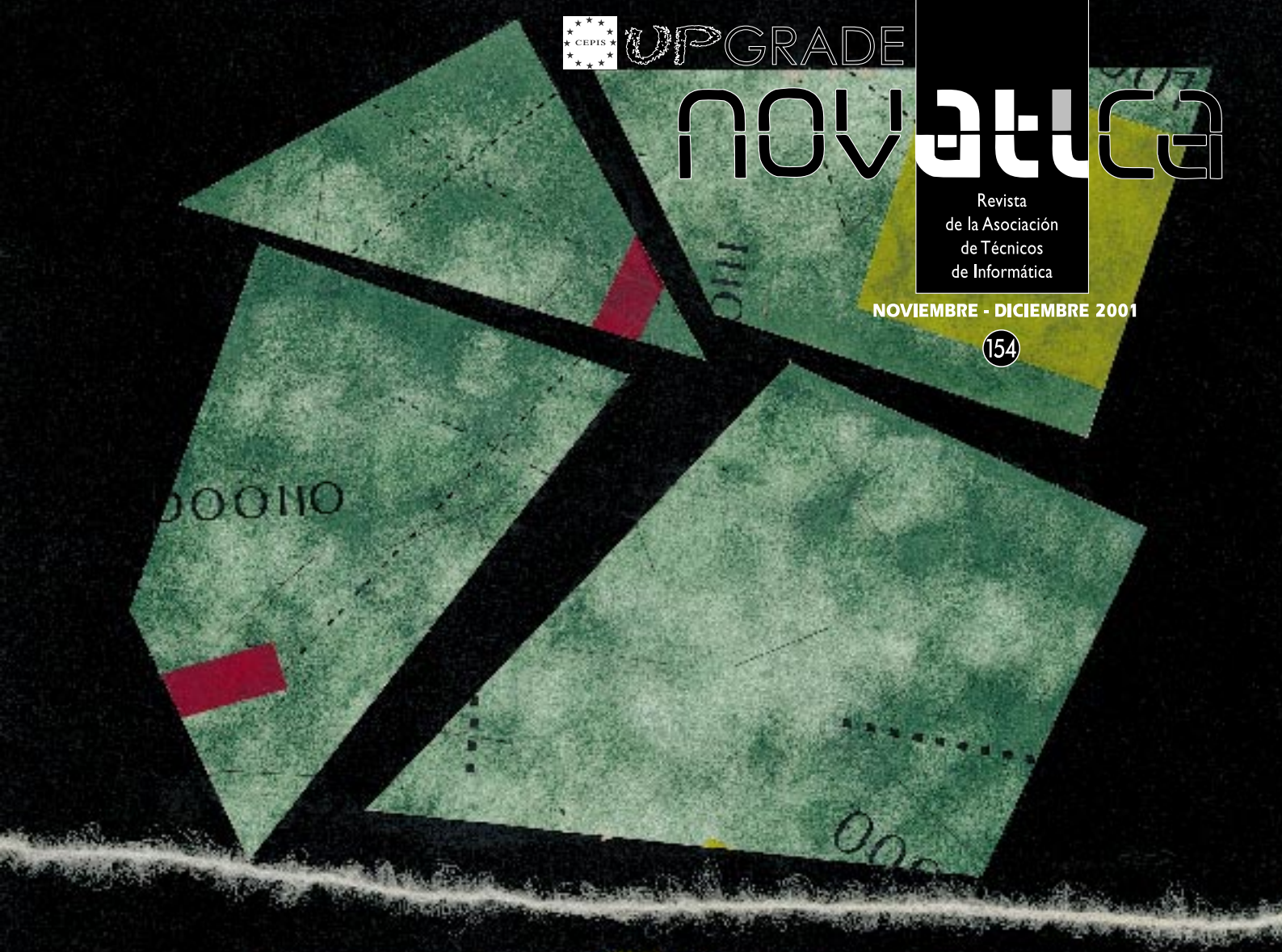
CEPIS UPGRADE
novática
Revista
de la Asociación
de Técnicos
de Informática
NOVIEMBRE - DICIEMBRE 2001
154



Software Libre/Fuente Abierta: hacia la madurez

UPGRADE
novática
Revista de la Asociación de Técnicos de Informática
NOVIEMBRE - DICIEMBRE 2001
154

Novática, revista fundada en 1975, es el órgano oficial de expresión y formación continua de ATI (Asociación de Técnicos de Informática)

ATI es miembro de CEPIS (*Council of European Professional Informatics Societies*) y tiene un acuerdo de colaboración con ACM (*Association for Computing Machinery*). Tiene asimismo acuerdos de vinculación o colaboración con AdaSpain, AI² y ASTIC

<http://www.ati.es/novatica/>

Coordinación Editorial y Redacción Central (ATI Madrid)
Padilla 66, 3º, dcha., 28006 Madrid
Tlf.914029391; fax.913093685 <*novatica@ati.es*>
Composición, Edición y Redacción ATI Valencia
Palomino 14, 2ª, 46003 Valencia
Tlf./fax 963918531 <*secreval@ati.es*>
Administración, Subscripciones y Redacción ATI Cataluña
Via Laietana 41, 1º, 1ª, 08003 Barcelona
Tlf.934125235; fax 934127713 <*secregen@ati.es*>
Redacción ATI Andalucía
Isaac Newton, s/n, Ed. Sadiel, Isla Cartuja 41092 Sevilla
Tlf./fax 954460779 <*secreand@ati.es*>
Redacción ATI Aragón
Lagasca 9, 3-B, 50006 Zaragoza
Tlf./fax 976235181 <*secreara@ati.es*>
Redacción ATI Asturias-Cantabria <*gp-astucant@ati.es*>
Redacción ATI Castilla-La Mancha <*gp-clmancha@ati.es*>
Redacción ATI Galicia
Recinto Ferial s/n, 36540 Silleda (Pontevedra)
Tlf.986581413; fax 986580162 <*secregal@ati.es*>

Publicidad: Padilla 66, 3º, dcha., 28006 Madrid
Tlf.914029391; fax.913093685 <*novatica.publicidad@ati.es*>

Imprenta: Gráficas Sierra S.L., Atenas, 3, int. bajos, 08006 Barcelona.
Depósito Legal: B 15.154-1975
ISBN: 0211-2124; CODEN NOVAEC

Portada: Antonio Crespo Foix / © ATI 2001

SUMARIO

En resumen: **Libertad y madurez** 2

Monografía: *«Software Libre/Fuente Abierta: hacia la madurez»*
(En colaboración con ***Informatik/Informatique*** y ***Upgrade***)
Coordinada por *Joe Ammann, Jesús M. González-Barahona y Pedro de las Heras Quirós*

Presentación: hacia la madurez 3
Joe Ammann, Jesús M. González-Barahona, Pedro de las Heras Quirós

Actualidad del software libre 5
Pedro de las Heras Quirós, Jesús M. González-Barahona

El daño viene de La Haya 14
Richard Stallman

Iniciativas europeas sobre el uso de software libre en el Sector Público 17
Juan Jesús Muñoz Esteban

***Open Source* en un gran banco suizo** 22
Klaus Bucka-Lassen, Jan Sorensen

El proyecto GNU Enterprise: software de aplicación para la empresa 25
Neil Tiffin, Reinhard Müller

Contando patatas: el tamaño de Debian 2.2 30
Jesús M. González-Barahona, Miguel A. Ortuño, Pedro de las Heras, José Centeno, Vicente Matellán

La crisis del software libre científico 38
David Santo Orcero

El proyecto Debian GNU/Linux 41
Javier Fernández-Sanguino Peña

Sistemas de ficheros con *Journaling* en Linux 45
Ricardo Galli

Secciones técnicas

Ingeniería del Software
Un nuevo modelo de evaluación de procesos de software para PYMEs a partir de SPICE (ISO/IEC TR-15504-5) 52
Antonia Mas Pichaco, Ángel Igelmo Ganzo, Esperança Amengual Alcover, Gabriel Fontanet Nadal

Profesión informática
El futuro de la Ingeniería del Software 57
Karol Frühauf

Seguridad
De mí misma líbreme Dios, que del Sircam ya me libro yo (y II) 59
Mª del Carmen Ugarte García

Tecnología de Objetos
¿Es conveniente la Orientación a Objetos en un primer curso de programación? 64
Jesús J. García Molina

Referencias autorizadas 69

Sociedad de la Información

Programar es crear
Ancho de banda en Internet 72
Concurso de Programación ACM 2000: programa E
«Fila y asociados»: solución 73
Álvaro Martínez Echevarría

Asuntos Interiores

Coordinación editorial / Programación de Novática 76
Normas de publicación para autores / Socios Institucionales 77

Monografía del próximo número: «***Gestión del Conocimiento y TIC***»

En resumen

Libertad y madurez

Rafael Fernández Calvo
Coordinación Editorial de Novática
<rfcalvo@ati.es>

El mercado del software, cuyos productos son puro conocimiento y constituyen por tanto un fruto paradigmático de la llamada Sociedad de la Información (o del Conocimiento), sufrió una tremenda revolución con el advenimiento de la Informática (o Computación) Personal a principios de los años 80 del pasado siglo y se vio ulteriormente conmocionado por la popularización de Internet a mediados de los 90. Desaparecieron empresas desarrolladoras de programas que parecían de enorme solidez y aparecieron otras que en pocos años se hicieron con cuotas consistentes de dicho mercado. Una de ellas, Microsoft, se destacó y en pocos años, por méritos propios y por errores de los demás, se convirtió en la dominadora casi en exclusiva del sustancioso campo de los sistemas operativos y las aplicaciones para ordenadores personales, manteniendo hasta nuestros días esta situación de preeminencia.

Curiosamente la respuesta a dicha situación no ha venido de ninguna empresa competidora sino de una galaxia difusa: el mundo del **software libre** (frente al habitual **software propietario**), que, nacido en ambientes universitarios norteamericanos más bien contestatarios a principios de los años ochenta, era considerado hasta hace no mucho poco menos que como una extravagancia radical. Aunque por fortuna siga manteniendo en su base la frescura contestataria de sus orígenes, este movimiento se ha visto apoyado en los últimos años por las empresas competidoras de Microsoft y por algunos Gobiernos que se declaran temerosos de la dependencia de un solo suministrador, y ha dado lugar incluso a un sector especifico de empresas de software libre y **fuente abierta** (*open source*) basadas en la comercialización empaquetada de sistemas y aplicaciones desarrolladas en dicho entorno (especialmente de Linux, Apache, Perl, Sendmail, que son por el momento los productos más destacados del mismo), que está penetrando lentamente en el mundo empresarial.

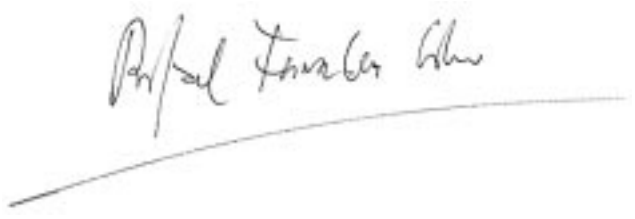
Novática, que siempre ha querido ser fiel observadora de las novedades informáticas y parainformáticas, publicó en 1997 una monografía sobre la materia titulada escuetamente «Software Libre», que fue pionera en los medios especializados españoles (ver el número 126, marzo-abril 1997, disponible en <http://www.ati.es/novatica/1997/126/indice.html>). La monografía, que tuvo una repercusión muy amplia, fue incluso causa de alguna incomprensión hacia nuestra revista nacida de la falta de información que ese momento existía sobre el tema y que llevaba a muchos a confundir software libre con piratería e ilegalidad. Coordinada por ***Jesús M. González Barahona*** y ***Pedro de las Heras Quirós***, era una demostración de que el software libre (que no gratuito, como clamaba en su artículo el padre de este movimiento, ***Richard Stallman***) empezaba a asomar su cabeza tímidamente fuera del útero académico. A raíz de ella se creó en ***Novática*** una sección técnica sobre este tema.

Más de cuatro años después, cuando se ve con claridad que en el mercado, en interés de todos (incluida la empresa hegemónica), hay y debe haber sitio para todos, Jesús y Pedro, acompañados del especialista suizo ***Joe Ammann***, vuelven a ser los editores invitados de una monografía sobre el mismo asunto, publicada esta vez en colaboración con ***Upgrade*** e ***Informatik/Informatique***, que refleja el camino recorrido en este periodo y que por eso mismo se titula «*Software Libre/Fuente Abierta: hacia la madurez*». Ellos, con su autoridad moral y su poder de convocatoria, han reunido a una serie de destacados especialistas, entre los que repite el ya citado Richard Stallman, que cubren distintas facetas de este ya amplio universo. No podemos dejar de expresar nuestro agradecimiento a todos ellos y de reiterarlo a los miembros del Grupo de Lengua e Informática de ATI que se han encargado de nuevo, de forma desinteresada, de la traducción de los artículos foráneos.

Esperamos y deseamos que esta monografía sea para nuestros lectores de la misma utilidad que lo fue la de 1997.

Pero este número navideño de ***Novática*** no se agota en absoluto en la monografía sino que incluye también valiosas aportaciones en sus secciones técnicas de Ingeniería del Software, Profesión informática, Seguridad y Tecnología de Objetos, así como los habituales y enigmáticos ejercicios de programación que nos vienen proponiendo desde hace un año los jóvenes componentes del equipo de participantes españoles en el Concurso Internacional de Programación de ACM del pasado año 2000.

Termina el año 2001 y comienza el 2002, que es capicúa (para los que lo ignoren, palabra catalana compuesta, donde *cap* significa «cabeza» o «comienzo» y *cúa* «cola» o «final»). Que sea para todos de mayor paz, libertad y prosperidad que lo fue el año que ahora acaba.



Presentación: hacia la madurez

Joe Ammann[1], Jesús M. González-Barahona[2], Pedro de las Heras Quirós[2]
[1] /ch/open (Open Systems User Group, Suiza); [2] Universidad Rey Juan Carlos (Madrid)

<joe@pyx.ch>
{<pheras, jgb@gsyc.escet.urjc.es>}

Hace ya varios años, a principios de 1997, ***Novática*** publicó un monográfico sobre software libre. Con ello se adelantó a la mayoría de las revistas dedicadas a la Informática, que sólo comenzaron a incluir este tema en sus contenidos bastante más tarde. Hoy ***Novática***, esta vez en colaboración con ***Informatik/Informatique*** y ***Upgrade***, vuelve a dedicar un número a esta nueva forma de desarrollar programas.

En estos años, desde ese primer monográfico, han cambiado muchas cosas. El mundo del software libre ya no está compuesto fundamentalmente por programadores que escriben programas en sus ratos de ocio. Ahora hay muchas empresas para las que el software libre es parte de su estrategia (como productores o como consumidores), y el público en general se está empezando a interesar por sistemas libres (como GNU/Linux). El software libre ya no es «algo raro» y tanto individuos como organizaciones buscan formas de utilizarlo y beneficiarse de sus ventajas. Hoy día, el software libre ha pasado ya sus años de niñez y está entrando a toda velocidad en su madurez, a la vez que se introduce, como un elemento más, en el complejo entramado social.

Por eso, la visión que hemos tratado de presentar en este número es también compleja, y comprende varios enfoques y puntos de vista. Agrupando los artículos incluidos por temas, ofrecemos al lector los siguientes:

·Panorámica general: «*Situación actual del mundo del software libre*» repasa el estado de los programas, proyectos, empresas e iniciativas más relevantes del mundo del software libre.

· Asuntos legales: «*El daño viene de La Haya*» nos explica los peligros del Tratado de la Haya y cómo éste puede convertirse, si se aplica a asuntos relacionados con la información, en una puerta trasera por la que se nos cuelen leyes que hemos de cumplir aunque no hayamos tenido la posibilidad de votar (por estar promulgadas en otros países). Sus implicaciones sobre asuntos como los derechos de autor o las patentes de software son muy importantes y pueden afectar directamente al status legal del software libre.

· Software libre en el mundo real: «*Iniciativas europeas sobre el uso de software libre en las Administraciones Públicas*» nos detalla la situación en Europa con respecto al uso de software libre en instituciones públicas y describe algunas experiencias interesantes. Por otro lado, «*Open Source en un gran banco suizo*» analiza el uso de software libre en una gran empresa, que puede servir a modo de ejemplo para muchas otras organizaciones similares.

· Artículos técnicos: también se han incluido varios artículos que explican detalles técnicos de programas o proyectos libres. «*El proyecto GNU Enterprise: software de aplicación para la empresa*» y «*El proyecto Debian GNU/Linux*» describen estos dos proyectos, para que el lector pueda hacerse una idea, a modo de ejemplo, de dos estrategias bien diferentes de desarrollo de software libre. «*Sistemas de ficheros transaccionales en Linux*» y «*La crisis del software libre científico*» muestran la panorámica actual del software libre en estos dos campos. «*Contando patatas: el tamaño de Debian 2.2*» ofrece una estimación de la cantidad de software libre incluida en Debian, una de las distribuciones de software libre más extendidas.

Si el lector tiene la paciencia de ojear todos estos artículos, esperamos que pueda hacerse una idea personal de la situación del software libre en la actualidad y disponer de suficien-

Editores invitados

Joe Ammann obtuvo el título de Ingeniería Eléctrica en la ETH de Zürich (Suiza) en 1990. Es miembro de la Junta Directiva de /ch/open (Grupo de Usuarios de Sistemas Abiertos de Suiza), país donde ha estado promoviendo el uso de software de fuente abierta durante los últimos años.

Jesús M. González Barahona es profesor en la Universidad Rey Juan Carlos, y colaborador de BarraPunto.Com. Comenzó a trabajar en la promoción del software libre en 1991, en el grupo PDSOFT (más adelante grupo Sobre). Desde entonces, ha realizado diversas actividades en este área, como la organización de seminarios, la realización de cursos y la participación en grupos de trabajo sobre software libre. Actualmente colabora con varios proyectos de software libre (entre ellos Debian y La Espiral), participa en el Grupo de Trabajo sobre Software Libre promovido por la DG-INFO de la Comisión Europea, colabora con asociaciones como Hispalinux y EuroLinux, escribe en varios medios sobre temas relacionados con software libre, y asesora a empresas en sus estrategias relacionadas con estos temas. Coordinador de la Sección Técnica de Software Libre de Novática.

Pedro de las Heras Quirós es profesor en la Universidad Rey Juan Carlos, y colaborador de BarraPunto.Com. Desde principios de los 90 ha sido usuario de software libre, habiendo colaborado en el grupo Sobre dedicado al Software Libre desde su creación. Ha participado en la organización de congresos, expos y cursos relacionados con software libre, y ha sido editor y autor de varias publicaciones relacionadas con el software libre. Coordinador de la Sección Técnica de Software Libre de Novática.

tes elementos de juicio para decidir si, en su caso particular, esta nueva forma de hacer las cosas puede ayudarle en su trabajo. Con esa esperanza hemos incluido también unas cuantas referencias existentes en la Web, donde sin duda podrá completar la visión que le ofrecemos aquí.

Para terminar, nos gustaría agradecer su colaboración a todos los autores que han preparado sus artículos para este número especial. Sin su contribución desinteresada, no sería posible hacéroslo llegar. Y por supuesto, es preciso agradecer también a François Louis Nicolet y Rafael Fernández Calvo, editores de ***Informatik/Informatique*** y ***Novática***, respectivamente, y coeditores de ***Upgrade***, la oportunidad de haber colaborado con ellos y la confianza que depositaron en nosotros cuando nos encargaron este trabajo.

Referencias en la Red

Definiciones de Software Libre y *Open Source*:
- Normas de Debian sobre qué es software libre: <http://www.debian.org/social_contract>
- Definición de *Open Source*: <http://opensource.org/docs/definition.html>
- ¿Qué es software libre?, según la *Free Software Foundation*: <http://www.fsf.org/philosophy/free-sw.html>

Organizaciones que promueven Software Libre o software de Fuente Abierta:
- *Free Software Foundation:* <http://fsf.org>
- *Open Source Initiative*: <http://opensource.org>

Información sobre algunos programas libres:
- Apache: <http://apache.org>
- Linux: <http://linux.com>
- GNOME: <http://gnome.org>
- KDE: <http://kde.org>
- OpenOffice: <http://openoffice.org>
- Programas de GNU: <http://gnu.org>

Distribuciones de sistemas operativos libres:
- Debian: <http://debian.org>
- RedHat: <http://redhat.com>
- Mandrake: <http://mandrake.com>
- Suse: <http://suse.com>
- FreeBSD: <http://freebsd.org>

Noticias relacionadas con Software Libre:
- Slashdot: <http://slashdot.org>
- Advogato: <http://advogato.org>
- BarraPunto: <http://barrapunto.com>

Software Libre/Fuente Abierta para la empresa

Actualidad del software libre

Pedro de las Heras Quirós, Jesús M. González-Barahona
Universidad Rey Juan Carlos

{<pheras, jgb@gsyc.escet.urjc.es>}

Resumen: *el mundo de software libre es sorprendentemente dinámico. La velocidad a la que se anuncian nuevos desarrollos, nuevas versiones, nuevas empresas, es realmente alta. Casi cualquier evaluación de programas libres de hace un año prácticamente carece de valor, por las constantes mejoras que han experimentado muchos de ellos en este tiempo. Proyectos que hace dos años estaban empezando, y aún prácticamente no habían producido software, están hoy completamente establecidos y sus programas tienen suficiente calidad como para satisfacer los requisitos de muchos usuarios. La postura de muchas grandes empresas con respecto al software libre está variando cada pocos meses. Nuevas empresas con pocos años de existencia se están convirtiendo rápidamente en gigantes que cotizan en bolsa. Por eso, si se quiere tener una idea realista sobre el mundo del software libre, es imprescindible que los datos en los que se base estén muy actualizados. Cualquier impresión basada en datos de hace unos meses es, posiblemente, equivocada. En este artículo pretendemos precisamente mostrar la situación actual, para que el lector pueda estar al tanto y formar su propia opinión.*

Palabras clave: *software libre, fuente abierta, situación actual, novedades.*

1. ¿Qué hay de nuevo?

Los últimos tiempos han sido pródigos en acontecimientos relacionados con el software libre, tanto desde el punto de vista técnico como empresarial. Ya hay un flujo continuo de noticias en los medios de información tradicionales cubriendo casos de uso de software libre o de empresas que se dedican a ese negocio. Varios nichos tecnológicos están siendo conquistados por productos libres. Y cada vez más empresas e instituciones consideran que las soluciones basadas en software libre son las más adecuadas para llevar sus negocios.

En las siguientes secciones trataremos de exponer la situación actual de los proyectos de software libre más interesantes, de las empresas «tradicionales» del sector informático y de las nuevas empresas que se están dedicando directamente al software libre.

2. ¿Qué hacen las empresas «tradicionales»?

Muchas empresas «tradicionales» del sector de la informática están desarrollando una estrategia completa relacionada con el mundo del software libre. En algunos casos están colaborando abiertamente en proyectos libres, en otros están liberando programas que ya tenían desarrollados, en otros incluyen programas libres como parte de las soluciones comerciales que están vendiendo, a veces proporcionan servicios relacionados con el software libre, etc. Algunos casos especialmente interesantes son los siguientes:

· **IBM**, a pesar de seguir teniendo una estrategia tradicional basada en el desarrollo de software propietario, es una de las grandes empresas informáticas que lleva tiempo haciendo movimientos claros en el mundo del software libre. IBM está vendiendo productos basados en software libre (la solución para servidores de Internet que comercializa, por ejemplo, está construida alrededor de Apache, al que IBM también contribuye notablemente). También ha liberado el código de programas como Jikes (un compilador de Java a Java *byte-code*), que está siendo desarrollado como un proyecto de software libre. Ha contribuido a Linux con el **JFS** (*Journaling File System*), liberando también su código. Se ha encargado de portar Linux a sus ordenadores de arquitectura S/390, para el que incluso ofrece programas de soporte. En <http://www.ibm.com/developerworks> puede encontrarse más información sobre la relación de IBM con el software libre.

Autores

Pedro de las Heras Quirós es profesor en la Universidad Rey Juan Carlos y colaborador de BarraPunto.Com. Desde principios de los 90 ha sido usuario de software libre, habiendo colaborado en el grupo Sobre dedicado al Software Libre desde su creación. Ha participado en la organización de congresos, expos y cursos relacionados con software libre, y ha sido editor y autor de varias publicaciones relacionadas con el software libre. Es coordinador de la Sección Técnica de Software Libre de Novática.

Jesús M. González Barahona es profesor en la Universidad Rey Juan Carlos y colaborador de BarraPunto.Com. Comenzó a trabajar en la promoción del software libre en 1991, en el grupo PDSOFT (más adelante grupo Sobre). Desde entonces, ha realizado diversas actividades en este área, como la organización de seminarios, la realización de cursos y la participación en grupos de trabajo sobre software libre. Actualmente colabora con varios proyectos de software libre (entre ellos Debian y La Espiral), participa en el Grupo de Trabajo sobre Software Libre promovido por la DG-INFO de la Comisión Europea, colabora con asociaciones como Hispalinux y EuroLinux, escribe en varios medios sobre temas relacionados con software libre y asesora a empresas en sus estrategias relacionadas con estos temas. Es coordinador de la Sección Técnica de Software Libre de Novática.

· **AOL-Time Warner** empieza a obtener resultados de su inversión en Mozilla, después de un par de años en los que se ha dicho mucho sobre este proyecto. La principal ventaja que posiblemente van a conseguir del proyecto Mozilla es el disponer de un navegador que con seguridad será muy usado en Internet y que puede ser usado en combinación «especial» con sus portales. La apuesta de Netscape (empresa comprada luego por AOL, antes de que compraran Time Warner) cuando decidió liberar el código de su navegador y dedicar muchos esfuerzos a construir como software libre su siguiente generación, fue considerada como el comienzo de la atención de las grandes empresas al software libre. Hoy día sigue siendo uno de los proyectos libres más grandes financiado por una empresa.

· **Intel** ha desarrollado el porte de Linux para el microprocesador Merced (ia64), que se ha convertido así en algo estratégico para ellos (como lo es el chip Merced).

· **Apple** ha liberado varios programas, como el Streaming Server de QuickTime, y está usando programas libres en sus productos más estratégicos, como el núcleo de Darwin, que constituye el prototipo de la nueva generación de sistemas operativos para sus máquinas.

· **Sun Microsystems** dio un paso muy importante al distribuir a partir de octubre su producto StarOffice como software libre, bajo licencia GPL, con el nombre OpenOffice. El sitio donde se va a llevar la coordinación del desarrollo está ya montado (OpenOffice) y ya se ha empezado a colaborar entre otros con el proyecto **GNOME**, con la intención de integrar OpenOffice en su modelo de componentes. Esta iniciativa de Sun es especialmente interesante, porque podría marcar un cambio estratégico en una empresa que desde hace tiempo ha probado estrategias similares al software libre, pero sin llegar a apostar completamente por este modelo. También proporciona al mundo del software libre un juego de herramientas ofimáticas comparable en funcionalidad y madurez a los mejores juegos propietarios.

3. ¿Y las empresas «nuevas»?

Alrededor del software libre están surgiendo, literalmente, cientos de nuevas empresas. Es posiblemente uno de los subsectores más dinámicos dentro de un sector tan dinámico como el informático, aunque los vientos de crisis también se están llevando muchos proyectos por delante. Algunas de estas empresas tienen actividades más o menos tradicionales, como dar soporte, pero basadas en sistemas libres como GNU/Linux. Otras empresas están probando nuevos modelos de negocio. Sin embargo, la situación actual de los mercados financieros no es la más propicia, y muchas empresas que hace unos años parecían muy prometedoras están hoy al borde de la desaparición. A continuación incluimos una lista no exhaustiva de algunas de las empresas más interesantes relacionadas con el software libre.

· **RedHat** fue la primera empresa con un negocio basado en software libre que salió a bolsa (al NASDAQ, en EE.UU.). Y lo hizo con tal éxito que a partir de ese momento la prensa económica internacional no ha dejado de atender al software libre. Desde entonces también se ha hecho mucho más sencillo conseguir capital riesgo para empresas relacionadas con software libre. El negocio original de RedHat estaba basado en la venta de CDs con su distribución de GNU/Linux (posiblemente la más popular). Hoy día han expandido sus áreas de negocio a casi cualquier actividad relacionada con Linux y software libre, desde soporte hasta desarrollo de grandes proyectos. Ha colaborado también mucho en varios proyectos libres, notablemente GNOME. Si bien su cotización, más de dos años después de su salida a bolsa, ha bajado mucho, este año ha sido el primero en que RedHat ha tenido beneficios.

· **VA Linux** salió también al NASDAQ, algo después de RedHat. Fue también un gran éxito. Su negocio original era la venta de ordenadores preinstalados con GNU/Linux. Hoy día se dedican también a casi cualquier actividad relacionada con software libre, desde sitios Internet (compraron Andover.net), hasta soporte a grandes empresas. Su cotización en bolsa es hoy día también muy baja.

· **Alcôve** (Francia), **ID-PRO** (Alemania) y **LinuxCare** (EE.UU.) son empresas dedicadas al negocio del soporte y la consultoría sobre software libre y GNU/Linux, fundamentalmente para grandes empresas. Son buenos ejemplos de cómo ya hay un mercado de clientes deseando contratar soporte para sus sistemas basados en software libre, que este tipo de empresas está tratando de atender.

· **Helixcode** está desarrollando partes de GNOME, y adaptándolo a las necesidades de sus clientes, en coordinación con el resto del proyecto GNOME, pero mejorando fundamentalmente aspectos de usabilidad y cuidando mucho la calidad. Su objetivo es proporcionar un entorno ofimático completo, fácil de instalar y utilizar, y constituido completamente por software libre. Sus programadores componen el núcleo de desarrolladores de aplicaciones como Gnumeric y Evolution, entre otros (ver descripción del proyecto GNOME más abajo). Actualmente está también participando en el proyecto Mono, una implementación libre de la arquitectura .NET de Microsoft.

· **Digital Creations** es la empresa que promueve Zope, un sistema para la creación de sitios web. Lo más curioso de esta empresa es cómo decidieron dedicarse al negocio del software libre (dando lugar posteriormente a lo que hoy conocemos como Zope) cuando una firma de inversores en capital riesgo les animó a ello. Fue uno de los primeros casos donde la idea de entrar en el mundo del software libre no vino de unos técnicos conocedores de ese mundo, sino que fueron razones fundamentalmente financieras las que les llevaron a tomar la decisión.

· **Ars Digita** también se dedica a desarrollar software libre para la creación y el mantenimiento de sitios web. También ha sido capaz de conseguir capital riesgo (recientemente ha anunciado que ya ha conseguido 35 millones de dólares) para desarrollar su negocio. Además, esta empresa mantiene ya una línea saneada de ingresos por servicios prestados a empresas como Hewlet Packard y Siemens. Los servicios que propor-

cionan son fundamentalmente consultoría y mantenimiento sobre el software que producen (que al ser libre, sus clientes obtienen a un coste nominal).

4. ¿Y las instituciones?

Las Administraciones Públicas, en su doble vertiente como grandes consumidores de software y como instrumento para el fomento del desarrollo de la sociedad, tienen un gran papel potencial en el mundo del software libre. Las ventajas que se pueden conseguir del uso y la promoción del software libre son muy grandes, y así lo están empezando a reconocer en varios países, que actualmente constituyen la avanzadilla de un movimiento que, sin duda, se hará más y más común en los próximos años.

En los siguientes apartados se repasan algunas iniciativas que los poderes ejecutivo y legislativo están llevando a cabo en diversos países del mundo, centrándonos en las llevadas a cabo en Europa.

4.1. Grupo de trabajo sobre software libre en la Unión Europea

El grupo de trabajo sobre software libre promovido por la Comisión Europea está compuesto por profesionales procedentes de varios países de la Unión Europea,y cuenta también con la colaboración de representantes de la Dirección General sobre la Sociedad de la Información de la Comisión Europea. La lista completa de sus miembros, así como el informe que han preparado, puede consultarse en <http://eu.conecta.it>.

El grupo comenzó sus trabajos en junio de 1999, con una reunión en la que se estudió la situación en ese momento del mundo del software libre. También se discutieron algunas de las acciones que la Comisión Europea podría tomar en relación con él. Como plasmación de sus trabajos, el grupo decidió elaborar un informe que pudiese ser distribuido a las instituciones: «*Free Software / Open Source: Information Society Opportunities for Europe?»* (disponible en <http://eu.conecta.it/paper.pdf>. En él se estudia con cierto detalle el software libre (su historia, algunos aspectos legales, los modelos de negocio posibles, algunos de los impactos más significativos que produce) y se proponen algunas recomendaciones a la Comisión Europea y a las administraciones de los estados miembros.

El informe fue presentado en una reunión sobre software libre en marzo de 2000 en Bruselas, con la asistencia de representantes de la Comisión Europea, de algunos estados miembros, de empresas del sector de tecnologías de la información y de la comunidad del software libre. Después de incluir los resultados de esta reunión, este informe ha sido enviado a la Comisión para que considere su respaldo oficial.

A nivel institucional, la Comisión Europea está en la actualidad explorando las posibilidades que ofrece, en los planos económico y social, el software libre. Por un lado, desde el punto de vista de la industria de tecnologías de la información, el software libre podría cambiar las reglas del juego de forma que Europa ganase competitividad en un mercado que está claramente dominado por empresas de otras áreas geográficas. Por otro, desde el punto de vista de las ventajas para la sociedad, son muchas las ventajas que podrían derivarse de un uso amplio del software libre.

Por ahora, la Comisión ya ha adoptado algunas medidas destinadas a facilitar el uso de software libre en ciertos entornos, como los proyectos de investigación inscritos en el Programa Marco. En otros campos, como la reducción de la dependencia europea en asuntos relacionadas con la seguridad informática y el apoyo de estándares neutros y verdaderamente abiertos, las ventajas del software libre han sido ya reconocidas en algunos foros institucionales. El futuro dirá si la Comisión y los estados miembros de la Unión Europea son capaces de aprovechar las ventajas, y el cambio de panorama, que proporciona el software libre.

4.2. Proyectos de ley en Francia

Durante el año 2000 se presentaron en Francia dos proyectos de ley relacionados con el software libre: el de Laffitte, Trégouet y Cabanel, y el de Le Déaut, Paul y Cohen. El primero aboga por el uso obligatorio de software libre en la administración, previendo excepciones y medidas transitorias para aquellos casos en los que no sea aún técnicamente posible. El segundo persigue la compatibilidad e interoperabilidad del software, haciendo hincapié en la disponibilidad del código fuente del software utilizado en la administración. Sin embargo, no requiere que las aplicaciones desarrolladas sean software libre, entendido como aquél que se distribuye con licencias que garantizan la libertad de modificación, uso y redistribución del programa. Ninguno de ellos ha sido aprobado, pero su impacto ha sido indiscutible como ejemplo para los legisladores de todo el mundo.

4.2.1. Proyecto de ley de Laffitte, Trégouet y Cabanel

El proyecto de ley 495 (Laffitte, Trégouet y Cabanel) fue expuesto en el servidor web del senado de la república francesa a partir de octubre de 1999. Tras un proceso de discusión pública (disponible en <http://www.senat.fr/Vforum/5/forum.html>) a través de Internet que se prolongó durante dos meses, el proyecto sufrió algunas modificaciones. El resultado es el proyecto de ley número 117 (disponible en <http://www.senat.fr/grp/rdse/page/forum/index.htm>) que propone, entre otras cosas, la utilización preferente de sistemas libres en la administración. En particular, obliga a que a partir de enero de 2002 sólo se use software libre en la administración, a crear una Agencia del software libre y a asegurarse de que las modificaciones realizadas al software libre por la administración son redistribuidas.

Este proyecto, promovido por la oposición, es poco probable que consiga apoyos suficientes para convertirse en ley.

4.2.2. Proyecto de ley de Le Déaut, Paul y Cohen

En abril de 2000 los diputados Jean-Yves Le Déaut, Christian Paul y Pierre Cohen propusieron una nueva ley (disponible en

<http://www.osslaw.org/) cuyo objetivo es similar al proyecto de Laffitte, Trégouet y Cabanel: reforzar las libertades y la seguridad del consumidor, así como mejorar la igualdad de derechos en la sociedad de la información.

Sin embargo, a diferencia del proyecto de ley de Laffitte, Trégouet y Cabanel, este otro no fuerza a utilizar software libre en la administración. Este proyecto de ley pretende que el software utilizado en la administración tenga el código fuente disponible, si bien no obliga a que éste se distribuya con licencias de software libre.

