177.1)
TCP Wrapper	Wietse Venema	ftp cert.org:/pub/tools/tcp_wrappers/tcp_wrappers.*
TIS Firewall ToolKit	TIS	ftp ftp.tis.com:/pub/firewalls/toolkit (192.94.214.101)
TripWire	Gene Kim/Gene Spafford	ftp://coast.cs.purdue.edu/pub/COAST
watcher	Matt Madisson	http://www.wku.edu:/www/fileserv/fileserv.html
xwatchwin	George D. Drapeau	ftp ibmserv.edu.tw:/pub1/xwatchwin (140.111.3.11)
YPX	Rob Nauta	USENET comp.sources.misc y ftp.rediris.es
La relación es prácticamente interminable y además de los anteriores queremos destacar, *Mail Gateway, AT&T Inet, Chkacct, Tiger, MLS, rpcbind, nfstrace y nfswatch.*		

Tabla 1: Relación de productos de software libre orientado a la seguridad

8. Conclusiones

Hemos presentado una relación casi enunciativa, sin entrar en demasiados aspectos técnicos que despistaran la atención del verdadero objeto del presente trabajo que es el *freeware* orientado a la seguridad. No hemos podido exponerlos todos ya que, realmente, cualquiera de ellos requeriría el espacio de todo este artículo para analizarlo en profundidad.

Queremos destacar que la mayor parte de este software de dominio público ha sido desarrollado en Universidades que son, posiblemente, las organizaciones que más ataques sufren y, sin lugar a dudas, las que más *hackers* y *crackers* producen.

Algunas son soluciones realmente potentes y, todas juntas, pueden garantizar unos índices de seguridad bastante elevados. Pero es necesario tener muy presente que, igual que todo este software está disponible para los Administradores de Sistemas y Comunicaciones, también lo está para los que se dedican a actividades deshonestas pervirtiendo su idea original. Por tanto, la solución no está en instalarlos en nuestros sistemas y descansar, sino en un seguimiento continuo de las novedades que aparecen y en una formación adecuada para intentar estar siempre por delante de usuarios deshonestos. Es necesaria, por tanto, una preocupación en su justa medida y sin caer en la obsesión que acabe siendo demasiado restrictiva e incómoda para los usuarios honestos que, no lo olvidemos, son mayoría.

Cuidando los tres grupos o aspectos generales de inseguridad que hemos utilizado para realizar la presentación de los productos, acabaría obteniéndose la necesaria seguridad en los SI. Si conocemos y atajamos los agujeros de nuestro sistema; evitamos los ataques a través de la Red y nos concienciamos como usuarios, tendremos ganada gran parte de la partida. De hecho, las estadísticas del CERT indican que casi el 100 % de los ataques provienen de los *sniffers* y programas *crackers* de *passwords*. Como hemos visto, los *sniffers* pueden ser detectados y los *crackers* no se atreven con *passwords* bien elegidas. Además, cuando manejemos información confidencial, tenemos la criptografía.

Nada más queda agradecer a los autores de todo este software de dominio público su inestimable aportación y rogar a todos los que se dedican a "pervertir" su *freeware* que los utilicen como modelo a seguir en el futuro.

9. Referencias

Guías de instalación de cada una de las herramientas de dominio público comentadas.

Seguridad en VLANs. *Institute for International Research.* **González J. L.** (Marzo, 1996).
Seguridad de la mensajería electrónica en Internet. *Revista Seguridad en Informática y Comunicaciones,* 21. **González J.L., Caballero J., Sánchez M.** (Septiembre, 1996).
Protección de la información en las nuevas organizaciones (I y II). RedesLan Nº 95 y 96. **González J.L., Caballero J., Sánchez M.** (Octubre y Noviembre, 1996).
Redes informáticas seguras. **Jeffrey I. Schiller**. *Investigación y Ciencia.* Enero, 1995.
http://avalon.ira.uka.de/eiss/indexe.html.
ftp://info.cert.org/pub/tech_tips/email_bombing_spamming y *~email_spoofing.*
http://www.alw.nih.gov/Security/
Grupos de *news alt.security* y *comp.security.misc*

10. Bibliografía

S.M. Bellovin; *Security Problems in the TCP/IP Protocol Suite,* Apr. 89.
Simson Garfinkel and Gene Spafford; *Practical UNIX Security,* Jun. 91.
B. Chapman, E. Zwicky; Internet Security Firewalls, 95.
Seguridad en Redes y Sistemas. **Francisco Cruz Argudo** Boletín *RedIRIS* Nº 35 (Abril 96).
Communications of then ACM Vol. 32, Nº 6, Jun. 89.
A public key cryptosystem and a signature scheme based on discrete logarithms. Taher ElGamal IEEE transactions on information theory.
RFC 822 Crocker, D. *Standard for the Format of ARPA Internet Text Messages,* Ago. 82.
RFC 1421 Linn, J. *Privacy Enhancement for Internet Electronic Mail; part I: Message Encryption and Authentication Procedures.* Feb. 93.
RFC 1824: *The exponential security System TESS.*
RFC 1847 Galvin J. *Security Multiparts for MIME:Multipart/signed and Multipart/Encripted.* Oct. 95.

Software Libre

Experiencias en el desarrollo de software de dominio público para uso docente en las enseñanzas universitarias (Herramienta de diseño lógico)

Lourdes Peñalver, A. Pont, J. A. Gil
Departamento de Ingeniería de Sistemas, Computadoras y Automática; Universidad Politécnica de Valencia

{lourdes, apont, jagily}@disca.upv.es

Resumen: *La realización de prácticas en la universidad conlleva muchas veces el uso de herramientas software/hardware, en general, muy costosas para la misma. Además, sobre todo en lo que se refiere a las herramientas software, el alumno muchas veces las solicita para poder realizar sus propias experiencias en casa. Todo esto, unido a que cualquier estudio de ingeniería requiere por parte del alumno la realización de trabajos/proyectos final de carrera, hace que sea interesante dirigir estos trabajos hacia el diseño de herramientas que permitan disponer de software de forma gratuita tanto para profesores como para alumnos. Este tipo de proyectos final de carrera servirán a su vez de experiencia a los alumnos a la hora de realizar un programa con características profesionales. Siguiendo esta idea, aquí se presenta una experiencia encaminada a dotar de una herramienta de dominio público para la realización de las prácticas de las asignaturas de Estructura de Computadores para las titulaciones de Ingeniero Técnico en Informática de Gestión, Ingeniero Técnico en Informática de Sistemas e Ingeniero en Informática de la Universidad Politécnica de Valencia.*

Palabras claves: *diseño lógico, estructura de computadores, programación orientada a objetos, docencia, software libre.*

1. Introducción

Es una práctica común, sobre todo en universidades extranjeras, el desarrollo de software de dominio público que permita a la comunidad científica y estudiantes su uso y modificación, adaptándose a las necesidades del entorno y del momento. Debido a las limitaciones de presupuesto que, en general, disponen estas comunidades y a la idea intrínseca de intercambio de información, estas herramientas de dominio público son especialmente oportunas. En esta línea parece interesante implementar herramientas útiles tanto para el profesor en el desarrollo de su labor docente, esto es, para las prácticas de las asignaturas, como para la distribución de programas a los alumnos que les permitan valorar los conocimientos adquiridos en las aulas. Siguiendo este razonamiento nos encontramos con la necesidad de disponer de una herramienta de simulación gráfica que permita la realización de las prácticas de las asignaturas que imparte el grupo de profesores firmantes. Las asignaturas para las que se diseñó la herramienta que aquí se presenta son: *Fundamentos de Computadores* (FCO), *Estructura de Computadores I* (EC1) y *Estructura de Computadores II* (EC2), correspondientes a los tres primeros cuatrimestres de las titulaciones de Ingeniero Técnico en Informática de Sistemas (ITIS) e Ingeniero Técnico en Informática de Gestión (ITIG) de la E. U de Informática , así como en la titulación Ingeniero en Informática (II) de la Facultad de Informática de la Universidad Politécnica de Valencia. Sin embargo, su ámbito de aplicación es mucho mayor ya que puede ser útil en asignaturas similares de otras titulaciones.