Para conseguir sus objetivos los legisladores pretenden garantizar el «derecho a la compatibilidad» del software, proporcionando mecanismos que lleven a la práctica el principio de interoperabilidad de la Directiva Europea sobre el sSftware de 1991.(disponible en <http://europa.eu.int/eur-lex/es/lif/dat/1991/es_391L0250.html>). Esta propuesta, promovida por el grupo parlamentario que actualmente gobierna en Francia, tiene muchas posibilidades de acabar convertida en ley.

4.3. Proyecto de ley en Brasil

El diputado Walter Pinheiro presentó en diciembre de 1999 un Proyecto de Ley sobre el Software Libre en la Cámara Federal de Brasil. Este proyecto afecta a la utilización de software libre en la administración pública y en las empresas privadas controladas accionarialmente por el estado.

Se recomienda el uso de software libre en estas entidades que no tenga restricciones en cuanto a su préstamo, modificación o distribución. El articulado de la ley describe pormenorizadamente qué se entiende por software libre y cómo deben ser las licencias que lo acompañen. Las definiciones coinciden con la definición clásica de software libre del proyecto GNU. En la exposición de motivos se repasa la historia del proyecto GNU, analizando sus ventajas y logros. Así mismo se hace referencia a la situación actual del software libre, utilizando como ejemplo el sistema operativo GNU/Linux.

El proyecto de ley del diputado Walter Pinheiro puede consultarse en portugués en <http://www.guiasoft.com.br/leilinux.htm>.

4.4. Iniciativas del gobierno alemán

En noviembre de 1999 el gobierno alemán anunció sus planes para financiar con 163.000 euros un proyecto de desarrollo de software libre para el cifrado de mensajes en Internet. El dinero se entregó al Grupo de Usuarios Unix de Alemania (GUUG), <http://www.guug.de/> con el objetivo de mejorar el software *GNU Privacy Guard* (**GnuPG**). GnuPG es una aplicación de seguridad informática basada en algoritmos de cifrado. Está escrito principalmente por programadores alemanes, siendo una de las mejores herramientas de seguridad existentes actualmente.

GnuPG implementa el estándar OpenPGP, que extiende PGP, el estándar de facto para herramientas de cifrado en Internet. GnuPG se distribuye bajo la licencia GNU General Public License (GNU GPL).

La financiación del Ministerio de Economía y Tecnología alemán pretende fomentar la producción de aplicaciones que utilicen cifrado fuerte y de GnuPG en particular. Así mismo pretenden que las herramientas programadas sean fáciles de utilizar por los usuarios del sistema operativo propietario Microsoft Windows. Habrá versiones que se integrarán con programas de correo electrónico. Actualmente GnuPG está disponible para sistemas operativos tipo Unix.

También se pretende financiar mediante ésta y futuras ayudas una versión que se integre con el producto propietario de gestión de documentos Lotus Notes. Este producto de la empresa estadounidense IBM se distribuye con criptografía débil debido a la regulación de la administración estadounidense que prohibe la exportación de criptografía fuerte. En 1997 este hecho provocó un escándalo en la administración de Suecia, (ver <http://catless.ncl.ac.uk/Risks/19.52. html#subj1>) cuando se descubrió que IBM entregaba a la administración estadounidense parte de las claves utilizadas para cifrar en Lotus Notes. Lotus Notes se utilizaba hasta entonces en múltiples dependencias de la administración sueca, incluyendo el ministerio de defensa y altas instancias del gobierno.

Ésta es una de las razones por las que el gobierno alemán ha escogido el modelo de desarrollo del software libre para abastecerse de este software crítico de cifrado: la disponibilidad de código fuente hace que se pueda auditar qué hace el software, convirtiéndolo así en más seguro.

En declaraciones realizadas por Werner Mueller, ministro alemán de Economía y Tecnología de Alemania, a la revista alemana **c't** <http//www.heise.de/ct/>, éste declaraba la intención de su gobierno no sólo de tolerar, sino de promocionar la tecnología de cifrado. En el plan del gobierno federal «Innovación y empleo en la sociedad de la información del siglo 2» se destaca la seguridad como un factor importante para que los usuarios puedan utilizar aplicaciones con confianza. Asimismo el ministro se muestra convencido de que la disponibilidad del código fuente de un programa es la mejor forma de asegurarse de que no oculta puertas traseras, lo que incrementa la seguridad del producto.

En la política de seguridad alemana adoptada a partir del 2 de junio de 1999, el gobierno federal manifestó que los productos de cifrado fuerte pueden desarrollarse, producirse, venderse y usarse sin ningún tipo de restricción. El gobierno identificó la necesidad de disponer de dichos productos para que el comercio electrónico se incremente en Alemania. Sin embargo detectaron que el producto más utilizado internacionalmente era el programa **PGP** (*Pretty Good Privacy*), que en la actualidad se distribuye como producto propietario, no pudiéndose examinar su código fuente.

Este razonamiento fue el que llevó al ministerio alemán a financiar el proyecto de software libre GnuPG. Puede encontrarse más información en:

· Artículo en New York Times (<http://www.sicherheit-im-

internet.deshowdoc.php3?doc=bmwi_theme_doc_2000947923935&page=1>, en inglés)
- Nota de prensa del anuncio de la iniciativa (<http://www.gnupg.de/presse.en.html>, en inglés).
- Entrevista al ministro de economía alemán (<http://www.heise.de/ct/99/21/046/>, en alemán)
- Páginas de GNU Privacy Guard (GnuPG) (<http://www.gnupg.org/>, en inglés y alemán)
- Información mantenida por el ministerio de Economía y Tecnología alemán sobre GnuPG (<http://www.sicherheit-im-internet.de showdoc.php3?doc= bmwi_theme_ doc _ 1999943531307 &page=1>, en alemán)

Por otro lado, la Agencia de coordinación de tecnologías de la información del gobierno alemán (*Koordinierungs und Beratungsstelle*, **KBSt**, < http://www.kbst.bund.de/>) recomendó el uso de Software Libre en la administración alemana mediante la circular número 2/2000. Este informe está disponible en <http://linux.kbst.bund.de/index2.html> (en alemán). La KBSt es una agencia dependiente del Ministerio del Interior alemán. Sus circulares se publican regularmente y tienen como objetivo ayudar a coordinar la planificación en tecnologías de la información entre las diferentes agencias federales, así como ayudar a los organismos del gobierno a estar al día de los desarrollos y experiencias en el campo de las tecnologías de la información.

Según la KBSt, muchas agencias se enfrentan en la actualidad con la tarea de reemplazar sistemas obsoletos. En la circular 2/2000 se hace referencia también a la importancia del software libre en la industria.

La KBSt recomienda el uso de GNU/Linux en los ordenadores personales de los empleados del gobierno. Afirman que no puede mantenerse la situación actual producida por la naturaleza propietaria del software: la instalación de las últimas versiones de los paquetes de software suele requerir nuevo hardware, ya que el existente no suele tener las prestaciones requeridas por las nuevas aplicaciones. Esta dependencia conlleva numerosos inconvenientes. Los productos suelen ser muy caros y las actualizaciones se distribuyen frecuentemente. Los documentos suelen almacenarse en formatos propietarios y los documentos creados por las nuevas versiones de los programas no pueden ser procesados por las viejas versiones.

La circular de la KBSt concluye: «hoy ya existe un entorno informático para uso profesional en entornos de oficina, que es estable, económico y seguro, y que economiza los recursos y está soportado por un número suficiente de compañías consultoras. Este entorno se puede obtener mediante el uso de sistemas operativos libres como GNU/Linux o FreeBSD, complementándolos con software comercial y libre. Esto es cierto también para sistemas clientes y servidores. Linux en particular, gracias a su gran aceptación en el mercado de las tecnologías de la información, ofrece una inversión segura hoy día».

5. ¿Cómo van los proyectos?

El software libre no sería más que un modelo posible, pero vacío de contenido, si no fuera porque existen productos utilizables. Y detrás de cada uno de estos programas hay un proyecto que lo desarrolla y lo mantiene. Son estos proyectos los que aseguran que los programas libres siguen mejorando técnicamente, los que aseguran su calidad y los que integran la colaboración de las comunidades de usuarios. En algunos casos, los proyectos funcionan de forma relativamente informal, dirigidos por voluntarios. En otros, hay empresas colaborando con muchos recursos a que los proyectos continúen adelante. Y en otros, están directamente promovidos por alguna empresa. A continuación, pasamos revista a los más destacados.

5.1. Linux sigue vivo y con buena salud

El proyecto Linux es posiblemente el proyecto libre más conocido. Como es bien conocido, fue iniciado por Linus Torvalds, un estudiante finlandés, que, con la ayuda de cientos de programadores de todo el mundo, construyó a principios de los años 90, desde cero, un *kernel* de sistema operativo similar a Unix. Al portar a ese *kernel* muchas aplicaciones libres ya disponibles en esa época (y en especial las producidas por el proyecto GNU), vio la luz el sistema operativo Linux, o como muchos prefieren nombrarlo, GNU/Linux. Hoy día el proyecto sigue siendo coordinado por Linus Torvalds, aunque otros desarrolladores, como Alan Cox, tienen en él un papel destacado. Ya desde hace unos años, han decidido liberar series de versiones con más frecuencia, para seguir más de cerca los nuevos desarrollos[1]. En cualquier caso, se va a continuar con la política de mantener dos series en paralelo, la estable (pensada para usuarios finales, actualmente la 2.4) y la inestable (pensada para desarrolladores que quieren probar las últimas características del *kernel*, que en breve será la 2.5). Alguno de los mejores sitios con información sobre el *kernel* Linux son Linux.com, <http://www.linux.com>, y Kernelnotes, <http://www. kernelnotes.org>.

La serie 2.4, introducida en el año 2001, no presenta muchos cambios radicales, pero incluye más y más características avanzadas. Varios subsistemas han sido rediseñados, mejorando su rendimiento y sus capacidades. Incluye muchos más manejadores para diferentes tipos de hardware. Algunas características esperadas desde hace tiempo están disponibles por fin (por ejemplo, soporte completo para *plug-and-play*, **USB**, sistemas de ficheros transaccionales, etc.). En

Figura 1. Logo de Linux

Figura 2. Logo de GNU

general, la mayoría de estas mejoras buscan hacer Linux más apto para su empleo en ordenadores de sobremesa, que es por ahora el segmento en el que mostraba más problemas. Puede encontrarse información más detallada sobre Linux 2.4 en el artículo «Wonderful World of Linux 2.4», <http://linuxtoday.com/stories/10698.html>, de Joe Pranevich.

5.2. GNU produce más y más software

El proyecto GNU, <http://www.gnu.org/>, fue iniciado por Richard Stallman en 1984, con la idea de producir un sistema operativo completo libre. Comenzó construyendo sobre todo herramientas para programadores (el compilador **gcc**, el editor **emacs**, el depurador **gdb** y muchas más) y utilidades típicas de sistemas operativos. Cuando estas herramientas fueron utilizadas en el proyecto del *kernel* Linux a principios de los años 90, permitieron el nacimiento del sistema operativo GNU/Linux. Desde entonces el proyecto GNU no ha dejado de producir cientos de programas libres. Actualmente ha desarrollado también un *kernel* (Hurd), que ya es muy utilizable, y es la base, por ejemplo, para una distribución Debian (ver más abajo). Muchos de sus programas están entre los mejores en su campo. Por ejemplo, gcc es sin lugar a dudas el compilador que genera código en más plataformas, utilizado por compañías como Intel, Texas Instruments, Palm Computing o Sony. El proyecto más innovador desarrollado dentro del marco del proyecto GNU es GNOME, que tiene tanta entidad como para ser considerado un proyecto por sí mismo.

Y a pesar de esta enorme producción de software, desde muchos puntos de vista la mayor contribución de GNU ha sido de otro tipo: sentar parte de las bases legales (con la licencia **GPL**, una de las más usadas por los programadores de software libre) y filosóficas (al menos parcialmente) del movimiento de software libre. Actualmente GNU está embarcado en la promoción de una nueva licencia, ésta para proteger documentación, y con una filosofía muy similar a la GPL. También se está trabajando en una modernización de la licencia GPL, que incluya las nuevas modalidades de uso de software, como por ejemplo los servidores de aplicaciones en Internet. Cuando sea publicada, esta licencia será la GPL 3.0.

5.3. Apache domina su nicho

Apache, <http://www.apache.org>, es el servidor de web usado en más sitios de Internet (más del 58% de los sitios web usan Apache, según la encuesta de Netcraft, <http://www.netcraft.com/survey/>), y desde el punto de vista técnico es sin duda uno de los más completos y estables. La nueva serie de Apache (la 2.0) está ya disponible como beta. Su rediseño ha sido muy completo y ahora es más modular. También se ha cuidado especialmente el rendimiento y la configuración, que se ha mejorado sustancialmente, y se ha cuidado mucho el soporte para Windows NT y Windows 95. Alrededor de Apache están floreciendo otros proyectos, como los que integran Java y Java *servlets* con el servidor de Web, que hacen que se mantenga en el frente tecnológico en este mercado.

5.4. Mozilla empieza a dar resultados

El proyecto Mozilla, <http://www.mozilla.org>, fue iniciado por Netscape (hoy parte de AOL-Time Warner) como el primer proyecto de software libre de gran escala iniciado por una empresa. Los recursos puestos por Netscape han sido enormes, incluso para proyectos «tradicionales» (propietarios). Cientos de programadores, docenas de herramientas auxiliares, y sobre todo una gran apuesta. Durante mucho tiempo, Mozilla fue considerado como un fracaso, hasta que hace unos meses empezaron a ver la luz las primeras versiones beta del producto. Parece que Mozilla va a ser un navegador que nos va a mostrar cómo va a ser la nueva generación de navegadores. Y a su alrededor ya está apareciendo toda una constelación de nuevas aplicaciones que usan componentes suyos (como Gecko, su motor de HTML), o están derivados de su código (como Chatzilla, un cliente de IRC que ha sido desarrollado usando gran parte del código de Mozilla).

5.5. Debian incluye 4.500 paquetes

Debian, <http://www.debian.org>, es una distribución de GNU/Linux (el *kernel* Linux más miles de programas a su alrededor) que tiene la peculiaridad de no estar directamente promovida por una empresa, sino por cientos de desarrolladores repartidos por todo el mundo. Fue una de las primeras distribuciones, comenzada por unas decenas de desarrolladores a mediados de los años 90. Hoy día es un gran proyecto

Figura 3. Logo de Apache

Figura 4. Logo de Mozilla

coordinado en el que trabajan con diversos niveles de dedicación cientos de desarrolladores. El trabajo de estos desarrolladores consiste fundamentalmente en empaquetar aplicaciones para su inclusión en la distribución, y la creación de herramientas que simplifiquen la instalación y la administración del sistema.

La última versión (Debian 2.2, alias potato) vio la luz en agosto de 2000. Incluye más de 4.500 paquetes diferentes, entre los que se puede encontrar casi cualquier programa libre disponible para GNU/Linux. Una de las características más funda-

Figura 5. Logo de Debian

mentales de esta distribución es la facilidad con la que se pueden realizar las actualizaciones de forma prácticamente transparente usando CDs o directamente Internet. Pero lo que más diferencia a Debian de otras distribuciones es su énfasis en que todo el software de su distribución principal sea software libre. En esta línea, Debian está consiguiendo ser la distribución libre por excelencia, que incluye virtualmente el estado del arte en software libre. La nueva versión, que será probablmente Debian 3.0, está prevista para primeros de 2002, e incluirá más de 6000 paquetes.

5.6. KDE tiene cientos de aplicaciones

KDE, <http://www.kde.org>, es un entorno completo de escritorio que incluye ya cientos de aplicaciones que funcionan de forma integrada, incluyendo herramientas de ofimática (KOffice, que incluye procesador de texto, hoja de cálculo, navegador, etc), de programación (KDevelop, un entorno integrado para la programación en C y C++), etc. KDE funciona en sistemas tipo Unix, entre ellos en GNU/Linux. La versión 2.2 de KDE fue liberada en julio. KDE se incluye en muchas distribuciones de GNU/Linux como el entorno de escritorio por defecto, hasta el punto que muchos usuarios están ya identificando la apariencia habitual de KDE con Linux. Alrededor de KDE están surgiendo también empresas cuyo modelo de negocio está basado en el desarrollo, integración y mantenimiento de aplicaciones dentro de este entorno.

5.7. GNOME avanza con su modelo de componentes

GNOME, <http://www.gnome.org>, es otro entorno de escritorio que también incluye varios cientos de aplicaciones. Su característica principal es su énfasis en un diseño arquitectural completamente basado en componentes, que usan CORBA para integrarse y coordinarse entre ellos. Es notable cómo algunas empresas, notablemente Helixcode y Eazel (hasta su desaparición) están haciendo desarrollos basados en GNOME y contribuyendo muy activamente a su desarrollo. Algunas aplicaciones muy interesantes que están apareciendo en el marco de este proyecto son: Evolution (un programa para trabajo en grupo), Gnumeric (una hoja de cálculo), Nautilus (un gestor de ficheros), Gimp (un programa de tratamiento de imágenes), y Abiword (un procesador de textos). La constitución de la Fundación GNOME --en la que participan empresas como Compaq, Hewlet Packard, Sun, IBM y Helixcode-- demuestra el gran interés de la industria informática por este proyecto.

5.8. XFree86 ya ha liberado la versión 4.1

XFree86, <http://www.xfree86.org/> es la implementación de X Window que utilizan casi todos los sistemas operativos

Figura 7. Logo de GNOME

libres sobre procesadores derivados del i386. Proporciona la infraestructura sobre la que están construidas la mayoría de las aplicaciones gráficas libres. Acaba de liberar una nueva versión, la 4.1, con un rediseño casi completo. El camino hacia esta nueva versión comenzó hace casi tres años (pasando por la versión 4.0), y ha conseguido una mayor modularización del sistema, mejoras en el rendimiento y nuevos servicios.

Figura 6. Logo de KDE

6. Pero también hay problemas

No todo son noticias positivas en el mundo del software libre. También hay problemas que se están haciendo cada vez más evidentes. Entre ellos cabe destacar los siguientes:

· Técnicas **FUD** (miedo, desconocimiento, duda). Se están empezando a ver campañas de grandes empresas de software propietario con las que se intenta desprestigiar el software libre en general, o algún producto en particular (por ejemplo, GNU/Linux). Hasta ahora, estas campañas han mostrado no ser muy problemáticas de cara al desarrollo y aceptación del software libre. Pero según más y más productos libres van convirtiéndose en competidores peligrosos para productos propietarios, algunas empresas que comercializan estos últimos tratan de dirigir su maquinaria de mercadotecnia a evitar perder su mercado. Sólo el tiempo dirá si estas campañas serán capaces de sembrar el desconcierto en los usuarios potenciales de software libre.

Figura 8. Logo de XFree86

· Fracasos empresariales. La crisis de las *puntocom* también ha afectado a muchas empresas relacionadas con el software libre. En muchos casos, esto no ha afectado a la cantidad ni a la calidad del software desarrollado. Sin embargo, en algunos si ha tenido un impacto importante. Por ejemplo, la suspensión de pagos de Eazel ha puesto en entredicho la continuación del proyecto Nautilus (aplicación de GNOME), que esta empresa lideraba. Sólo el tiempo nos mostrará si estos problemas empresariales tendrán un impacto importante sobre el software libre.

· «Disolución» (sistemas que pueden confundirse con el software libre). Hay muchas empresas explorando los límites del software libre y cómo utilizar modelos de negocio que no sean exactamente de software libre, pero que a la vez sean lo suficientemente similares como para que puedan confundirse con él. El problema principal que causa esto es que muchos individuos pueden creer que están tratando con software libre, dando por supuestas muchas cosas que no son ciertas. Un ejemplo de estas estrategias es la **SCSL** (*Sun Community Source License*), que aunque mantiene algunas de las ventajas del software libre, incluye ciertos puntos que ponen de facto cualquier proyecto realizado usando código SCSL bajo el control de Sun. Desgraciadamente, es poco común entre los profesionales informáticos interesarse por los detalles legales del software que usan o desarrollan, y eso hace que este problema sea especialmente grave. Si los integrantes de la comunidad del software libre no reaccionan frente a estas prácticas, puede llegarse a su fraccionamiento y desde luego a la pérdida de muchas ventajas del modelo de software libre.

· Desconocimiento (pérdida de visión). Muchos individuos utilizan programas libres sin ser conscientes de cómo es posible que esos productos existan y, en muchos casos, ni siquiera de qué ventajas tiene el hecho de que sean libres. Cada vez más se corre el riesgo de que productos como GNU/Linux se conviertan simplemente en una «moda», donde los usuarios no son conscientes del modelo del software libre ni de los motivos por los que ese software les da derechos y libertades radicalmente nuevos. De hecho, el sentimiento de pertenencia a una comunidad (la comunidad del software libre) es en muchos casos extraña a mucha gente. Sin embargo, es la existencia de esta comunidad lo que hace posible que exista software libre y que, por ejemplo, GNU/Linux tenga el éxito que tiene. Es necesaria una amplia labor formativa, que explique a los usuarios y desarrolladores el funcionamiento del mundo del software libre, y especialmente por qué ellos, los usuarios, son muy importantes.

· Impedimentos legales. El software libre ya ha demostrado que es capaz de desarrollarse en un entorno con pocas o ninguna ayuda por parte de las administraciones en general, o del sistema legal en particular. Pero se están empezando a percibir problemas debidos a legislaciones que alteran específicamente algunos de los factores que hacen posible el software libre. Por ser especialmente importantes, les dedicamos a continuación un apartado.

7. Impedimentos legales

Los principales impedimentos legales que se encuentra el software libre son:

· Patentes de software. Hay legislaciones que impiden el libre desarrollo de software, entre las que cabe destacar las patentes de técnicas software. Estas patentes son aceptadas en los EEUU desde hace más de una década y, desde hace algunos años, en Japón. En Europa hay fuertes presiones para cambiar el Tratado Europeo sobre Patentes (que actualmente prohibe explícitamente este tipo de patentes). Las patentes de software, que probablemente son perjudiciales para el desarrollo de software en su conjunto, propietario o libre, son especialmente dañinas en el caso del software libre. Un desarrollador puede, de forma completamente inadvertida, utilizar una técnica patentada. Si lo hace, incluso si quisiera conseguir una licencia para el uso de esa técnica en su programa, no habría forma de que la consiga y de que el programa siga siendo libre (salvo naturalmente que el poseedor de la patente ceda sus derechos gratuitamente). Esto pone fuera del alcance del software libre muchísimas técnicas (hay más de 15.000 patentes de software en EE.UU.). Pero lo más grave es que las patentes hacen posible que alguien consiga poner fuera de la circulación un programa libre por acusaciones de infringir una patente. En Europa aún estamos a tiempo de evitar estos problemas, pero, a medio plazo, son posiblemente los que más pueden dañar el desarrollo futuro del software libre (y posiblemente, del software en general).

· En algunos países se están introduciendo normativas para

regular las transacciones comerciales que hacen muy difícil la redistribución de software libre tal y como es posible hacerlo hoy. Entre ellas puede destacarse la **DMCA** en Estados Unidos, <http://www.gnu.org/press/2001-09-24-CPI.txt>, y en menor medida, la Directiva Europea sobre Copyright. La DMCA, por ejemplo, se ha utilizado en el caso **DeCSS** para perseguir a los programadores y difusores de software libre para reproducir **DVD** bajo el sistema operativo GNU/Linux y otros no soportados actualmente por los reproductores aprobados por la industria de contenidos. Este software permite eliminar las restricciones técnicas impuestas por los estudios cinematográficos, como los códigos de región que impiden ver una película en ciertas regiones del mundo.

· Pero aún se están promoviendo medidas «un paso más allá». Entre ellas cabe destacar el proyecto de ley de EE.UU. denominado **SSSCA**.

8. Un final para este comienzo ...

Desde muchos puntos de vista, estamos empezando a ver el comienzo de una nueva forma de producir software: el software libre. Los principales proyectos libres están demostrando que son capaces de realizar programas perfectamente utilizables, de alta calidad y que pueden competir en cuanto a prestaciones con los mejores programas propietarios («tradicionales») actuales. Las empresas que se dedican a la Informática no pueden ignorar esta situación y en su mayor parte están elaborando estrategias que les permitan, como mínimo, seguir la evolución de este mundo y empezar a aprender cómo participar en él. Cientos de nuevas empresas están apareciendo por todo el mundo, inventando nuevos modelos de negocio y cubriendo las necesidades que ya están reclamando miles de grandes y pequeños clientes que ya han identificado que el software libre puede convertirse en un elemento clave para ellos.

En el futuro a corto plazo seguiremos viendo cómo estas tendencias se van reforzando conforme más y más programas libres se van haciendo un hueco en el mercado. A largo plazo, sólo el tiempo podrá decir hacia donde vamos. Algunos piensan que, pasados unos años, las aguas volverán a su cauce y el software libre volverá a la marginalidad en la que ha estado instalado hasta hace bien poco. En el otro extremo, otros opinan que va a revolucionar la producción de software hasta puntos hoy difícilmente imaginables, llegando a dominar completamente como modelo de producción toda la industria. Posiblemente la realidad estará en algún punto medio entre estos dos extremos, con nichos dominados por programas propietarios y otros donde el más usado sea claramente un programa libre. Lo que parece claro en casi cualquier escenario es que las empresas, instituciones e incluso países que sean capaces de identificar cuanto antes cómo pueden aprovechar las ventajas que les ofrece el software libre tendrán muchas ventajas competitivas frente a sus competidores. Por eso es tan importante, igual que casi siempre en tecnología, no sólo estar ahí, sino estar ahí antes que otros.

Nota

1 Una serie de versiones está compuesta por todas las versiones con características y diseños similares. Actualmente, la serie estable es la 2.4, que incluye por ejemplo a las versiones 2.4.1, 2.4.5, 2.4.8, etc.

Software Libre/Fuente Abierta: hacia la madurez

Richard Stallman

<rms@gnu.org>

Traducción: Pedro de las Heras Quirós, Jesús M. González Barahona (URJC, Novática)

El daño viene de La Haya

Resumen: *en este artículo se denuncian los peligros que para la libertad de expresión, y también para la competitividad de las empresas europeas, tendría la aprobación de cambios en la legislación europea sobre patentes de software (Convenio Europeo de Patentes, también llamado Tratado de la Haya).*

Palabras clave: *software libre, propiedad intelectual, Tratado de La Haya, patentes de software, libertad de expresión, flujo libre de Información*

Los europeos se han opuesto de manera enérgica al intento de introducir patentes de software en Europa, y lo han frustrado. Se ha propuesto un tratado, aún en fase de negociación, que amenaza con someter a los desarrolladores de software de Europa y de otros países a las patentes de software de los Estados Unidos, y a otras leyes perjudiciales del resto del mundo. El problema no sólo se les plantea a los programadores, los autores de cualquier tipo se enfrentarán también a nuevos peligros. Incluso las leyes de censura de varios países podrían tener un efecto global.

De hecho, el tratado de La Haya no se refiere específicamente a patentes, a derechos de autor o a censura, aunque afecta a todos ellos. Es un tratado sobre jurisdicción, sobre cómo un país debe tratar las sentencias judiciales de otros países. La idea básica es bastante razonable: si alguien choca con tu coche en Francia o incumple un contrato con tu compañía francesa, puedes demandarle en Francia y luego hacer que se cumpla la sentencia por orden de los juzgados del país en el que viva el demandado (o en el que tenga sus activos).

El tratado se convierte en un problema cuando se aplica a la distribución de información, porque hoy día la información viaja con normalidad a todos los países (por ejemplo, pero no sólo, mediante Internet). La consecuencia es que podrás ser demandado por la información hayas distribuido utilizando las leyes de **cualquier** país que firme el tratado de La Haya y la sentencia se aplicaría probablemente en tu propio país.

Por ejemplo, si publicas un paquete de software (libre o no) en Alemania y hay gente que lo utiliza en los EE.UU., podrías ser demandado por infringir alguna patente de software absurda de los EE.UU. Esto no es fruto de La Haya, podría ocurrir ahora. Pero hoy día podrías ignorar la sentencia de los EE.UU. estando a salvo en Alemania y el dueño de la patente lo sabe. Bajo el tratado de La Haya se puede requerir a cualquier juzgado alemán para que te aplique la sentencia estadounidense. De hecho, las patentes de software de cualquier país firmante del tratado se aplicarían en todos los países firmantes del mismo. No basta con evitar las implantación de las patentes de software en Europa si las patentes de software estadounidenses, japonesas o egipcias te pueden alcanzar de esta forma.

Pero la legislación sobre patentes no es el único área de la legislación que puede causar problemas si se mundializa mediante el tratado de La Haya. Supongamos que publicas unas declaraciones criticando a un personaje público. Si llegan copias de tus declaraciones al Reino Unido, ese personaje público podría demandarte amparándose en la estricta legislación antilibelo del Reino Unido. Puede que las leyes de tu país garanticen el derecho a criticar a personajes públicos, pero bajo el tratado de La Haya ya no es seguro que te protejan.

O supongamos que publicas una comparativa de tus precios con los de la competencia. Si la leen en Alemania, donde es ilegal la publicidad en la que se comparan productos, podrías

Autor

Richard Stallman es el gurú, pionero y más conocido ideólogo y activista del software libre. Es el fundador del Proyecto GNU, iniciado en 1984 para desarrollar el sistema operativo libre GNU (un acrónimo de "GNU's Not Unix") y, por lo tanto, dar a los usuarios de computadoras la libertad que la mayoría de ellos han perdido. GNU es software libre: todos tienen libertad para copiarlo y redistribuirlo, así como para hacerle cambios grandes o pequeños. En la actualidad se usan ampliamente diversas variantes del sistema GNU basadas en Linux, cuyo núcleo de sistema [*kernel*], Linux, fue desarrollado por Linus Torvalds. Se estima que hoy en día existen más de 10 millones de usuarios de sistemas GNU/Linux. Richard Stallman es el autor principal del Compilador C de GNU, un compilador optimizador transportable que fue diseñado para apoyar diversas arquitecturas y múltiples lenguajes. El compilador ahora apoya más de 30 arquitecturas distintas de computadoras y 7 lenguajes de programación. Stallman también escribió el depurador simbólico GNU (GDB), GNU Emacs y algunos otros programas GNU. Stallman recibió el Premio *Grace Hopper* de la *Association for Computing Machinery* de 1991 por el desarrollo del primer editor Emacs en la década de los setenta. En 1990 fue premiado con una beca de la *MacArthur Foundation* y en 1996 se le concedió un doctorado honorario del *Royal Institute of Technology* de Suecia. En 1998 recibió el *Pioneer Award* de la *Electronic Frontier Foundation* conjuntamente con Linus Torvalds. En 1999 recibió el Premio *Yuri Rubinski*. Página personal: <http://www.stallman.org/rms.es.html>

ser demandado en Alemania y la sentencia te sería aplicada allá donde estuvieses (esta ley podría haber sido derogada en Alemania, pero la idea general sigue siento válida).

O supongamos que publicas una parodia. Si la leen en Corea te podrían demandar allí, ya que Corea no tiene reconocido el derecho a parodiar a la gente (de nuevo, es posible que Corea cambie esta legislación, pero el fondo del asunto sigue siendo el mismo).

O supongamos que tienes opiniones políticas que están prohibidas por algún gobierno. Podrías ser demandado en su país y la sentencia contra ti te sería aplicada allá donde vivas.

No hace mucho, Yahoo fue demandada en Francia por tener enlaces en sus páginas a sitios de los EE.UU. que subastaban recuerdos nazis, algo que está permitido en EE.UU. Cuando un juzgado francés pidió a Yahoo Francia que bloquease dichos enlaces, Yahoo acudió a los tribunales estadounidenses para pedir que dictasen una sentencia en la que se reconociese que la decisión judicial francesa no podía aplicársele a la compañía matriz en los EE.UU.

Richard Stallman

Puede resultar sorprendente descubrir que los disidentes chinos en el exilio se personaron en la causa a favor de Yahoo. Sin embargo, ellos sabían lo que estaban haciendo pues su movimiento democrático depende del resultado.

Y es que el nazismo no es la única postura política cuya expresión está prohibida en algunos lugares. Criticar al gobierno chino también está prohibido (en China).

Si una sentencia de un tribunal francés en contra de posturas nazis puede ser ejecutada en los EE.UU., o en tu país, quizá una sentencia china contra posturas contrarias al gobierno chino podría ser ejecutada allí también. (Quizá por ello China se ha sumado a las negociaciones del tratado de La Haya). El gobierno chino puede adaptar fácilmente su legislación sobre censura para que el tratado de La Haya pueda aplicarse a dicha legislación. Basta con que conceda a individuos particulares (y a agencias gubernamentales) el derecho a demandar a publicaciones disidentes.

China no es el único país que prohibe criticar a su gobierno. En la fecha de publicación de este artículo, el gobierno de Victoria (Australia) está litigando para que se retire un libro titulado *Victoria Police Corruption* («Corrupción en la policía de Victoria»), argumentando que «difama a los tribunales». Este libro puede obtenerse fuera de Australia a través de Internet. Australia participa en el tratado de La Haya. Si el tratado se aplica a estos casos, una sentencia de un tribunal australiano que obligue a retirar el libro podría utilizarse para retirarlo de cualquier otro lugar.

Mientras tanto, las obras que critican el Islam están siendo cada vez más censuradas en Egipto, otro país que participa en el tratado de La Haya. Esta censura también podría mundializarse mediante el tratado de La Haya.

Los norteamericanos pueden ampararse en la Primera Enmienda de la Constitución para protegerse de las sentencias foráneas en contra de su libertad de expresión. El borrador del tratado permite que un tribunal desestime una sentencia de otro país si es «manifiestamente incompatible con sus principios legales». Este es un criterio riguroso, por lo que no puedes contar con que te proteja sólo porque tu conducta sea legal allá donde estés. Es potestad del juez definir si te protege o no. Es poco probable que te sea de ayuda frente a interpretaciones foráneas amplias de la legislación sobre *copyright*, sobre marcas registradas o sobre patentes de software, aunque los tribunales estadounidenses podrían usarlo para rechazar por completo sentencias en las que se intente censurar.

Sin embargo, incluso esta vía no te sería de ayuda si publicas en Internet, bien porque tu proveedor de servicios de Internet (PSI) tenga activos en otros países, o porque se comunique con el resto del mundo mediante otros PSI mayores que los tengan. Podría aplicarse una sentencia en la que se censure tu sitio, o cualquier otro tipo de sentencia, litigando contra tu PSI, o contra el PSI de tu PSI, en cualquier país en el que tengan activos, y en el que no haya una Carta de Derechos o la libertad de expresión no goce del mismo status privilegiado que tiene en los EE.UU. Como respuesta, tu PSI cerrará tu sitio. El tratado de La Haya mundializaría los pretextos para querellarse en contra de alguien, pero no las protecciones de las libertades civiles, por lo que cualquier protección local podrá ser evitada.

¿Crees que la demanda a tu PSI es poco probable? Ya está ocurriendo. Cuando la multinacional Danone anunció sus planes de cierre de factorías en Francia, Olivier Malnuit abrió un sitio, <jeboycottedanone.com>, para quejarse (el nombre significa «Yo boicoteo Danone»). Danone no sólo le demandó a él, sino también a la compañía que hospedaba su sitio y a la que registró el nombre de dominio, por «falsificación de bienes», y en abril de 2001 se dictó una sentencia en la que se prohibía a Malnuit mencionar el nombre «Danone» en el nombre de dominio y en el texto del sitio. Lo que es más revelador aún, la empresa que registraba el nombre lo eliminó, temiendo ser condenada por el tribunal.

La respuesta natural de los disidentes franceses es publicar sus críticas a Danone fuera de Francia, al igual que hacen los disidentes chinos que pubican sus críticas a China fuera de su país. Pero el tratado de La Haya permitiría a Danone atacarles en cualquier lugar. Quizá incluso este mismo artículo podría ser retirado utilizando su PSI, o el PSI de su PSI.

Los efectos potenciales del tratado no se limitan a las leyes que existen hoy día. Cuando 50 países sepan que las sentencias de sus tribunales podrían ser ejecutadas en Estados Unidos, Europa y Asia, sentirán la tentación de aprobar leyes justo con ese propósito.

Supongamos, por ejemplo, que Microsoft quisiese poder aplicar derechos de autor a lenguajes y protocolos de comunicaciones. Podrían acudir a un país pequeño y pobre, y ofrecerles una inversión de 50 millones de dólares EE.UU. al año durante un período de 20 años a cambio de que el país apruebe una ley que diga que implementar un lenguaje o un protocolo de Microsoft supone una infracción del *copyright*. Seguramente pueden encontrar algún país que aceptase la oferta. Cuando alguien implementase un programa compatible, Microsoft podría demandarle en ese país y ganar. Cuando el juez emitiese la sentencia a su favor, prohibiendo la distribución de tu programa, los tribunales de tu país aplicarían dicha sentencia, de acuerdo al tratado de La Haya.

¿Parece esto poco probable? En 2000, Cisco presionó a Liechtenstein, un pequeño país europeo, para que legalizase las patentes de software. Y el principal grupo de presión de IBM amenazó a muchos gobiernos europeos con la retirada de sus inversiones si no reconocían las patentes de software. En la misma época, el agregado comercial de los EE.UU. en Jordania (Oriente Medio) presionó hasta que se reconocieron las patentes matemáticas.

En una reunión de organizaciones de consumidores (<http://www.tacd.org>) recomendaron en mayo de 2001 que las patentes, los *copyright*s y las marcas registradas (la «propiedad intelectual») fuesen excluidas del ámbito del tratado de La Haya, porque esas leyes varían considerablemente de país a país.

Es una buena recomendación, pero sólo resuelve parte del problema. Las patentes y las extensiones extrañas a los derechos de autor son sólo dos de las muchas excusas que se utilizan para prohibir la publicación en ciertos países. Para resolver el problema completamente, deberían excluirse del ámbito de aplicación del tratado todos los casos relativos a la legalidad de la distribución o transmisión de información y sólo debería tener jurisdicción el país en el que opere el distribuidor o transmisor de la información.

En Europa, la gente que se opone a las patentes de software está trabajando para cambiar el tratado de La Haya. Para obtener más información, vea <http://www.noepatents.org/hague>. En los EE.UU., el *Consumer Project for Technology* está llevando el liderazgo en este ámbito: <http://www.cptech.org/ecom/jurisdiction/hague.html>.

Hoy (6 de junio de 2001) está previsto que comience una conferencia diplomática para tratar los detalles del tratado de La Haya. Debemos hacer que los políticos y el público en general sean conscientes cuanto antes de los peligros posibles.

Iniciativas europeas sobre el uso de software libre en el Sector Público

Juan Jesús Muñoz Esteban
Jefe del Área de Explotación de la Subdirección de Informática del Ministerio de Justicia

<juange@inf.uc3m.es>

Resumen: *la Comisión Europea, en algunas de sus iniciativas estratégicas, empieza a contar con el software libre. El programa* ***IDA*** *(Intercambio de Datos entre Administraciones) ha realizado un simposium y un estudio sobre el uso del mismo en varias Administraciones Públicas de distintos países que analiza las condiciones que motivaron su uso y se plantea qué condiciones deberían darse para su generalización, y tiene previsto la creación de un foro de intercambio en Internet. La evitación de dependencias y el fomento de la transparencia son algunos de los objetivos, aunque a corto plazo la intención es dar a conocer el software libre a las distintas Administraciones.*

Palabras clave*: software libre, open source, estándares abiertos, Administraciones Públicas, Comisión Europea, IDA,* ***IST***

1. Introducción

Las Administraciones Públicas (AA.PP.) son organizaciones basadas en información. La medida de los indicadores del país, la gestión de la recaudación de impuestos, el seguimiento de inversiones y gastos, la elaboración de normativa, el control de su personal ... Podrán privatizar la prestación de servicios, externalizar la operación de esa información, pero en toda Administración, como el de la mayoría de las empresas, el valor reside en sus datos disponibles.

Las AA.PP. son grandes y compuestas por diversos organismos. Sus múltiples procedimientos surgen de forma independiente y aunque las iniciativas de portales y «ventanillas únicas» pretenden poner luz en ese caos percibido por el ciudadano de a pie, la mayoría están todavía en la etapa de crear una infraestructura de comunicaciones común, pero lejos de compartir servicios o de poder integrar los diferentes flujos de información que puedan ya haberse informatizado.

Quizá el único elemento que haya podido poner algo de compatibilidad entre la información gestionada por los diferentes organismos es la reducción a un pequeño conjunto de formatos de ficheros: los utilizados por las herramientas ofimáticas que han copado el mercado. Usar una herramienta distinta te puede dejar aislado si no tiene el adecuado «salvar como», que en todo caso hace que los documentos ya no queden con la misma calidad.

El papel de los datos es cada vez mayor. Inicialmente al gestor le preocupaba diseñar aplicaciones con las que pretendía hacer reingeniería de procesos. Después hay que extender infraestructura e implantarlas, y salvar el rechazo de los usuarios a los cambios.

Hoy día el gran volumen de datos acumulados a lo largo de los años exige tomar medidas. Muchos datos de los que nos tenemos que fiar han tenido un ciclo de vida no del todo controlado: la aplicación procura que su introducción sea amigable, que sea coherente. Pero la vida real no se comporta exactamente como dice el diseño funcional. La casuística aparece y con ella las distintas respuestas de los usuarios ante un sistema informático que siempre es más rígido de lo que quieren y más flexible de lo que querrían los informáticos. Al final la calidad de los datos se resiente.

En tecnología los datos también han adquirido cada vez mayor importancia. El intercambio de páginas en formato estándar **HTML** (*HyperText Markup Language*) se enriquece para plasmar no solamente los valores, sino también apariencias (**CSS***, Cascade Style Sheet*), comportamientos (**DHTML***, Dynamic HTML*), semántica (**XML**, *eXtended MarkupLanguaje*).

El cliente universal (la aplicación de navegación y de la que en teoría hay múltiples suministradores pues los datos intercambiados cumplen estándares) es un mero instrumento al servicio de estas estructuras de datos, que se limita a presentar el interfaz de usuario, incluyendo eso sí consideraciones sobre la seguridad en el acceso a la información (**SSL**, **PKI** ...).

Si lo que importa son los datos:

1. ¿Se les da suficiente importancia a su cuidado y gestión? No se trata únicamente de conducirlos por redes seguras, almacenarlos en dispositivos tolerantes a fallos, controlar la calidad de los programas que los transforman. Se trata de lograr ese nivel de confianza al que llamamos seguridad, que comienza por la propia confianza en el equipo humano suficientemente cualificado y motivado, capaz de vigilar, controlar, depurar ...

Autor

Juan Jesús Muñoz Esteban, Ingeniero de Telecomunicación y Master en Dirección de Sistemas y Tecnologías de la Información y las Comunicaciones por la Universidad Politécnica de Madrid. Es Funcionario del Cuerpo Superior de Sistemas y Tecnologías de la Información de la Administración del Estado. Actualmente es Jefe del Área de Explotación de la Subdirección de Informática del Ministerio de Justicia.

2. ¿Se es consciente de la dependencia que suponen para la propia supervivencia de la organización? ¿Qué sería de estas organizaciones sin sus bases de datos? ¿Qué pasaría si los datos se hubiesen confiado a una empresa que luego hubiese desaparecido? ¿Puede un estado soberano tener comprometido el acceso a sus datos en función del comportamiento de una empresa comercial de otro continente, para la que el organismo concreto es apenas un pequeño cliente?
3. ¿Que efectos puede tener la «protección» de formatos o protocolos propietarios de datos, que impidan a aplicaciones de terceros poder intercambiar información con las aplicaciones de las empresas que dominan el mercado? ¿Qué limitaciones puede imponer un suministrador de software para que el propietario de los datos los manipule como considere?

La dependencia de los productos es ya absoluta. Para no tener en explotación una versión fuera de soporte es necesario un presupuesto, del que no siempre se dispone pues el momento lo decide la empresa suministradora y no las propias exigencias de funcionalidad, y medidas de implantación a gran escala que suponen un gasto aún mayor. Incluso la forma de licenciar se cambia, como está ocurriendo en estos momentos, y las AA.PP. deben adaptarse al cambio en unos plazos que les vienen impuestos y que no se sincronizan con la ejecución del ejercicio presupuestario.

2. IDA

«Intercambio de Datos entre Administraciones» (**IDA**) [IDA] es una iniciativa estratégica de la Comisión Europea y en concreto de su Dirección General para Empresas (*Enterprise Directorate-General*). Su misión es coordinar el establecimiento de redes telemáticas transeuropeas. Al canalizar las tecnologías de la información y con objeto de mejorar la toma de decisiones, este organismo se preocupa de la convergencia de interfaces y formatos comunes, de la interoperabilidad... y por tanto le surgen preocupaciones como las siguientes:

1. Datos: las AA.PP. ven limitada su capacidad de acción sobre sus propios datos.
2. Inversiones: hay grandes inversiones que no suponen mayor coste para la empresa suministradora y sin embargo suponen una sobrecarga en los presupuestos que pagamos entre todos, sin tener siquiera como contrapartida un soporte adecuado que facilite la aplicación de soluciones genéricas a los problemas concretos de gestión de la información.
3. Dependencia: el mirar a Europa en conjunto se observa una enorme dependencia tecnológica, que, siendo grande en hardware, es abrumadora en la mayoría del software básico.

El propio comisario europeo para la Sociedad de la Información y para las Empresas, Erkki Likkanen, afirmó que el software libre puede suministrar nuevas oportunidades de interoperatividad y evitar la absoluta dominación a la que nos llevaría el que todos tuviésemos que usar la misma herramienta comercial y además en la misma versión.

2.1. Iniciativas de IDA relacionadas con el software libre

La primera acción consistió en un simposio sobre **OSS** (*Open Source Software*, programas de código fuente abierto) para estudiar casos de uso reales y posibles ventajas del uso de software libre. Tuvo lugar en Bruselas el 22 de febrero de 2001 y participaron alrededor de 100 representantes de las distintas AA.PP. (Comisión, Gobiernos Nacionales, Ayuntamientos...) y de empresas privadas y organizaciones involucradas.

Entre abril y agosto se realizó un estudio sobre el uso concreto de software libre en el sector público, que se ha publicitado el 3 de septiembre. Su ámbito de aplicación (Francia, Alemania, España, Bélgica, Italia y Suecia) permite sacar algunas conclusiones, constituyendo el informe un buen estudio de la situación comparada y de las posibilidades y problemática del software libre en las AA.PP..

Entre diciembre de 2001 y junio de 2002 se realizará una nueva acción encaminada a la creación de un *European OSS distribution site* donde organizar y concentrar software y *know-how*. Aportar este servicio es una condición previa para que se gesten grupos de colaboración (*web site for distribution*, *support*, *help-desk*, forum...) entre las instituciones europeas y las AA.PP. de los Estados miembro.

2.2. OSS Symposium

El simposio comenzó con una introducción a cargo del comisario europeo Erkki Likkanen, quien destacó que si en algunos parámetros como la amigabilidad o el soporte de hardware el software libre puede ir aún por detrás, en ciertas áreas como la seguridad y estabilidad puede tener una ventaja. Internet ha posibilitado un modelo de desarrollo que combina la independencia de actores individuales con una fuerte coordinación y gestión de los proyectos. Pero si la seguridad que da el acceso al código fuente es importante, más allá de estas cuestiones técnicas cuando el asunto se plantea en términos de persistencia del servicio aparece una dimensión política y económica.

Los beneficios que se derivarían de la trasparencia de la tecnología que utilizan las AA.PP. se extienden a la garantía de las condiciones de igualdad y a la defensa de los intereses generales. El acceso al código para facilitar la detección de *bugs*, troyanos, etc., y verificar el funcionamiento de la aplicación es algo inmediato si se utiliza software libre, si bien seguramente con los productos comerciales se podría conseguir el mismo efecto mediante su inspección durante el proceso de revisión en condiciones que garanticen la confidencialidad.

La ley de Linux («con infinitos ojos mirando, todos los errores saldrán a la vista») es un factor que, además de aportar calidad al código, genera una confianza que, en el momento de la verdad, para la mayoría de los usuarios se limita a poder seleccionar el suministrador de distribuciones de software libre.

Ponemos nuestra confianza en la pulcritud en la inclusión de programas y la revisión del código de esos suministradores, aunque realmente la actividad de los integradores de esas distribuciones se centra en lograr que funcionen sin problemas. En la práctica confiamos pues en el prestigio de las empresas, igual que ocurre con el software comercial. Por tanto este argumento debe utilizarse con cuidado hasta que las AA.PP. puedan implantar realmente un mecanismo de inspección en la selección y distribución de software libre,

tema este cuyas implicaciones legales se llegan a vislumbrar al final del estudio sobre el uso de **OSS** (*Open Source Software*, en el sector público, que será promovido por IDA como su siguiente acción.

2.2.1. El desarrollo de la jornada

Las ponencias mostraban ejemplos reales en los que la escalabilidad era un factor que formaba parte del diseño. Las iniciativas se debían a personas con inquietudes técnicas, aunque también hubo intervenciones con una perspectiva jurídica.

El uso de software libre se centraba en servidores y servicios gestionados por informáticos. El puesto de trabajo de usuario es aún es la asignatura pendiente, aunque empiezan a aparecer alternativas realmente viables, que en todo caso deben buscar compatibilidad con el enorme número de aplicaciones disponibles en el entorno dominante. La mayoría de los pequeños desarrolladores se ciñe al estándar de facto para no limitar su de por si concreta clientela. La propia situación dominante tiene un efecto autoamplificador: las academias sólo enseñan los programas ofimáticos que la mayoría tiene y la gente luego pide usar lo que aprendió, sin tener que aprender otra herramienta, de manera que pueda compartir con su entorno ficheros y trucos. Es la fuerza de la compatibilidad que tan buenos efectos ha traído al mercado informático y cuyo éxito se debe más al interés de los usuarios por intercambiar información que al buen hacer que hay que reconocer a las empresas que han sabido liderar este deseo de la mayoría.

Las distintas intervenciones coincidían en los valores que habían conducido al uso de software libre. Me gustaría destacar un par de ideas que me parecieron especialmente interesantes:

- Un gran ayuntamiento con miles de empleados públicos aprovechaba el amor propio de sus programadores como factor motivador en la producción de software de calidad. Al saber que se va a publicitar el código fuente, los autores tienen en mente que sus colegas les van a evaluar y en consecuencia ponen su máximo esfuerzo en ganarse ese reconocimiento profesional, teniendo en cuenta que este reconocimiento se produce en los niveles más altos de la pirámide de motivación dentro de organizaciones donde los niveles más bajos de dicha pirámide son muy sólidos.
- Otro organismo exponía la insoportable presión de una empresa que les había desarrollado una aplicación y que, al considerarse única conocedora de la misma, se permitía unos precios fuera de mercado ante la barrera de entrada que suponía su desconocimiento para la competencia. La publicitación del código permitió la concurrencia con conocimiento de causa de otras empresas, que de esta manera tuvieron posibilidad de ofrecer la mejor solución.

2.2.2. Conclusiones oficiales del simposio

De los casos de estudio presentados en el simposio se destaca la existencia de mucha experiencia práctica en el uso de software libre en las AAPP. europeas, que eligen este tipo de soluciones (para las que no existe presión comercial) por su funcionalidad, menor coste, independencia del vendedor, cumplir estándares y por interoperabilidad y seguridad.

Entre los temas tratados en la mesa redonda se destaca que hay requisitos como la trasparencia y fiabilidad que pueden lograrse mejor si el software utilizado es libre, aunque sólo en algunos campos hay ya soluciones alternativas y maduras. En cualquier caso, quienes toman las decisiones necesitan tener datos para confiar en este tipo de soluciones, limitándose en ocasiones a los productos más conocidos (Apache, Linux, Samba).

También se destaca la mayor importancia que este factor debería tener en la licitación pública, en sintonía con las declaraciones provenientes de la Dirección de Informática de la Comisión: «Hemos actualizado nuestras guías internas de evaluación para que se incluyan sistemáticamente productos de software libre en nuestras evaluaciones».

Los profesionales del sector deberían estar mejor informados de las disponibilidad de estas soluciones y cómo se han utilizado fructíferamente en organizaciones similares. Como consecuencia de esta propuesta surge la acción prevista para diciembre de 2001 para la creación de un foro de intercambios.

2.3. Estudio sobre el uso de software libre en el sector público

Los objetivos del estudio son:

1. Analizar el efecto que puede tener el software libre sobre los costes de propiedad (**TCO**, *Total Cost of Ownership*) en las AA.PP.
2. Difundir el conocimiento del mismo en el sector público, para que las AA.PP. exploren cómo estar más próximas a sus ciudadanos en los procesos de toma de decisiones.

2.3.1. Primera parte

Tras introducir el concepto de software libre, analiza su origen, los agentes que participan y factores para utilizarlo, mitos... Presta especial atención a las condiciones que imponen las licencias que limitan la responsabilidad de quienes lo distribuyen y sus implicaciones. Por último presenta una clasificación que permita localizar mejor las soluciones disponibles, y adjunta un amplio anexo con bastantes productos y sus características.

2.3.2. Segunda parte

Como resultado de unos cuestionarios y entrevistas realizados entre abril y agosto del 2001, el estudio describe las experiencias de distintas organizaciones de seis países europeos y de la Comisión Europea. El resumen de los datos recogidos se refleja en el siguiente cuadro:

- Francia, con sus iniciativas legislativas, es el país que más decididamente promueve el uso de software libre. El 5 de marzo de 2001 el Ministro responsable de los servicios públicos, Michel Sapin, en el primer «Día del software libre en el sector público», destacaba que en el complejo y dinámico mundo de las tecnologías de la información el respeto a los estándares abiertos es la principal garantía para preservar las inversiones públicas a largo plazo, y reconocía la positiva aportación de la comunidad que promueve el software libre para mejorar los servicios y aumentar la

trasparencia de las instituciones.
· Alemania es el otro país con mayores apoyos. El Ministro Federal de Economía y Tecnología (**BMWi**), Siegmar Mosdorf, afirma [ALE] que «el desarrollo de software libre puede ser el modelo base en Europa para la edad de la información». Entre las acciones concretas destaca el proyecto BerliOS [BerliOS], que proporciona una estructura para desarrolladores, distribuidores y proveedores de servicios, y el desarrollo de **GnuPG**, una herramienta de cifrado fiable y libre en cuya financiación ha participado el BMWi.

En el resto de países las experiencias se limitan a utilizaciones concretas de aplicaciones libres con mayor o menor extensión. En la Comisión Europea, donde se ha llegado a un cierto nivel de estandarización en las Direcciones Generales gracias a la labor de la Dirección de Informática (**DI**) que tiene su sede en Luxemburgo, no se contaba estratégicamente con el software libre.

La variedad de distribuciones Linux no favorece precisamente su objetivo unificador. Todo lo que sea unificar, como propone la coalición **LBS** (*Linux Standards Base*) [LBS] respecto a la «compatibilización» de la estructura de ficheros y librerías, ayudará en este objetivo ya que en una gran instalación se parte de una única distribución y luego se puede querer integrar otro paquete no incluido en la misma.

La DI estaba especialmente preocupada por contar con un soporte técnico garantizado y partía además de que las licencias suponen una parte no fundamental del coste de propiedad (TCO). La adquisición de PCs con Windows ha llevado a este alto nivel de estandarización y soporte, y es éste último el factor que más les preocupa. Pero recientemente han actualizado sus guías internas de evaluación para requerir sistemáticamente la inclusión de alternativas libres si están disponibles. Linux, Apache... están en proceso de pasar al estado de «con soporte». No se trata de renunciar a los mejores productos del mercado. Al contrario. El software libre surge en cierta medida para mejorar la calidad y fomentar la competencia.

2.3.3. Tercera parte

De las razones por las que se usa software libre en el sector público, las cuatro más importantes son la interoperatividad, la seguridad, el respeto a estándares y las funcionalidades. Pero la falta de personal especializado dificulta que los gestores de tecnologías de la información puedan sacar el mayor provecho del gran número de proyectos libres, cada uno de los cuales está en un diferente nivel de madurez.

Además de analizar por qué se usa o no en el sector público analiza cómo evoluciona su mercado, se centra en las implicaciones que tiene en cuanto a coste de propiedad (TCO), y en un primer análisis de los factores legales que tanta importancia tienen en las AA.PP.

El uso de software libre en muchas organizaciones comienza siendo anecdótico, posibilitando proyectos que no se podrían arrancar si fuese precisa la previa adquisición de productos comerciales. Los técnicos ponen su interés y sortean la limitación utilizando herramientas sin coste y con el soporte que da Internet. En ocasiones se ven reducidos en funcionalidad y deben integrar distintas herramientas para completar sus necesidades, pero con software libre siempre hay otras alternativas. Por ello y por la inmadurez de las soluciones para el puesto de trabajo de usuario, su uso se concentra en soluciones de infraestructura que al usuario le resultan transparentes.

Un factor fundamental en la adopción de software libre es la escalabilidad. En grandes instalaciones donde el presupuesto es limitado, la dependencia de un modelo de licencias ha de ser tenida en cuenta cuando se diseña o elige cada componente de la solución. Los costes a los que comprometen las futuras implantaciones pueden llegar a ser considerables, cuando lo que realmente implica el crecimiento del parque es el aumento de demanda de servicio. Y la situación se complica con el exceso de complejidad que se incorpora a los paquetes ofimáticos, la frecuencia de las actualizaciones, parches, etc.

La flexibilidad del software libre, debida a la posibilidad de comprender y modificar los fuentes e incluso difundir las modificaciones para que otros puedan mejorarlas, facilita la integración incluso con sistemas propietarios, aunque requiere una cualificación relativamente elevada (en especial porque exige conocer cómo funciona y no limitarse a instalar cajas negras). El resultado de ese sobreesfuerzo es proporcionarunos sistemas en los que se termina confiando porque se conocen bien y realmente funcionan: nadie hablaría de Linux ni de Apache si no fuesen tan robustos y estables. Hay muchas cosas gratis en el mundo, y no por ello les otorgamos ese valor.

Sin derivar de una decisión estratégica (aunque ya hay directivos en diversas Administraciones Europeas que sí promueven su uso de software libre, e incluso normativas en varios países que animan a su uso si es viable como alternativa), las organizaciones van dependiendo cada vez más de soluciones mixtas, donde el software libre convive con el software comercial. No se trata de eliminar uno por el otro. Los periódicos hace años que hacen convivir los dos modelos de propiedad de la información: la que es de dominio público y las exclusivas.

Por supuesto hay también casos en los que tras una experiencia piloto desarrollada sin medios económicos se decide invertir y acudir a un producto comercial, con una marca conocida que lo respalde, de manera que la responsabilidad del correcto funcionamiento quede fijada por un contrato. La escasez de funcionarios cualificados en TIC, sobre los que se concentran cada vez más proyectos, y su movilidad laboral pueden percibirse como una dependencia peligrosa. El soporte comercial a productos libres es aún incipiente, y en general motivado por la propia demanda de los clientes. Y los escurridizos anuncios de las grandes empresas de servicios no han creado en los directivos una sensación de confianza. Apostar definitivamente por un modelo en el que las modificaciones que puedan hacerse sobre el código no tienen por qué convertirse en propietarias, ni quedar sin mantenimiento o bajo la dependencia de quien las codificó, es un salto que algunas empresas han dado porque el modelo tradicional no

les era favorable, o porque el núcleo de su negocio no es el software, o porque persiguen un modelo de negocio alternativo. Quizá pueda ser también un modelo válido para las AA.PP. como generadoras de software en continuo proceso de mejora, y donde el servicio es lo que importa.

Disponer del código fuente es además garantía de continuidad. Además la propia evolución del grupo que le da soporte permite prever a la evolución de la funcionalidad en ese proceso de innovación continua. Las empresas justifican como estrategia comercial los cambios en los formatos que no están justificados técnicamente, y esto hace que el resto del software tenga que ir un paso por detrás. Es el momento de plantearse las ventajas de estar a la última («versionitis») frente a no depender de forma tan absoluta.

2.3.4. Recomendaciones

El informe termina reconociendo la falta de datos y la importancia de que los gobiernos tomen conciencia del tema. El uso de software libre implica un giro en el modelo, pasando de «de licencias a servicios». Debería eliminarse la obligatoriedad del uso de herramientas propietarias aunque sean gratuitas, por crear discriminación. El efecto «compatibilizador» debe basarse en estándares abiertos y los gobiernos deberían invertir en la creación de estructuras de datos en XML y en compartir para evitar reinventar la rueda una y otra vez.

3. IST

IDA no es el único programa europeo que fomenta el uso de software libre; el programa **IST** (*Information Society Technologies*) [IST] tiene un presupuesto de unos 3.600 millones de Euros (1998-2002), gestionado por la Dirección General para la Sociedad de la Información de la Comisión Europea. También participa un comité que cuenta con representantes de los estados miembros. En el contenido del programa de trabajo asesoran veinticinco expertos independientes. IST unifica y extiende los programas ACTS, Esprit y Telematics Applications.

Su objetivo estratégico es maximizar los beneficios de la sociedad de la información para Europa, acelerando su implantación y asegurando que se satisfagan las necesidades de sus empresas y ciudadanos. Las orientaciones de este programa se pueden resumir en «comenzar a crear las condiciones para implantar servicios y aplicaciones en Europa de forma cohesionada en base a bancos de pruebas y software libre, fomento de la amigabilidad y desarrollo y convergencia de las infraestructuras de redes europeas a nivel mundial».

Dentro de las acciones propuestas hay un fuerte impulso para el desarrollo de software libre. Las referencias a este tipo de software están diseminadas por el sitio web de IST [IST2]. Por ejemplo en <http://www.cordis.lu/ist/bwp_en5.htm> se encuentra la siguiente línea de acción:
- IST2001 - IV.3.3 Desarrollo de software libre: hacia una masa crítica.
Objetivos:
1. Fomentar en Europa la constitución de una masa crítica para el desarrollo de software que se libere bajo licencias compatibles con GPL.
2. Lograr la disponibilidad en Europa de servicios de soporte a proyectos de software libre.

El 18 de mayo tuvo lugar una primera sesión informativa [IST3] dentro de esta línea de acción.

4. Conclusiones

Lo importante de estas iniciativas es que actúan en sustitución de una actividad comercial que no existe para el software libre. Es importante dar a conocer, promocionar, generar confianza en quien tiene que decidir su uso. En ese sentido la actuación de IDA y su estudio de casos reales que afrontan problemáticas similares es muy importante. «Ya no tiene usted que 'tirarse a la piscina' el primero. Alguien le ha precedido y tuvo éxito». El siguiente paso sería «Si va a hacer usted lo mismo que él, pónganse en contacto y colaboren».

En España el Estado de las Autonomías ha fomentado que distintas administraciones compitan por ofrecer a los ciudadanos de su región geográfica servicios comparativamente mejores a los de las regiones vecinas o a los ofrecidos por la Administración General del Estado. La interconexión entre administraciones (local, autonómica, estatal y comunitaria) se ve dificultada por la variedad de sistemas de información, lo que además supone un gasto adicional para el ciudadano, que paga distintos programas para gestionar la misma tramitación sólo por estar en zonas geográficas limítrofes.

Las restricciones presupuestarias podrían superarse con una mayor rentabilidad del gasto público mediante la creación de grupos coordinados y proyectos comunes, en los que el software no solo quede libre entre los participantes, sino incluso para terceros. Hay quienes rechazan la financiación pública para el desarrollo de software libre, arguyendo que sólo el mercado decidirá sobre el futuro del software libre. También es rechazada esta financiación por defensores del mismo, como Eric Raymond, quien defiende que el éxito del llamado «modelo bazar» se deberá a la calidad lograda y por tanto no desea intervencionismo ni regulación. Pero ¿por que no puede una organización que trabaja para el bien social beneficiarse de los bienes que nos depara la sociedad? Aunque sólo un pequeño porcentaje de usuarios colabore en el desarrollo de software libre, el mero hecho de la difusión de su uso y la demanda de correcciones tiene un efecto beneficioso tan importante como para acallar a quienes puedan pensar que es negativo que los impuestos locales puedan servir para favorecer a terceros.

5. Referencias

[IDA] <http://europa.eu.int/ISPO/ida/>
[ALE] <http://www.internetnews.com/intl-news/article/0,,6_408271,00.html>
[LBS] <http:// http://www.linuxbase.org/>
[BERLIOS] < http://www.berlios.de/index.php.en>
[IST] <http://www.cordis.lu/ist/>
[IST2] <http://www.cordis.lu/ist/bwp_en6.htm#fet>
[IST3] <http://www.cordis.lu/ist/ka4/tesss/prog.htm>

Open Source en un gran banco suizo

Klaus Bucka-Lassen, Jan Sorensen
aragost AG

{<klaus.bucka-lassen, jan.sorensen@aragost.com>}

Traducción: Julio Ayesa, Agustín Palomar (Grupo de Lengua e Informática de ATI)

Resumen: *¿qué podría inducir a un gran banco suizo a usar un producto* ***Open Source*** *(código fuente abierto)? Este artículo usa el ejemplo de* ***Jakarta Struts*** *para explicar las ventajas y desventajas del software Open Source y remarcar cuáles de éstas son significativas para un proveedor de servicios financieros. En él se describen los problemas que aparecen y que es los que convence a la dirección a usar Struts para el desarrollo de su aplicación Web.*

Palabras clave*: Jakarta Struts, Open Source, gran banco suizo, ventajas y desventajas, experiencia.*

1. Los comienzos

Al igual que otros «productos de moda», *Open Source* (código fuente abierto) nació en un entorno académico con productos como **Emacs**, **LaTeX**, **Mosaic,** varios compiladores (**C**, **C++**, **Smalltalk**, etc.) y software asociado [GNU, Mosaic]. Y fue adoptado a continuación por usuarios personales.

No hace mucho tiempo, alguien que se hubiera atrevido a comparar **Linux** con **Windows** hubiera causado risa . Hoy, a la vista del impresionante éxito experimentado por este sistema operativo, sus críticos están siendo silenciados. La cuota de mercado de Linux está aumentando constantemente, incluso en el entorno comercial.

Linux ha dejado de ser un producto para idealistas y enemigos de Microsoft y se ha convertido en un serio opositor de Windows. El valor de un sistema operativo es mas difícil de establecer que el de una aplicación de software. Un sistema operativo *per se* casi no se usa --sólo las aplicaciones que son desarrolladas para este sistema operativo hacen que sea interesante. El mejor sistema operativo del mundo no tendría éxito sin el software. puesto que requiere una masa critica de usuarios y desarrolladores. A pesar de esto, Linux ha sido un éxito. Esto demuestra que *Open Source* no es simplemente una moda pasajera. Las razones del éxito de Linux son probablemente las muchas ventajas, inherentes en parte, que el software *Open Source* ofrece en comparación con el software propietario. Esto queda remarcado mas abajo en un ejemplo tomado de nuestra propia experiencia.

2. Struts

Struts es una estructura *Model-View-Controller* (**MVC**) de *Open Source* para aplicaciones Web que está construida usando **Java** y **JSP/Servlets** [Struts]. Un gran banco suizo recientemente decidió usarlo como marco para su desarrollo de aplicaciones Web internas .

Model-View-Controller es un patrón de diseño ampliamente aceptado que define la construcción lógica , o arquitectura, de una aplicación.

Model describe el objeto de negocio y la lógica asociada. *Wiew* refleja su presentación en la pantalla (**GUI**). MVC estipula que estos dos elementos sólo pueden comunicarse a través de un C*ontroller* y, por tanto, están separados uno del otro. Esto significa, por ejemplo, que es relativamente sencillo modificar la vista (*view*) sin que el modelo se vea afectado. La experiencia ha mostrado que esto es un enfoque muy bueno que hace más económico el desarrollo y, en un nivel inferior, el mantenimiento de la aplicación.

Struts es una estructura que soporta principalmente y simplifica el desarrollo del *Controller* y la *View.* El desarrollo de la lógica de negocio no se ve influenciado por Struts.

Struts, junto con otros productos como **Log4J** y **Tomcat**, llega bajo los auspicios del proyecto **Jakarta** promovido por la Fundación de Software **Apache**. Es relativamente nuevo, pues la versión 1.0 fue liberada en Junio de 2001.

Existen varias maneras diferentes de trabajar con Struts. En primer lugar, uno puede involucrarse activamente en las discusiones sobre Struts como usuario de listas de correo. Se puede informar de fallos y sugerir nuevas características. Pero también uno mismo puede desarrollar nuevo código para Struts -- en este caso un comité decide si será incluido en la próxima versión.

Autores

Klaus Bucka-Lassen es Master en Informática por la Universidad de Aarhus (Dinamarca). Ha estado trabajando durante cinco años como desarrollador de Smalltalk y Java en Suiza y Australia. Actualmente trabaja como arquitecto de sistemas para un proyecto piloto de Struts en uno de los principales bancos suizos.

Jan Sorensen es Master en Informática por la Universidad de Aarhus (Dinamarca). Ha estado trabajando durante siete años como desarrollador de C++, Smalltalk y Java en Dinamarca, Suiza y los Estados Unidos. “Contribuyente” a Struts.

Ambos son co-fundadores y co-propietarios de aragost AG, empresa especializada en la consultoría y el soporte de productos *Open Source* en un entorno orientado a objetos (Smalltalk y Java). <http://www.aragost.com/>

3. Struts en funcionamiento

Desde el principio de año, Struts ha sido usado en un proyecto piloto en este gran banco. ¿Qué ventajas se ha obtenido el banco debido al hecho que Struts sea software *Open Source*?

Seguridad. El hecho de que el código fuente está disponible públicamente asegura que no ocurre nada que no debería ocurrir. Esto es fundamental para un banco, puesto que no debería haber ningún agujero en la seguridad de una aplicación bancaria. Es casi imposible construir «puertas traseras» en Struts, pues sería claramente visible para cualquiera que mirara el código.

Robustez y fiabilidad. Incluso teniendo en cuenta que Struts es un producto totalmente nuevo, da la impresión de ser altamente estable. Los fallos son rápidamente localizados y solucionados puesto que el código fuente está siendo examinado constantemente. Por otra parte, los desarrolladores escriben mejor código si saben que va a ser publicado. Uno de los autores de este articulo identificó recientemente un fallo en Struts. Se refería específicamente a un método que debería estar «a salvo de amenazas» pero no lo estaba. Descubrió el fallo por casualidad mientras leía código fuente**,** corrigió el fallo y envió una versión corregida. Hasta entonces nadie había encontrado ese problema porque el código sólo se comportaba erróneamente bajo ciertas circunstancias no habituales . Esto significa que fue prácticamente imposible reproducir el fallo haciendo mucho más difícil localizarlo. ¡Así es como los programas *Open Source* llegan a ser más robustos y estables!

Formación, documentación y soporte. Se ha escrito ya un considerable número de artículos, recomendaciones, instrucciones y ejercicios en relación con Struts. El soporte mediante las listas de correo funciona de una manera considerablemente mas rápida efectiva y amigable que lo que uno puede esperar de la mayoría de los más importantes suministradores de software.

Facilidad para encontrar fallos. Localizar fallos en el código propio es un arte (no estamos hablando de fallos en el código Struts). Generalmente se usa un *debugger* para este propósito, con el cual la aplicación puede ser ejecutada línea por línea. Este procedimiento sólo es posible porque Struts es un producto *Open Source*.

Modificabilidad y Extensibildad. El código Struts puede ser modificado, sin embargo, si uno quiere mantener la posibilidad de posteriores migraciones a nuevas versiones de Struts sin grandes problemas, no es recomendable hacer modificaciones permanentes en el código Struts. Sabido esto, pueden ser muy útiles modificaciones temporales, por ejemplo, para registro de datos (*logging*). También pueden ser solventados de esta manera toda clase de fallos durante un corto periodo de tiempo a la espera de las «soluciones oficiales».

Evaluación. Como Struts es un producto *Open Source*, se puede probar cualquier clase de situación sin el riesgo de infringir la licencia. Esto significa que se pueden hacer evaluaciones en profundidad. Si Struts fuera un software propietario la decisión de usarlo o no hubiera debido tomarse antes de que fuera posible probar una versión completa del producto.

Precio de compra. Para hablar de todo, el precio de compra también debe ser mencionado aquí. comparado con las otras ventajas mencionadas anteriormente, el hecho de que Struts esté libre de cargo no es un argumento particularmente significativo en su favor -- en particular no para un gran banco.

4. El precio

La evaluación de costes forma parte de cualquier empresa seria. Los costes en un escenario Open Source son fáciles de confrontar con los siguientes hechos:

- **Falta de responsabilidad del producto.** Con *Open Source*, nadie es responsable del producto. Por lo tanto, nadie es culpable de ningún daño que pueda surgir como resultado de un producto *Open Source* defectuoso. En este contexto, sin embargo, habría que señalar que los productores de software propietario también afirman que no pueden asumir responsabilidades por ningún daño que pueda surgir del uso de su producto (*disclaimer*, o exención de responsabilidad).

- **No hay un punto de soporte oficial.** Los productores de productos propietarios ofrecen a menudo un servicio de soporte caro. Éste no es el caso habitual con el software *Open Source*. Por eso, es necesario establecer un punto de soporte interno, o hay que obtener soporte de una tercera empresa especializada en el producto *Open Source*. Además, para el banco que se menciona en este artículo (y probablemente para la mayoría de otras grandes empresas similares) las normas y reglas internas para empleados les prohíben en mayor o menor medida tomar parte en foros de discusión en Internet (lo que impide el uso de soporte directo mediante la lista de correo de usuarios de Struts, o al menos lo dificulta).