Dicho programa se ha realizado siguiendo una metodología de programación orientada a objetos, y cubre las necesidades de las prácticas de las asignaturas citadas. Debido a la complejidad del problema, este requiere ser llevado a cabo por un grupo de trabajo, en este caso formado por cuatro alumnos de Proyecto Fin de Carrera. Esta ha sido principalmente una de las razones por la que se ha elegido la metodología de diseño orientado a objetos ya que permite desglosar el trabajo en módulos independientes aunque íntimamente relacionados y coordinados.

La asignatura de FCO referida anteriormente, centra su estudio principalmente en el diseño lógico (cuyo nivel de abstracción más bajo es el concepto de puerta) y en el estudio de lenguaje ensamblador. Para este último ya se dispone de una herramienta de dominio público de simulación del ensamblador del procesador R2000 de MIPS. Las prácticas correspondientes al diseño lógico se realizan actualmente mediante un entrenador lógico, ya que se considera muy interesante que el alumno maneje también cables, chips, visualizadores, etc. y no se limite a un entorno de simulación. Por ello, para esta materia el programa sólo se utilizará como una herramienta complementaria que sirva al alumno para ampliar su estudio o realizar ejercicios en casa.

En las asignaturas de EC1 y EC2 se estudia la estructura del computador y las unidades funcionales que la integran, en EC1 se aborda la parte correspondiente a la Unidad Central de Proceso, o Procesador (Ruta de datos, unidad de control, unidad aritmética y lógica), y en la asignatura EC2 la Memoria y la unidad de Entrada/Salida. Las prácticas relativas a EC1, se basan principalmente en el diseño de una Unidad Central de Proceso (UCP) sencilla, utilizando los elementos lógicos estudiados en la asignatura de FCO. Asimismo, en la asignatura de EC2 las prácticas abordan la realización de circuitos que permitan dotar a la UCP ejemplo desarrollada en EC1, de una unidad de memoria y de unidades de entrada/salida, además de alguna práctica encaminada a la utilización de la entrada/salida de un procesador real, como un PC. Para la mayoría de estas experiencias y para trabajos complementarios que se puedan plantear, es necesario disponer de una herramienta software adecuada. Con los nuevos planes de estudios se plantea una reducción bastante considerable del tiempo que el alumno dedica a la universidad, compensado por el trabajo que este tiene que hacer en casa. La realización de dicho trabajo, requiere que el alumno disponga de herramientas que le ayuden en el

estudio de una forma creativa en casa, esto es, que le ayuden a realizar ejercicios y autoevaluar el resultado de los mismos. El alumno es el usuario final del software y es esencial que pueda disponer de él de forma gratuita; en el caso de los profesores, ya que cada universidad tiene, sobre todo hoy, con los nuevos planes de estudio unos planteamientos diferentes, es bueno que los programas permitan modificaciones de una forma sencilla para adaptarlos a sus necesidades.

Otra cuestión importante es la necesidad que tienen los alumnos de realizar proyectos final de carrera para obtener su título. Muchas veces, debido a la carga de los profesores en su trabajo docente e investigador, a los alumnos les resulta difícil encontrar proyectos final de carrera no directamente relacionados con la investigación de los propios profesores, dejando un poco de lado aquellos proyectos más usuales en otras ingenierías que requieran la realización de productos bien acabados y útiles. En esta línea parece interesante encaminar este esfuerzo a dotar de herramientas útiles para la docencia y la investigación y que, teniendo en cuenta el entorno en el que se desarrollan, sean de dominio público.

2. Descripción del programa

El programa se ha planteado, como se ha comentado anteriormente, siguiendo una metodología de programación orientada a objetos. Debido a la complejidad del mismo esta metodología permite hacer un estudio de cuáles son los elementos que deben componer el programa final y a partir de ahí distribuir el trabajo, en este caso en cuatro trabajos que presentará cada alumno como Proyecto Final de Carrera. A la hora de plantear el trabajo se consideró interesante dividirlo en varios apartados diferentes:

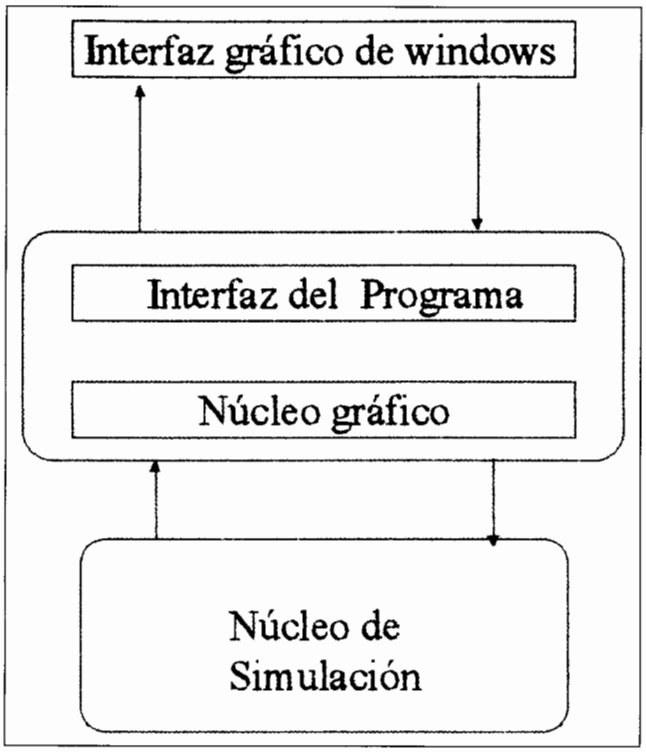


Figura 1. Esquema general de los diferentes módulos desarrollados.

- Una interfaz amigable con el usuario, que permitiera un uso sencillo e intuitivo del programa. Esta interfaz sería de tipo ventanas y menús desplegables con una ayuda en línea de los elementos disponibles.
- Un elemento que permitiera el diseño de circuitos; para reducir complejidad se consideró interesante distinguir entre circuitos combinacionales y secuenciales, aunque los dos trabajos debían estar íntimamente relacionados entre sí.
- Un módulo de simulación para los circuitos diseñados.

Al comienzo del trabajo se consideró oportuno abordar todo el conjunto al mismo tiempo, en trabajos diferentes pero coordinados. Teniendo en cuenta todo lo anterior, el trabajo se dividió en cuatro proyectos, cada uno de los cuales se encargó de uno de los módulos siguientes: La interfaz de usuario; Diseño de circuitos combinacionales; Diseño de circuitos secuenciales; Simulación de circuitos.

Otra cuestión importante era decidir qué tipo de programa se deseaba diseñar. Un programa de simulación y diseño profesional con una librería de los circuitos integrados existentes en el mercado como puede ser el **ORCAD,** o un programa educacional orientado a la simulación como el **CASCAD**, programa actualmente en uso en las prácticas consideradas. La elección fue claramente hacia la segunda alternativa, ya que el objetivo principal es, en este caso, el de disponer de una herramienta de simulación que permita a los alumnos realizar las prácticas, pero haciendo un diseño flexible que admita añadir en un futuro librerías actualizables que doten al programa de una entidad suficiente.