- Dificultades para planificar la disponibilidad de nuevas versiones. Como los desarrolladores disponibles en un proyecto *Open Source* difícilmente pueden ceñirse a una planificación, es difícil, por no decir imposible, establecer un plan de disponibilidad de nuevas versiones. La resolución de errores y la implementación de nuevas prestaciones no puede garantizarse, ni tampoco pueden darse fechas para arreglos (*fixes*). Antes de que esto ocurra, hay que encontrar un voluntario con tiempo y ganas suficientes. Para la mayoría de las aplicaciones, las ventajas parecen merecer la pena (pero a pesar de ello, no es del todo fácil convencer a los directivos de un gran proveedor de servicios financieros de que utilice software *Open Source*). Para conseguir la aprobación para utilizar Struts en un gran banco, hubo que sortear algunos obstáculos, no solamente los que hunden sus raíces en los prejuicios y la incertidumbre al enfrentarse a lo desconocido, sino también los que tienen que ver más con intereses personales.

5. Argumentos que convencieron finalmente a los directivos a favor de Struts

- La credibilidad de la organización que hay detrás de Struts.

La Apache Software Foundation es reconocida y ya ha llevado al mercado otros varios productos *Open Source* con éxito. El más bien conocido de ellos es el servidor http Apache, el líder del mercado entre los servidores Web. Incluso **Websphere** de IBM se basa en Apache.

- **Referencias.**Como ya se ha mencionado, innumerables artículos y ejercicios sobre Struts están disponibles en la Internet. También está disponible un libro que dedica un capitulo completo a Struts [JSP]. Pero lo que convenció a la dirección más que nada fue el hecho de que IBM canta las alabanzas de Struts en su página oficial [IBM].
- **Entrega definitiva.** Ahora está disponible una entrega definitiva de Struts. Pero un par de meses antes de la decisión, únicamente se disponía de una *pre-release*.
- **Simplicidad e independencia**. Struts es simple y los desarrolladores aprenden a usarlo rápidamente. Struts se construye exclusivamente sobre tecnologías no propietarias, ampliamente probadas y examinadas tales como Servlets, **JSP** y **XML**.
- **Compatibilidad.**Struts podrá ejecutarse en todos los principales servidores de aplicaciones Web incluyendo **Tomcat**, Websphere de IBM, **WebLogic** de BEA y **SilverStream**.

En el peor caso, el código de *Open Source* es «tan malo como cualquier código escrito en casa». O por decirlo de otra forma, usted puede descargar Stratus y utilizarlo como punto de inicio para su propio producto; nadie va a obligarle a migrar a nuevas versiones de Struts. Puede enmendar Struts para adaptarse a sus propias necesidades. La alternativa a esto sería reinventar la rueda una vez más, lo que podría significar unos costes considerablemente más altos y un «tiempo de puesta en mercado» más prolongado.

6. Conclusión

El software mencionado en este artículo, que se desarrolló usando Struts, ahora está ejecutándose con éxito en un entorno real. Sin Struts, el desarrollo habría tomado mucho más tiempo y la calidad, junto con la mantenibilidad, se habrían resentido. El éxito se debió en parte al hecho de que MVC es un buen patrón de diseño y Struts lo implementa excepcionalmente bien, pero no menos porque Struts es *Open Source* y por lo tanto tiene las ventajas mencionadas anteriormente. A diferencia de lo que podría parecer, no hubo problemas dignos de mención. La curva de aprendizaje adicional fue modesta. Lo que faltaron fueron casos de estudio y «patrones de la mejor práctica» de Struts. Tuvimos que aprender las cosas por el camino difícil.

7. El futuro

Según la lista de correo de usuarios de Struts, al menos otro gran banco internacional, Deutsche Bank, ha optado por Struts últimamente. Otros proveedores de servicios financieros, también en Suiza, están evaluando actualmente sus pros y contras.

8. Referencias

[Struts] Página principal de Jakarta Struts <http://jakarta.apache.org/struts/>

[OSD] Definición de *Open Source* <http://www.opensource.org/docs/definition.html>

[GNU] Página principal de GNU <http://www.gnu.org/≥

[Mosaic] Licencia Mosaic <http://archive.ncsa.uiuc.edu/SDG/Software/Mosaic/License/LicenseInfo.html>

[FSF] Fundación Software Libre <http://www.fsf.org/gnu/the-gnu-project.html>

[JSP] «JSP Profesional», 2ª Edición; Wrox Press Inc., 29 S. LaSalle St, Suite 520, Chicago, Illinois 60603, USA; **Simon Brown, Robert Burdick, Jayson Falkner, et al.**

[IBM] «Struts, an open-source MVC implementation» <http://www-106.ibm.com/developerworks/ibm/library/j-struts/>

[Linux] «The Cathedral and the Bazaar» <http://tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/>

Software Libre/Fuente Abierta: hacia la madurez

El proyecto GNU Enterprise: software de aplicación para la empresa

Neil Tiffin[1], Reinhard Müller[2]
[1]Performance Champions, Inc. (EE.UU.); [2]Bytewise Software, GmbH (Austria)

<neilt@gnue.org>
<reinhard.mueller@bytewise.at>

Traducción: José Alfonso Accino (Grupo de Lengua e Informática de ATI)

Resumen: *GNU-Enterprise (**GNUe**) es un proyecto de software libre que aspira a proporcionar una solución **ERP** (Planificación de Recursos de Empresa) comparable a **SAP R/3**. GNUe es un conjunto de herramientas y aplicaciones integradas de negocios con soporte para contabilidad, cadena de suministro, recursos humanos, ventas, producción y otros procesos de la empresa. Este artículo describe el proyecto, la idea y la motivación de los desarrolladores y usuarios que están detrás del mismo, así como su estado actual.*

Palabras clave: *aplicaciones integradas de negocios, SAP R/3, estándares abiertos, GNUe.*

1. Introducción

Imaginémonos la escena: Mr.Howard Harvard, Director General de Acme Magnetic Levitation, acaba de recibir malas noticias. La compañía está funcionando por debajo de las previsiones y necesita reducir costes temporalmente hasta que las ventas mejoren. Mr.Harvard echa una mirada a los costes más importantes, directivos, gastos de mantenimiento y gastos de licencias de software, y piensa para sus adentros: «¿No pagamos ya todo el software cuando lo compramos? Funciona bastante bien, no necesitamos prestaciones nuevas, no hemos utilizado el servicio de soporte técnico. ¿Por qué tenemos que seguir pagando licencias que no añaden valor a nuestra organización?». «Ah, bien» --se responde a sí mismo-- «si no pagamos esas licencias nos quitan el software. Parece un impuesto ¿verdad?, un impuesto por la utilización del software. Sólo con que pudiera quitarme este impuesto de encima, podría superar esta mala racha, conservar el personal clave y estar preparado para cuando la situación mejore».

La ventana se abre con un estallido y entre el viento y los cristales rotos aparece *GNUeman*, la solución al problema del impuesto sobre el software, y en su espalda está la clave: Software Libre. «Seguramente no puede ser tan simple», piensa Mr.Harvard, «ya he oído algo antes acerca del software libre pero, ¿qué es realmente?, ¿cuál es su economía?, ¿funciona? y ¿cómo conseguiré que mi superior lo comprenda?».

Mr.Harvard recuerda entonces lo que ha oído sobre algunos de los éxitos del software libre: **GNU/Linux**, **KDE**, **Gnome**, **Apache**, **PostgreSQL** y **MySQL**. «Pero todo esto son sólo herramientas», piensa, «¿cómo voy a llevar adelante mi negocio con software libre?».

GNUeman se sienta y empieza a hablarle de **GNUe**, un conjunto de herramientas y aplicaciones de negocios que automatizan las funciones empresariales.

2. Tome el control de su negocio

GNUe consta de los paquetes Contabilidad, Cadena de Suministro y Ventas. Todos los paquetes están estructurados para soportar múltiples monedas e idiomas. El paquete de Contabilidad incluye funciones básicas de libro mayor, pagos y cobros. El de Cadena de Suministro cubre pedidos, recepción, almacenamiento y salida de productos. Ventas incluye facturación y elmantenimiento de información sobre clientes. Futuros paquetes incluirán CRM, gestión de proyectos, fabricación y comercio-e.

«Estas funciones están bastante estandarizadas en el ámbito empresarial, así que ¿qué distingue GNUe de otras aplicaciones de empresa como **SAP**, **PeopleSoft** y **JDEdwards**?», pregunta Mr.Harvard.

Empecemos por algunos problemas típicos. Todo sabemos lo que es tener costes y pagar derechos sin recibir valor a cambio. ¿Qué ocurre cuando se llama al servicio de atención al cliente? ¿Consigue una respuesta rápida y acertada a lo que pregunta? Mr.Harvard niega con la cabeza. «Sólo cuando por fin damos con la persona adecuada que tiene la información correcta y, a veces, no ocurre nunca».

Así que, cuando está en un verdadero apuro, generalmente tiene que esperar y algunas veces ni siquiera consigue la

Autores

Neil Tiffin (EE.UU.) es presidente de Performance Champions, Inc., una firma de consultoría de gestión y negocios especializada en la mejora de la competitividad. Posee una licenciatura en Informática por la Universidad de Missouri. Su experiencia incluye la implementación de sistemas a gran escala en operaciones internacionales y responsabilidades de gestión en empresas de fabricación y distribución. Actualmente coordina el desarrollo de Objetos de Negocio para GNUe.

Reinhard Müller (Austria) es Director de Gestión de Bytewise Software GmbH, una firma austríaca que se ocupa del mantenimiento del software de negocios de más de 150 pequeñas y medianas empresas en Austria, Suiza y Alemania. Actualmente, coordina el desarrollo del Servidor de Aplicaciones de GNUe.

solución. Con GNUe, y con el software libre en general, los desarrolladores le dan toda la información que necesita para resolver el problema usted mismo, incluyendo todo el código fuente, todas las instrucciones para construir la aplicación y toda la documentación, incluyendo comentarios al código y notas sobre las interioridades del diseño. Los vendedores de software patentado consideran toda esta información como secretos comerciales.

«Y qué ocurre si mi equipo no tiene tiempo para aprender lo suficiente como para resolver el problema?» Entonces puede llamar a un consultor de GNUe. Lo más probable es que consiga acceder a un desarrollador si es necesario y no tenga que limitarse a un estudiante principiante de los que atienden las ventanillas de servicio al cliente.

«¿Así que el software libre no confía en los estudiantes?». GNUe hace buen uso de los estudiantes pero, al contrario que con los vendedores de software patentado, usted siempre sabrá cuándo está tratando con un estudiante, debido a la naturaleza abierta del proceso de desarrollo y soporte. Cuando se utilizan estudiantes, generalmente están comprometidos en áreas muy específicas y trabajando bajo la supervisión de un miembro veterano del equipo de GNUe.

«¿Se le paga al equipo de GNUe por su trabajo? Si no, ¿cómo puedo estar seguro de que habrá alguien disponible para responder mis preguntas?». Buena pregunta, que implica varias cuestiones. Primero, GNUe se basa en estándares de la industria, así que la solución a su problema no tiene por qué requerir necesariamente el equipo de GNUe. Ya que GNUe utiliza estándares abiertos, cualquiera tiene acceso a las muchas personas que trabajan con ellos, además de la gente que trabaja con GNUe. Por ejemplo, toda la comunicación multi-grado entre cliente y servidor emplea la funcionalidad del estándar **CORBA 2.3** proporcionado por ORBit. Segundo, GNUe tiene desarrolladores en Europa, Australia y Estados Unidos. Esto significa que 24 horas al día y, usualmente, 7 días a la semana, hay alguien en línea en el canal GNUe IRC. Nuestro tiempo de respuesta para resolver problemas ha sido tradicionalmente muy breve. Tercero, se dispone de un soporte de consultoría de pago de, al menos, cuatro organizaciones extendidas por Europa, Nueva Zelanda y Estados Unidos. Esperamos que el número de organizaciones de soporte vaya en aumento conforme nuestra solución se extienda por la industria.

El desarrollo de GNUe comenzó en julio de 1999 con el proyecto **Obelisk**. Este proyecto se fusionó con el proyecto Sanity en marzo de 2000 para convertirse en GNUe. Que nosotros sepamos, GNUe es el único proyecto multinacional de empresa activo con actualizaciones mensuales continuas desde octubre de 2000. Somos un proyecto formal auspiciado por la organización Free Software Foundation. Este impulso es una importante consideración para la madurez de un proyecto de software como GNUe.

Los desarrolladores de GNUe son una experimentada mezcla de usuarios, desarrolladores de software y consultores que conocen las necesidades empresariales diarias de compañías que van desde el empresario individual a la corporación multinacional. Los usuarios están más próximos al desarrollo

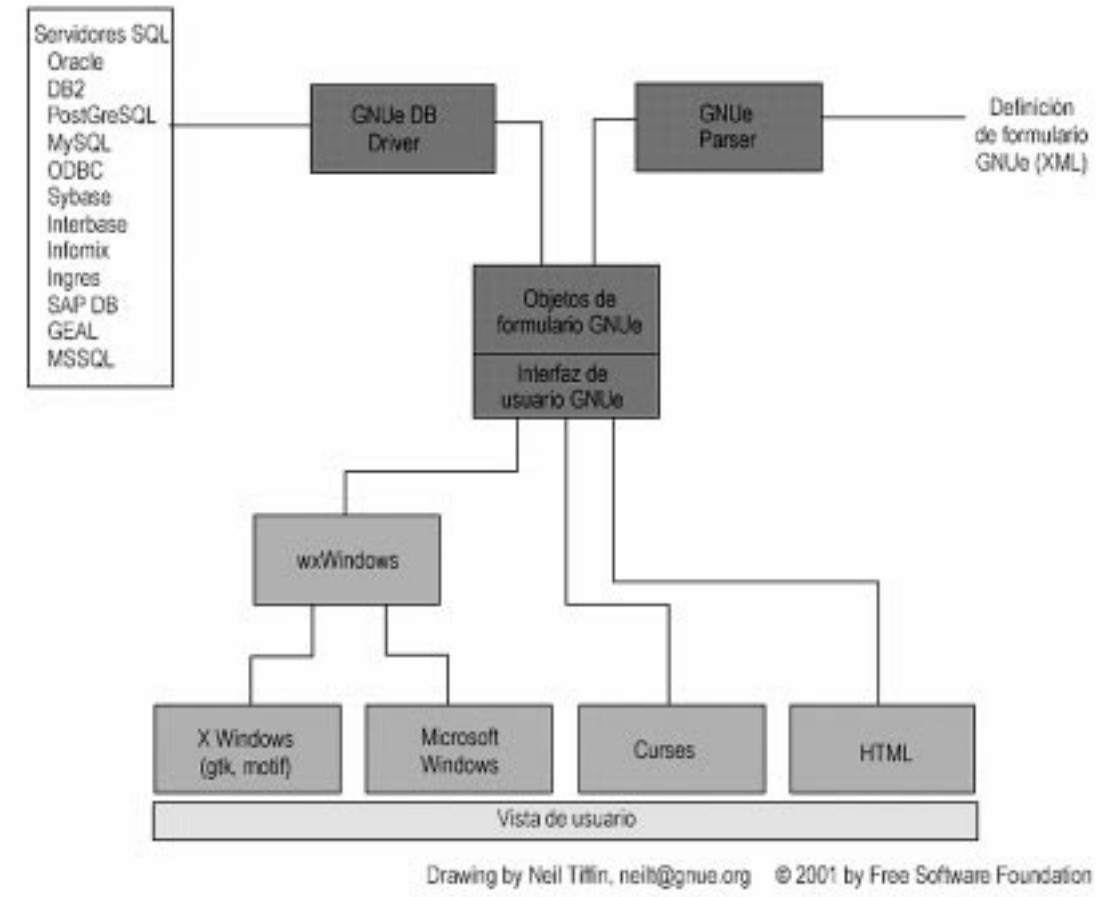


Figura 1. Arquitectura de formularios de GNUe

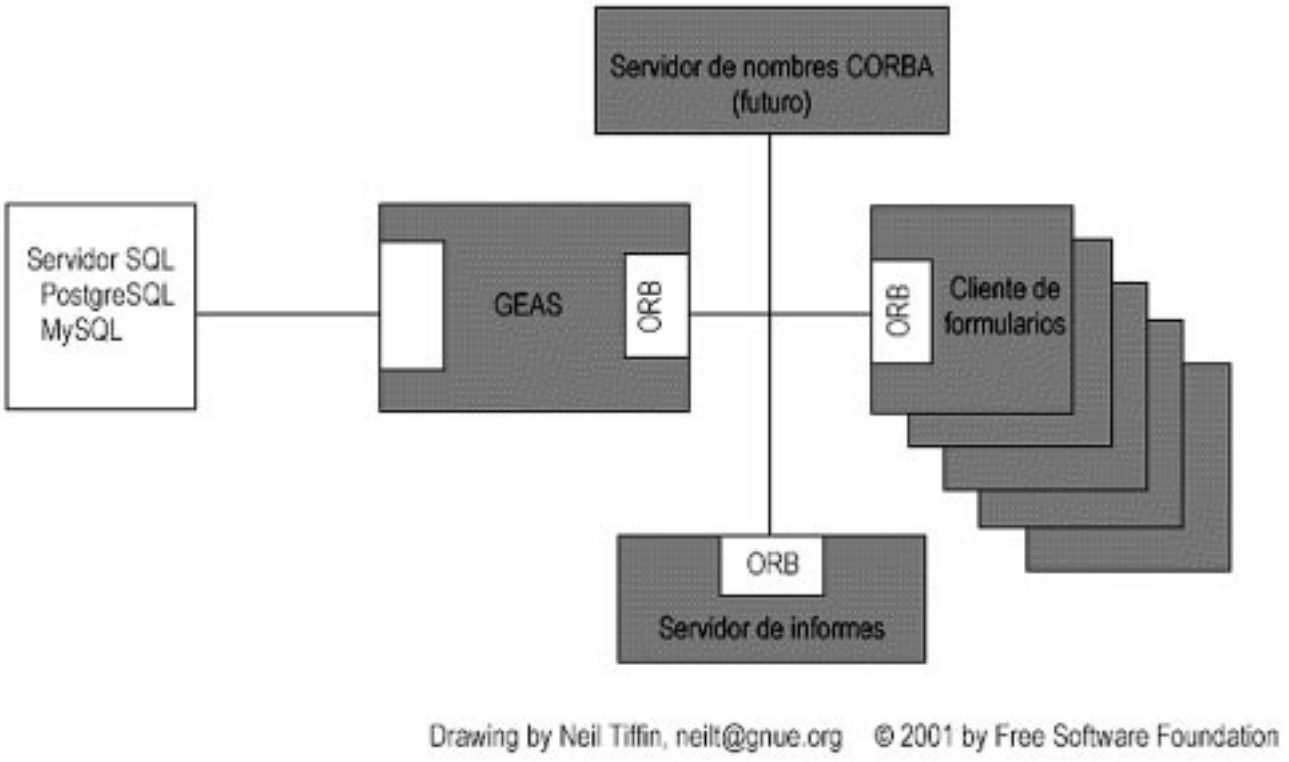


*Figura 2. Esquema simple de la arquitectura GEAS (*GNU Enterprise Application Server*) multi-grado*

de GNUe de lo que lo están a la mayoría de los vendedores de software patentado. Como ejemplo, véase en Computerworld de 17 de septiembre de 2001: «Oracle rehúsa participar en los congresos bianuales del Grupo de Usuarios de Aplicaciones de Oracle y ha propuesto que se integren en su evento AppsWorld, sugerencia que el grupo independiente de usuarios ha rechazado hasta el momento».

3. Herramientas GNUe

«Impresionante», dice Mr.Harvard, «pero ¿cómo encaja GNUe en mi infraestructura técnica y de negocio?». GNUe está estructurado para sostener dos tipos de infraestructura, cliente-servidor grado-dos y multi-grado.

Ambas versiones utilizan nuestro diseñador de formularios, cliente de formularios y cliente de informes. El diseñador de formularios es el centro de nuestras herramientas de productividad. Proporciona desarrollo de formularios por medio de un interfaz gráfico de usuario y es un instrumento básico para proporcionar a los desarrolladores del negocio una herramienta de construcción de sistemas que no necesita programación.

Las herramientas de formulario e informes de GNUe son independientes de plataformas y se ejecutan en **Windows**, **GNU/Linux**, **Mac OS X** y hasta viejos terminales de texto. Todos los formularios e informes se definen por medio del Lenguaje de Definición de Formularios de GNUe (**XML**). Ver la **figura 1**.

Los clientes GNUe de grado dos se conecta directamente a un servidor de bases de datos SQL, incluyendo ODBC, PostgreSQL, MySQL, Oracle y DB2.

GNUe de grado dos está diseñado para proporcionar una elaboración rápida de prototipos y la construcción de soluciones basadas en SQL para pequeñas y medianas empresas. Muchos ejemplos se incluyen con la distribución, como gestión de contactos, pedidos y facturación. GNUe de grados dos es la parte más madura de GNUe.

Las principales aplicaciones de negocios de GNUe se desarrollan alrededor de nuestra solución multi-grado, que nosotros llamamos **GEAS** (*GNU Enterprise Application Server*), que se muestra en la **figura 2**. El equipo de GNUe se encuentra en estos momentos en el proceso de integrar **Bayonne**, para aplicaciones de telefonía, y **Double Choco Latte**, para acceso web, en nuestro servidor. Esto permitirá acceder a todos los objetos de GNUe desde cualquier teléfono, fax o web.

Multi-grado utiliza el mismo cliente de formularios, escritura de informes y diseño de formularios que el de grado dos. La única diferencia es que el cliente utiliza un *driver* GEAS (y una interfaz **CORBA**) en lugar de un *driver* para servidor SQL. En un momento posterior proporcionaremos otros mecanismos de red para interfaz de clientes.

La principal razón para utilizar la versión multi-grado es la escalabilidad y la naturaleza, orientada a objetos, de los objetos de negocios.

Eventualmente, GEAS podrá soportar múltiples bases de datos con reparto de carga, bases de datos orientadas a objetos, objetos distribuidos y otras funciones multi-grado sofisticadas y escalables.

GEAS está diseñado para integrarse con muchos sistemas diferentes y no necesita cambiar los sistemas heredados a GNUe. Como nuestro cliente de formularios, se ejecuta en la mayor parte del hardware más popular incluyendo **Microsoft Windows**, **Apple Macintosh**, **Sun Solaris**, **GNU/Linux** y otros **Unix**.

4. Proporcionando funcionalidad empresarial a GNUe

A diferencia de otros sistemas, GNUe no está atiborrado de funciones, campos o datos innecesarios para una determinada instalación, sino que se puede configurar y utilizar sólo lo que se necesite.

El sistema GEAS está estructurado en *Plantillas*, *Paquetes*, *Módulos* y *Objetos de Negocio* (ver **figura 3**). Las Plantillas se utilizan para configurar GNUe para una industria específica y son una manera de modificar los Módulos. Este proceso de configuración puede añadir, modificar o suprimir funcionalidades básicas.

Los Paquetes son simplemente colecciones convenientes de Módulos. Los Módulos son grupos lógicos de objetos de negocio diseñados para ser utilizados conjuntamente, aunque no obligatoriamente. Los Objetos de Negocio, como Cliente, Artículo, Pedido y Factura, se modelan según las necesidades del negocio y contienen definiciones de datos y reglas de negocio. Los módulos también pueden añadir nuevos Objetos de Negocio o ampliar los existentes. Por «ampliar» queremos decir añadir campos adicionales y reglas de negocio. Los Objetos de Negocio se diseñan para una fácil mezcla e integración con otras soluciones de GNUe o de terceras partes. Las Reglas de Negocio son métodos de objetos que se pueden ejecutar localmente o en un servidor. Actualmente pueden escribirse en **C** o en **Python** y esperamos añadir **Java** y otros lenguajes en cuanto sea posible.

Para cada Módulo, se puede decidir si se desea instalarlo o no (naturalmente, teniendo en cuenta las posibles dependencias, que se han reducido al mínimo en cada caso). Sus Objetos de Negocio pueden verse de un modo diferente dependiendo de los Módulos que tenga instalados. Para poner en este sistema tanta potencia como sea posible, hemos hecho los módulos con granularidad muy fina.

Algunos de los primeros módulos desarrollados fueron Artículo, Moneda y Lenguaje. Estos módulos forman parte de un paquete básico, muy pequeño, que acompaña a todas las instalaciones. Por ejemplo, todas las cantidades monetarias se representan como objetos moneda que incluyen una cantidad, representada por un entero de 64 bits, junto con un decimal implícito de 16 bits. Por tanto, los cálculos financieros no utilizan datos de coma flotante. Cada objeto monetario debe tener una divisa asignada. En un sistema muy básico con sólo una moneda, esto ocurre de forma transparente. Así, los usuarios de tales sistemas no están obligados a tratar con otras monedas en formularios e informes. No obstante, cualquier sistema monodivisa puede convertirse a un sistema multidivisa simple-

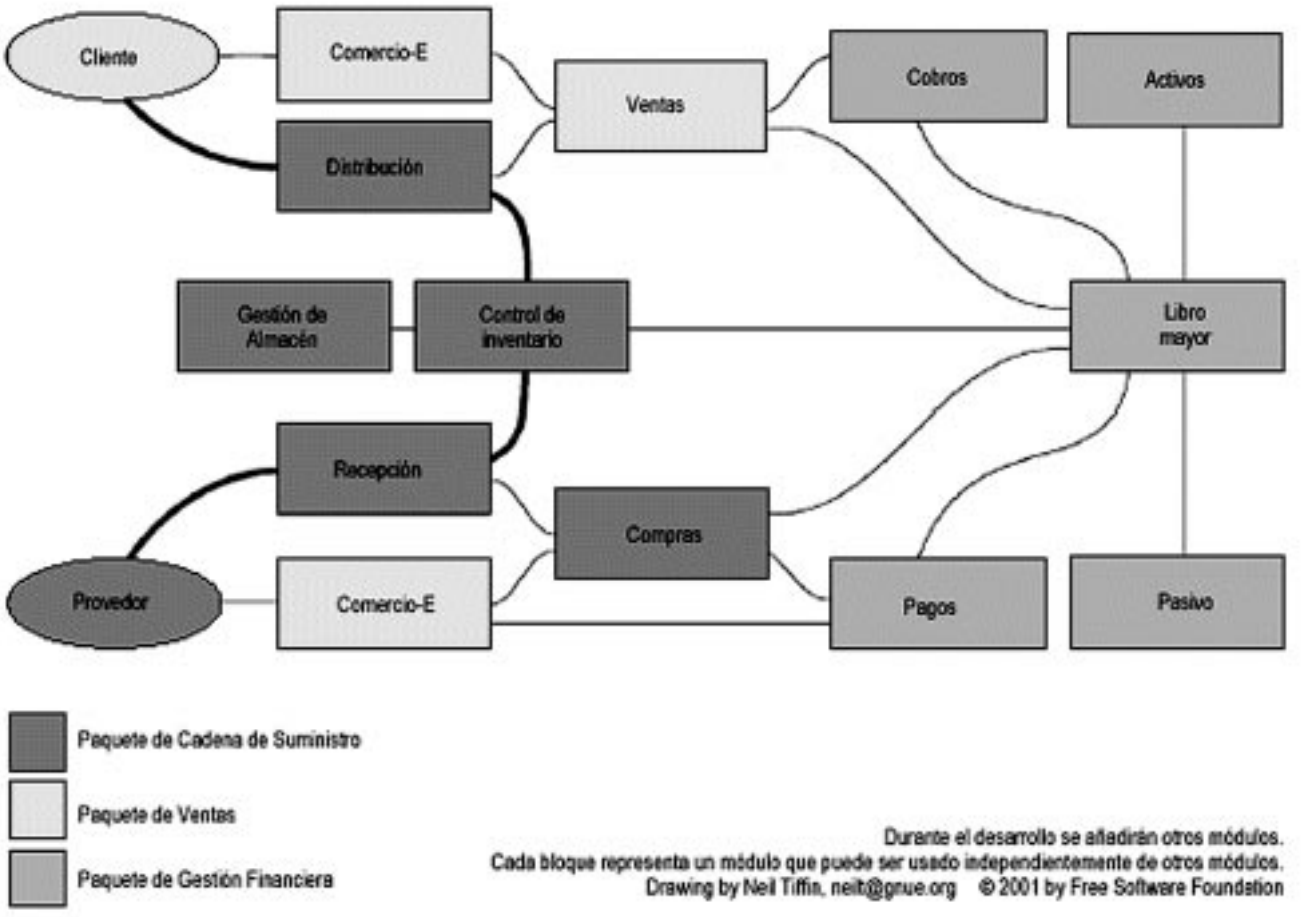


Figura 3. Arquitectura de Paquetes de la Aplicación Base de GNUe

mente añadiendo nuevas definiciones de moneda.

5. ¿Quién está utilizando GNUe?

«GEAS realmente encaja en mi negocio, pero simplemente no comprendo por qué la gente va a seguir apoyando el software libre si no ganan ningún dinero con él». Hay realmente tres tipos de negocios utilizando y apoyando GNUe. Las pequeñas empresas utilizan GNUe porque no tienen suficientes recursos para comprar o desarrollar una solución patentada y/o porque quieren la libertad y funcionalidad que GNUe les ofrece. Generalmente, adaptan sus negocios a GNUe y, de esta forma, pueden beneficiarse del trabajo de grandes empresas y consultores. Este modelo ayuda a los pequeños negocios proporcionando software competitivo a un precio que ellos pueden satisfacer.

El segundo tipo de usuario GNUe son las grandes empresas. Éstas utilizan GNUe porque no quieren verse encerrados en el software patentado. Las grandes empresas pueden hacer una pequeña inversión en GNUe proporcionando una pequeña cantidad de recursos. A cambio, obtienen un paquete que satisface sus necesidades a un coste mucho menor que si lo escribiesen ellos mismos. El tercer tipo de negocio incluye los consultores. El software libre es un excelente producto para consultores porque así pueden proporcionar soporte de pago cuando el usuario lo necesita. La organización del usuario no tiene que pagar continuamente un cuota y, además, es también libre para desarrollar su propia unidad interna de soporte si así lo desea.

6. Conclusión

GNUe es realmente beneficioso para compañías que quieran conservar la máxima libertad con la idea de controlar y sostener su inversión en sistemas de información, ya sean compañías de rápido crecimiento o grandes empresas que requieren la máxima flexibilidad.

Las ventajas reales de GNUe son su bien probada funcionalidad, los estándares abiertos, el no atarse a un solo vendedor, escalabilidad, documentación, ausencia de gastos de licencias, acceso a todo el código fuente, y la posibilidad de controlar los gastos de soporte por medio de los mecanismos de competencia del mercado. Estos beneficios se aplican a todas las empresas, cualquiera que sea su tamaño, y favorecen una atmósfera de abierta confianza y asociación entre los desarrolladores y los usuarios.

7. Referencias

Steve Cardell, Neil Tiffin; «Taming the beast» (Domesticando a la bestia), *Industrial Management*, Mayo-Junio 1999, pags.23-28. Un artículo de seis páginas subrayando los seis pasos necesarios para una implementación con éxito de sistemas de empresa.
Free Software Foundation, <http://www.fsf.org/>
GNU GPL, <http://www.fsf.org/licenses/licenses.html>
GNU Enterprise (GNUe), <http://www.gnuenterprise.org/>
Bayonne, <http://www.gnu.org/software/bayonne/>

Contando patatas: el tamaño de Debian 2.2

Jesús M. González-Barahona, Miguel A. Ortuño Pérez, Pedro de las Heras Quirós, José Centeno González, Vicente Matellán Olivera
Universidad Rey Juan Carlos

{<jgb, mortuno, pheras, jcenteno, vmo@gsyc.escet.urjc.es>}

Resumen: *Debian es la mayor distribución de software libre, en su última distribución estable supera ampliamente los 2.800 paquetes de código fuente. Es mayor que las demás, pero ¿cuánto exactamente? En este artículo usamos el sistema «sloccount» de David Wheeler para determinar el número de lineas físicas de código fuente (*physical **SLOC**, Physical Source Lines of Code*) de Debian 2.2 (conocida como «potato»).*

Veremos como Debian 2.2 incluye más de 56.000.000 líneas físicas de código, casi el doble que Red Hat 7.1, distribuida ocho meses después. Esto muestra que para distribuciones de este tamaño, el modelo de desarrollo de Debian, basado en el trabajo de un gran grupo de voluntarios repartidos por el mundo es, al menos, tan capaz como otros modelos, por ejemplo el usado por Red Hat o Microsoft, que es más centralizado y que está basado en el trabajo de empleados.

Palabras clave: *Debian, software libre, SLOC, líneas de código, Linux*

1. Introducción

El 14 de agosto de 2000 el Proyecto **Debian** anuncia **Debian GNU/Linux** 2.2, la distribución «Joel 'Espy' Klecker» [Debian22Ann] [Debian22Rel]. Su nombre familiar es *potato* y es la última distribución (hasta la fecha) del Sistema Operativo Debian GNU/Linux. En este trabajo hemos medido la distribución, mostrando su tamaño y comparándolo con otras distribuciones.

Debian no es sólo la mayor distribución GNU/Linux, es también una de las más fiables, con varios premios basados en las preferencias de los usuarios. Aunque es difícil de estimar el número de usuarios (el Proyecto Debian no vende CDs ni ningún otro soporte con el software), es sin duda importante dentro del mercado **Linux**. Se preocupa especialmente de beneficiar a los usuarios con una de las ventajas fundamentales del software libre: la disponibilidad del código fuente. Así, los paquetes fuente se preparan cuidadosamente para permitir la reconstrucción de los fuentes originales (conocidos como *upstream*). Estas precauciones resultan muy convenientes para hacer mediciones y, en general, para obtener estadísticas.

La idea de este artículo surgió del interesante trabajo de David Wheeler [Wheeler2001]. Animamos al lector a, al menos, ojearlo, y comparar los datos que ofrece con los aportados aquí.

La estructura de este artículo es la siguiente: la próxima sección proporciona algunas ideas de contexto sobre el proyecto Debian y la distribución Debian 2.2 GNU/Linux. Posteriormente, examinaremos el método empleado para reco-

Autores

Jesús M. González Barahona es profesor en la Universidad Rey Juan Carlos y colaborador de BarraPunto.Com. Comenzó a trabajar en la promoción del software libre en 1991, en el grupo PDSOFT (más adelante grupo Sobre). Desde entonces, ha realizado diversas actividades en este área, como la organización de seminarios, la realización de cursos y la participación en grupos de trabajo sobre software libre. Actualmente colabora con varios proyectos de software libre (entre ellos Debian y La Espiral), participa en el Grupo de Trabajo sobre Software Libre promovido por la DG-INFO de la Comisión Europea, colabora con asociaciones como Hispalinux y EuroLinux, escribe en varios medios sobre temas relacionados con software libre y asesora a empresas en sus estrategias relacionadas con estos temas. Es coordinador de la Sección Técnica de Software Libre de *Novática*.

Miguel A. Ortuño Pérez es Ingeniero en Informática, profesor de la Universidad Rey Juan Carlos (Madrid), donde su área de interés es la computación móvil y el software libre. Previamente trabajó en la Universidad de Oviedo en varios proyectos relacionados con sistemas de formación a distancia.

Pedro de las Heras Quirós es profesor en la Universidad Rey Juan Carlos y colaborador de BarraPunto.Com. Desde principios de los 90 ha sido usuario de software libre, habiendo colaborado en el grupo Sobre dedicado al Software Libre desde su creación. Ha participado en la organización de congresos, expos y cursos relacionados con software libre, y ha sido editor y autor de varias publicaciones relacionadas con el software libre. Es coordinador de la Sección Técnica de Software Libre de *Novática*.

José Centeno González es profesor en la Universidad Rey Juan Carlos. Se incorporó al Departamento de Informática de la Universidad Carlos III de Madrid en 1993, donde trabajó hasta 1999. Sus intereses en investigación incluyen la programación de sistemas distribuidos, los protocolos de comunicaciones y la movilidad. También está interesado por el impacto del software libre en la docencia en Ingeniería Informática y en la industria. Es Ingeniero de Telecomunicación por la Universidad Poliécnica de Madrid, en la que espera finalizar el doctorado este año.

Vicente Matellán Olivera es profesor en la Universidad Rey Juan Carlos (Madrid, España). Ha realizado varias actividades relacionadas con el software libre, entre ellas la creación de OpenResources.com y BarraPunto.com.

pilar los datos mostrados, para después ofrecer los resultados de la medición de Debian 2.2 (incluyendo totales, tamaños máximos, mediciones por lenguaje, etc). En la siguiente sección se comentan algunas de las cifras y cómo deben ser consideradas, así como algunas comparaciones con Red Hat Linux y otros sistema operativos. Finalizaremos con conclusiones y trabajo relacionado.