Una vez decidido este aspecto se debía definir, teniendo en

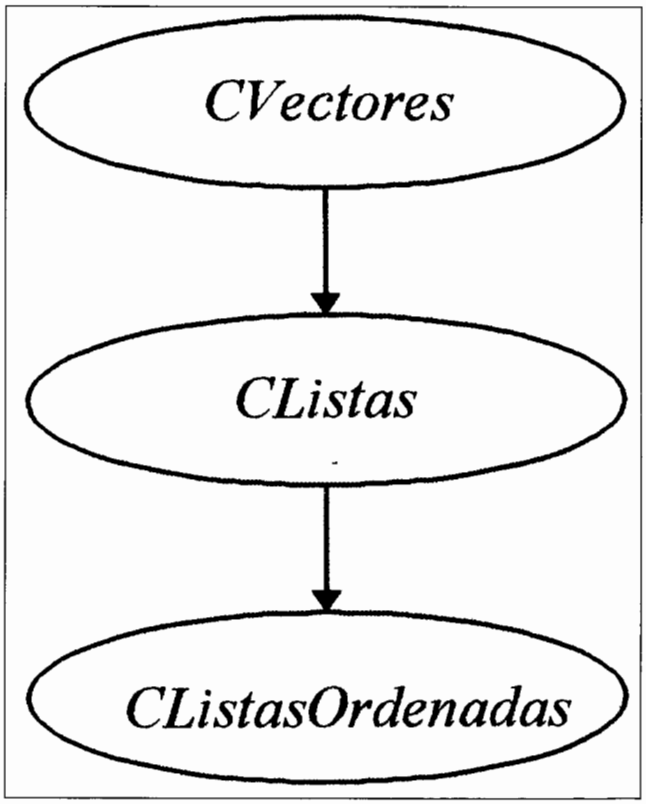


Figura 2 : Estructura de los objetos básicos

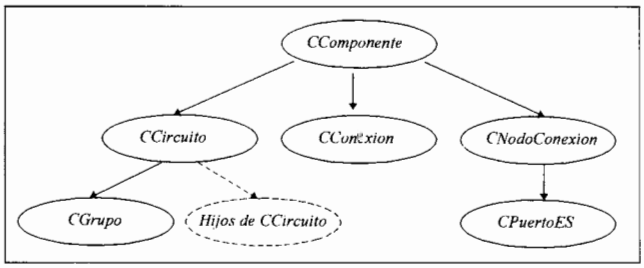


Figura 3: Esquema de los objetos específicos para el diseño de circuitos

cuenta los objetivos de las asignaturas, el tipo de prácticas y de alumno, cuáles serían las características del programa. Por ejemplo: el elemento básico de diseño;las utilidades del diseño como la capacidad de agrupar circuitos y de utilizarlos como elementos para realizar circuitos más complejos o añadirlos a la librería de circuitos definidos; las utilidades de simulación que permitieran al alumno; comprobar el funcionamiento de los circuitos y las utilidades de diseño que permitieran mover y borrar elementos fácilmente, etc.

En suma, todos aquellos elementos que permitieran al alumno realizar como fin último el diseño de un computador ejemplo a partir de elementos sencillos.

Una vez definidos todos estos puntos, había que elegir el lenguaje y el entorno de trabajo del desarrollo del programa. El primero, teniendo en cuenta que la idea era realizar un programa siguiendo la metodología de programación orientada a objetos, debía ser un programa que tuviera esta posibilidad y de uso lo suficientemente extendido que facilitara mejoras posteriores, por lo que se utilizó el C++ para los módulos de diseño de circuitos y simulación. La interfaz gráfica como era una parte estrechamente relacionada con el entorno de trabajo, se ha de considerar junto con éste. Como entorno se decidió utilizar, *Windows 3.x* o *Windows 95* ya que las herramientas de que se dispone en el Departamento de Ingeniería de Sistemas Computadores y Automática para realizar las prácticas son PC's que trabajan con estos entornos. Por lo tanto, la interfaz se debía realizar con programas que facilitaran su uso: *VisualBasic, Visual C++, Delphi*, etc. Se optó por esta última por la facilidad de uso y su definición claramente orientada a objetos.

Con todo esto se planteó el programa siguiendo los pasos descritos a continuación.

En primer lugar había que diseñar *grosso modo* cada uno de los módulos del programa, y definir la interacción existente entre ellos. Para realizar esta tarea se partió de la idea de tener un objeto común a todos los módulos, que era el circuito, con una serie de características como: número de entradas; número de salidas; representación gráfica; tiempo de retardo, etc.

Además debía soportar operaciones tales como: obtener la salida en función de las entradas, trabajo del módulo de simulación, dibujar el circuito con sus entradas y salidas, objetivo de la interfaz, etc. Todo este trabajo se fue refinando en sucesivas reuniones, hasta que el grupo de alumnos fue capaz de trabajar por sí sólo. El resultado se describe a continuación y un **diagrama general** de los diferentes módulos se muestra en el esquema de la **Figura 1**.

A continuación se describirá de forma resumida cada módulo haciendo especial hincapié en los elementos comunes a todos ellos ya que constituyen la base para el resto de los módulos, principalmente los de circuitos combinacionales, secuenciales y de simulación.

Base Común

Para el diseño de estos elementos, en primer lugar, se ha definido un conjunto básico divididos en dos grupos:

- ***Objetos básicos***, donde se agrupan aquellos elementos útiles a la hora de desarrollar el resto de objetos pero que no están directamente relacionados con el problema, entre estos los más importantes son los vectores y listas, estructurados como muestra la **Figura 2**. La diferencia entre los tres objetos es básicamente las operaciones que soporta cada uno, así, el objeto *CVectores* es una zona de memoria que se redimensiona dinámicamente para poder almacenar un determinado número de elementos, mientras que a *CListas,* con la misma implementación, se le ha añadido la posibilidad de insertar y eliminar elementos y *CListasOrdenadas* se diferencia en que cuando se inserta un elemento, este se coloca en la posición correspondiente para que la lista resulte ordenada. Sobre estos objetos básicos se construyen algunos objetos como vectores de texto, listas de punteros, de enteros, etc.
- ***Objetos directamente relacionados con el problema***, para el diseño de estos circuitos se ha construido un objeto base denominado *CComponente*, a partir del cual se han añadido una serie de objetos como se muestra en la **Figura 3**. Tomando como base el objeto *CGrupoCircuitos*, este objeto debe tener una lista de todos los *CComponente* que forman el circuito, además *CGrupoCircuitos* por ser hijo de *CCircuito* debe poseer unos puertos de conexión con el exterior (*CPuertoES)*, los cuales se deben mantener mediante una lista. Además cada *CPuertoES* por ser hijo de *CNodoConexion* debe de tener una lista de conexiones (*Cconexion)* para poder saber con quien está conectado. Finalmente, cada *CConexion* debe mantener información sobre entre qué dos nodos *CNodoConexion* se ha realizado la conexión. A partir de esta estructura se construirá cada tipo de circuito. Hay que destacar que a partir de estos objetos y de los anteriores, aparecerán otros nuevos como son: *CListaComponentes*, *CListaPuertosES*, etc.

Se han añadido otros objetos como *CDatosPropiedades*, lista de *CPropiedad* que permiten variar las características de los componentes, es decir, el nombre, el retardo asociado, el número de entradas y salidas del circuito,etc. Con todo esto y algunos objetos más, se constituye la base que ha permitido desarrollar el resto de módulos, donde cada tipo de circuito descenderá de la clase *Ccircuito* **(Figura 3).**

Circuitos combinacionales

Este módulo es el encargado de implementar a partir del

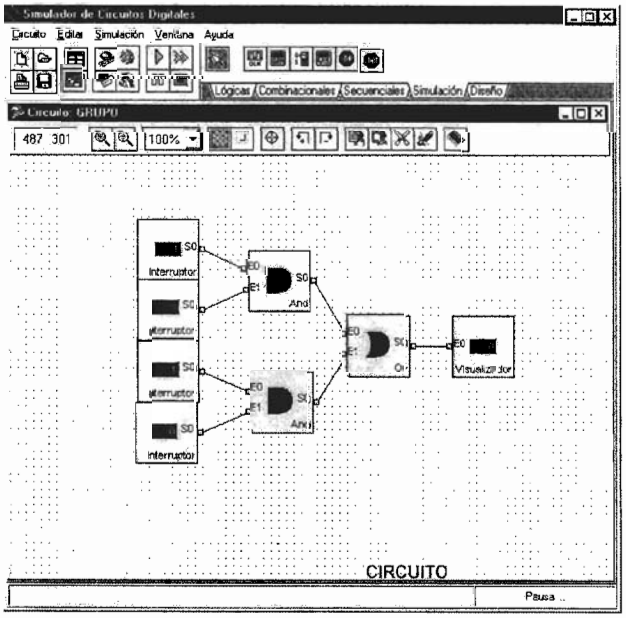


Figura 4. Aquí se muestra un ejemplo de un circuito combinacional que implementa la función, donde a cada entrada del circuito se ha conectado un interruptor que permite cambiar el valor de las entradas, y a la salida un visualizador.

conjunto de clases básico, una serie de elementos útiles para el diseño de circuitos combinacionales como son las puertas lógicas. Además, se han realizado una serie de circuitos básicos, ya predefinidos como son codificadores y decodificadores, multiplexores y demultiplexores, etc. Estos circuitos están disponibles para que el usuario del programa diseñe sus propios circuitos. Sobre los mismos puede definir el valor de las variables tales como el número de entradas y/o salidas, el retardo asociado al circuito etc. Estos elementos están disponibles en las pestañas de: Lógicas y Combinacionales, En la **Figura 4** se muestra un sencillo ejemplo de diseño de circuitos combinacionales.

Una vez diseñado el circuito, este se puede agrupar todo o parte del mismo en una caja con el número de entradas y salidas adecuado a modo de encapsulamiento. Esta opción permite un diseño modular en el caso de circuitos con alto grado de complejidad.

Circuitos secuenciales

De forma similar, se han definido los elementos básicos secuenciales, como los biestables, a partir de los cuales se puede construir cualquier circuito simple como son registros y contadores, útiles para la realización de las unidades funcionales del computador. Así mismo, este módulo dispone también, de manera predefinida, de una unidad de memoria a la que se le puede indicar el tamaño de palabra y el

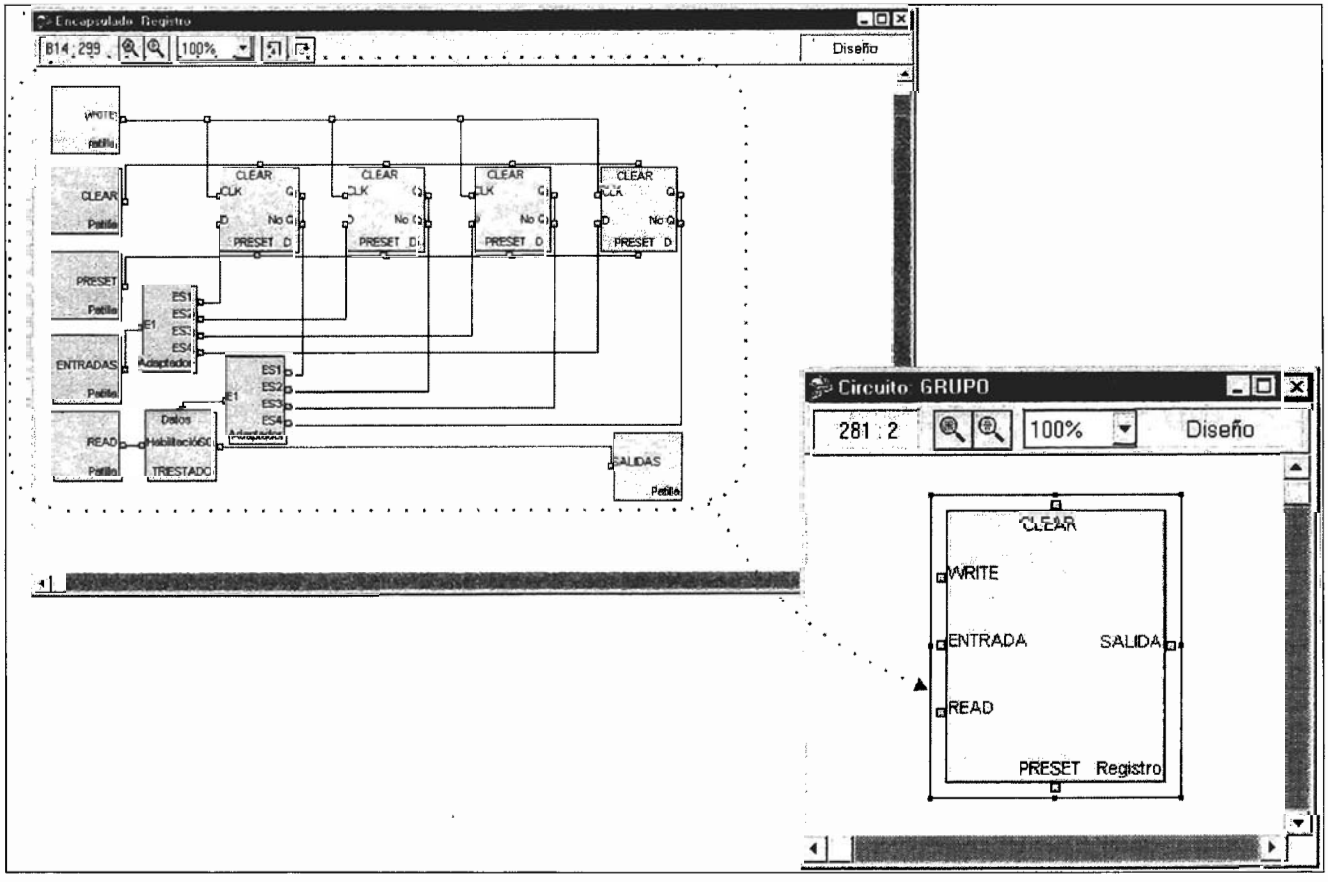


Figura 5. Registro de cuatro bits con entrada paralela. Este registro puede ser utilizado directamente o agrupándolo en una caja con entrada/salida de cuatro bits y dos señales una de lectura y otra de escritura. El grupo se muestra en la parte derecha de la figura.

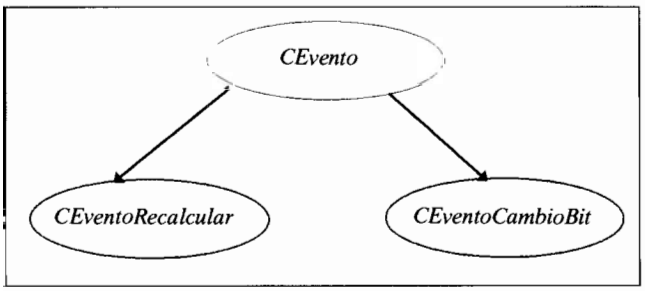


Figura 6. Objetos relacionados con los eventos

de la dirección. En la **Figura 4** se muestra un ejemplo de un registro de cuatro bits.

Como en el caso anterior, este tipo de circuitos también soporta, de forma separada o conjunta, la posibilidad de agrupamientos para su posterior utilización. Todos los circuitos se pueden almacenar y recuperar para la realización de otros más complejos, de esta forma, se le añade la posibilidad de crear una biblioteca de circuitos creados por el propio alumno o de una biblioteca externa de circuitos comerciales que se puede ir añadiendo de forma paulatina.