2. Algunas ideas de contexto sobre Debian

La distribución Debian 2.2 GNU/Linux está recopilada por el Proyecto Debian, quien también se encarga de su mantenimiento. En esta sección aportaremos alguna información básica sobre Debian como proyecto, y sobre la distribución 2.2.

2.1. El Proyecto Debian

Debian es un sistema operativo libre, que actualmente usa el *kernel* Linux como núcleo de todo el software de la distribución Debian GNU/Linux (si bien se esperan en un futuro próximo distribuciones basadas en otros núcleos como Hurd). La distribución está disponible para varias arquitecturas: Intel x86, ARM, Motorola 680x0, PowerPC, Alpha y SPARC. El núcleo de la distribución Debian es la sección principal (*main*). Constituye con mucho el grueso de los paquetes, y está compuesto sólo por software libre, conforme con lo que la **DFSG** (*Debian Free Software Guidelines*) [DFSG] entiende por software libre.

Esta distribución se puede descargar de la red y muchos redistribuidores la venden en CD u otros formatos. La distribución Debian es recopilada por el Proyecto Debian, un grupo de 900 desarrolladores voluntarios repartidos por todo el mundo y colaborando a través de Internet. No sólo se ocupan de adaptar y empaquetar los programas de la distribución, sino también de la infraestructura web, el sistema de control de errores, la «internacionalización» (adaptación a países e idiomas), las listas de correo de Debian de desarrollo y mantenimiento, y en un sentido amplio, a toda la infraestructura que hace la distribución Debian posible.

Los desarrolladores de Debian empaquetan el software que obtienen de los autores originales (*upstream*), asegurándose de que funciona correctamente con el resto de los programas Debian. Para ello, hay un conjunto de reglas que todo paquete debe cumplir, el Manual de Política Debian (*Debian Policy Manual*) [DebianPol]. La mayor parte del esfuerzo de empaquetar un determinado programa generalmente consiste en hacerlo compatible con estas normas. Los desarrolladores también gestionan los errores en los programas, intentan solucionarlos (informando de problemas y soluciones a los autores originales), siguen el desarrollo de nuevos programas y construye todo el software intermedio necesario para que el sistema Debian funcione. Los fallos y los problemas de seguridad se discuten abiertamente, y diariamente se ponen a disposición de los usuarios actualizaciones para las distribuciones estables, para solucionar problemas importantes de forma que los sistemas permanezcan tan seguros y libres de errores como sea posible.

Debian es único por muchos motivos. Es destacable su dedicación al software libre, su naturaleza sin ánimo de lucro y su modelo abierto de desarrollo (donde la mayor parte de las discusiones se hacen en listas de correo públicas). El Proyecto Debian está comprometido con el software libre, como refleja el Contrato Social Debian. La definición de lo que Debian considera software libre se encuentra en las Directrices del Software Libre de Debian (**DFSG**, *Debian Free Software Guidelines*), que esencialmente es el mismo software que entra en la categoría *open source* (lo que no resulta extraño, ya que la definición *open source* deriva de la DFSG).

2.2. Debian *potato*

Debian 2.2 (*potato*) es la última distribución oficial, la que actualmente es considerada «estable». Vió la luz en agosto de 2000 e incluye todo el software libre de cierta relevancia disponible en ese momento. Tan sólo la distribución principal, compuesta únicamente por software libre (según la DFSG), consta de más de 2.600 paquetes de código fuente. La distribución completa incluye cerca de 4.000 paquetes binarios, que el usuario puede instalar fácilmente desde diversos soportes o desde servidores en Internet.

Debian 2.2 está divido en dos archivos, el «normal» y el **non-US**. Este último incluye paquetes que tienen algún impedimento legal para ser exportados desde Estados Unidos (generalmente la legislación estadounidense sobre criptografía) Cada archivo está compuesto de varias «distribuciones»: Las llamadas **main**, **contrib** y **non-free**.

En el trabajo al que hacemos referencia en este artículo hemos considerado sólo la distribución **main** del archivo «normal». Es con mucho la mayor parte del archivo, está compuesto sólo por software libre y no tiene restricciones de exportación. En muchos aspectos, es la mayor colección coordinada de software libre disponible en Internet.

3. Captura de Datos

En resumen, el enfoque empleado para la captura de los datos presentados en este artículo es el siguiente:

1. Código Fuente de una Distribución Debian: por fortuna, el código fuente de las distribuciones Debian presentes y pasadas está archivado, disponible en Internet para cualquiera. El único problema es determinar la lista de paquetes de código fuente para una determinada distribución, y dónde acceder a ellos.
2. Descarga y captura datos: una vez que conocemos los ficheros que nos interesan, debemos descargarlos para obtener los datos. Pero no los descargamos todos al tiempo (lo que ocuparía mucho espacio en disco), si no que, secuencialmente, descargamos un paquete, lo desempaquetamos, analizamos y borramos antes de pasar al siguiente.
3. Análisis final: análisis de los datos recogidos y obtención de estadísticas, atendiendo al número total de SLOC de la distribución, las SLOC para cada uno de los diversos lenguajes de programación considerados, etc.

En las siguientes secciones estos tres pasos están descritos con mayor detalle.

3.1. Código fuente de una distribución Debian

En el sistema de paquetes de Debian hay dos tipo de paquete: fuente y binario. De cada paquete fuente, de forma automática, se pueden construir uno o más paquetes binarios. Para este trabajo, sólo los paquetes fuente son relevantes, con lo que en lo sucesivo no volveremos a hacer referencia a los binarios.

Al construir un paquete fuente, un desarrollador de Debian comienza por el directorio con el código fuente «original» del programa en cuestión. En terminología Debian, éste es el fuente *upstream* («corriente-arriba»). El desarrollador Debian crea modificaciones en forma de parche de estos fuentes originales si es necesario y crea un subdirectorio **debian** con todos los ficheros de configuración Debian (incluyendo los datos necesarios para construir los paquetes binarios). Se tiene entonces el paquete fuente, que generalmente (pero no siempre) consta de tres ficheros: los fuentes originales (un fichero **tar.gz**), los parches para obtener el directorio Debian fuente (un fichero **diff.gz** con los parches y el directorio **debian**), y un fichero de descripción con extensión **dsc**. (aunque sólo las últimas distribuciones incluyen este fichero) Los paquetes Debian «nativos» (desarrollados por Debian, en los que no hay fuentes *upstream*) no incluyen parches.

Los paquetes fuente de las distribuciones Debian actuales forman parte del archivo de Debian. En cada distribución están en el directorio **source**. En Internet hay servidores con los paquetes fuente de cada distribución Debian actual (generalmente *mirrors* de <http://archive.debian.org>). Desde Debian 2.0, en cada distribución hay un fichero **Sources.gz** en el directorio **source**, con información sobre los paquetes fuente de la distribución, incluyendo los ficheros que componen cada paquete. Ésta es la información que usamos para determinar qué paquetes fuente, y qué ficheros deben ser tenidos en cuenta para Debian 2.2.

De todas formas, hay que tener en cuenta que no todos los paquetes en **Sources.gz** deben ser analizados al contar líneas de código. La razón principal para no hacerlo es la existencia, en ocasiones, de varias versiones del mismo programa. Por ejemplo, en Debian 2.2 tenemos paquetes fuente **emacs19** (para **emacs-19.34**), y paquetes fuente **emacs20** (para **emacs-20.7**). Contar ambos paquetes implicaría contar Emacs dos veces, que no es lo que se pretende. Por tanto para cada distribución es necesaria una inspección manual, detectando aquellos que son esencialmente distintas versiones de un mismo programa y eligiendo un «representante» para cada familia de versiones.

Estos casos pueden hacer infraestimar el número de líneas de la distribución, ya que las diferentes versiones de un paquete puedan compartir una buena parte del código pero no todo. (Pensemos por ejemplo en **PHP4** y **PHP3**, donde el primero está reescrito casi por completo). De todas formas consideramos este defecto asumible y compensado por otras sobreestimaciones (como veremos después).

En otras ocasiones, hemos decidido analizar paquetes que pueden tener cantidades significativas de código en común. Este es el caso, por ejemplo de **emacs** y **xemacs**. Si bien el segundo es una ramificación del segundo y ambos comparten una buena parte del código, sin ser iguales son evoluciones del mismo «antepasado». Otros casos similares son **gcc** y **gnat**. El segundo, un compilador Ada, está construido sobre el primero (un compilador C), añadiendo muchos parches y mucho código nuevo. En ambos casos consideramos que el código es lo bastante distinto como para considerarlo paquetes diferentes. Esto probablemente lleva a sobreestimar el número de líneas de código de la distribución.

El resultado de este paso es la lista de paquetes (y los ficheros que la componen) que consideramos para analizar el tamaño de una distribución Debian. Esta lista está hecha a mano para cada distribución (con ayuda de unos *scripts* muy simples).

3.2. Descarga y captura de datos

Una vez que se establecen los paquetes y ficheros de Debian 2.2 a analizar, se descargan de alguno de los servidores de la red de *mirrors* Debian. Para este paso usamos algunos sencillos *scripts* Perl. El proceso consta de las siguientes fases:
· Descarga de los ficheros que componen el paquete.
· Extracción del directorio fuente correspondiente al paquete original (descomprimiendo el fichero **tar.gz**). Tras esto se consiguen los datos sobre los fuentes originales.
· Aplicación del parche al directorio original con el fichero **diff.gz** file, para obtener el directorio fuente Debian. Tras la extracción, se obtienen los datos de este directorio.
· Borrado del directorio **debian** para evitar contar los *scripts* que contiene, que son los hechos por el desarrollador de Debian.

No todos los paquetes tienen versión original (*upstream*), por lo que en este proceso hay que prestar atención a estas situaciones.

La captura de los datos se hace usando los *scripts* de **sloccount**, tres veces por paquete (una vez en cada fase, ver el párrafo anterior), que almacenan número de líneas de código de cada paquete en un directorio diferente, preparado para un análisis e informe posterior.

La razón de recoger datos tres veces por cada paquete es el analizar el impacto del desarrollo Debian en el paquete original. Este impacto puede ser en forma de parches al original (con frecuencia para hacerlo más estable y seguro, para hacerlo consistente con la política de instalación Debian o para añadir alguna funcionalidad) o de *scripts* de instalación (que pueden ser identificados al medir los fuentes sin el directorio **debian**).

El resultado final de este paso es la recopilación de todos los datos obtenidos de los paquetes descargado, organizados por paquete, y listos para ser analizados. Estos datos consisten fundamentalmente en listas de ficheros, con el número de líneas de código que corresponden a cada uno, detallando subtotales por lenguaje de programación.

3.3. Análisis final

El último paso es la generación de informes, usando **sloccount** y algunos *scripts*, para estudiar los datos obtenidos. Ya que en este punto todos los datos obtenidos están disponibles localmente y resulta sencillo procesarlos, el análisis puede ser hecho rápidamente y reiterado automáticamente de forma sencilla, buscando diferentes tipos de información.

El resultado final de este paso es un conjunto de informes y análisis estadísticos, usando los datos obtenidos en el paso anterior, y considerándolos desde diferentes puntos de vista. Estos resultados se presentan en la siguiente sección.

4. Resultados de la medición de Debian

Los resultados principales de nuestro análisis de la distribución Debian 2.2 GNU/Linux se pueden organizar en las siguientes categorías.
· Tamaño de Debian Potato.
· Relevancia de los lenguajes de programación más usados.
· Análisis de la evolución en el tamaño de los paquetes más importantes.
· Estimaciones de esfuerzo.

4.1. Tamaño de Debian Potato

Hemos contado el número de líneas físicas de código fuente de Debian GNU/Linux 2.2 de tres formas, con los siguientes resultados (todas las cifras son aproximadas, consultar el apéndice para más detalles):
· Número de líneas en paquetes originales *upstream* «tal cual»: *52.810.000* SLOC
· Número de líneas en paquetes fuente Debian: *56.180.000* SLOC
· Número líneas en paquetes fuente Debian sin directorio **debian**: *55.920.000* SLOC

Para más detalles sobre el significado de cada categoría, el lector puede volver a la subsección «Descarga y Captura de datos». En resumen, la medida de los paquetes *upstream* originales puede ser considerado como el tamaño del software original usado en Debian. La medida de los paquete fuente Debian representa la cantidad de código presente actualmente en la distribución Debian 2.2, incluyendo tanto el trabajo de los autores originales como el de los desarrolladores Debian. Este último incluye los *scripts* relativos a Debian y los parches. Los parches pueden estar hechos por los desarrolladores Debian (por ejemplo, aquellos que adaptan un paquete a la política Debian) o pueden ser obtenidos de algún otro sitio. La medida de los paquetes Debian sin el directorio **debian** excluye los *scripts* relacionados con Debian, por lo que son una buena medida del tamaño de los paquetes tal y como aparecen en Debian, excluyendo los *scripts* referentes a Debian.

Es importante tener en cuenta que los paquete desarrollados específicamente para Debian en general no tienen paquete fuente *upstream* original. Este es por ejemplo el caso de **apt**, presente sólo como paquete fuente Debian.

4.2. Lenguajes de programación

El número de SLOC físico para los paquetes fuente originales, clasificado por lenguajes de programación es, redondeando:
· **C**: *39.960.000* SLOC (71,12%)
· **C++**: *5.500.000* SLOC (9,79%)
· **LISP**: *2.800.000* SLOC (4,98%)
· **Shell**: *2.640.000* SLOC (4,70%)
· **Perl**: *1.330.000* SLOC (2,36%)
· **FORTRAN**: *1.150.000* SLOC (2.04%)
· **Tcl**: *550.000* SLOC (0.99%)
· **Objective C**: *425.000* SLOC (0.76%)
· **Ensamblador**: *425.000* SLOC (0.75%)
· **Ada**: *405.000* SLOC (0.73%)
· **Python**: *360.000* SLOC (0.65%)

Por debajo del 0,5% encontramos otros lenguajes: Yacc (0,46%), Java (0,20%), Expect (0,20%), Lex (0,13%) y otros con porcentaje inferior al 0,1%.

Cuando contamos las líneas de los paquetes fuente Debian sin el directorio **debian** (que contiene ficheros de configuración del paquete y *scripts* hechos por el desarrollador de Debian) las cifras son similares, lo que significa que estos *scripts* no representan una parte significativa de la distribución. La principal diferencia está en las lineas de Shell (unas 150.000 menos) y en la líneas Perl (unas 80.000 menos), lo que revela los lenguajes preferidos para estos *scripts*.

A pesar de esto, al contar los paquetes *upstream* originales hay algunas diferencias destacables: unas 2.000.000 líneas de código C, 300.000 líneas de LISP, 200.000 líneas FORTRAN y pequeñas variaciones en otros lenguajes. Estas diferencias se deben generalmente a los parches a los fuentes originales hechos por los desarrolladores Debian. Por tanto, consultado estos resultados, podemos saber en qué lenguajes están escritos la mayoría de los paquetes a los que se han aplicado parches.

4.3. Los mayores paquetes

Los mayores paquetes en la distribución Debian Potato son:
· **Mozilla (M18)**: 2.010.000 SLOC (2.010.000). Un total de 1.260.000 SLOC en C++, 702.000 en C. Mozilla es un navegador WWW muy conocido.
· **Linux kernel (2.2.19)**: 1.780.000 SLOC (1.780.000). 1.700.000 SLOC en C, 65.000 en ensamblador. Los núcleos Linux 2.x eran la serie estable en la época de Debian 2.2.
· **XFree86 (3.3.6)**: 1.270.000 SLOC (1.265.000). Fundamentalmente 1.222.000 SLOC de C. Es una implementación de X Window que incluye un servidor de gráficos y programas básicos.
· **PM3 (1.1.13)**: 1.115.000 SLOC (1.114.000). 983.000 SLOC de C, 57.000 de C++. PM3 es la distribución de Modula-3 de la Escuela Politécnica de Montreal, incluye compilador y librerías.
· **OSKit (0.97)**: 859.000 SLOC (859.000). Un total de 842.000 SLOC de C. OSKit es el "Flux Operating System Toolkit" un entorno de trabajo para el diseño de sistemas operativos.
· **GDB (4.18)**: 801.000 SLOC (800.000). Incluye 727.000 líneas

de C y 38.000 de Expect. GDB es el depurador GNU a nivel de fuente.
· **GNAT (3.12p)**: 688.000 SLOC (687.000). Unas 410.000 SLOC de C y 248.000 SLOC de Ada. GNAT es el compilador GNU de Ada 95, incluyendo una serie de librerías.
· **Emacs (20.7)**: 630.000 SLOC (629.000). 454.000 SLOC de LISP, 171.000 SLOC de C. Emacs es, entre otras muchas cosas, un editor de texto muy conocido .
· **NCBI Librerías (6.0.2)**: 591.000 SLOC (591.000). La mayoría en C, 590.000 SLOC. Este paquete incluye librerías para aplicaciones de Biología.
· **EGCS (1.1.2)**: 578.000 SLOC (562.000). Incluye 470.000 SLOC de C y 55.000 **SLOC de C++**. Este paquete incluye las librerías GNU de extensión de C++.
· **XEmacs, base support (21)**: 513.000 SLOC (513.000). Casi todo en LISP , 510.000 SLOC. Includes the base extra Emacs LISP files needed to have a working XEmacs. Incluye una serie de ficheros en Emacs LISP necesarios para la parte básica de XEmacs

Los números entre paréntesis representan aproximadamente el SLOC de los paquetes fuente originales, el resto de las cifras es el SLOC aproximado de los paquetes fuente Debian. Sólo se muestran los datos de los lenguajes más relevantes en cada paquete. El lector puede advertir que en la mayor parte de los casos, las cifras en ambos casos son aproximadamente iguales, lo que evidencia que, en esos casos, lo añadido por los desarrolladores Debian es mínimo (aunque estas modificaciones puedan ser importantes).

Las versiones de las distribuciones no son, obviamente, las actuales, pero eran las disponibles cuando Debian 2.2 fue congelado (primavera de 2000). La clasificación podría ser diferente si los desarrolladores Debian tuvieran que empaquetar las cosas de otra forma. Por ejemplo, si todas las extensiones de Emacs estuvieran en el paquete Emacs, hubiera sido mucho mayor. En todo caso, los paquetes fuente de Debian generalmente encajan bien con la idea de paquete que se tiene en general, que es la que suelen tener los autores originales.

Los siguientes paquetes según su tamaño (entre 350.000 y 500.000 _ SLOC) son Binutils (ensamblador GNU, linker y utilidades binarias), TenDRA (compiladores y comprobadores C y C++), LAPACK (un conjunto de librerías para álgebra lineal) y el Gimp (el paquete GNU de manipulación de imágenes). La mayoría de estos paquetes están escritos en C, excepto LAPACK, escrito principalmente en FORTRAN.

4.4. Esfuerzo y estimaciones del coste

Usando el modelo **COCOMO** básico [Boehm1981], se puede estimar el esfuerzo necesario para construir un sistema con el tamaño de Debian 2.2. Esta estimación supone un modelo de desarrollo de software propietario "clásico", con lo que no es válido para estimar el esfuerzo aplicado a la construcción de este software. Pero al menos puede darnos un orden de magnitud del esfuerzo que sería necesario si se hubiera empleado un modelo de desarrollo de software propietario.
Aplicando el contador de SLOC para los paquetes fuente Debian los datos que proporciona el modelo COCOMO básico son los siguientes:
· SLOC físicas totales: 56.184.171
· Esfuerzo estimado: 171.141 personas-mes (14.261 personas-año)
Formula: *2.4 * (KSLOC**1,05)*
· Tiempo estimado: 72.53 meses (6.04 años)
Formula: 2,5 * (Esfuerzo**0,38)
· Coste estimado del desarrollo: 1.848.225.000 dólares

Para calcular la estimación de costes, hemos usado el salario medio para un programador de sistemas a tiempo completo en el año 2000, que según Computer World [ComWorld2000] es de 54.000 dólares USA al año, con un factor de conversión de 2,4.

5. Algunos comentarios y comparaciones

Los números ofrecidos en las secciones anteriores no son más que estimaciones. Pueden darnos al menos órdenes de magnitud y permitir las comparaciones, pero no deben ser tomados como datos exactos, pues hay demasiadas fuentes de error, así como margen para diversas interpretaciones. En esta sección discutiremos algunas de las presunciones más importantes que se han hecho, con el objetivo de aportar contexto al lector para interpretar los números.

5.1. Qué es una línea de código fuente

Ya que dependemos de la herramienta **sloccount** de David Wheeler para medir el SLOC, también dependemos de su definición de «líneas físicas de código fuente». Por tanto, podemos decir que contabilizamos una SLOC cuando **sloccount** lo hace, si bien **sloccount** ha sido cuidadosamente diseñado para responder a la definición habitual de SLOC físicas: *" una línea física de código fuente es una línea que acaba en un marcador de nueva línea o de fin de fichero, y que contiene al menos un carácter que no es un espacio en blanco ni un comentario"*.

Hay otra medida similar que se prefiere en ocasiones, el SLOC «lógico». Por ejemplo, una línea escrita en C con dos punto y coma sería considerado como dos SLOC lógico, mientras que sería un solo SLOC físico. En todo caso, para los propósitos de este artículo (como para casi cualquier otro), las diferencias entre ambos SLOC son despreciables, especialmente comparadas con otras fuentes de error e interpretación.

5.2. Fuentes de imprecisión en la medición de SLOC

Las mediciones de líneas de código presentadas en este artículo no son más que estimaciones. En ningún caso pretendemos afirmar que sean exactas, especialmente cuando se refieren a agregación de paquetes. Hay varios factores que causan imprecisiones, algunas debidas a las herramientas empleadas para contar, otras, debidas a la selección de los paquetes.

· Algunos ficheros pueden no haber sido medidos de forma precisa. Si bien **sloccount** incluye heurísticos cuidadosa-

mente diseñados para detectar ficheros de código fuente, y para distinguir las líneas de código de los comentarios, estos heurísticos no siempre funcionan de la manera prevista. Además, con frecuencia es difícil distinguir los ficheros generados automáticamente (que no deberían ser contados), aunque **sloccount** hace un buen esfuerzo para reconocerlos.
- No todos los lenguajes de programación son reconocidos. Para capturar los datos usamos la versión 1.9 de **sloccount**, que reconoce unos 20 lenguajes distintos. De todas formas, algunos de los lenguajes empleados en Debian (como Modula-3 o Erlang) no están soportados. Obviamente esto lleva a una infra-estimación en los paquetes con ficheros escritos en estos lenguajes.
- Distintas percepciones en la agregación de familias de paquetes y en la selección de los representativos. Como comentamos en la subsección donde tratamos sobre la selección de la lista de paquetes a medir, las razones para incluir o excluir un paquete no son incuestionables. ¿Deberíamos contar distintas versiones del mismo paquete? ¿Deberíamos contar sólo una vez el código presente en varios paquetes o no? El criterio habitual para medir SLOC es «líneas de código fuente distribuido». Desde este punto de vista, todos los paquetes debieran ser considerados tal y como aparecen en la distribución Debian. En todo caso esto es difícil de aplicar cuando algunos paquetes son claramente evoluciones de otros. En vez de considerarlos todos como «distribuidos», parece más productivo considerar los más antiguos como «versiones beta». De todas formas en el ámbito del software libre es bastante frecuente el distribuir versiones estables cada 6 o 12 meses. Estas versiones estables representan mucho trabajo sólo para asegurar su estabilidad, aunque sólo sean la base para posteriores versiones.

En la mayoría de los casos, hemos adoptado una decisión intermedia: Contar sólo familias de paquetes que sean una línea de evolución (como en el caso de **emacs19** y **emacs20**, pero contar por separado familias de paquetes que comparten algo de código pero que son en sí mismas desarrollos destinos (como en el caso de **gcc** y **gnat**).

5.3. Estimación de esfuerzo y coste

Los modelos de estimación actual, y específicamente COCOMO, sólo consideran modelos clásicos de desarrollo de software propietario. Pero los modelos de desarrollo de software libre son bastante distintos. De esta forma sólo podemos estimar el coste del sistema si hubiera sido desarrollado por métodos convencionales, pero no los costes reales (en esfuerzo o dinero) del desarrollo del software incluido en Debian 2.2.

Algunas de las diferencias que hacen imposibles el uso de estos modelos de estimación son:
- Proceso continuo de distribuciones. El modelo COCOMO está basado en el concepto de «SLOC de versiones distribuidas» , lo que implica un punto en el ciclo de vida del proyecto en que el producto es distribuido. A partir de ese momento, el principal esfuerzo de desarrollo está dedicado al mantenimiento. Por el contrario, la mayoría del software libre libera versiones con tanta frecuencia que podría ser considerado como un proceso de distribuciones continuas. Esto conlleva una casi continua estabilización del código, al mismo tiempo que evoluciona. Los proyectos de software libre suelen mejorar y modificar los programas al mismo tiempo que los preparan para los usuarios finales.
- Control y solución de error. Mientras que todos los sistemas de software propietario precisan de costosos ciclos de depuración, el software libre cuenta con la ayuda de voluntarios ajenos al proyecto, en forma de valiosos informes de errores e incluso soluciones para ellos.
- Reutilización, evolución e intercambio del código. En los proyectos de software libre es común la reutilización de código de otros proyectos de software libre como una parte central del sistema en desarrollo. También es frecuente que varios proyectos evolucionen del mismo sistema base, e incluso todos usen código de todos al mismo tiempo. A veces esto mismo puede suceder en desarrollos propietarios, pero incluso en grandes empresas con muchos proyectos abiertos, no es común, es mucho más frecuente en proyectos de software libre.
- Modelo de desarrollo distribuido. Aunque algunos sistemas propietarios son desarrollados por equipos distribuidos geográficamente, el grado de distribución que suelen presentar los proyectos de software libre es varios órdenes de magnitud superior. Hay excepciones, pero en general los proyectos de software libre corren a cargo de personas de diferentes países, que no trabajan en la misma empresa, que dedican distinta cantidad de esfuerzo al proyecto, que cooperan principalmente a través de Internet y que, la mayor parte de las veces (especialmente en proyectos grandes), el equipo de desarrollo nunca ha coincidido físicamente en un mismo sitio.

Algunos de estos factores incrementan el esfuerzo necesario para construir el software, mientras que otros lo decrementan. Sin analizar detalladamente el impacto de estos (y otros) factores, los modelos de estimación en general, y COCOMO en particular no son directamente aplicables al desarrollo de software libre.

5.4. Comparación con los tamaños estimados de otros sistemas

Para poner en contexto las cifras mostradas anteriormente, aportamos estimaciones del tamaño de algunos Sistemas Operativos y una comparación más detallada con las estimaciones de la distribución Red Hat Linux.

Como se indica en [Lucovsky2000] (para Windows 2000) [Wheeler2001] (para Red Hat Linux) [Schneier2000] (para el resto de los sistemas) este es el tamaño estimado para varios Sistemas Operativos, en líneas de código. (Siempre en cifras aproximadas):
- Microsoft Windows 3.1: 3.000.000
- Sun Solaris: 7.500.000
- Microsoft Windows 95: 15.000.000
- Red Hat Linux 6.2: 17.000.000
- Microsoft Windows 2000: 29.000.000

- Red Hat Linux 7.1: 30.000.000
- Debian 2.2: 56.000.000

Casi todas las estimaciones (en realidad, todas, excepto las de Red Hat) no son detalladas, por lo que es difícil saber qué consideran una línea de código. De todas formas, las estimaciones deberían ser lo suficientemente próximas a las mediciones de SLOC como para que sean válidos para una comparación.

Nótese que mientras que Red Hat y Debian incluyen muchas aplicaciones, con frecuencia varias aplicaciones del mismo tipo de programa, tanto los sistemas operativos de Microsoft como los de Sun son mucho más limitados en este sentido. Si las aplicaciones más usadas en estos entornos fueran contabilizadas juntas, el tamaño sería mucho mayor. En todo caso, también es cierto que todas esas aplicaciones ni son desarrolladas ni integradas por el mismo equipo de desarrollo, como en el caso de las distribuciones basadas en Linux.

A partir de estas cifras, queda claro que las distribuciones basadas en Linux, en general, y Debian 2.2 en particular, son unas de las mayores colecciones de software nunca integradas por un equipo de desarrollo.

5.5. Comparación con Red Hat Linux

El único sistema operativo para el que hemos encontrado medidas detalladas de líneas de código fuente es Red Hat Linux (consultar *Estimating Linux's Size* y *More Than a Gigabuck: Estimating GNU/Linux's Size*;). Siendo también una distribución basada en Linux, donde los paquetes software incluidos en Debian y en Red Hat son bastante similares, la comparación puede ser ilustrativa. Además, siendo Red Hat Linux una distribución muy común, probablemente la más conocida, la comparación con ellas puede aportar un contexto apropiado para el lector familiarizado con ella.

Lo primero que nos sorprendió cuando medimos Debian 2.2 fue su tamaño, comparado con Red Hat 6.2 (distribuido en marzo de 2000) y con Red Hat 7.1 (distribuido en abril de 2001). Debian 2.2 aparece en agosto de 2000 y su tamaño es casi el doble que el de Red Hat (distribuido unos seis meses después) y más de tres veces el tamaño de Red Hay 6.2 (distribuido cinco meses antes). Algunas de estas diferencia pueden ser debidas a diferentes consideraciones sobre qué paquetes incluir al medir, pero con todo aportan una buena idea de los tamaños relativos.

El principal factor causante de las diferencias es el número de paquetes incluidos en cada distribución, en el caso de Debian hemos considerado 2630 paquetes fuente (con una media de unos 21.300 SLOC por paquete), mientras que Red Hat 7.1 incluye sólo 612 paquetes (de unos 49.000 SLOC por paquete) Cuando se comparan los paquetes mayores en ambas distribuciones, encontramos en Debian todos los incluidos en Red Hat, lo que no es cierto a la inversa: varios paquetes que suman una buena cantidad de SLOC en Debian no están presentes en Red Hat. Por ejemplo, entre los 11 mayores paquetes en Debian 2.2, los siguientes no están en Red Hat 7.1: PM3 (unos 1.115.000 SLOC), OSKit (unos 859.000 SLOC), GNAT (688.000), NCBI (591.000). Por el contrario, entre los 11 paquetes mayores en Red Hat 7.1, no echamos ninguno en falta en Debian 2.2.

De todas formas hay una gran colección de paquetes software presentes en Red Hat 7.1 y no en Debian 2.2: el escritorio **KDE** y sus utilidades. Debido a problemas de licencia, Debian decidió no incluir software KDE hasta después de Debian 2.2, cuando la licencia para Qt cambió a **GPL**. Por tanto, podemos decir que Debian 2.2 es mayor, incluso sin todo el código de KDE. Sólo para dar una idea, los mayores paquetes KDE en Red Hat 7.1 son **kdebase**, **kdelibs**, **koffice** y **kdemultimedia**, que suman sobre 1.000.000 SLOC. Todos ellos están ausentes en Debian. Esto sugiere que si las medidas hubieran sido hechas en la distribución actual Debian (aún no distribuida oficialmente), las diferencias hubieran sido mayores.

Las diferencias entre el mismo paquete en cada distribución son atribuibles a las diferentes distribuciones incluidas en ellos. Por ejemplo, el kernel Linux suma 1.780.000 SLOC (Versión 2.2.19) en Debian 2.2, mientras que el mismo paquete suma 2.437.000 SLOC (Versión 2.4.2) en Red Hat 7.1. XFree incluye 1.270.000 SLOC (Versión 3.3.6) en Debian 2.2, mientras que la versión incluida en Red Hat 7.1 (XFree 4.0.3) suma 1.838.000 SLOC. Estas diferencias hacen difícil el comparar directamente Red Hat y Debian.

El lector debe percatarse de que hay una diferencia metodológica entre el estudio de Red Hat y el nuestro sobre Debian. El primero extrae todos los paquetes fuente y usa *checksum* MD5 para evitar duplicados entre todo el código de la distribución. En el caso de Debian, hemos extraído los paquetes uno a uno, evitando paquetes duplicados pero no ficheros individuales repetidos. De todas formas, la suma total no debería verse muy afectada por esta diferencia.

6. Conclusiones y Trabajo Relacionado

Es importante darse cuenta de que estas mediciones pueden representar aproximadamente toda la colección de paquetes de software libre estable disponibles para GNU/Linux en el momento de la aparición de Debian 2.2 (agosto de 2000). Naturalmente, hay software libre no incluido en Debian, pero si hablamos de programas bien conocidos, usables y estables, la mayoría han sido empaquetados por un desarrollador Debian e incluidos en la distribución Debian. Por tanto, con ciertas precauciones puede decirse que el software del que estamos hablando suponía unos 60.000.000 SLOC en agosto de 2000. Usando el modelo COCOMO, esto implicaría un coste (usando las técnicas convencionales de desarrollo de software propietario) de cerca de 2.000 millones de dólares y un esfuerzo de más de 170.000 personas/mes.

También podemos comparar esta medición con otras distribuciones Linux, especialmente Red Hat. En números redondos, el tamaño de Debian 2.2 es el doble que el de Red Hat 7.1, que es unos ocho meses posterior. También es mayor que el último sistema operativo de Microsoft (aunque como vimos en la sección correspondiente, esta comparación puede ser engañosa). Entrando en los detalles, pueden verse algunos datos

interesantes. Por ejemplo, el lenguaje más empleado en la distribución es C (más del 70%), seguido de C++ (cerca del 10%), LISP y Shell (sobre el 5%), Perl y FORTRAN (un 2% aproximadamente). Los mayores paquetes en Debian 2.2 son Mozilla (unas 2.000.000 SLOC), el núcleo Linux (unas 1.800.000 SLOC), XFree86 (1.250.000) y PM3 (Más de 1.100.000).

No hay muchos estudios detallados sobre el tamaño de sistemas operativos modernos y completos. Entre ellos, el trabajo de David Wheeler midiendo Red Hat 6.2 y Red Hat 7.1 es la referencia más parecida al presente articulo. Otros trabajos interesantes, con puntos en común con este es [GodfreyTu2000], un estudio de la evolución a través del tiempo del núcleo Linux. Algunos otros artículos, ya citados, proporcionan mediciones globales de algunos sistemas operativos de Sun y Microsoft, pero no son lo bastante detallados, excepto para indicar estimaciones.

Por último, es importante repetir una vez más que estamos dando sólo estimaciones, no cifras reales. Estas dependerían demasiado en la elección del software a medir y de algunos otros factores que ya hemos tratado aquí. Pero creemos que tienen la suficiente precisión como para extraer algunas conclusiones y para compararlos con otros sistemas.

7. Agradecimientos

Este artículo está evidentemente inspirado en el de David A. Wheeler [Wheeler2001] del que hemos usado su herramienta **sloccount**. Sin su trabajo, el nuestro hubiera sido completamente impensable. También queremos agradecer los comentarios y sugerencias de muchos desarrolladores de Debian, que nos han ayudado a mejorar este artículo.

8. Bibliografía

[Boehm1981] Barry W., Boehm, 1981, *Software Engineering Economics*, Prentice Hall.

[ComWorld2000] Computer World, *Salary Survey 2000*, <http://www.computerworld.com/cwi/careers/surveysandreports>.

[Debian22Ann] Debian Project, *Debian GNU/Linux 2.2, the "Joel 'Espy' Klecker" release, is officially released*, <http://www.debian.org/News/2000/20000815>.

[DebianPol] Debian Project, *Debian Policy Manual*, <http://www.debian.org/doc/debian-policy/>.

[Debian22Rel] Debian Project, *Debian GNU/Linux 2.2 release information*, <http://www.debian.org/releases/2.2/>.

[DFSG] Debian Project, *Debian Free Software Guidelines*, <http://www.debian.org/social_contract#guidelines>.

[GodfreyTu2000] Michael W., Godfrey, Qiang, Tu, Software Architecture Group (SWAG), Department of Computer Science, University of Waterloo, August 3-4, 2000, *Evolution in Open Source Software: A Case Study*, 2000 Intl Conference on Software Maintenance <http://plg.uwaterloo.ca/~migod/papers/icsm00.pdf>.

[Lucovsky2000] Mark, Lucovsky, August 3-4, 2000, *From NT OS/2 to Windows 2000 and Beyond - A Software-Engineering Odyssey*, 4th USENIX Windows Systems Symposium, <http://www.usenix.org/events/usenix-win2000/invitedtalks/lucovsky_html/>.

[Schneier2000] Bruce, Schneier, March 15, 2000, *Software Complexity and Security*, Crypto-Gram Newsletter, <http://www.counterpane.com/crypto-gram-0003.html>.