Módulo de simulación

Es el encargado de implementar todas las funciones que permiten que los circuitos funcionen como corresponde. Para construir el módulo de simulación se toma como base el concepto de evento, que es un suceso que ocurre en un determinado instante, por tanto todos los eventos tienen un valor asociado que los identifica en el tiempo. Los eventos que se producen en este problema son los intercambios de valores a través de las mallas entre los diferentes componentes. Estos intercambios se producen en instantes determinados por el instante de llegada más el tiempo de respuesta de cada componente. Una vez definidos los eventos, estos se

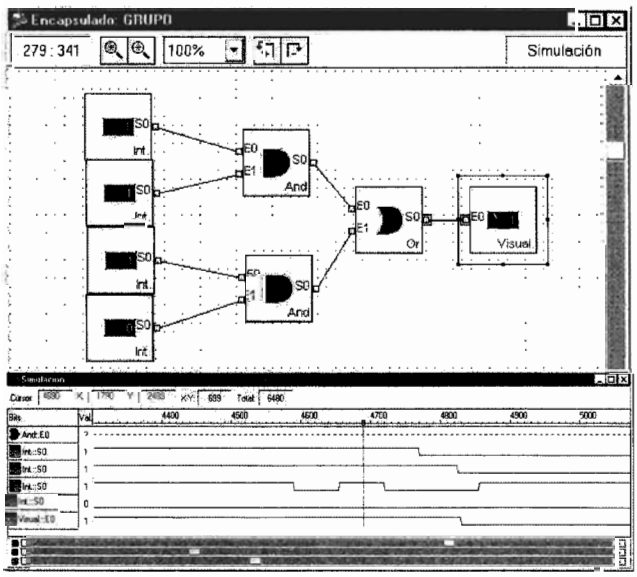


Figura 7. Se muestra una imagen con dos ventanas una con un circuito ejemplo y otra con la simulación del valor de las entradas y salidas

almacenan en una lista ordenada por el factor tiempo de cada evento, de forma que el simulador se limite a obtener los eventos de la lista con el tiempo más pequeño posible y a forzar el intercambio de valores. Esta lista se realimenta con los nuevos eventos que se generan al recalcular las salidas de los componentes afectados por el intercambio de valores entre entradas y salidas. Esto se ha realizado a partir de tres objetos básicos que se muestran en la **Figura 6**.

Para poder tener todos los eventos ordenados en el tiempo se utiliza como base *CListaBaseOrdenada* que a partir de un método virtual *Comparar* puede determinar dónde se insertarán los nuevos eventos. Por tanto, para modelizar el simulador se ha optado por un objeto *CSimulador*, descendiente de *CListaBaseOrdenada*, el cual aporta información sobre el tiempo de simulación actual y donde se planificarán los nuevos eventos. El objeto *CSimulador* dispondrá de métodos que permitan simular uno o varios eventos de la lista.

Para la adquisición de datos durante la simulación se han añadido objetos que permitan construir un cronograma, *CCronograma*, además se han añadido generadores y visualizadores de señales, así como elementos de control, como son el inicio y la parada del simulador. Alguno de estos elementos se muestran en la **Figura 7**, donde se observa un circuito y un cronograma de funcionamiento del mismo.

Esta ventana de simulación dispone además de interruptores, relacionados con los elementos de control comentados, que permiten detener la simulación o cambiar el valor de las entradas. El tiempo transcurre utilizando el reloj del sistema, pero se puede definir una entrada de reloj en la que se puede establecer la sincronización y el tamaño de los diferentes pulsos. Esta parte es muy útil para el alumno a la hora de evaluar el funcionamiento del circuito, no hay que olvidar que tanto a las puertas básicas como a los circuitos ya predefinidos les pueden ser asignados unos retardos equivalentes al retardo real que permitan tener una visión más realista del funcionamiento del circuito, aunque también pueden ser tratados como ideales dándoles un valor de retardo nulo.

Interfaz gráfica

Como ya se ha comentado, este módulo se ha realizado mediante *Delphi* sobre *windows*, con un esquema familiar para el alumno tal y como se muestra en la **Figura 8**. En esta figura se ha activado la pestaña de las puertas lógicas. Si se selecciona la pestaña de circuitos combinacionales, se muestran algunos circuitos básicos como pueden ser decodificadores, codificadores, disponibles ya para su utilización. Sobre todos estos circuitos se puede definir el número de entradas/salidas a utilizar por el circuito. Para cada circuito existe también una hoja de ayuda donde se explica el funcionamiento y las posibilidades de cada tipo de circuito. Asimismo existe una ayuda con globos donde se indica la función de cada ítem de la ventana.

La interfaz está todavía en fase de desarrollo aunque ya se dispone de una versión que ha sido útil para la evaluación del resto de los módulos.

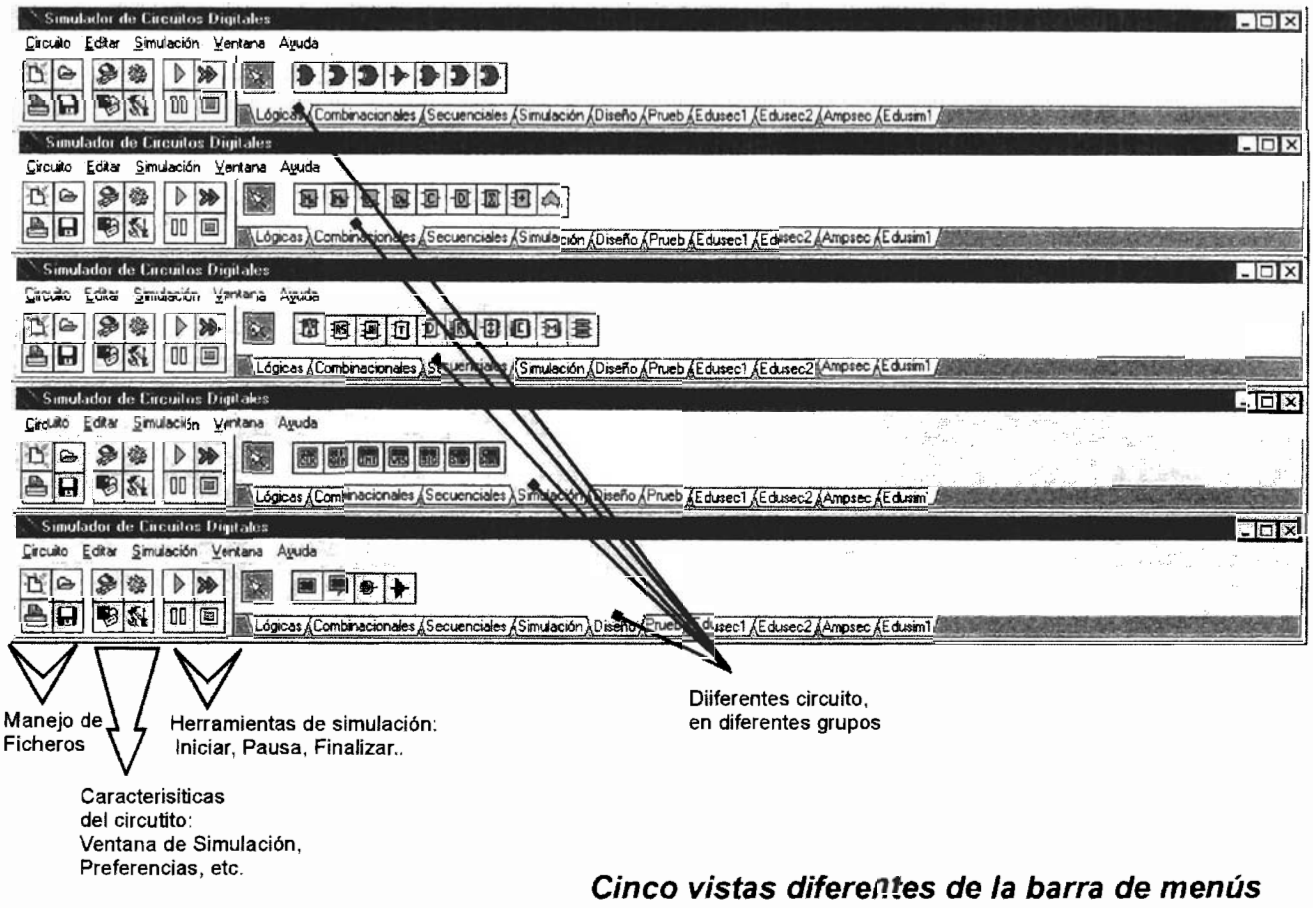


Figura 8. Cinco vistas diferentes de la ventana principal, donde se muestran los principales iconos de las herramientas, y los circuito básicos

3. Conclusiones

En este trabajo se han presentado las experiencias adquiridas en el desarrollo de una herramienta de diseño lógico, que permite la realización de las prácticas correspondientes a las asignaturas de Fundamentos y Estructura de Computadores de la Escuela Universitaria de Informática (EUI) y Facultad de Informática (FI) de la Universidad Politécnica de Valencia.

De esta experiencia cabe destacar el hecho de que los alumnos de proyecto final de carrera de la FI que han realizado el programa, han trabajado con planteamientos profesionales en la realización de un programa disponible de forma gratuita para todos los alumnos y profesores que así lo soliciten.

La utilización de la metodología de programación orientada a objetos así como el uso de lenguajes muy extendidos permitirá la ampliación o modificación del mismo fácilmente. Además como trabajos futuros se plantea, entre otros, añadir una librería de circuitos comerciales disponibles, en cada momento en el mercado.

Como se ha comentado anteriormente el programa está prácticamente finalizado, aunque falta por refinar el módulo de la interfaz gráfica, están disponible tanto sus fuentes como el ejecutable en la dirección: *http:\www. ec.upv.es.*

4. Referencias

Logic Works, Interactive Circuit Design Software. Capilano Computing Systems, LTD.

Stroustrup, B. (1993). *El C++ Lenguaje de Programación.* Addison Wesley/Díaz de Santos. Segunda Edición.

Pressman, R. (1995). *Ingeniería del Software. Un enfoque práctico.* McGraw-Hill. Tercera Edición.

Agradecimientos

A los alumnos que han trabajado en el proyecto:

Alejandro Santos Atienza, Antonio Manuel Estruch Ivars, Ignacio Rodríguez Ortí y Pedro José Antequera.

Software Libre

Simuladores de Robo-Fútbol y otro Software Libre en la Inteligencia Artificial

Vicente Matellán, Camino Fernández
Departamento de Informática, Universidad Carlos III de Madrid

vmate@ia.uc3m.es
camino@inf.uc3m.es

***Resumen**: El presente artículo intenta dar una visión de las aportaciones que el software de dominio público puede hacer al campo de la investigación en informática, usando como ejemplo la rama de la Inteligencia Artificial. Para ello se describen las ventajas que aporta el software de libre distribución como filosofía general, a las disciplinas científicas en particular y en especial a las áreas relacionadas con las ciencias de la computación como la Inteligencia Artificial. Para ilustrar algunas de dichas portaciones se utilizan distintos ejemplos, en especial el de los simuladores de libre distribución de las competiciones de robots autónomos inteligentes en el que trabajan los autores.*

1. Introducción

En el campo de la **Inteligencia Artificial**, en especial en el área de los robots autónomos, existen desde hace años diversas competiciones que tienen como objeto probar la eficiencia de las distintas técnicas de resolución de problemas. Una de las competiciones más llamativas que existen actualmente es la de fútbol entre robots, denominada **RoboCup** [1]. RoboCup es el nombre coloquial de la ***World Cup Robot Soccer Initiative***, un intento de promover la investigación en el área de los robots autónomos cooperantes [2].

En paralelo a la competición con robots reales se celebrara una competición basada en un simulador (**figura 1)** de libre distribución [3]. Esta es nuestra herramienta habitual de trabajo (sobre una plataforma basada totalmente en software de libre distribución) y será, por tanto, el ejemplo que usaremos más frecuentemente en el resto de este artículo para introducir las ventajas, que a nuestro juicio, aporta la filosofía del software de libre distribución a este campo.

En primer lugar parece lógico que si el objetivo es comparar las diferentes arquitecturas, se disponga de una plataforma común que permita la comparación. No seria razonable que cada uno probase su sistema en su propio simulador. Por otra parte, si el simulador es único, se tiene que garantizar que todos los participantes tengan igualdad de oportunidades en la competición, es decir, que no haya nadie que, conociendo la estructura del simulador, se aproveche de esa información privilegiada para diseñar su arquitectura.

Una solución podría haber sido garantizar bajo palabra, por escrito u otro método tradicional que el código del simulador no sería conocido por ningún participante. Este tipo de soluciones es habitual en muchos otros órdenes de la vida, como los exámenes, concursos públicos, subastas, etc.

Sin embargo, para este caso los organizadores han decidido que existe una solución mejor: hacer publico el código del simulador que se empleará. De esta forma se garantiza perfectamente la igualdad de oportunidades y se ayuda a la difusión de la propia competición y del conomiento.

Al permitir que el simulador se distribuya (copie) libremente, se facilita enormente la difusión de la competición. Si alguien tiene el simulador se lo puede copiar, con toda tranquilidad, a sus compañeros. Tanto es así que se puede conseguir este simulador mediante *ftp anónimo* por Internet o a través de la correspondiente página *HTML*.

Desde otro punto de vista, se puede decir que con esta política de distribución también se está contribuyendo a difundir el conocimiento, en este caso sobre cómo construir simuladores, a todo el mundo, pues los organizadores facilitan el código. De esa forma cualquier interesado puede leerlo y obtener el conocimiento que busca.

Este ejemplo puede hacer ver que no siempre es conveniente proteger el software con las tradicionales licencias, entregando solamente el código objeto y guardando el fuente como un tesoro y que esto es incluso cierto en situaciones como las competiciones o la ciencia. La pregunta es si esta idea de distribuir los programas con su código, permitiendo además que los que lo adquieran lo redistribuyan libremente (siempre manteniendo los mismos derechos para los siguientes usuarios), como establece, por ejemplo, la licencia de GNU [4], es siempre beneficiosa. Para intentar responderla, en la siguiente sección se analizaran con más detalle los motivos que pueden llevar a considerarlo así.

2. Motivaciones

El caso de la RoboCup no deja de ser un caso particular, que no tiene por que ser aplicable en todas las ramas de la informática. Por ello conviene analizar que motivos hacen recomendable el software de libre distribución en los ambientes académicos y de investigación, y cuales desaconsejan su uso.

En primer lugar, hay que tener en cuenta que en gran parte de las actividades humanas, la competitividad parece ser un catalizador eficiente a la hora de aumentar la productividad de las mismas. La ciencia no ha sido, ni es, ajena a este principio. Así, a lo largo de su historia se pueden encontrar multitud de ejemplos de trabajos paralelos y enormemente competitivos entre científicos que perseguían los mismos objetivos, bien con fines militares, comerciales o de prestigio personal. Dentro de esas carreras una de las armas tradicionales ha sido el secreto de los trabajos que cada investigador estaba realizando, la discreción sobre las herramientas empleadas y la protección de los resultados obtenidos.

Así mismo, el catalizador de la competencia parece no tener rival a la hora de mejorar el rendimiento en el campo de la investigación y de igual forma las medidas tradicionales de protección de los trabajos de investigación parecen ser estrictamente necesarios para garantizar la rentabilidad de esa competencia.

El mundo de la informática tampoco es ajeno a esta situación. Así, multitud de grupos trabajan actualmente sobre los mismos temas compitiendo, bien por las mismas subvenciones, bien por las mismas cuotas de mercado o de prestigio. De igual forma, las mismas teorías sobre la protección de los resultados mediante licencias y de los métodos empleados (el código fuente de los programas) se ha generalizado.