[Wheeler2001] David A., Wheeler, *More Than a Gigabuck: Estimating GNU/Linux's Size*, <http://www.dwheeler.com/sloc>.

La crisis del software libre científico

David Santo Orcero
Consultor

<irbis@orcero.org>

Resumen: *el mundo científico produjo gran cantidad de software libre en los 70 y 80. Mas la producción de software libre se ha reducido en esta última década, hasta el punto de que hay áreas donde no se desarrollan herramientas libres desde los 80, o incluso donde no existen herramientas libres. En este artículo analizaremos las posibles causas y algunas propuestas de soluciones a este problema.*

Palabras clave: *software libre, GPL, software científico*

1. La ciencia y el software libre

Richard Stallman es la primera persona que ha definido el concepto de software libre. Después de la infructuosa relación con el software propietario que sucedió al abandono del **ITS** --libre-- por el laboratorio de inteligencia artificial del **MIT**, Richard Stallman dimitió de su posición y creó la **FSF** (*Free Software Foundation*) para difundir el software libre, tarea que comenzó con su célebre manifiesto (<http://www.gnu.org/gnu/manifesto.html>) y viene realizando desde entonces.

De cualquier modo, aunque Richard Stallman es el responsable de formalizar el concepto de software libre y de haber desarrollado la licencia **GPL** *(General Public Licence,* <http://www.gnu.org/copyleft/gpl.html>, de distinguirlo claramente del software gratuito y de haber sido uno de sus principales teóricos, no es el primer programador en desarrollar software libre. En el ámbito científico encontramos grandes proyectos colaborativos de desarrollo de software libre anteriores a Stallman, como es el caso de **GAMESS**. Además, en las épocas en que el peso del software propietario era más grande en el mundo «comercial», la mayor parte del software libre se concentraba en el área científica.

Tradicionalmente, la ciencia ha sido uno de los nichos de mercado más importantes del software libre. Permitiendo al científico una más rápida verificación de sus teorías aprovechando código de los otros, y asegurando la posibilidad de acceder a el código a bajo costo --ya que hay una amplia tradición histórica de que los científicos compartan código apenas a cambio de reconocimiento--, el esfuerzo de desarrollo compartido ha permitido acelerar el adelanto científico en muchas áreas, como por ejemplo en la química computacional con el proyecto **QCPE** (*Quantum Chemistry Program Exchange*, <ftp://qcpe6.chem.indiana.edu/>) o el **CCL** (*Computational Chemistry List*, <http://www.ccl.net/>); y al mismo tiempo ha permitido que pueda ser realizada ciencia también en lugares con bajo presupuesto.

El científico es conceptualmente afín a muchos de los conceptos del software libre: la formación de una meritocracia, donde eres más importante cuanto más has contribuido --código, en el caso del software libre; descubrimientos, en el caso del científico. Los científicos están acostumbrados a hacer públicos sus conocimientos publicando en revistas, donde son analizados entre sus semejantes; como es el caso de los programadores de software libre. Por ello, gran parte del código desarrollado por científicos tenían licencias que por los criterios actuales pueden ser consideradas libres.

Sin embargo, la situación ha comenzado a mudar. En el área comercial, el software libre se hace popular hasta el punto de comenzar a plantar cara al software propietario. Por otro lado, en el área científica cada vez son más frecuentes los paquetes propietarios, y vemos al software libre ahogarse y retroceder lentamente. El software científico bajo licencia GPL en muchas áreas actualmente es, desafortunadamente, prácticamente inexistente.

2. La causa de la crisis del software científico

Hay varias razones que pueden ser propuestas como causa de la crisis. No son mutuamente excluyentes; de facto, es posible que la verdadera causa sea una mezcla de todas ellas. Sin embargo, todas tienen en común que son causadas por el método de valorar los avances científicos y el mecanismo actual de financiación de la actividad científica.

La ciencia en los últimos años se viene haciendo más competitiva. Por razones que quedan fuera del ámbito de este trabajo --quizás desempleo, quizás más valorización de la carrera científica-- cada vez son más las personas que al terminar su graduación se deciden por cursar un curso de posgraduación y que prestan concurso para la carrera docente en alguna universidad. El hecho de aumentar la competencia ha hecho que la competición se haga más difícil. Y, de entre los distintos factores que intervienen en un concurso --prueba didáctica,

Autor

David Santo Orcero estudió Ingeniería Informática. Descubrió el software libre en 1994, cuando comenzó a usar Linux. Desde entonces ha estado trabajando con software libre. Ha sido becario de investigación durante los últimos cinco años, desarrollando aplicaciones para biofísica y física del estado sólido. Ha intentado mostrar las ventajas de usar y producir software libre donde ha trabajado, y ha publicado más de 150 artículos. Actualmente trabaja de consultor. Página personal: <http://www.orcero.org/irbis>

experiencia docente...-- o que intervienen en la obtención de una beca de investigación --experiencia investigadora--, aquel en que es más fácil marcar diferencia entre personas de la misma promoción es el número de artículos publicados en revistas indexadas.

Por otro lado, entre los grupos ya consolidados la situación es similar. Cada vez existen más grupos de investigación y la financiación no ha crecido proporcionalmente, por lo que la competencia es cada vez mayor. En muchos países, no llega a manos del investigador una perra gorda si no presenta un rendimiento en publicaciones, vía artículos publicados en revistas indexadas.

Este sistema «publica o perece» perjudica a los que desarrollan software libre, dado que no existen revistas indexadas específicas de software libre, por lo que es difícil conseguir una publicación de una nueva herramienta desarrollada en una revista indexada. El único método es publicar en una revista de informática teórica, lo que no siempre es posible. En la práctica, esto beneficia a los que trabajan en informática teórica, ya que les basta con la idea, el algoritmo en pseudolenguaje y, en el peor caso, una comparativa. Desarrollar una herramienta libre completa involucra también codificación, documentación de uso, verificación de código y, muchas veces, una interfaz gráfico. Por otro lado, si no implementa un algoritmo nuevo, no puede ser publicado; de nada vale decir: «mi programa es el primer y único programa libre que hace tal cosa».

Si alguien ya planteó el problema, aunque no haya desarrollado la aplicación, él publicó y la aplicación completamente desarrollada, funcional, no generará ninguna publicación. Queda publicar en revistas de otra área; por ejemplo, publicar en revistas de biofísica los programas libres de biofísica. Sin embargo, contamos con los mismos problemas que en las revistas de informática teórica, junto con los problemas de que tenemos que hacer un trabajo bien por encima de la media de los de la revista por estar fuera de área. Por si esto fuera poco *handicap*, además, muchas universidades y agencias de fomento de investigación ponderan negativamente, o incluso ni valoran aquellos artículos indexados realizados en revistas de fuera del área --en nuestro caso, informática.

Además, los mecanismos para avalar el rendimiento son ahora a muy corto plazo. Por ejemplo, la agencia de becas que mi última beca de investigación exigía un informe anual de rendimiento científico, con número de publicaciones en revistas indexadas. Si el rendimiento no es bueno, perdía la beca. Este hecho me obligaba a mantener dos líneas de trabajo, una a corto plazo para mantener la beca y otra a largo plazo, que era el objetivo real de la investigación. En la práctica, era preciso el doble de trabajo que el de cualquier otro investigador del departamento para mantener una evaluación de que mi trabajo era aceptable.

Por otro lado, los proyectos de software científico que deben ser encarados actualmente en muchas áreas --física, química, medicina, entre otras-- son de gran porte, necesitando años para su desarrollo.

Antiguamente era rentable para un grupo consolidado tener uno o varios científicos trabajando a tiempo completo desarrollando software libre. El placer de crear, de descubrir, suponía ya suficiente aliciente como para perder un poco de rentabilidad científica durante unos años, ya que parte del grupo estaba «entretenida» en tareas que no iban a suponer publicaciones --desarrollo de software científico. Sin embargo, actualmente, cada vez se da menos este caso. La supervivencia del grupo y el mantenimiento de la beca de cada uno de los investigadores depende de las publicaciones con vista a un año. Esto supone que en los grupos muchas veces se prioriza la investigación con resultados inmediatos. Rob Pike, en su artículo *Systems Software Research is Irrelevant* <http://www.cs.bell-labs.com/who/rob/utah2000.ps> sobre por qué la investigación de Informática de Sistemas estaba muerta, ya daba la voz de alarma. En áreas que no sean la informática, como es el caso de la ciencia base, la investigación en herramientas informáticas sufre más todavía de los problemas expuestos por Rob Pike.

La licencia GPL, aunque excelente, no ayuda tampoco a desarrollar software científico libre. Por un lado, es legal según la GPL cambiar el nombre del autor --aunque no sea ético. La GPL no permite realizar restricciones adicionales a la licencia, cuando sería necesario establecer algún mecanismo para que se citen los trabajos del autor del programa. Por otro lado, a diferencia de otras áreas, no tiene sentido en muchas áreas de ciencia base abrir una empresa de servicios de valor añadido. Aunque esto funcione bien en algunos casos --como es el caso de **HelixCode**, o de cualquiera de las empresas de distribuciones-- la masa crítica de posibles usuarios es lo suficientemente baja como para que no sea posible la supervivencia dando consultoría. Sí, existen universidades y algunas empresas con grandes paquetes de software cerrado, pero esos paquetes llevan años desarrollándose --algunos más de 20 años--, con el principio de que no tenían competencia; y cada día que pasa, los paquetes son mejores y es más difícil que el software libre pueda presentar competencia real.

Un ejemplo es el **Whatif**. Es un excelente paquete de ingeniería de proteínas, pero es propietario. Existen proyectos que intentan producir sustitutos, como es el caso de **Platon**, que son lo suficientemente poco conocidos y lo suficientemente poco desarrollados como para no ser tenidos en cuenta. Y cada vez la distancia se hace más grande, y ahora que el Whatif se ejecuta maravillosamente en **Linux**, es probable que acabe con la competencia. Esto supone que en un área tan crítica como el diseño de medicinas a partir de proteínas sintéticas no contaremos con software libre plenamente desarrollado. Por tanto, una primera causa para esta desaceleración del crecimiento del software científico es el sistema de valoración del rendimiento científico. Mas aunque seamos lo suficientemente idealistas como para jugar a la ruleta rusa con nuestra beca, las cosas no suelen ser tan simples. Universidades y centros de investigación suelen estar rígidamente jerarquizados. A diferencia de estructuras como el ejército, donde desde el principio está claro quien es el jefe, en los centros de investigación las relaciones de poder suelen ser delicadas, difusas y necesitan de una gran cantidad de diplomacia. Modificar la licencia de un proyecto científico para licencia

libre supone, como mínimo, convencer al orientador y al jefe de grupo de investigación de la idoneidad de liberar un código a cambio de nada. También hay que convencer a los compañeros de grupo, que ayudan incorporando su conocimiento a nuestra aplicación. En caso de responsabilidades compartidas --varios jefes--, la cosa se complica. En el momento que uno ponga reticencias, todos se cerrarán en banda. Y si el viejo gurú del laboratorio dice que eso no sirve para nada, necesitaremos un milagro para continuar con el proyecto.

3. Posibles soluciones al problema del software científico

La primera solución puede ser la existencia de una licencia libre, mas que recoja las singularidades del software científico. Esta licencia debería compartir el espíritu de la GPL y gran parte de su letra; y debería asegurar la preservación de las libertades propuestas por Stallman. Por otro lado, también debería incluir algunas restricciones adicionales, fáciles de cumplir y que no limitan la libertad final del código según los criterios expuestos por Stallman en «Qué es el Software Libre» <http://www.gnu.org/philosophy/free-sw.es.html>, como son:

· Si se usa el programa para realizar una investigación, y esa investigación termina en uno o varios artículos publicados, los artículos de dicha investigación deben reconocer qué programas fueron usados, así como citar los artículos en los que fueron descritos. Esta restricción no impide la libertad de libre uso, y, de hecho, la totalidad de los programas libres científicos la incluyen; este sistema hasta ahora parece funcionar. La principal objeción a esta restricción es similar a la crítica que Stallman hizo a la licencia BSD (disponible en <http://www.gnu.org/philosophy/bsd.html>); sin embargo, en este caso no se produce el exceso de verborrea en las citaciones; ya que, en caso de que alguien pueda publicar un artículo en una revista indexada, por la naturaleza de revisadas por asesores externos de estas revistas, ha de ser cierta importancia la aportación al código. Y para un cálculo en particular no es tan grande el número de citaciones derivadas. Por ejemplo, el **CCP4** es un proyecto libre, con modelo de desarrollo de bazar y una cantidad muy alta de personas que contribuyeron. Para un uso particular del programa --por ejemplo, cálculo de la estructura de una proteína por sustitución molecular-- es apenas preciso citar, a lo sumo, tres artículos. Observamos que cualquier uso no científico del software --como es el caso del uso educativo-- no entra dentro de esta restricción, y puede ser usado en condiciones iguales a la GPL.

· Si se modifica el código, se ha de identificar adecuadamente la parte modificada como tal. Esto no impide la libertad de modificación, mas permite que, en caso de que la modificación suponga una mejora en el rendimiento o la calidad de los resultados, el autor de la modificación tenga el debido reconocimiento; y, en caso de que la modificación introduzca errores, evita manchar el nombre del programa y del grupo que lo desarrolló. El hecho de comentar el cambio en el código fuente y en el *Changelog* es suficiente. Seguimos manteniendo el espíritu de la GPL, por la que la posibilidad de mejorar el código debe ser libre.

La segunda solución que puede tener el problema del software científico libre es la creación de una revista indexada destinada específicamente a productos cuya licencia sea libre. Debería ser todo lo amplia que fuera posible --es decir, no aceptando apenas la GPL, sino también licencias como la BSD, la artística o la de domínio público--, con la restricción de que la licencia del software asegure las libertades propuestas por Stallman, <http://www.gnu.org/philosophy/free-sw.html>. Sería una buena idea que dicha revista se hiciese ya con todos los requisitos para ser aceptada por indexadores e indexada, <http://www.isinet.com/isi/about/index.html>. Estos requisitos suelen ser de tipo lingüístico, desgraciadamente --el más común es que el título, las palabras clave y el resumen estén disponibles en inglés--. La mejor licencia para esta revista indexada de software libre sería la licencia GPL de documentación, <http://www.gnu.org/copyleft/fdl.html>. Esa hipotética revista de software libre publicaría artículos sobre software libre consistentes en descripciones de software libre, siendo como condición de publicación que el paquete esté terminado, funcional, y que los autores del artículo coincidan con los autores del código.

La tercera solución es el apoyo de las instituciones gubernamentales. En una primera solución, financiando proyectos libres, como ya hace el gobierno alemán, solo que en el área de software científico libre. Otra posibilidad es abrir grandes proyectos temáticos de software libre científico, como ya realizó el Reino Unido con los famosos **CCP**. Es fundamental el ofrecimiento de ofertas de becas sobre áreas informáticas de interés para los gobiernos, exigiendo que el código realizado lo sea bajo licencias libres.

La mejor solución, draconiana, es que aquellos proyectos financiados con dinero del estado deben ser libres. Esta restricción ya la tiene la legislación de los EEUU --y es por eso por lo que el GAMESS es libre--, y es de sentido común: si todos estamos pagando con nuestros impuestos el desarrollo de un programa, deberíamos beneficiarnos todos del desarrollo, y no apenas la institución donde se desarrolla el trabajo y el jefe del que desarrolla el código, que suelen ser los dos únicos beneficiados.

La última corresponde a la universidad: liberando los resultados de proyectos fin de carrera, de resultados de investigación y de tesis de doctorado bajo GPL de documentación, y el código desarrollado bajo una hipotética GPL científica con las características anteriormente comentadas, ganamos todos. La comunidad porque cuenta con más código libre para compartir y el investigador porque ve compensado su trabajo con el reconocimiento, los artículos y las citaciones que necesitará para conseguir becas y fondos para seguir investigando.

Software Libre/Fuente Abierta: hacia la madurez

El proyecto Debian GNU/Linux

Javier Fernández-Sanguino Peña
Coordinador del proyecto de internacionalización al castellano de la distribución Debian GNU/Linux

<jfs@debian.org>

Resumen: *el proyecto Debian GNU/Linux es uno de los más ambiciosos proyectos de software libre en la actualidad, agrupando a un gran número de desarrolladores dispersos geográficamente que trabajan con un único objetivo: crear un sistema operativo totalmente libre.*

Palabras clave: *Debian, sistema operativo, Linux, software libre.*

1. Introducción

Debian comenzó como un proyecto fundado por el proyecto **GNU** <http://www.gnu.org> de la *Free Software Foundation* <http://www.fsf.org>, dirigido por Ian Murdock en agosto de 1993, para construir un sistema operativo basado completamente en programas libres. El sistema operativo es el conjunto de programas básicos y utilidades que hacen que funcione un ordenador. En el corazón de un sistema operativo está el núcleo. El núcleo (*kernel*) es el programa más importante de la computadora, hace todas las cosas de mantenimiento básico y le permite ejecutar otros programas. Debian es independiente del núcleo. Actualmente usa el núcleo de Linux <http://www.linux.org/>, pero Debian también puede funcionar con el Hurd <http://www.gnu.org/software/hurd/hurd.html>.

Debian ha crecido mucho desde sus principios hasta convertirse en una distribución usada por, se estima, cerca de un millón de usuarios. Además, debido al inclinamiento de Debian en ser una distribución enteramente formada por software libre (con el compromiso de su Contrato Social <http://www.debian.org/social_contract>), la totalidad de los paquetes integrantes del cuerpo principal de la distribución (*main*) son libres, estando disponibles en código fuente.

Aquí hay algunas de las fechas importantes de Debian:

- Las versiones 0.01-0.90 vieron la luz entre agosto y diciembre de 1993.
- La versión 0.91 salió en enero de 1994, contaba con cerca de 30 desarrolladores y un sistema de paquetes primitivos.
- La versión 0.93R5 salió en marzo de 1995, en ella apareció el programa **dpkg**.
- La versión 0.93R6 surgió en noviembre de 1995, con cerca de 60 desarrolladores, soportaba el sistema **a.out** y tenía la primera versión de **dselect**.
- La versión 1.0 nunca salió. Se convirtió posteriormente en la versión 1.1.
- **Buzz**, la versión 1.1 salió en junio de 1996, con 474 paquetes, el *kernel* 2.0 de Linux y soporte completo de **ELF**.
- **Rex** (1.2) aparecería en diciembre de 1996 con 848 paquetes y 120 desarrolladores.
- **Bo** (1.3) saldría en julio de 1997, tendría 974 paquetes y 200 desarrolladores.
- La versión **Hamm** (2.0) vio la luz en julio de 1998, cuenta con más de 1500 paquetes en los que trabajan más de 400 desarrolladores, tiene pleno soporte de **libc6**, aunque aún mantiene librerías para los programas compilados con **libc5**.
- La siguiente versión: **Slink** (2.1) se distribuyó el 9 de marzo de 1999. Contando con más de 2500 paquetes, y consta de cuatro CD-ROMs, dos de binarios y dos de fuentes. Para solventar problemas de seguridad descubiertos así como errores relacionados con el año 2000 se realizaron hasta cinco revisiones de esta versión.
- La versión actual **Potato** (2.2) se distribuyó 15 de agosto del año 2000. Cuenta con más de 4.000 paquetes de software y, sólo en binarios, iguala a la versión anterior (binarios y fuentes). Ha habido tres revisiones de esta versión, solucionando problemas graves y de seguridad detectados.
- La versión en preparación **Woody** (3.0), que está próxima a ser distribuida. Supera a la anterior con más de 6.000 paquetes de software.

Pero el crecimiento de Debian no ha sido solamente en la cantidad de programas incluidos en la distribución, sino también en el número de personas que trabaja en ella; actualmente cuenta con más desarrolladores que cualquier otra distribución de GNU/Linux. Y ha sido reconocida de forma internacional por multitud de entidades:

- El Gobierno Australiano <http://www.debian.org/News/2001/20010727>
- HP <http://http://www.computerworld.com/cwi/story/0,1199,NAV47_STO60507,00.html>
- IBM Global Services <http://www.debian.org/News/weekly/2000/20000408>
- Compaq <http://www.debian.org/News/weekly/2000/20000302>
- France Telecom. <http://www.debian.org/News/weekly/

Autor

Javier Fernández-Sanguino es miembro del proyecto Debian desde enero de 1998. Dentro del proyecto es el coordinador del proyecto de internacionalización al castellano de la distribución Debian GNU/Linux, miembro del grupo de trabajo del sitio web, y mantiene más de sesenta paquetes de software que varían desde aplicaciones para usuarios finales a herramientas de desarollo y seguridad. Igualmente es colaborador habitual de revistas relacionadas con GNU/Linux.

1999/19990917>
· Corel <http://www.debian.org/News/weekly/1999/19990421a>
· EBay <http://www.debian.org/News/weekly/1999/19991020>
· VA Linux Systems <http://www.debian.org/News/weekly/1999/19991012>
· Linux Hardware Solutions <http://www.debian.org/News/weekly/1999/19990225a>

Esto sin contar las múltiples donaciones de equipamiento que ha recibido de múltiples compañías, entre otras: Novare, VA Linux, Compaq, y Sun.

Debian se trata de la única distribución importante de GNU/Linux mantenida solamente por voluntarios, es decir, sin un enfoque comercial, esto tiene ventajas y desventajas.

En primer lugar, las personas que se dedican a Debian tienen una alta motivación en participar en la misma, y se actualiza la distribución diariamente, apareciendo paquetes nuevos de software constantemente. Al mismo tiempo, existe un compromiso de calidad, no se desea distribuir software con errores. Por esta razón, se ha retrasó, por ejemplo, la versión 2.0 de Debian hasta que se han arreglado *bugs*, tanto por los problemas de pasar todos los programas a libc6 como de agujeros de seguridad que se hubieran descubierto entonces. Algo que, por ejemplo, RedHat <http://www.redhat.com> no hizo en el mismo momento, habiéndose visto obligada a distribuir RedHat 5.1 inmediatamente después de 5.0 para arreglar muchos de los problemas de seguridad existente en los programas con los que se distribuía.

En segundo lugar, dada su actitud abierta a la participación de todos, en el mismo espíritu original de Linux, constantemente hay personas que se unen a Debian para participar aportando su granito de arena, no solamente haciendo paquetes de programas, sino colaborando en el servidor de web, traduciendo ocumentación de Debian, documentando fallos, o ayudando a los usuarios, a través de listas de correo <http://www.debian.org/MailingLists> o de otras herramientas como la FAQ-o-matic <http://www.debian.org/cgi-bin/fom>, a resolver los problemas que tienen con la distribución. Este servicio es, sin duda, mucho más ágil, rápido y eficaz que el ofrecido en muchas ocasiones por las compañías de software.

En el lado de desventajas hay que decir que Debian tiene un mayor componente técnico que otras distribuciones. También, dada la naturaleza voluntaria de los desarrolladores, es posible que ciertos paquetes no estén tan actualizados como debieran, quizás porque sus desarrolladores han dejado de actualizarlos y nadie se ha hecho cargo. Sin embargo esto es algo que todos los desarrolladores tratan de evitar y, aunque cada desarrollador mantiene sus paquetes, no es raro que otro desarrollador (incluso un usuario) envíe una actualización del paquete para arreglar un problema o actualizarlo.

2. Organización

La organización de Debian puede parecer inexistente de cara a las personas que no conocen el grupo, pero esto no es así. Debian está amparada por una asociación sin ánimo de lucro llamada Software en el Interés Público (**SPI**) <http://www.spi-inc.org>, que tiene su sede en Estados Unidos y cuyo objetivo es ayudar a las organizaciones a desarrollar y distribuir software y hardware abierto. La asociación fue creada el 16 de junio de 1997 y Debian no es el único proyecto apoyado por esta organización, también están:
· **GNOME** (GNU *Network Object Model Environment*) <http://www.gnome.org> el proyecto que intenta desarrollar un entorno de escritorio dirigido al usuario sobre y con software libre.
· **LSB** (*Linux Standard Base*) <http://www.linuxbase.org> proyecto que desarrolla y promueve la de_nición de estándares que favorecen la compatibilidad entre distribuciones de Linux y que permiten a las aplicaciones funcionar en cualquiera de éstas. También coordina esfuerzos para animar a las compañías de software a desarrollar y portar aplicaciones a Linux.
· **Open Hardware** <http://www.openhardware.org/> es un sistema de certificación para hardware que permite a los desarrolladores controladores de dispositivos para sistemas operativos libres.
· **Berlin** <http://www.berlin-consortium.org> un entorno de ventanas desarrollado por miembros de la comunidad del software libre que pretende desarrollar el entorno de usuario más _exible y más potente posible.

SPI gestiona las marcas registradas: Open Source, Open Hardware y Debian.

Sin embargo, Debian tiene a su vez su propia organización, con un Presidente (actualmente Ben Collins), un Secretario y un Comité Técnico; así como un gran número de personas encargados de distintas áreas del desarrollo de Debian, desde la producción de CDs, a la seguridad pasando por el sistema de seguimiento de errores (*bugs*). Y, a continuación, todo el grupo de desarrolladores que, de forma altruista al igual que los anteriores, donan parte de su tiempo para contribuir en la distribución. Estas contribuciones son de distintos tipos aunque, fundamentalmente se refiere a la creación de paquetes de programas que se incluyen en la distribución siguiendo la política de ésta. La facilidad con la que los nuevos desarrolladores pueden contribuir en la distribución y pueden introducir nuevos programas ha hecho, a lo largo de su historia, que Debian creciera en ambos aspectos. Debian cuenta, en el momento de escribir este artículo, con cerca de 600 desarrolladores (consulte el mapa de desarrolladores <http://www.debian.org/devel/developers. loc>) distribuidos por todo el globo. Evidentemente, no todos tienen el mismo nivel de compromiso, dentro de éstos hay personas dedicadas a trabajar en exclusiva para Debian (financiados por empresas externas) y personas que colaboran de forma esporádica. Los desarrolladores toman las decisiones que sean necesarias para el funcionamiento del proyecto, a través de un sistema de votaciones que se inauguró para votar su Constitución, y ha sido utilizado en muchas ocasiones: elección de nuevos presidentes, selección del logotipo, decisiones sobre la política de Debian, etc...

La gran mayoría de las discusiones se realizan a través de correo electrónico ya que es el único medio que puede permitir a tantas personas colaborar de forma organizada.

Los desarrolladores Debian están involucrados en una gran variedad de tareas, incluyendo la administración del Servidor

de Web <http://www.debian.org/> y Servidor de FTP <ftp://ftp.debian.org/>, diseño gráfico, análisis legal de licencias de software, escribir documentación y, por supuesto, mantener paquetes de software.

En el interés de comunicar su filosofía y atraer desarrolladores que creen en los principios que Debian protege, el Proyecto Debian ha publicado un número de documentos que contienen nuestros valores y sirven como guías de lo que significa ser un «Desarrollador Debian»:

- El Contrato Social de Debian <http://www.debian.org/social_contract> es una afirmación del compromiso de Debian a la Comunidad del Sofware Libre. Cualquier que esté de acuerdo en acogerse al Contrato Social puede convertirse en un desarrollador <http://www.debian.org/doc/maint-guide/>. Cualquier desarrollador puede introducir nuevo software en Debian siempre que éste cumpla este criterio de software libre, y cumpla con sus éstandares de calidad.
- El documento *Debian Free Software Guidelines* (Las Guías de Software Libre de Debian) <http://www.debian.org/social_contract\#guidelines> es un informe claro y conciso sobre los criterios de Debian sobre software libre. La DFSG es de gran in_uencia en el Movimiento del Software Libre, y proporciona las bases de la definición Open Source <http://opensource.org/osd.html >.
- La Política de Debian Policy <http://www.debian.org/doc/debian-policy/>. es un especificación extensiva de los estándares de calidad del Proyecto Debian.

Los desarrolladores de Debian también están involucrados en otros proyectos; algunos específicos a Debian, otros en los que está involucrado parte o toda la comunidad Linux. Algunos ejemplos incluyen:

- El *Linux Standard Base* (**LSB**) <http://www.linuxbase.org/>. El LSB es un proyecto que pretende estandarizar el sistema básico de Linux, lo que permitiría a desarrolladores de software y hardware ajenos a desarrollar fácilmente programas y controladores de dispositivos para Linux en general, más que para una distribución de Linux en particular.
- El *Filesystem Hierarchy Standard* (**FHS**) <http://www.pathname.com/fhs/>es un esfuerzo para estandarizar la distribución del sistema de ficheros de Linux. El FHS permitirá a desarrolladores de software a concentrar sus esfuerzos en diseñar programas, sin tener que preocuparse sobre cómo se instalará su paquete en diferentes distribuciones de Linux.
- **Debian Jr**.<http://www.debian.org/devel/debian-jr/> es nuestro proyecto interno, orientado hacia asegurarnos de que Debian tiene algo que ofrecer a nuestros usuarios más jóvenes. Para más información general sobre Debian, vea las Preguntas Frecuentes de Debian <http://www.debian.org/doc/FAQ/>.

3. La distribución

La combinación de la filosofía y metodología de Debian, las herramientas GNU, el núcleo de Linux y otros programas importantes de software libre, forma una distribución de software única llamada **Debian GNU/Linux**. Esta distribución está formada por un gran número de paquetes. Cada paquete en la distribución contiene ejecutables, *scripts*, documentación e información de configuración y tiene un mantenedor que es el principal responsable de mantener el paquete al día, seguir informes de error, y comunicar con los autores principales del software empaquetado. La gran base de usuarios, combinada con nuestro sistema de seguimiento de errores, se asegura de que los errores se encuentren y arreglen rápidamente.

La atención de Debian al detalle permite producir un distribución de alta calidad, estable y escalable. La instalación puede configurarse fácilmente para servir muchos perfiles, desde cortafuegos reducidos con el menor número de servicios imprescindible, pasando por estaciones de trabajo científicas a servidores de red de alta gama.

El sistema que distingue a Debian de otras distribuciones GNU/Linux es su sistema de gestión de paquetes. Estas herramientas dan al administrador de un sistema Debian control completo sobre los paquetes instalados en su sistema, incluyendo la capacidad de instalar un sólo paquete o actualizar el sistema operativo por completo. Los paquetes individuales también pueden protegerse para no ser actualizados. También puede decir al sistema de gestión de paquetes qué software ha compilado Vd. mismo y qué dependencias cumple.

Para proteger su sistema contra caballos de Troya y otros programas malévolos, Debian verifica que los paquetes provienen de sus mantenedores Debian auténticos utilizando sistemas de clave pública. Los empaquetadores de Debian también ponen gran cuidado en configurarlos de forma segura. Si se declara un problema de seguridad con los paquetes entregados, los parches están por lo general rápidamente disponibles. Con el sencillo sistema de actualización de Debian, se pueden descargar e instalar arreglos de seguridad automáticamente a través de Internet. Los usuarios también pueden verificar que el software descargado no ha sido manipulado dado que incorpora los mismos mecanismos de sistemas de clave pública para verificar su integridad.

El principal, y mejor, método para obtener soporte de un sistema Debian GNU/Linux y comunicarse con los Desarrolladores Debian es a través de las muchas listas de distribución <http://www.debian.org/MailingLists/> mantenidas por el Proyecto Debian (hay más de 90 en el momento de escribir estas líneas).

4. La última estable: Debian 2.2

Todas las arquitecturas de Debian se basan, a partir de Debian 2.2, en el núcleo de Linux 2.2.16 (última versión estable). También está basada en la nueva versión 2.1.2 de la biblioteca C de GNU. Aunque la nueva **glibc** hace que los nuevos paquetes no se puedan instalar en la versión anterior, mantiene compatibilidad binaria hacia atrás con los viejos paquetes compilados con la glibc 2.0 de Debian versiones 2.1 y 2.0, y compatibilidad fuente prácticamente completa con esas fuentes antiguas.

En esta publicación, la mayoría de las utilidades básicas del sistema han empezado a usar **PAM** (*Pluggable Authentication Modules* o módulos de autentifiación enchufables), que suministran a los administradores de sistemas un poderoso

método de controlar el acceso al sistema y los métodos de autentificación. PAM permite administrar desde un único punto la autentificación y la gestión de cuentas. Si desea cambiar sus programas de autentificación a un esquema diferente (p. ej. **OPIE**, **Kerberos**, etc...) sólo necesita modificar los ficheros de configuración de PAM para esos programas, en lugar de recompilar el programa en si.

La publicación Debian 2.2 es la primera versión de Debian que incluye soporte completo para nuestros usuarios japoneses, que tenían que usar paquetes adicionales de Debian JP hasta ahora para obtener soporte a caracteres *multi-byte*. Adicionalmente, se ha aumentado el nivel de internacionalización, y mejorado el soporte para la mayoría de los idiomas con alfabeto no latino. También se ha mejorado el soporte de idiomas europeos, con más y mejores traducciones a muchos idiomas.

El número de paquetes que incluye la distribución principal está ahora alrededor de 4.000, aumentando en más de 50%, como es habitual (de hecho son 1.500 paquetes nuevos). Algunos de los nuevos paquetes son:

· **postfix**. Un nuevo agente de transporte de correo seguro.
· **openssh**. Una implementación libre de la *shell* segura.
· **openldap**. Paquetes de cliente y servidor **LDAP**.
· **w3m**. Un nuevo navegador basado en texto, con soporte de tablas.
· **gdm**. El gestor de ventanas de GNOME.
· **cvsup**. Un sistema eficiente de réplicas para **CVS**.
· **everybuddy**. Un cliente de mensajería integrado.
· **reportbug**. Una herramienta para avisar de problemas a Debian GNU/Linux.
· **zope**. Un servidor de aplicaciones web para sitios dinámicos.
· **xmms**. El sistema de multimedia X.
· **kaffe**. Una máquina virtual para código Java libre, capaz de hacer **JIT**.
· **gnapster**. Un interfaz al popular sistema de compartición MP3.

5. La diferencia entre la versión estable y la inestable

Debian sigue unas fases claramente definidas en el desarrollo y publicación de sus versiones. En primer lugar, existe una versión en desarrollo, conocida como «inestable», a la cual se van incorporando nuevos programas, nuevas versiones de programas y se van corrigiendo *bugs* y errores. Tras ese periodo, la versión inestable se «congela» y a partir de ese momento ya no se aceptan nuevos programas y se dedica todo el esfuerzo a realizar diversos ciclos de prueba, notificación y corrección intensiva de *bugs* o errores. Una vez corregidos todos los errores críticos o retiradas aquellas aplicaciones cuyos errores críticos no ha dado tiempo a corregir, se «libera»: se tiene entonces la «versión estable».

Sin embargo, «inestable» en Debian no significa «peligroso» o «inseguro» sino «en desarrollo». Para usos no críticos es perfectamente utilizable y en general un usuario encontrará que a menudo la versión inestable de Debian no lo es más que las versiones definitivas de otras distribuciones comerciales. No obstante, para servicios críticos es recomendable utilizar la versión estable, que es la que está probada intensivamente y depurada. El tiempo de salida de una nueva versión de Debian es alto y esto se debe a su estricto sistema de depuración de errores, a carecer de urgencias comerciales, a mantenerse con el trabajo sin remunerar y en ratos libres de los desarrolladores, y al inmenso número de paquetes que, en los últimos tiempos, se ha hecho difícilmente manejable. El objetivo de la versión estable de Debian no es «estar a la última», sino conseguir un sistema robusto y fiable con los menos fallos posibles. Quien necesite tener acceso a las últimas versiones de los programas y a nuevas aplicaciones puede usar la versión inestable, o bien puede buscar paquetes no oficiales o incluso realizar uno mismo el paquete o la compilación.

6. Conclusiones

Realmente puede haber muchos puntos a favor y en contra de Debian pero el mayor punto a favor de soportar Debian, haciendo uso de ésta, colaborando en la documentación, o aportando nuevos paquetes, es que se trata de un proyecto en el que tienen cabida todo tipo de iniciativas. A diferencia de otras distribuciones con interés comerciales, en las que se puede llegar a una división de recursos entre lo que sería en realidad, esencialmente, la misma distribución, esto no sucede en Debian.

Notorio es por ejemplo, el hecho de los esfuerzos duplicados en distribuciones derivadas de RedHat en el entorno hispano, como Esware, Hispafuentes, o Eurielec Linux, debido a la imposibilidad de contribuir los muchos esfuerzos realizados por las personas a la distribución «original»,lo que tiene como resultado el desarrollo de productos divergentes. De igual forma, se puede hablar de las distribuciones derivadas de RedHat en entornos más globales como SuSE o Mandrake.

Debian es, al contrario, un punto de concentración. Si se quiere, se puede entender como una «enana Blanca», en lugar de explotar como una «supernova» extendiendo sus restos por el universo conocido, o llevar al olvido todo lo que pase próximo como haría un «agujero negro»; seguirá absorbiendo materia, gracias a su fuerte gravedad, y viviendo mucho, mucho más tiempo.