Entonces, ¿cual es la ventaja que aporta el software de libre distribución? ¿No sería más lógico ocultar a los demás el código escrito? ¿No dar ventajas a los posibles competidores, ni pistas sobre las líneas de trabajo? Para muchos desde luego es cierto, pues no permiten que los demás conozcan el código de sus programas, ni permiten que se copien o que se regalen a otro usuario al adquirir una nueva versión, etc.

En el bando opuesto, el colectivo de desarrolladores de software de libre distribución consideran que ese no es el camino adecuado para el caso del software. Para argumentar su opinión se pueden esgrimir variados argumentos, desde los más filosóficos a los más técnicos. Un buen resumen de estos principios, así como una critica a la forma cómo las compañias de software están tratando la copia del software se puede encontrar en [5]. Además de las razones esgrimidas en ese artículo, que se puede calificar como manifiesto, en el caso del software de investigación se podrían considerar otros aspectos. Como ejemplo citaremos algunos.

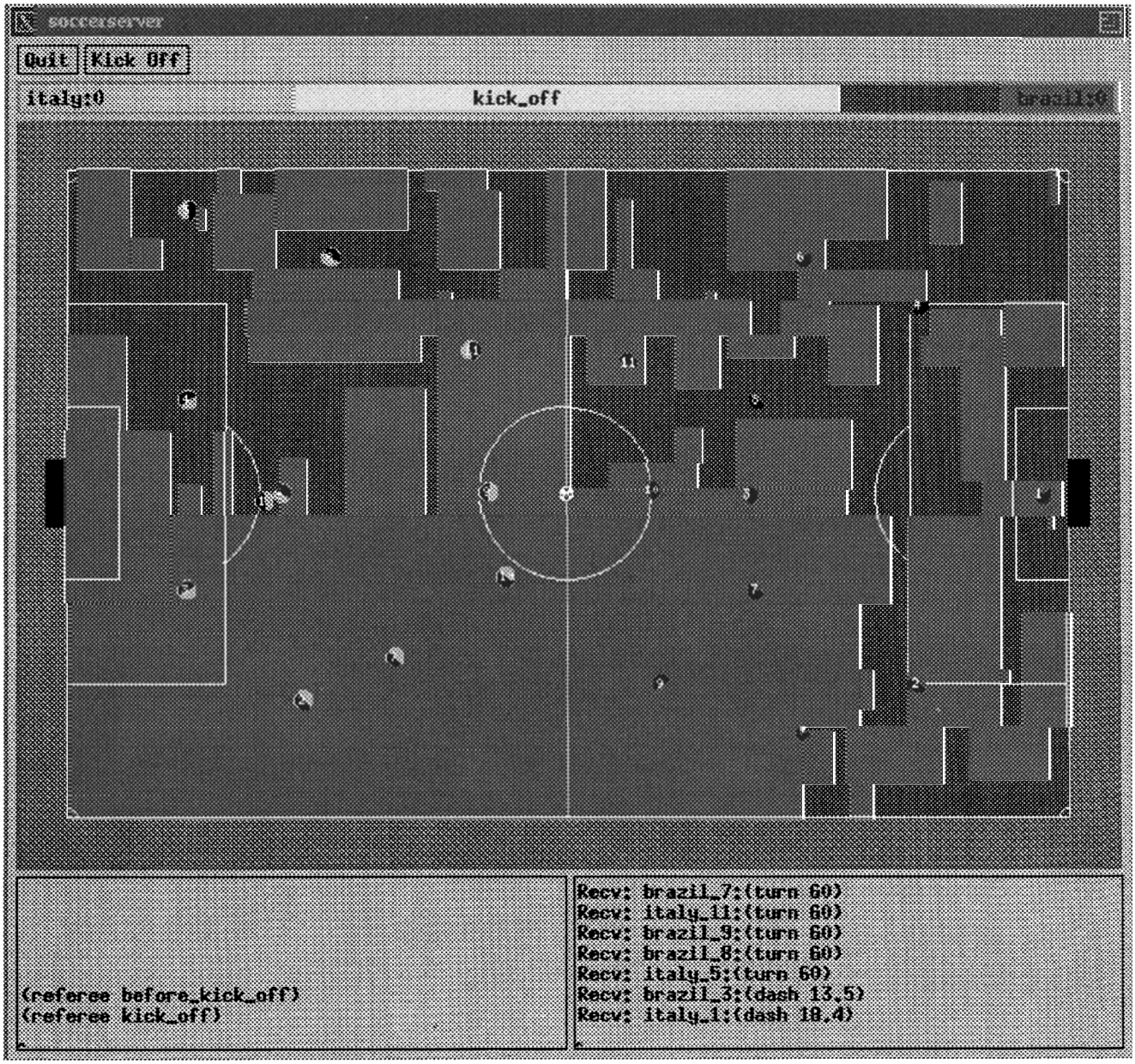


Figura 1: El simulador de la Robo Cup

2.1. El aspecto social

Un aspecto que debería tener en cuenta cualquier investigador es el alcance social de su trabajo. Los desarrollos que se realizan en los ambitos académicos, de investigación y en general de innovación, tienen influencia en la vida de todos los ciudadanos, por lo que la información sobre sus trabajos, descubrimientos y aplicaciones debería ser conocida. Por tanto, los avances científicos deberían mostrar cómo se han obtenido y que necesitan para mantenerse todos sus trabajos.

De igual forma, los programas deberían poder ser inspeccionados, los usuarios deberían saber que está haciendo realmente el software que están utilizando. Alguien podría diseñar, por ejemplo, editores de texto que enviasen copias (sin avisar al usuario) de todo lo que se teclea a determinados lugares donde pudiera ser investigado. Este ejemplo que parece utópico no lo es tanto si consideramos que el gobierno americano intentó instalar un chip en todos los ordenadores que le permitiera intervenir cualquier comunicación electrónica.

Además, hay que considerar que los resultados de la investigación deberían contribuir al desarrollo de la humanidad, mejorando el nivel de vida de las regiones más desfavorecidas y acercándolo al de las más avanzadas. En el caso del software, al permitir que cualquiera pueda acceder al software desarrollado en los lugares más avanzados se está facilitando que los menos desarrollados avancen con mayor rapidez. Este es un arma de doble filo, pues muchos opinan que esta idea favorece a los, en teoría, competidores. No suele ser cierto. Llevándolo a sus limites, poco podrían hacer en Somalia con el software de los transbordadores espaciales, aunque quizá si les resultase útil algún software de gestión. El problema sería que Japón se aprovechara de los desarrollos americanos, pero eso es también relativamente falso, como se comenta en los siguientes apartados.

2.2. Las ventajas individuales

Siguiendo con la idea anterior, hay que ser realistas: la mayor parte de la gente (investigadores incluidos) no son idealistas, buscan su desarrollo personal y como mucho el de su grupo, empresa, nación o en general, organización. Parece ser un sentimiento humano generalizado. Por tanto, no verán con buenos ojos que otros prosperen a costa de su esfuerzo, sin verse ellos recompensados o reconocidos.

En contra de lo que muchos pueden pensar, el software de dominio publico también puede ser positivo desde ese punto de vista. ¿Que mejor publicidad puede tener un programador que decir que su software se utiliza en todo el mundo? ¿Alguien duda de la calidad de un programa como emacs? Además, es bastante evidente que los productos de dominio publico hacen bastante más conocidos a los desarrolladores que los productos de las casas comerciales.

En el campo de la investigación esto es todavía más cierto. De hecho, en todos los campos de la ciencia existen multitud de revistas que se dedican precisamente a ello: hacer público el resultado de los trabajos de los científicos. En el caso del software es aún más sencillo, pues los desarrolladores pueden generalmente enviar sus resultados (sus programas) por medios electrónicos (Internet) a cualquier colega interesado en probarlos, lo que no es tan fácil en el caso de otras ciencias (como la biología o la química).

Desde otro punto de vista, el de los preocupados por garantizar la autoría de los trabajos científicos, está claro que la mejor forma (ya desde hace siglos) de garantizar la autoría, la primera realización, de un trabajo científico (de un programa en la mayoría de los casos informaticos) es publicándolo en alguna revista del campo. ¿Que mejor publicación, en el caso informático, que la publicación al mundo entero a través de la Internet? ¿Que mejor garantía para un científico que el hecho de que existan otros colegas que prueben haber recibido el trabajo (software) directamente de su creador antes que otro pueda afirmar haberlo realizado él?

Pero aún hay más. Muchas empresas invierten fortunas en publicidad tratando de convencer a los usuarios de las bondades de sus productos. Utilizando para ello la técnica del consejo: unos usuarios se los recomiendan a otros que confían en ellos. Del mismo modo ¿que garantía es mejor que recibir una copia, idéntica, del producto que esta usando el recomendador? Eso es lo que se consigue mediante el software de libre distribución: que los usuarios recomienden con libertad, sin tener que autojustificarse, los productos que les parezcan adecuados.

El mismo mecanismo es aplicable al mundo de la investigación, en particular a la informática y dentro de ella a la Inteligencia Artificial. La mejor forma de mostrar que el sistema que se ha desarrollado resuelve los problemas que alguien afirma que resuelve y de explicar cómo lo hace es, a la vez que contarlo literariamente, hacerlo mediante el código que implementa esa idea. Pudiéndose verificar si la solución realmente funciona, si es escalable para otros problemas, etc.

Por ultimo, no se puede dejar de citar el enorme *feed-back* que en el caso del software de libre distribución se obtiene de los usuarios. En general, los mejores depuradores de un programa son los usuarios del mismo. Además, en el caso de usuarios avezados, como suele ser el caso de los investigadores en Inteligencia Artificial, si disponen del código pueden hacer sugerencias, comunicar mejoras, etc. al creador del software. De hecho, para los paquetes que se califican como de libre distribución se suele crear una lista de correo que facilite la comunicación entre los usuarios de ese software. Por ejemplo, en el caso de la RoboCup existe la lista *RoboCup@csl.sony.co.jp*, a la que cualquiera puede suscribirse, en la que se anuncian las nuevas versiones, se comentan los errores detectados o se hacen consideraciones sobre las nuevas mejoras o modificaciones que se deberían incorporar al simulador.

2.3. El realismo práctico

Los que aún tengan reparos en hacer público su software, pueden asumirlo pensando que la mayor parte de los progra-

madores que dejan su software libre lo hacen cuando consideran que el producto está terminado, lo cual generalmente quiere decir que ya están trabajando en otro tema, o en otra versión.

Traducido al campo de la investigación quiere decir que no están regalando su último trabajo. Los investigadores, en general, hacen público lo que ya tienen terminado, no aquello en lo que están trabajando. De esta forma, cualquiera tiene todo el derecho a fijar el ritmo con el que va dejando al resto acceder a su información.

Este es el caso de los autores que deciden depositar sus aplicaciones en el repositorio de inteligencia artificial de la Universidad Carnegie Mellon. El software contenido en este lugar se distribuye bajo las condiciones de libre redistribución, libre uso y uso bajo la propia responsabilidad, más otras condiciones que impongan los propios autores siempre que no contradigan las anteriores, como GNU [4] o restricciones a su uso comercial, militar, etc.

3. Conclusiones

En las secciones anteriores se ha tratado de explicar que entienden los autores como ventajas del software de libre distribución en el campo de la investigación en general y en particular en la Inteligencia Artificial. Además de las justificaciones genéricas se han comentado casos particulares, como el de la RoboCup, donde se pueden apreciar muchas de estas ventajas. En cualquier caso, la única conclusión que se debería sacar de este artículo es que existe una forma alternativa de trabajar en el mundo del software, incluido el software de investigación, al *copyright* tradicional. La elección de la filosofía de trabajo debe ser personal, pero tomada desde la reflexión y el conocimiento.

Por nuestra parte, hemos aceptado desarrollar software de libre distribución, siempre que sea posible. De esta forma haremos públicas aquellas herramientas que creamos que pueden interesar a alguien más. Como ejemplo, sirva citar el simulador distribuido de robots que estamos desarrollando en el **Laboratorio de Agentes Inteligentes** [6], que esperamos dejar como de libre distribución tan pronto como esté terminado. Esa creemos que es la forma de progresar y de hacerlo además en la dirección correcta, gracias a las opiniones de los demás.

4. Direcciones de interés

Como ya se ha comentado, el software de libre distribución tiene, entre otras ventajas, la facilidad para obtener todo tipo de programas a través de Internet, incluso de forma anónima al no existir ningún problema de distribución. A continuación se encuentran las direcciones del software, por ejemplo sobre simuladores de fútbol entre robots, mencionados en este trabajo, alguna sobre software de dominio publico para la inteligencia artificial y sobre el software libre en general:

http://www.robocup.org/RoboCup/RoboCup.html: página oficial de laRoboCup. En ella se pueden obtener las normas de la competición y el simulador.

http://wwwi3s.unice.fr/om: Olivier Mitchel organiza un concurso basado en el mini-robot Khepera (información de este robot comercial se puede encontrar en: ***htp://www.epfl.ch/***). A través de la página de Olivier se puede obtener la última versión de su simulador.
http://www.mirosot.org/MIROSOT96/index.html: Información sobre los encuentros previos de la RoboCup.
http://forum.swarthmore.edu/jay/learn-game/projects/microb.html: Otra versión del fútbol entre robots: MICROB. En este caso es una versión francesa, donde también han desarrollado su propio simulador.
http://www.cs.ubc.ca/nest/lci/soccer: EL LCI Dynamite Testbed. Uno de los primeros entornos, en este caso canadiense, para la competición entre robots [7]. También tienen su propio simulador.
ftp://irisa.irisa.fr/pub/gnu/: Dirección europea donde se puede conseguir software de libre distribución de la *Free Software Foundation*, como emacs, gcc, etc.
http://www.inf.uc3m.es: Página HTML del Departamento de Informática de la Universidad Carlos III de Madrid, donde se puede obtener más información sobre los autores.
http://www.ptf.com/ptf/products/AI: Repositorio de Inteligencia Artificial de la Universidad Carnegie Mellon. Contiene implementaciones de lenguajes, software diverso, tesis, libros, anuncios de conferencias, etc.

5. Referencias

[1] H. Kitano, M. Asada, Y. Kuniyoshi, I.Noda, and E. Osawa. Robocup: The robot world cup initiative. In *Proceedings of the IJCAI-95 Workshop on Entertainment and AI/Life*, pages 19-24, 1995.

[2] Third call for participation. Micro-robot world cup soccer tournament, 1996.

[3] Itsuki Noda. Soccer server: A simulator of robocup. In *Proceedings of AI Symposium'95*. Japanese Society for Artificial Intelligence, December 1995.

[4] Free Software Foundation. Gnu general public license, 1991.

[5] Richard Stallman. Why software should not have owners. In *Comunicaciones del Primer Taller sobre el Software de Libre Distribución*, pages 13-17, Septiembre 1995.

[6] Lorenzo Sommaruga, Ignacio Merino, Vicente Matellán, and Manuel Molina. A distributed simulator for intelligent autonomous robots. In *Proceedings of the Fourth International Symposium on Intelligent Robotic Systems*, Lisbon, (Portugal), 1996.

[7] Michael K. Sahota, Alan K. Mackworth, Rod A. Barman, and Stewart J. Kingdon. Real-time control of soccer-playing robots using off-board vision: the dynamite testbed. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 3690-3663, 1995.