7. Más información

Puede encontrarse más información sobre el Proyecto Debian en:

· El servidor principal de Debian <http://www.debian.org>
· Las páginas de la última versión estable <http://www.debian.org/releases/stable/>
· Información sobre cómo conseguir Debian <http://www.debian.org/distrib/>
· El proyecto de documentación de Debian <http://www.debian.org/doc/ddp>

Software Libre/Fuente Abierta: hacia la madurez

Sistemas de ficheros con *Journaling* en Linux

Ricardo Galli
Dept. de Matemàtiques i Informàtica (Universitat de les Illes Balears)

<gallir@uib.es>

Resumen*: en este artículo se realiza una comparativa de rendimiento de sistemas de ficheros tradicionales con sistemas con* journaling *en Linux. Hay que señalar ante todo que no hay un ganador claro, XFS es mejor en algunos aspectos, ReiserFS en otros, y ambos son mejores que Ext2 en el sentido que son comparables en rendimiento pero ellos son sistemas de ficheros con* journaling*, y ya se sabe cuales son sus ventajas. Quizás la moraleja mas importante es que el sistema de* buffer-cache *del Linux sorprende por su rendimiento, afectó, positivamente, a todos los resultados de mis compilaciones, copias y lecturas/escritas aleatorias. Así que diría, compra memoria RAM e instala un sistema con* journaling *lo antes posible ...*

Palabras clave: *Linux, sistemas operativos*, page-cache*,* buffer-cache*,* journal file systems*, Ext3, XFS, ReiserFS, JFS.*

1. Introducción

El párrafo que aparece en el resumen es un extracto del que fue el segundo de una serie de artículos publicados en el web de **BULMA**, <http://bulmalug.net/body.phtml?nIdNoticia=642>, que es la Asociación de Usuarios de Linux en Baleares que describió los resultados de una segunda ronda de comparativas de rendimiento de sistemas de ficheros tradicionales con sistemas con *journaling*.

Aunque informales, ambas comparativas analizaron el Fat32, Ext2, ReiserFS, XFS y JFS para diferentes casos: las herramientas Mongo de Hans Reiser, copia de ficheros, compilación del *kernel* y un pequeño programa en C que simula patrones de acceso de sistemas de base de datos de tamaño medio.

Ambos artículos estuvieron entre los primeros en hacer dichas comparativas de los sistemas de ficheros con *journaling* (registro o diario). Nuestro servidor web de Bulma se vio sobrecargado debido a su publicación en <slashdot.org>. Estas pruebas sirvieron principalmente para hacer desaparecer el mito que los sistemas de ficheros con *journaling* son significativamente más lentos que los sistemas tradicionales de Unix (UFS) y del Ext2, derivado del anterior y estándar de Linux durante varios años.

Desde esos días se han publicado más comparativas, pero la verdad es todavía la misma: no hay un claro ganador. Algunos sistemas se comportan mejor que otros en algunos casos, pero no en todos. Por ejemplo, ReiserFS es muy bueno leyendo ficheros pequeños a medianos, XFS tiene mejor rendimiento para ficheros grandes y se dice que JFS facilita mucho la migración de sistemas con OS/2 Warp y AIX a Linux. En este artículo describimos todos los sistemas con *journaling* disponibles para Linux: Ext3, ReiserFS, XFS y JFS. También introducimos los conceptos básicos de *buffer-cache*, y *page-cache* del núcleo del Linux. El rendimiento de los diferentes sistemas de ficheros es afectado de forma importante por el uso de las dos técnicas anteriores. Y aún más importante, no sólo el rendimiento se ve afectado, sino también la re-programación de módulos importantes originalmente desarrollados para otros sistemas operativos. Por ejemplo, Silicon Graphics (SGI) tuvo que programar un nuevo módulo, *pagebuf*, que hace de interfaz entre su propia técnica de *buffering* en el XFS y los módulos del *page-cache* de Linux.

2. El sistema de ficheros virtual en Linux

Un *fichero* es una abstracción muy importante en programación. Los ficheros sirven para almacenar datos de forma permanente y ofrecen un pequeño conjunto de primitivas muy potentes (abrir, leer, avanzar puntero, cerrar, etc.). Los ficheros se organizan normalmente en estructuras de árbol, donde los nodos intermedios son directorios capaces de agrupar otro ficheros.

El sistema de ficheros es la forma en que el sistema operativo organiza, gestiona y mantiene la jerarquía de ficheros en los dispositivos de almacenamiento, normalmente discos duros. Cada sistema operativo soporta diferentes sistemas de ficheros. Para mantener la modularización del sistema operativo y proveer a las aplicaciones con una interfaz de programación (API) uniforme, los diferentes sistemas operativos implementan una capa superior de abstracción denominada *Sistema de Ficheros Virtual* (*VFS: Virtual File System*). Esta capa de software implementa las funcionalidades comunes de los

Autor

Ricardo Galli (<http://m3d.uib.es/gallir/>), nacido en 1965, es profesor de la Universidad de las Islas Baleares desde 1992. Tiene un título de Ingeniería en Informática (Argentina) y es Doctor en Informática por la Universidad de las Islas Baleares. Su tesis doctoral se tituló *Data Consistency Methods for Collaborative 3D Editing*. Trabaja en el área de gráficos por ordenador y CSCW (*Computer Supported Collaborative Work*). Ha publicado más de 40 artículos en conferencias nacionales e internacionales, revistas y capítulos de libros. También ha participado y liderado varios proyectos europeos de I+D. también es coordinador del programa Socrates/Erasmus. Fue co-fundador del proveedor de servicios Internet Atlas-IAP S.L. Ha dirigido y coordinado el desarrollo de 6 ediciones digitales de periódicos de las Baleares, varios proyectos de comercio electrónico y colabora en diversos proyectos de Open Source. Es miembro de Bulma, la Asociación de Usuarios de Baleares. Actualmente está coordinando el proyecto europeo *e-Content: MNM - Minority News Paper to Multimedia* y es también el director técnico de ISA3D, una empresa europea (Portugal, Holanda, España), surgida del proyecto europeo M3D (<http://www.m3d.org>).

diversos sistemas de ficheros implementados en la capa inferior.

Los sistemas de ficheros soportados por Linux se clasifican en tres categorías:
1. Basados en disco: discos duros, disquetes, CD-ROM. Estos sistemas son Ext2, ReiserFS, XFS, Ext3, UFS, ISO9660, etc.
2. Sistemas remotos (de red): NFS, Coda, y SMB.
3. Sistemas especiales: procfs, ramfs y devfs.

El modelo general de ficheros puede ser interpretado como orientado a objetos, donde los objetos son construcciones de software (estructura de datos y funciones y métodos asociados) de los siguientes tipos:
· **Super bloque**: mantiene información relacionada a los sistemas de ficheros montados. Está representado por un bloque de control de sistema almacenado en el disco (para sistemas basados en disco).
· **i-nodo**: mantiene información relacionada a un fichero individual. Cada i-nodo contiene la meta-información del fichero: propietario, grupo, fecha y hora de creación, modificación y último acceso, más un conjunto de punteros a los bloques del disco que almacenan los datos del fichero.
· **Fichero**: mantiene la información relacionada a la interacción de un fichero abierto y un proceso. Este objeto existe sólo cuando un proceso interactúa con el fichero.
· **Dentry**: enlaza una entrada de directorio (*pathname*) con su fichero correspondiente. Los objetos *dentry* recientemente usados son almacenados en una caché (*dentry cache*) para acelerar la translación desde un nombre de fichero al i-nodo correspondiente.
Todos los sistemas Unix modernos permiten que los datos del sistema de ficheros sean accedidos de dos formas distintas (ver **figura 1**):

1. Asociación (*mapping*) de memoria con **mmap**: la llamada de sistema mmap() permite a los procesos de usuario acceder de forma directa a los datos del *page-cache* mediante asociación de memoria. El objetivo de mmap es permitir el acceso a los datos mediante direcciones del sistema de memoria virtual, por lo que los datos de ficheros pueden ser tratados como estructuras de memoria estándares. Los datos del fichero son leídos y copiados a la *page-cache* bajo demanda (*lazily*) cuando se generan fallos de página por intentos de acceso a páginas no residentes en RAM.
2. Llamadas de sistema de acceso directo al sistema de E/S de bloques, tales como **read** y **write**: La llamada de sistema read() lee los datos de los dispositivos de bloques a la caché del núcleo (para CDs y DVDs se puede evitar esta copia mediando el parámetro O_DIRECT del *ioctl*), luego se copian al espacio de direcciones del proceso. La llamada de sistema *write*() copia los datos en la dirección opuesta, desde la memoria del proceso a la caché del núcleo y eventualmente, en un *futuro próximo*, a los bloques correspondientes del disco. Estas interfaces son implementadas usando el *buffer-cache* o el *page-cach*e para almacenar temporalmente en la memoria del núcleo.

2.1. *Page-cache* y *buffer-cache* del Linux

En versiones anteriores del Linux (y en Unix en general), las operaciones de correspondencia de memoria (*mapping*) eran gestionadas por el sistema de memoria virtual (VM o MM), mientras que las llamadas de E/S por bloques fueron gestionadas independientemente por el subsistema de E/S. Por ejemplo, en Linux hasta la versión 2.2.x, el sistema de memoria virtual y el de E/S tenían cada uno sus propios mecanismos de caché para mejorar el rendimiento: *page-cache* y *buffer-cache* respectivamente.

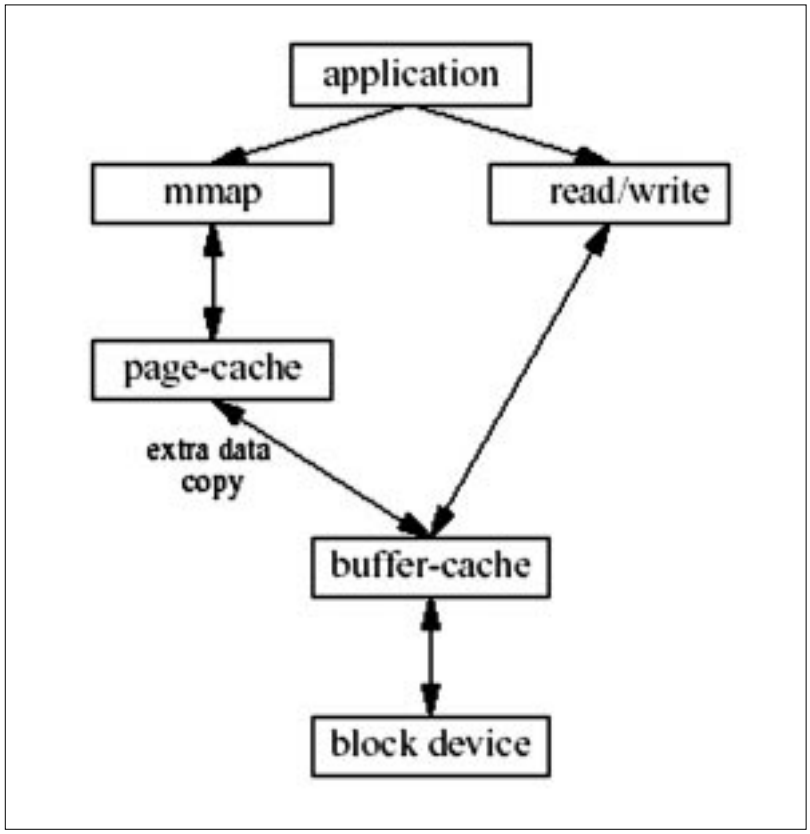


Figura 1. Buffer-cache *y* page-cache

2.1.1. Buffer-cache

El *buffer-cache* mantiene copias de bloques de disco individuales. Las entradas del caché están identificadas por el dispositivo y número de bloque. Cada *buffer* se refiere a cualquier bloque en el disco y consiste de una cabecera y un área de memoria *igual* al tamaño del bloque del dispositivo. Para minimizar la sobrecarga, los *buffers* se mantienen en una de varias listas enlazadas: sin usar (*unused*), libres (*free*), no modificadas (*clean*), modificadas (*dirty*), bloqueadas (*locked*), etc.

Por cada operación de lectura, el subsistema de *buffer-cache* debe buscar si el bloque en cuestión ya está copiado en la memoria. Para hacerlo de forma eficiente se mantienen tablas de dispersión para todos los *buffers* presentes en la caché. El *buffer-cache* es también usado para mejorar las operaciones de escritura, en vez de copiar todos los bytes inmediatamente al disco, el núcleo los almacena temporalmente en el *buffer-cache* e intenta agrupar varias operaciones para escribirlas al disco simultáneamente. Un *buffer* que está esperando por ser copiado al disco se denomina *modificado* (*sucio* o *dirty*).

2.1.2. Page-cache

A diferencia del anterior, el *page-cache* mantiene páginas completas de la memoria virtual (4 KB en la plataforma x86). Las páginas pertenecen a ficheros en el sistema de ficheros, de hecho las entradas en el *page-cache* están parcialmente indexadas por el número de i-nodo y su desplazamiento en el fichero. Pero una página es casi siempre más grande que un bloque de disco, y los bloques que conforman dicha página pueden estar almacenados de forma discontinua en el disco. El *page-cache* es principalmente usado para satisfacer los requerimientos de interfaz del subsistema de memoria virtual, que usa páginas de tamaño fijo de 4KB, con el VFS, que usa bloques de tamaño variable u otro tipos de técnicas tales como *extents* en XFS y JFS.

2.2. Unificación del *page-cache* y *buffer-cache*

Los dos mecanismos anteriores operaron de forma semi-independiente uno del otro. El sistema operativo tenía que tener un especial cuidado para mantener sincronizados los dos caches y así prevenir que las aplicaciones reciban datos inválidos. No sólo eso, si el sistema estaba escaso de memoria, el sistema operativo tenía que tomar decisiones muy difíciles para liberar memoria del *buffer-cache* o del *page-cache*.

El *page-cache* tiende a ser más sencillo de administrar debido a que representa más directamente los conceptos usados en los niveles superiores del código del núcleo. El *buffer-cache* además tiene las limitaciones que los datos siempre deben ser asociados al espacio de direcciones del núcleo, lo que agrega límites artificiales a la cantidad de datos que pueden mantenerse en caché, ya que el hardware moderno puede tener fácilmente más RAM que el espacio de memoria del núcleo. Con el tiempo, partes importantes del núcleo han pasado de usar el *buffer-cache* al *page-cache* por razones de simplicidad de codificación. Pero los bloques individuales del *page-cache* estaban todavía gestionados por el *buffer-cache*, lo que creaba confusiones entre el uso y programación de ambos niveles.

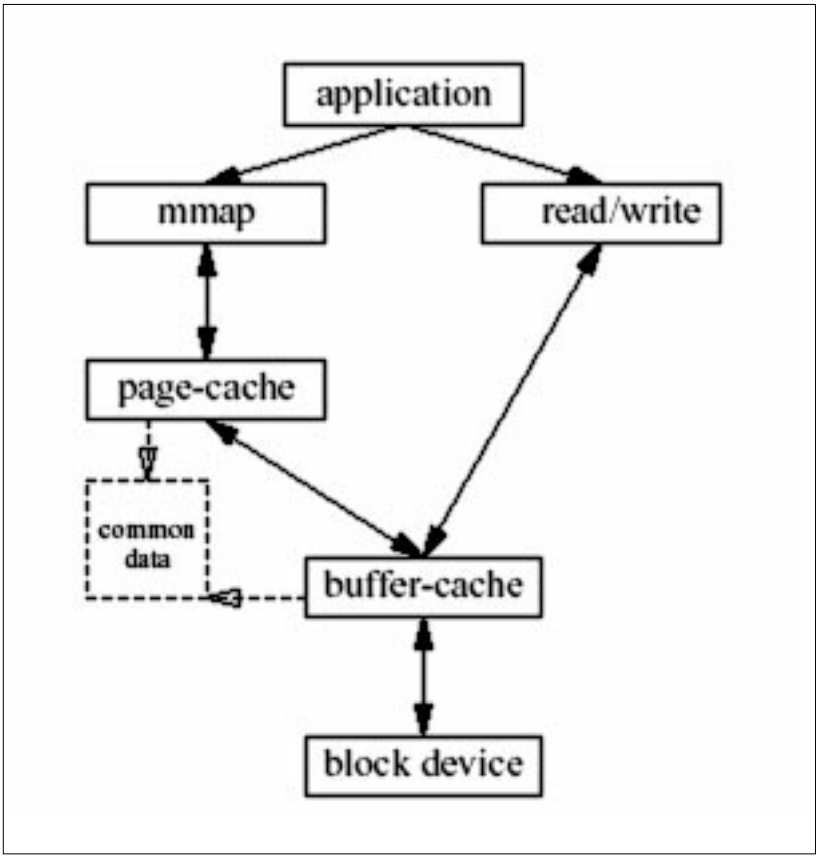


Figura 2. Los datos son compartidos por el page-cache *y* buffer-cache

Esta falta de integración llevó a un rendimiento ineficiente y falta de flexibilidad. Para lograr un buen rendimiento es importante que los sistemas de memoria virtual y E/S estén bien integrados. La forma elegida en Linux para reducir las ineficiencias de las dobles copias es almacenar los datos sólo una vez (en el *page-cache)* y mantener punteros de cada elemento en el *buffer-cache* (ver **figura 2**).

Las correspondencias temporales de memoria de las páginas en el *page-cache* para soportar *read*() y *write*() son raramente necesarias ya que Linux traduce permanentemente toda la memoria física al espacio de direcciones del núcleo. Linux además usa una técnica interesante, el número de bloque donde está una página en el disco está almacenada en forma de una lista en memoria con una estructura denominada *buffe_head*. Cuando una página modificada tiene que ser grabada en disco, las solicitudes de E/S pueden ser pasadas directamente al gestor del dispositivo, sin necesidad de hacer ningún tipo de operación adicional para determinar donde deben ser escritos los datos.

La unificación del page-cache

Siguiendo los conceptos del *Unified I/O and Memory Caching Subsystem for NetBSD*, Linus Torvalds quiso cambiar radicalmente el comportamiento del *page-cache* en el núcleo de Linux. El 4 de mayo de 2001, escribió el siguiente mensaje a la lista de desarrolladores (linux-kernel):

Quiero re-escribir block_read/write para que use el page cache, pero no porque impactará nada en esta discusión. Quiero hacerlo al principio del 2.5.x porque:
- acelerará los accesos
- reusará mejor el código existente y conceptualizará las cosas más claramente (i.e. convertiría un disco en sistema de ficheros _realmente_ simple con sólo un fichero enorme ;-)
- hará que la gestión de memoria sea mucho mejor para cosas como el fsck - la presión de memoria está diseñada para trabajar en cosas como el page-cache.
- será una cosa menos que use el buffer cache como una «caché» (quiero que la gente piense y use el buffer cache como una entidad de _E/S_, no como una caché).
No cambiará el «caché al arranque» para nada (porque aún en el page cache, no hay nada en común entre la representación virtual de un _fichero_ (o metadata) y la representación virtual de un _disco_).

Aunque estos cambios no estaban programados hasta el 2.5.x, Linus finalmente se decidió por integrar una cantidad considerable (más de 150) de parches de Andrea

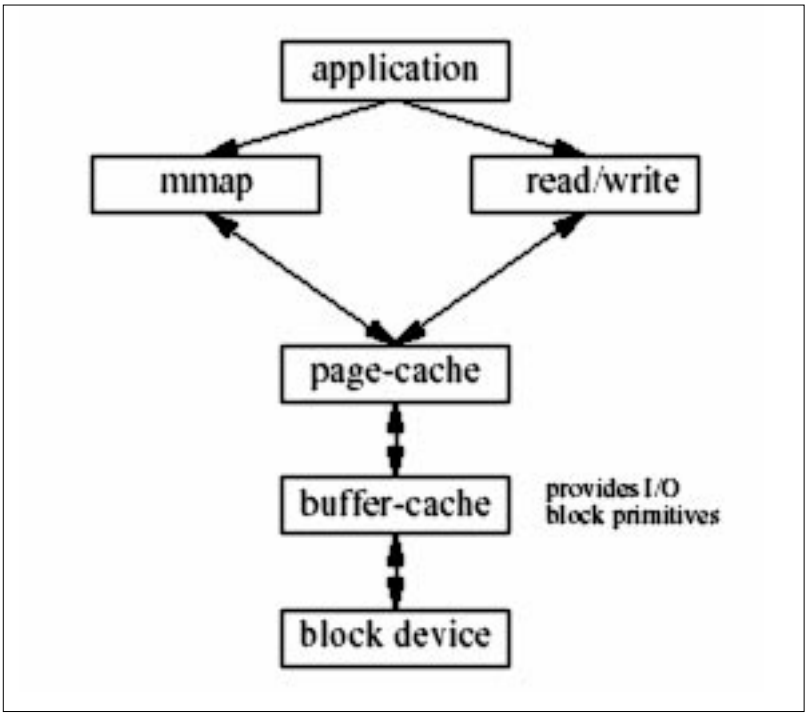


Figura 3. Unificación a page-cache

Arcangeli, más sus propios cambios, y liberó el 2.4.10, que finalmente unificaba el *page-cache* y *buffer-cache* (ver **figura 3**). A raíz de estos cambios, se esperan importantes mejoras en el sistema de E/S, sobre todo cuando se ajusten mejor los parámetros del sistema de memoria virtual (al momento de escribir este artículo, la versión disponible del Linux es 2.4.13, que ya muestra importantes mejoras de rendimiento, sobre todo para sistemas con poca memoria, y un consumo bastante menor de RAM para el *page* y *buffer cache*).

3. Sistemas de ficheros con *Journaling*

Durante mucho tiempo, el sistema de ficheros estándar en Linux fue el Ext2. Éste fue diseñado por Wayne Davidson con la colaboración de Stephen Tweedie y Theodore Ts'o. Es una mejora al sistema anterior, Ext, diseñado por Rémy Card. El Ext2 está basado en i-nodos (asignación indexada). Cada i-nodo mantiene la meta-información del fichero y los punteros a los bloques con los datos «reales».

Como hemos mencionado antes, para mejorar el rendimiento de las operaciones de E/S, los datos del disco son temporalmente almacenados en la memoria RAM a través del *page-cache* y *buffer-cache*. Los problemas surgen si hay un corte de suministro eléctrico antes que los datos modificados en la memoria (*dirty buffers*) sean grabados nuevamente al disco. Se generaría una inconsistencia en el estado global del sistema de ficheros. Por ejemplo, un nuevo fichero que todavía no fue «creado» en el disco u otros que hayan sido borrados pero sus i-nodos y bloques de datos todavía permanecen como «activos» en el disco.

El **fsck** (*file system check*) fue la herramienta que resolvía dichas inconsistencias, pero el fsck tiene que analizar la partición completa y verificar las interdependencias entre i-nodos, bloques de datos y contenidos de directorios. Con la ampliación de la capacidad de los discos, la recuperación de la consistencia del sistema de fichero se ha convertido en una tarea que requiere mucho tiempo, por lo que crea problemas serios de disponibilidad de los servidores afectados. Esta es la razón principal de que los sistemas de ficheros hayan importado de las bases de datos las técnicas de transacciones y recuperación, y así hayan aparecido los sistemas de ficheros con *journaling*.

Un sistema con *journaling* es un sistema de ficheros tolerante a fallos en el cual la integridad de los datos está asegurada porque las modificaciones de la meta-información de los ficheros son primero grabadas en un registro cronológico (*log* o *journal*) antes que los bloques originales sean modificados.

En el caso de un fallo del sistema, un sistema con *journaling* «integral» asegura que la consistencia del sistema de ficheros es recuperada. El método más común es el de grabar previamente cualquier modificación de la meta-información en un área especial del disco, el sistema realmente grabará los datos una vez que la actualización de los registros haya sido completada. A la hora de recuperar la consistencia luego de un fallo, el módulo de recuperación analizará el registro y sólo repetirá las operaciones incompletas en aquellos ficheros inconsistentes, es decir que la operación registrada no se haya llevado a cabo finalmente.

Los primeros sistemas de ficheros con *journaling* fueron creados a mediados de los ochenta e incluyen a Veritas (VxFS), Tolerant y JFS de IBM. La demanda de sistemas de ficheros que soporten terabytes de datos, miles de ficheros por directorios y compatibilidad con arquitecturas de 64 bits ha hecho que en los últimos años haya crecido el interés de la disponibilidad de sistemas con *journaling* en Linux.

Linux tiene ahora cuatro nuevos sistemas de ficheros: ReiserFS de Namesys (<http://www.namesys.com>), XFS de Silicon Graphics (SGI, <http//oss.sgi.com/projects/xfs/>), JFS de IBM (<http://oss.software.ibm.com/developerworks/opensource/jfs/>) y Ext3 desarrollado por Stephen Tweedie, co-desarrollador del Ext2 (<http://e2fsprogs.sourceforge.net/ext2.html>).

Mientras que ReiserFS es un sistema totalmente nuevo escrito desde cero, XFS, JFS y Ext3 son derivados de sistemas ya existentes. XFS está basado, y comparte parcialmente el mismo código, en el sistema desarrollado por Silicon Graphics para sus estaciones de trabajo y servidores.

JFS fue desarrollado por IBM para su sistema OS/2 Warp, que a su vez es derivado del sistema JFS de AIX. ReiserFS es el único incluido en el árbol de desarrollo estándar del núcleo, está programado que los otros sean incluidos en la versión 2.5, aunque ya están en fase totalmente estable y funcional.

Ext3 es una extensión a Ext2 y agrega dos módulos independientes: un módulo de transacciones y un módulo de registro. Ext3 está en un estado muy avanzado y su inclusión en el árbol estándar es prioritaria. La distribución RedHat 7.2 ya la incluye como opción y será el sistema de ficheros oficialmente soportado por dicha empresa.

3.1. *B-Trees*

La técnica básica para mejorar el rendimiento comparado a sistemas de ficheros tradicionales de Unix es evitar el uso de listas enlazadas o mapas de bits (para bloques libres, entradas de directorios y direccionamiento a bloques de datos) ya que sufren de problemas inherentes de escalabilidad. La complejidad típica para búsquedas es O(n), no muy apropiada para discos muy grandes. La mayoría de los nuevos sistemas usan *Árboles Balanceados* (*B-Tree*) o variaciones de ellos (*B+ Tree*). La estructura de los árboles balanceados están bien estudiadas, son más robustos en rendimiento pero al mismo tiempo los algoritmos de gestión y balanceo son más complejos. La estructura *B+ Tree* ha sido usada para indexación de base de datos desde hace tiempo ya que les provee de formas rápidas escalables para acceder a sus registros. El símbolo «+» en *B+ Tree* significa, normalmente, que es una versión modificada del original que:
- Coloca todas las claves en las hojas.
- Las hojas pueden estar enlazadas entre ellas.
- Los nodos pueden tener tamaños diferentes.
- No se necesita cambiar un padre si una clave en un hijo ha sido borrada.
- Hace que las búsquedas secuenciales sean más simples y rápidas.

3.1.1. ReiserFS

ReiserFS está basado en *B+ Trees* para organizar los objetos del sistema de ficheros. Dichos objetos son estructuras que mantienen información de ficheros: permisos, tiempos de creación, modificación y acceso, datos de ficheros, etc. En otras palabras, información tradicionalmente mantenida en los i-nodos, directorios y bloques de datos. ReiserFS llama a esos objetos: *1. stat data items*, *2. directory items* y *3. direct/ indirect items,* respectivamente.

ReiserFS provee *journaling* de sólo la meta-información de los ficheros. En caso de un reinicio no planificado, los bloques de datos de ficheros que se estaban usando en ese momento podrían estar corruptos.

Los *nodos no formateados* son bloques lógicos sin un formato determinado y son usados para mantener los datos de ficheros, los *direct items* consiste de los propios datos de ficheros. Dichos items son de tamaño variable y almacenados en las hojas del árbol, algunas veces junto a otros items si hay espacio suficiente en el nodo. La meta-información de los ficheros es almacenada próxima a los propios datos, ya que el sistema intenta ubicar a los *stats items* y los *direct/indirect items* del mismo fichero en ubicaciones contiguas.

Contrariamente a los *direct items*, los datos de ficheros apuntados por *indirect items* no se almacenan dentro del árbol. Esta gestión especial de los *direct items* se debe al soporte para ficheros pequeños: *empaquetado de colas* (o *tail packing*).

El empaquetado de colas es una características especial del ReiserFS, las colas son ficheros más pequeños que un bloque lógico, o los últimos bytes de los ficheros que no rellenan un bloque completo. ReiserFS empaqueta dichos ficheros o paquetes en nodos hojas únicos para ahorrar espacio en disco. Generalmente esto permite un ahorro del 5% comparado con el Ext2. Además ReiserFS tiene un rendimiento excelente para ficheros pequeños ya que puede incorporar dichas colas en el *B-Tree*, así se encuentran muy próximos a su meta-información (*stat data*).

El problema de esta técnica es que puede producir fragmentación externa. Además, el empaquetado de colas consume más CPU y puede afectar negativamente al rendimiento, especialmente en procesadores más lentos. Esto se debe a la necesidad de desplazamiento de memoria cuando se agregan datos a un fichero. Namesys conoce este problema y permite que el empaquetado sea desactivado especificando *notail* como opción de montaje para el montaje (o inclusive si se hace un *remount*).

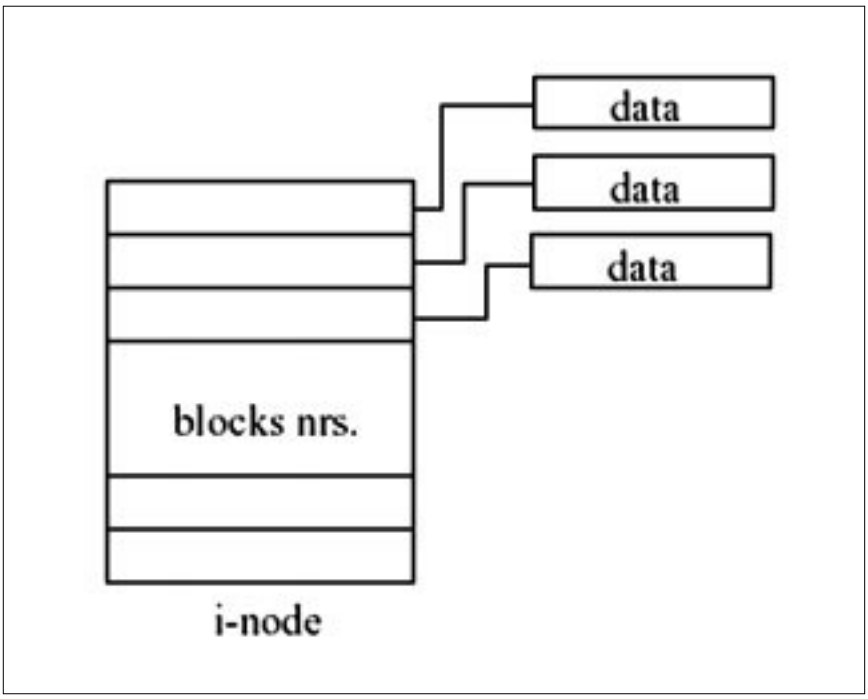


Figura 4. Asignación basada en bloques

Para la asignación de espacio, ReiserFS usa bloques de tamaño fijo (4KB, ver **figura 4**) que afecta negativamente al rendimiento en operaciones sobre ficheros grandes. El otro punto débil de ReiserFS es que el rendimiento sobre ficheros *esparcidos* (*sparse*, ficheros que no tienen todos los bloques de datos ocupados) es significativamente peor que Ext2, aunque Namesys está trabajando en este tema.

3.1.2. XFS

El 1 de mayo de 2001, SGI puso a disposición la versión 1.0 de sus sistema de *journaling* XFS para Linux. XFS es reconocido por su buen soporte a ficheros grandes con velocidades de transferencias muy altas (verificado hasta 7GB/ seg). XFS fue desarrollado para su variante de Unix, Irix, versión 5.3, cuya primera versión fue introducida al mercado en diciembre de 1994.

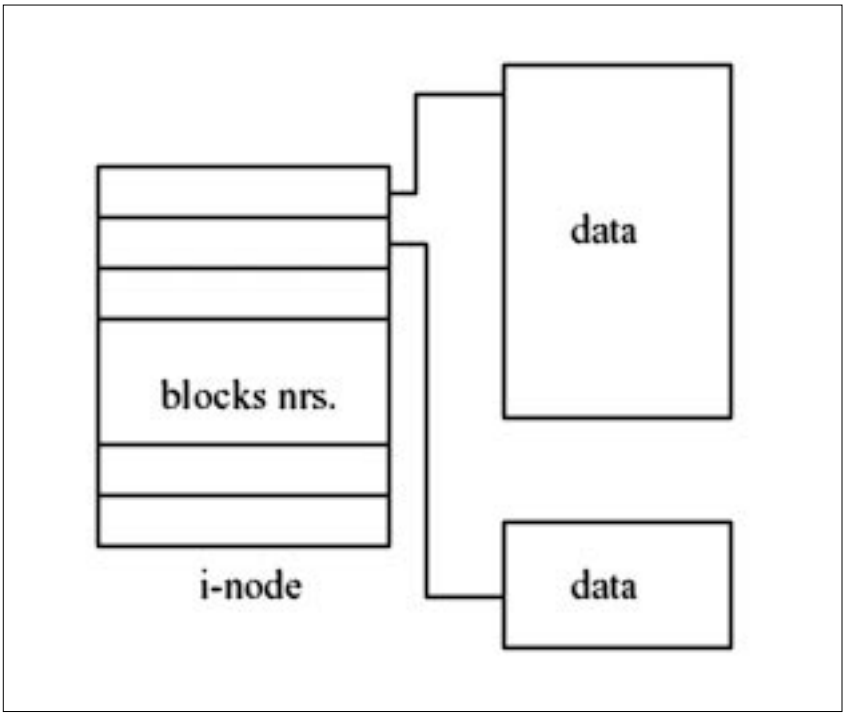


Figura 5. Asignación basada en extents

El objetivo del sistema fue soportar ficheros muy grandes para tratamiento de vídeo en tiempo real y generación de imágenes sintéticas para la industria de Hollywood.

XFS usa *B+ Trees* extensivamente, mantiene la información de *extents* libres, índices de directorios y la asignación dinámica de los i-nodos que se encuentran distribuidos por todo el sistema de ficheros. Además, XFS usa técnicas de registro asincrónicas para proteger la modificación de la meta-información de ficheros.

XFS usa una asignación de espacio basada en *extents* (ver **figura 5**), y tiene características como asignación retrasada (para acelerar las operaciones de creación y ampliación de ficheros), reagrupamiento de espacio luego del borrado de ficheros e intenta usar *extents* del máximo tamaño posible durante la asignación de espacio. Para hacer más eficiente la gestión de espacio contiguo, XFS usa grandes descriptores de *extents*.

Cada uno de ellos puede describir hasta dos millones de bloques del sistema, lo que ahorra mucho tiempo de CPU al no tener que buscar por bloques contiguos, puede simplemente leer la longitud del *extent* en vez que verificar para cada entrada si es contigua con la anterior.

XFS permite bloques de tamaño variable por cada sistema, de 512 bytes a 64 kilobytes. El cambio de tamaños de bloques varía la fragmentación, sistemas con ficheros pequeños suelen usar bloques más pequeños para evitar el desaprovechamiento de espacio por fragmentación interna. Sistemas con ficheros grandes suelen elegir la opción inversa para reducir la fragmentación externa.

El código fuente e XFS es muy complejo y muy orientado a Irix, lo que obligó a re-escribir la interfaz para poder portarlo a Linux. El resultado es el módulo *pagebuf*, que provee la interfaz entre XFS y el sistema de memoria virtual como así también entre XFS y la capa de dispositivos de bloques del núcleo.

El XFS soporta listas de acceso (*ACL*), integradas con el servidor Samba, y sistemas de cuotas transaccionales. En Linus soporta cuotas por grupo, mientras que en Irix es por proyecto. Otras características avanzadas, especialmente para aplicaciones de vídeo bajo demanda en tiempo real, todavía no han sido portadas a Linux.

Como mencionamos anteriormente, el modo normal de trabajo es la grabación asincrónica del registro, que aún así asegura que los datos de ficheros no pueden ser grabados al disco antes que haya hecho lo mismo con los datos del registro.

Con la grabación asincrónica del registro se gana en rendimiento por dos factores principales:
1. Múltiples modificaciones pueden ser agrupadas en un sólo operación de grabación del registro.
2. El rendimiento de las modificaciones de la meta-información (que deben ser registradas) se hace independiente de la velocidad del dispositivo. Esta independencia está limitada por la cantidad de memoria que se asigna para los *buffers* del registro.

3.1.3. JFS

IBM introdujo su sistema de ficheros JFS (*Journaled File System*) con la versión 3.1 del AIX. Dicho sistema luego fue modificado para su gama de sistemas corporativos OS/2 Warp Server, que es el que comparte código con la versión para Linux.

JFS está optimizado para altas velocidades de transferencias. Usa, como el XFS, asignación basada en *extents* (Figura 5), junto con políticas de asignación de bloques agrupados (4), para producir estructuras eficientes para hacer corresponder desplazamientos lógicos dentro de los ficheros a direcciones físicas del disco. Los extents en JFS son una secuencia de bloques contiguos asignados a un fichero como una unidad y descriptos por la tripla <desplazamiento lógico, longitud, física>. La estructura de direccionamiento es un árbol *B+ Tree* de descriptores de *extents*, cuya raíz es el i-nodo y la clave es el desplazamiento dentro del fichero. Los registros del JFS son mantenidos en cada sistema de ficheros y registran información de operaciones en la meta-información de ficheros. La semántica tradicional del registro es sincrónica, si una operación que involucra cambios en la mata-información retorna satisfactoriamente, los efectos de la operación ya han sido enviadas al sistema de ficheros y serán visibles aún si el sistema falla inmediatamente después de la operación.

En términos de rendimiento, la técnica sincrónica es una desventaja comparada con otros sistemas de ficheros que usan técnicas asincrónicas (tales como Veritas y XFS). JFS ha sido mejorado y ahora también provee operaciones de registro asincrónicas, el cual mejora sustancialmente el rendimiento.

JFS soporta tamaños de bloques de 512, 1024, 2048 y 4096 bytes, únicos por cada sistema de fichero. Los efectos de fragmentación son similares a los descriptos para XFS. El tamaño por defecto

es 4096 bytes. Al igual que XFS, JFS asigna espacio para los i-nodos dinámicamente y los libera cuando ya no son necesarios.

Para los directorios se usan dos organizaciones:
1. Para pequeños directorios, el contenido se almacena dentro del mismo i-nodo, esto ahorra la necesidad de otra operación de E/S y uso de espacio adicional.
2. Para directorios más grandes, cada directorio es un árbol *B+ Tree*, donde la clave es el nombre del fichero. Este sistema provee búsquedas, inserción y borrado más eficientes.

JFS soporta ficheros *esparcidos*, el tamaño de fichero reportado por el sistema es el byte más alto que ha sido escrito, sin embargo no se asignan bloques de discos hasta que haya operaciones de escritura sobre dichos bloques.

3.1.4. Ext3

Ext3 es compatible con Ext2, en realidad es Ext2 con un fichero adicional de registro. Ext3 es una capa adicional sobre Ext2 que mantiene un fichero de registro (por defecto en el directorio /jfs). Debido a que está integrado en el Ext2, sufre algunas de las limitaciones de dicho sistema, y no explota las posibilidades de los sistemas de *journaling* puros. Por ejemplo, todavía usa asignación basada en bloques (**figura 4**) y búsqueda secuencial de directorios, aunque se está trabajando es esta área para mejorarla. Sus mayores ventajas son:
- Ext3 mantiene la consistencia tanto en la meta-información como en los datos de los ficheros. A diferencia de los demás sistemas de *journaling* mencionados, la consistencia de los datos también está asegurada.
- Las particiones Ext3 no tienen una estructura de ficheros diferentes a los de Ext2, por lo que no sólo se puede pasar de Ext2 a Ext3, sino que lo opuesto también funciona, útil sobre todo si en algún caso el registro se corrompe accidentalmente, por ejemplo debido a sectores malos del disco.

Ext3 reserva uno de los i-nodos especiales de Ext2 para el registro, pero los datos del mismo pueden estar en cualquier conjunto de bloques, y en cualquier sistema de ficheros. Inclusive se puede compartir el registro entre sistemas distintos. Tres tipos de bloques de datos son grabados en el registro:
1. Meta-información: contiene el bloque de meta-información que está siendo actualizado por la transacción. Cada cambio en el sistema de ficheros, por pequeño que sea, es escrito en el registro. Sin embargo es relativamente barato ya que varias operaciones de E/S pueden ser agrupadas en conjuntos más grandes y pueden ser escritas directamente desde el sistema *page-cache* usando la estructura *buffer_head*.
2. Bloques descriptores: estos bloques describen a otros bloques del registro para que luego puedan ser copiados al sistema principal. Los cambios en estos bloques son siempre escritos antes que los de meta-información.
3. Bloques cabeceras: describen la cabecera y cola del registro más un número de secuencia para garantizar el orden de escritura durante la recuperación del sistema de ficheros.

4. Rendimiento y conclusiones

Las diferentes pruebas de rendimiento (ver Enlaces al final) demuestran que ReiserFS y XFS obtienen excelentes resultados comparados con el ya bien ajustado y optimizado Ext2. Ext3 es más lento pero aproximándose al rendimiento de Ext2. Suponemos que el rendimiento de Ext3 será mejorado considerablemente en los meses siguientes. Por otro lado, JFS obtiene los peores resultados globales, además de problemas de inestabilidad en algunas versiones anteriores de Linux.

XFS, ReiserFS y Ext3 han demostrado que son excelentes y muy confiables. Hay un área donde XFS es el claro vencedor, especialmente si se compara con ReiserFS: operaciones sobre ficheros grandes. Sin embargo esto es posible que cambie con el tiempo, ReiserFS usa las operaciones de lectura y escritura genéricas del núcleo, mientras que XFS ha heredado las operaciones sofisticadas de Irix, como el uso de *extents*, asignación retrasada y operaciones de E/S directas. Además, el código de ReiserFS usado para las pruebas hace un recorrido completo del árbol por cada 4KB que escribe, y luego inserta un puntero, lo que genera una sobrecarga importante del sistema al tener que balancear el árbol mientras copia datos. ReiserFS ha mostrado sus mejores resultados para ficheros entre 100 y 10.000 bytes, especialmente si los ficheros aún no están en el *page-cache* (como ocurre durante el arranque del ordenador). En caso que se lean ficheros que ya están en la caché, la diferencias son prácticamente despreciables para todos los sistemas de ficheros mencionados.

Entre todos los sistemas de ficheros con *journaling*, ReiserFS es el único incluido en el núcleo estándar desde la versión 2.4.1 (y SuSE lo soporta desde hace unos dos años). Sin embargo, Ext3 será el estándar de una distribución tan conocida como Red Hat, y XFS está siendo usado en grandes servidores, especialmente en la industria del cine y los efectos especiales. Tanto Ext3 y XFS son prioritarios para su inclusión en el núcleo. La inclusión de XFS es un poco más complicada debido a los grandes cambios que introduce en el *page-cache*. Por otro lado, los programadores de Linux están siguiendo con mucho interés el esfuerzo que dedica IBM para solucionar los problemas de estabilidad del JFS, aunque ya se dice que es una alternativa válida para migrar de OS/2 y AIX a Linux.

5. Enlaces

Ext3 architecture: <ftp://ftp.kernel.org/pub/linux/kernel/people/sct/ext3/>
Introduction to Linux Journal File Systems: <http://www. linuxgazette.com issue55/florido.html>
XFS Home Page: <http://oss.sgi.com/projects/xfs/>
JFS Home Page: <http://oss.software.ibm.com/develo perworks/opensource jfs/>
ReiserFS Home Page: <http://www.namesys.com/>
Storage Foundry: <http://sourceforge.net/foundry/storage/>
OS News http: <//www.osnews.com/story.php?news_ id=69>

6. Pruebas de rendimiento (*benchmarks*)

Pruebas con Ext2, Ext3, ReiserFS, XFS y JFS: <http://www.mandrakeforum.org/article.php?sid=1212>
Ext2, ReiserFS and XFS Benchmarks: <http://bulmalug.net/body.phtml?nIdNoticia=642>
Pruebas con XFS, ReiserFS, Ext2FS, y FAT32: <http://bulmalug.net/body.phtml?nIdNoticia=626>
Pruebas Namesys: <http://www.namesys.com/benchmarks/benchmark-results.html>
Pruebas con Ext2, XFS, ReiserFS y JFS Mongo: <http://bulmalug.net/body.phtml?nIdNoticia=648>

Ingeniería del Software

Un nuevo modelo de evaluación de procesos de software para PYMEs a partir de SPICE (ISO/IEC TR-15504-5)

Antonia Mas Pichaco, Ángel Igelmo Ganzo, Esperança Amengual Alcover, Gabriel Fontanet Nadal
Universitat de les Illes Balears

<antonia.mas@uib.es>

Este artículo fue seleccionado para su publicación en Novática de entre las ponencias presentadas en las VI Jornadas sobre Innovación y Calidad del Software, promovidas por ATI y celebradas en julio de 2001 en la Universidad Europea CEES (Madrid). Su selección se hizo en base a la puntuación otorgada tanto por los revisores técnicos como por el público asistente.

Resumen: *en este trabajo se presenta un modelo de evaluación de procesos de software, orientado a pequeñas y medianas empresas de desarrollo de software y basado en el propuesto en* ***SPICE ISO/IEC TR 15504-5****, que permita deducir la capacidad de los procesos del ciclo de vida del software, a partir de los resultados reflejados en los cuestionarios que conforman una guía evaluadora, que deberá ser cumplimentada por personal especializado de la empresa y analizada por un evaluador cualificado.*

Palabras claves: *calidad del software, procesos del ciclo de vida del software, modelos de evaluación y mejora de procesos de software, SPICE.*

1. Introducción

La Ingeniería del Software es una disciplina relativamente reciente y en muchos casos, aunque se conozcan los fundamentos predicados en la materia, no se ponen en práctica en las empresas. Quizás, podría fomentarse su aplicación, proporcionando al gestor de la organización y a los gestores de los proyectos unos mecanismos que les permitan establecer un conjunto de indicadores que les den a conocer el estado de sus procesos de software, a la vez que unos modelos, métodos, técnicas y herramientas que les posibiliten la mejora de los mismos.

Otro agravante que impide su aplicación efectiva en las pequeñas y medianas empresas y organizaciones, es que los avances generados en la disciplina son a menudo aplicables a grandes empresas de desarrollo de software, con múltiples departamentos y con altos presupuestos en I+D, pero poco extrapolables a entornos con menos recursos, tanto humanos como materiales.

La intención de tomar el pulso a esta actividad plantea la necesidad de un modelo y de unas herramientas que permitan, por una parte abordar la evaluación de la calidad de los procesos de estas empresas en base a unos criterios prefijados y que ofrezca unos resultados comparables [1], y por otra, que posibilite la extrapolación de las conclusiones y de los resultados obtenidos tanto a nivel nacional como internacional. Inicialmente, se barajan simultáneamente [6] el modelo americano **CMM** (*Capability Madurity Model*) y el modelo **SPICE (ISO/IEC TR-15504**), tomando éste último como punto de partida de este trabajo. Se descarta ya desde el inicio la norma **ISO 9000**, porque en el momento de iniciar este estudio, se encontraba a punto de aparecer la nueva versión 2000 de la norma y no se consideró oportuno aplicar la todavía vigente.

Ante la complejidad del Modelo para su aplicación intrínseca en las pequeñas empresas y organizaciones [5], se decidió abordar la realización de una guía evaluadora. En los siguientes apartados se describe por una parte, el método seguido para desarrollar el conjunto de cuestionarios que componen dicha guía evaluadora, indicando las variaciones al modelo de evaluación propuesto en la parte 5 de SPICE, y por otra, el método para el cálculo de la capacidad de los procesos a partir de las respuestas obtenidas en los cuestionarios.

2. SPICE (ISO/IEC TR 15504)

SPICE (ISO/IEC TR 15504), *Software Process Improvement and Capability dEtermination*, es un estándar emergente a nivel internacional que se encuentra en la fase de Informe Técnico (**TR**, *Technical Report*) y que es aplicable a cualquier organización/empresa que quiera mejorar la capacidad de sus procesos de software, independientemente del tipo de organización, del modelo del ciclo de vida adoptado, de la metodología de desarrollo y de la tecnología utilizada [3], [4].

El modelo de Referencia de SPICE (ISO/IEC TR 15504-2), propone un modelo bidimensional para valorar la capacidad de los procesos del software. Por un lado, en la **dimensión de procesos**, que se agrupa en cinco categorías, se describen todos los procesos que una organización puede realizar para comprar, suministrar, desarrollar, operar, mantener y soportar el software. Por otro lado, la **dimensión de capacidad**, formada por seis niveles de capacidad y nueve atributos de procesos, proporciona una base para medir la capacidad de dichos procesos, en función del grado de cumplimiento de sus atributos.

En la **figura 1** se muestran ambas dimensiones.

3. Desarrollo del Modelo de Cuestionarios

3.1. Introducción. Necesidad de un modelo de cuestionarios

Para conocer el estado de madurez de los procesos es necesario averiguar las prácticas que se siguen en las empresas para llevar a cabo estos procesos [2]. Resulta impensable recopilar toda la información necesaria para realizar una evaluación y una propuesta de mejora directamente a partir del modelo de evaluación de SPICE [5], sin haber preparado antes unos cuestionarios, que deberán ser cumplimentados por personal de la empresa y a ser posible sin la ayuda de un asesor o evaluador especializado.

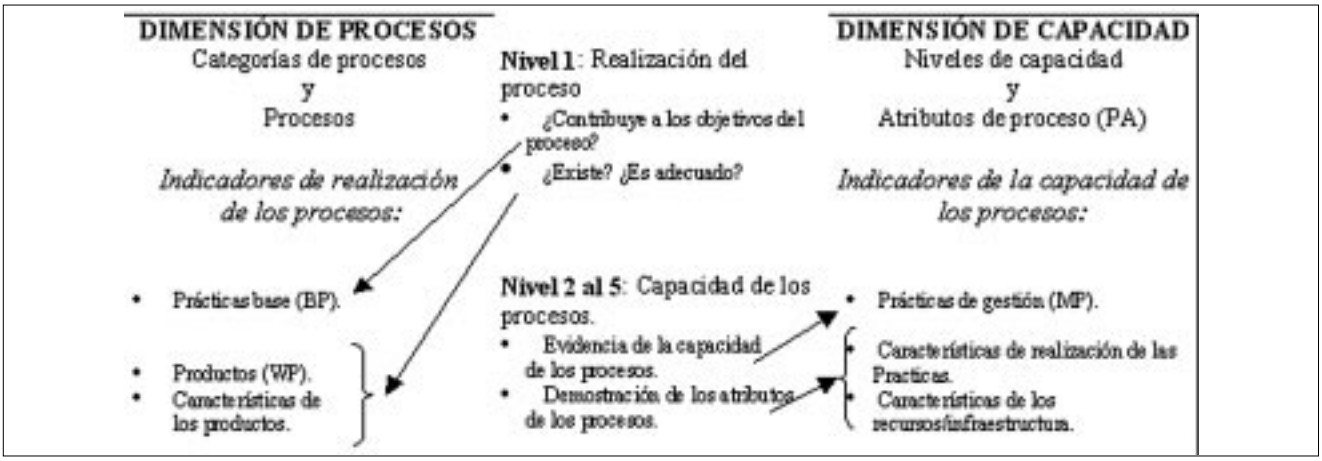


Figura 1. Las dos dimensiones del Modelo de Referencia de SPICE

Inicialmente, las expectativas eran que la guía evaluadora permitiera determinar si una categoría de procesos se encuentra en el nivel de capacidad 1 (proceso realizado). En el momento de decidir la manera de realizar las evaluaciones de los procesos con niveles de capacidad superiores a 1, se observó que esta cuestión podría quedar en gran parte resuelta ampliando el cuestionario y desarrollándolo de forma más detallada y precisa.

La guía evaluadora desarrollada en este trabajo se fundamenta en la parte normativa de SPICE ISO/IEC TR 15504-2: «*A reference model for processes and process capability*», y principalmente en la parte informativa SPICE ISO/IEC TR 15504-5: «*An assessment model and indicator guidance*» [4], en la cual se ofrecen algunas pautas para realizar una guía compatible con el modelo de referencia que permita evaluar los procesos software de una empresa.

3.2. Elaboración de los cuestionarios de evaluación. Procedimiento aplicado

En la parte normativa ISO/IEC TR 15504-2 se detallan cinco categorías de procesos, totalmente alineadas con las definidas en el estándar ISO/IEC 12207, *Information Technology - Software life cycle processes*, que son: de Cliente-Proveedor **(CUS),** de Ingeniería **(ENG)**, de Soporte **(SUP)**, de Gestión **(MAN)**, y de Organización **(ORG),** así como los procesos que componen cada una de estas categorías, especificándose los propósitos de todos estos procesos así como los resultados que se deben obtener mediante una correcta implantación de los mismos.

Pero es en el modelo de evaluación, parte 5, en el que se detallan los indicadores que permitirán deducir si se han cumplido los objetivos del proceso, y que son: unos productos (**WP**, *Work Products)* de entrada y otros de salida, junto con un conjunto de características asociadas, y un conjunto de prácticas base para evaluar cada uno de los procesos.

· El primer paso para realizar el cuestionario consistió en determinar qué productos de entrada y de salida de los sugeridos en el Anexo A de la parte informativa de SPICE formarían parte de cada uno de los procesos de cada una de las categorías.

Por ejemplo, los productos de entrada y de salida propuestos para el proceso **ENG.1.2**, son los que se muestran a continuación en la **tabla 1**.

Las modificaciones que se realizaron sobre el modelo son las siguientes:

- Se añadió la siguiente entrada: **Metodología de desarrollo del software** (1)**.** Se consideró que antes de empezar el proceso de análisis de requerimientos de software, sería conveniente considerar la metodología que se va a utilizar como entrada del proceso.

Table A.10 – ENG.1.2 - Software requeriments analyis process- Associated Work Products

Input	***Output***
44)Product need assessment	***8) Interface***
52)Requirement specification (customer)	*21) Analysis result*
52)Requirement specification (maintenance)	*31) Review record*
53)System design / architecture	*52) Requirement specification (software)*
58)Traceability record / mapping	*58) Traceability record / mapping*
83)Customer request	*69) Release strategy /plan*
87)Communication mechanism	*87) Communication mechanism*
84)Problem report	*93) Configuration item*
94)Change request	*100) Product configuration*
101)Database design	*105) Customer documentation*

Tabla 1. Productos asociados al proceso ENG.1.2

- Se eliminaron los siguientes productos de las entradas del proceso: Especificación de requerimientos de mantenimiento (52). En el proceso **ENG.2**, Mantenimiento del software y del sistema, ya se considera dicho producto como entrada. Peticiones del cliente (83). Se reflejarán en la especificación de requerimientos del cliente (52). Peticiones de cambio (94). Se tratarían en el proceso ENG.2 antes citado.
- Se añadió el siguiente producto como salida del proceso ENG.1.2: Modelo conceptual de datos (111). Se consideró oportuno añadir un nuevo producto, no contemplado en el Anexo C. Este nuevo producto ha sido denominado «Modelo conceptual de datos» y el código identificativo asignado ha sido el número secuencial inmediato.
- Se eliminó el producto Interfaces (8) como salida del proceso ENG.1.2, por considerar más apropiado tratar este aspecto en el proceso de diseño, **ENG.1.3**.

· Cada uno de los productos tiene asociadas un conjunto de características que aparecen detalladas en el Anexo C de ISO/IEC TR 15504-5. El segundo paso para la elaboración del cuestionario consistió en examinar en detalle dichas características, formular cuestiones [2] a partir de las que fueron consideradas y adaptar cada una de estas preguntas a los diferentes procesos en que interviene un mismo producto, además de tener en cuenta si dicho producto forma parte de la entrada del proceso, de su salida o de ambas partes.

El siguiente ejemplo ilustra como se ha realizado el cuestionario para el producto de salida «Casos de prueba» (61), en el proceso de construcción del software ENG.1.4. Las características que aparecen en el Anexo C para los casos de pruebas, se muestran en la **tabla 2**. A partir de las características anteriores y teniendo en cuenta que el proceso de construcción del software **ENG.1.4.** tiene como objetivo producir unidades de software ejecutables y verificar que éstas reflejan convenientemente el diseño del software, se genera el cuestionario del producto 61, Casos de pruebas, para la salida del Proceso de construcción del software ENG.1.4.que se muestra a continuación en la **tabla 3**.

· En tercer lugar, se desarrolló una importante labor de revisión del cuestionario, tanto para evitar la repetición de las mismas preguntas en diferentes apartados o procesos de la guía evaluadora, como para lograr una coherencia interna de la misma.

4. Desarrollo de utilidades para el cálculo de la capacidad de los procesos

En este apartado se plantea el método seguido para la valoración de la capacidad de los procesos, una vez que las empresas han revisado y cumplimentado la guía evaluadora y se han realizado las entrevistas necesarias con el personal apropiado de la empresa.

ID	*WP type*	*WP Characteristics*
61	*Test case*	*Provides executable set of test instructions.* *Purpose defined.* *Defines the expected test result.* *Mapped to test scripts, requirements.*

Tabla 2. Características del producto «Casos de pruebas»

La dimensión de capacidad del modelo de referencia SPICE define una escala de valoración para la capacidad de los procesos. Esta escala consta de cinco niveles, cada uno de ellos caracterizado por unos atributos de proceso. Dependiendo de los valores de los atributos que alcance un proceso, éste se encontrará en una u otra posición de la escala o nivel de capacidad. Cada uno de estos niveles se identifica pues con un porcentaje de posesión del atributo.

Cada atributo valora un aspecto particular de la capacidad del proceso y viene identificado por las siglas **PA**, *Process Attributes*, seguidas de dos números: el primero indica el nivel de capacidad al que caracteriza este atributo y el segundo indica el número de atributo dentro de un mismo nivel.

La escala para la valoración de los atributos se compone de cuatro valores: N: no alcanzado; P: parcialmente alcanzado; L: en gran parte alcanzado; y F: plenamente alcanzado.

SPICE obliga a evaluar empezando desde el nivel 1 y, en caso de que los atributos de los procesos asociados a cada nivel sean alcanzados en gran parte (L) o plenamente (F), permite evaluar un nivel más alto.

Los atributos de proceso vienen definidos en la parte normativa del modelo, pero las prácticas de gestión **MP**, *Management Practices*, así como las características que permitirán deducir su realización, son únicamente de carácter informativo.

Las prácticas de gestión se etiquetan con un identificador formado por las siglas MP y un número de tres dígitos. Los dos primeros hacen referencia al atributo de proceso al que caracterizan y el tercero los distingue dentro de un mismo atributo. Así, la relación entre niveles de capacidad, atributos de proceso y prácticas de gestión, es la siguiente para el nivel 1 de capacidad:

Nivel 1: Proceso realizado.
PA1.1 Atributo de realización del proceso.
- **MP 1.1.1**. Identificar los productos de entrada y de salida.
- **MP 1.1.2**. Asegurar que el ámbito de trabajo está identificado para cada proceso y que los productos de entrada y de salida se utilizan/generan en cada proceso.
- **MP 1.1.3.** Asegurar que se implementan las prácticas base, obteniendo los productos que garantizan el alcance de los resultados.

4.1. Evaluación del nivel 1

El procedimiento seguido para valorar si un proceso alcanza el nivel 1 es el siguiente:
· Para cada una de las características de los productos se han formulado numerosas y diversas preguntas. Según las respuestas de estas cuestiones, el evaluador deducirá la existencia o no de cada uno de los productos de entrada y de salida establecidos para el proceso en cuestión y las prácticas base que se realizan, que son los dos elementos que definen la realización de un proceso, nivel 1. Así por ejemplo, a partir de las respuestas obtenidas en el cuestionario de la ilustración 4, para el producto «Casos de prueba», el evaluador podría deducir si en la empresa se contempla o no esta salida para el

Cuestionario del producto (61), que forma parte de las salidas del Proceso de Construcción del software ENG.1.4				
Casos de prueba (61)	SI	NO	NS/NC	Otros
¿Se han diseñado casos para las pruebas unitarias?				
¿Se han definido los objetivos de los casos de pruebas?				
¿Se han identificado los componentes del proyecto que deberán ser probados?				
¿Se ha indicado la configuración sobre la que se deberán realizar las pruebas?				
¿Se han tenido en cuenta las técnicas existentes para la generación de casos de prueba?				
¿Los casos de prueba generados permitirán detectar errores del tipo: inicialización o valores implícitos erróneos nombres de variables incorrectas tipos de datos inconsistentes excepciones de desbordamiento o de direccionamiento cálculos incorrectos comparaciones incorrectas flujos de control inapropiados comparaciones entre tipos de datos distintos operadores lógicos o de precedencia incorrectos terminación de bucles inapropiadas o inexistentes otros tipos de errores				
¿Se han probado todos los caminos de tratamiento de errores, recomendados por las técnicas de generación de casos de prueba?				
¿Se ha acompañado cada caso de prueba con los resultados esperados?				
¿Se dispone o se ha desarrollado, en caso de que haya sido necesario, un software (resguardo o programa similar) que acepte los datos de los casos de prueba, que pase estos datos al módulo a probar y que imprima los resultados obtenidos?				
¿Se han registrado los informes de problemas generados a partir de los resultados de las pruebas?				

Tabla 3. Cuestionario del Producto (61)

Proceso de Construcción del software ENG.1.4.

· A continuación, se deberá realizar el recuento de respuestas afirmativas y de negativas, se procederá a asignar el porcentaje correspondiente sobre el total de los productos (de entrada y de salida a la vez) de cada proceso y, finalmente, este porcentaje permitirá deducir el valor del atributo para cada uno de los procesos del Modelo de Referencia, habiendo evaluado por tanto el Atributo de Realización del proceso (PA.1.1.) que es el que caracteriza el nivel de capacidad 1. Para poder decir que una categoría está a nivel 1, el Atributo de Realización del Proceso PA.1.1, tiene que estar plenamente (F) o en gran parte (L) alcanzado.

Así por ejemplo, si se hubieran obtenido 11 (61%) respuestas afirmativas y 7 (39%) respuestas negativas para el proceso ENG.1.4, le correspondería un valor L (entre un 51% y un 85%) al Atributo de Realización del proceso P.A.1.1, considerando pues que se encuentra como mínimo en el nivel de capacidad 1.

4.2. Evaluación de los niveles del 2 al 5

La asignación de un determinado nivel de capacidad se efectúa utilizando también las respuestas obtenidas en los cuestionarios, a partir de las cuales el evaluador deduce el valor de cada uno de los atributos de proceso correspondientes al nivel de capacidad que se desea evaluar. Las respuestas no deducibles directamente a partir de la guía evaluadora se deberán determinar mediante entrevistas complementarias con la persona o grupo de personas indicadas por la empresa.

- En primer lugar se calculan los porcentajes de realización de cada una de las prácticas de gestión asociadas a cada uno de los atributos de proceso del nivel de capacidad en estudio. Estos valores resultan de ponderar los porcentajes de realización de los procesos asociados a cada práctica de gestión, descritos en el Anexo B de la parte informativa.
- En segundo lugar, se ponderan los porcentajes resultantes de cada práctica de gestión y se obtiene el porcentaje del

atributo de proceso. A dicho valor es al que se le asigna la letra correspondiente de la escala definida por SPICE.

En esta primera versión de la guía evaluadora se ha dado el mismo peso a cada una de las prácticas de gestión de cada atributo de proceso, pero se está actualmente trabajando en este sentido para refinar el cálculo de la capacidad de los procesos.

5. Conclusiones y trabajo futuro

Se ha elaborado una primera versión de una guía evaluadora, mediante una adaptación de la guía propuesta en la parte 5 de SPICE, a la realidad de las pequeñas y medianas empresas, que permita deducir de una manera relativamente sencilla la capacidad de sus procesos.

La elaboración de las preguntas del cuestionario que forman una parte de esta guía parecía en principio una tarea menos laboriosa de lo que realmente ha sido, ya que se han obtenido finalmente más de 2.000 preguntas diferentes, y a que cada una de las características de los productos se ha tenido que adaptar a todos y cada uno de los procesos del modelo, sin olvidar además, que éstas podían corresponder tanto a un producto de entrada como a uno de salida.

A partir de la aplicación de la guía evaluadora a una pequeña o mediana empresa, se ha observado que sería conveniente realizar diferentes mejoras al cuestionario utilizado como base de la evaluación. Una de ellas consistiría en ajustar aún más los productos de entrada y los de salida de cada proceso, propuestos en la parte 5 de SPICE, (ISO/IEC TR 15504-5), a las características particulares de las pequeñas y medianas empresas.

Por otra parte, aunque el cuestionario actual cubre totalmente la determinación del nivel 1 de capacidad para los todos los procesos del ciclo de vida del software, para el resto de niveles lo hace parcialmente, ya que en general solamente puede deducirse si se realizan o no los procesos asociados a cada práctica de gestión (las prácticas de gestión definen los atributos que a su vez caracterizan los niveles de capacidad del 2 al 5).

Esto obliga a que la empresa evaluada deba responder al cuestionario completo, que abarca las cinco categorías de procesos, aunque solamente tenga interés en alguna de ellas. Además, como no es posible deducir del cuestionario toda la información necesaria para saber si las prácticas de gestión se realizan sobre un proceso concreto, se incrementa el número de entrevistas que el evaluador necesita realizar con el personal de la empresa.

Para evitar todo este proceso, podría desarrollarse un segundo cuestionario, orientado a deducir las prácticas de gestión que se realizan sobre cada proceso y que evitara tener que considerar los procesos asociados a las prácticas de gestión. Con ello se conseguiría reducir el esfuerzo que se debe realizar para reunir la información necesaria, tanto por parte del profesional que realiza la evaluación como del equipo de trabajo de la empresa evaluada.

Se está trabajando también en el desarrollo de un primer prototipo de herramienta de apoyo a la evaluación [3] que permita automatizar los resultados de las encuestas, que realice los cálculos necesarios para determinar la capacidad de los procesos y que ofrezca las propuestas de mejora de los procesos.

6. Bibliografía

[1] El Emam, K. et all, SPICE. The theory and practice of Software Process Improvement and Capability Determination, *IEEE Computer Society*, 1998.

[2] Humphrey, W., *Introduction to the Personal Software Process*, Addison – Wesley, 1997.

[3] ***Official Web Server for the SPICE Project*****,** <http://www.sqi.gu.edu.au/spice>

[4] ***SPICE Technical Report ISO/IEC 15504*****.** First Edition, 1999-E.

[5] ***Publicaciones SPIRE*** (*Software Process Improvement Case Study*) <http://www.cse.dcu.ie/spire>

[6] Zahran, S., *Software Process Improvement. Practical guidelines for Business Success*, Addison-Wesley, 1998.

Profesión informática

El futuro de la Ingeniería del Software

Karol Frühauf
Consultor, INFOGEM AG (Baden, Suiza)

<karol_fruehauf@acm.org>

Traducción: Luis Fernández Sanz, Antonio de Amescua Seco (Grupo de Lengua e Informática de ATI)

Este artículo estaba destinado a la monografía «Presente y futuro de la profesión informática» (***Novática*** nº 152, jul.-ago. 2001), pero no pudo ser publicado en nuestra revista por razones de espacio, aunque si lo fue en ***Upgrade***, en inglés, e ***Informatik/Informatique***, en alemán.

Resumen: *en este artículo se tratan muy brevemente tres asuntos: la naturaleza del software, con la hipótesis de que en un futuro próximo el software se utilizará en vez del dinero; la gente que crea software, y el valor de las certificaciones; y, finalmente, los factores esenciales en la creación de productos que contienen software.*

Palabras clave: *software, metodología, herramientas, certificación, entorno social*

La lectura de tres publicaciones ha estimulado mis ideas para este breve artículo. Los tres aspectos tratados en las mismas son la naturaleza del software, la gente que crea software y los factores esenciales en la creación de productos que contienen software

1. La naturaleza desconocida del software

La revista *Communications of the ACM* dedica su número de marzo de 2001, con gran modestia, a «los próximos mil años». La primera contribución, cuyo autor es Philip G. Armour, defiende que:
1. El software es un medio de almacenamiento de conocimiento, pero no un medio pasivo como un libro: el software pone en marcha el conocimiento.
2. El software es la futura moneda del mundo.

Es un hecho conocido el que vivimos en una época de trabajo intelectual. Pero la teoría de que el software puede usarse como moneda es, a la vez, asombrosa y seductora. El dinero simplifica el intercambio de bienes: yo no puedo construir una valla a base de manzanas, pero puedo cambiar mis manzanas por dinero y comprar tablillas, clavos, etc. para construir la valla. De acuerdo con Armour, el software se utilizará en vez del dinero en un futuro próximo. Todavía hace falta resolver algunos problemas, como el de los valores de cambio (estandarización de las interfaces entre productos de software o paquetes de software), el problema de la inflación (gran disponibilidad de ofertas para la misma funcionalidad, la ampliación del programa con código innecesario) o la protección de grandes valores nominales (por ejemplo, los programas que controlan las máquinas fresadoras para el torneado de los núcleos de bombas atómicas), por nombrar algunos.

Una vez que estos problemas se hayan resuelto, 100 kg. de manzanas permitirán comprar mil «MS-Word 2095», y un clavo de 10 mm. costará una licencia de usuario.

Una moneda debe cumplir ciertos requisitos. Esperamos que una moneda que ofrezca seguridad ante falsificaciones garantice que no pueda ser modificado por nadie una vez puesta en circulación. Debemos estar seguros también de que las modificaciones no sean necesarias; es decir, que el conocimiento encapsulado se aplicará correctamente siempre que se use el software. El software no necesitará estar libre de errores pero no debe hacer nada incorrecto. Así, todo software tendrá que tener una fiabilidad comparable, como sucede hoy con el software en las aplicaciones sensibles. Un software como ése no puede ser producido por aventureros, renegados o entusiastas carentes de conocimientos de Ingeniería del Software.

Una moneda debe ser fuerte ante la inflación. Un amigo mío, un escultor con títulos académicos que se integró en una compañía de Internet, me informó orgulloso de que diseñaba software que resolvía el problema del tratamiento de múltiples lenguas. Mi reacción fue decirle que no era el primero que resolvía el problema, lo que empañó de alguna manera su placer pero afortunadamente no frenó su autoconfianza. El software que resuelve problemas ya resueltos --que, por lo tanto, no encapsula el nuevo conocimiento--, acelera la inflación cuando se utiliza como moneda. Esto es algo que hoy todavía es tolerado por los informáticos e incluso lo piden los directivos.

Si el software está llamado a ser una moneda útil en el futuro, es necesario que todos conozcamos cómo se aplica coherentemente la Ingeniería del Software. También es necesario que continuemos desarrollando procedimientos, metodologías y herramientas para que nuestra moneda no sea tan cara. La facilidad de integración, la corrección, la portabilidad, la facilidad de prueba están en la vanguardia en este aspecto; que la interfaz sea «simpática» para el usuario, la eficiencia, la flexibilidad, la facilidad de mantenimiento, son aspectos que deben ser garantizados. En otras palabras, la calidad del software es crucial.

No es sorprendente que las «metodologías ligeras» que aparecieron en los últimos años no tegan ningún compromiso en relación con la calidad. Hay tres áreas en las que se requiere actuar:
1. La investigación en el área de Ingeniería del Software debe proporcionar mejores bases para los métodos que permitan desarrollo sin coste de software más fiable.
2. Los vendedores de software deben centrarse en la calidad del producto y comportarse en consecuencia.
3. Los desarrolladores de software deben conocer su negocio para desarrollar el software que cumpla los requisitos para ser moneda.

2. La gente que crea software

La Computer Society del **IEEE** ha desarrollado el programa de **CSEP** (*Certified Software Engineering Profesional* – Profesional Certificado en Ingeniería del Software), «una credencial que reconocerá el amplio conjunto de habilidades nece-

sarias para un ingeniero profesional de software» [IEEE, Computer, marzo 2001, pp. 82-83]. Esto cubrirá el amplio abanico de áreas temáticas que son importantes para los ingenieros de software más que centrarse en un estrecho rango de habilidades específicamente relacionadas con herramientas que es lo que reconocen los programas de certificación patrocinados por fabricantes de software. El CESP no es una licencia para ejercer, sino una certificación del reconocimiento entre iguales diseñado para medir la maestría individual en los conocimientos fundamentales requeridos para desarrollar las funciones de un ingeniero de software. Existen proyectos muy parecidos dentro de **CEPIS** (en concreto, **EPICS**, *European Profesional Informatics Competence Service*) y en Suiza (**I-CH**).

Las certificaciones son particularmente populares en los países angloparlantes. Existen ya desde hace tiempo certificaciones como la de jefe de proyecto de software y la de experto en pruebas de software. No se parecen a los certificados específicos de los fabricantes de software, que pueden ser adquiridas a un alto precio y que son licencias que autorizan a actuar como experto en ese conjunto de software propietario. Las licencias en el verdadero sentido de la palabra son aquéllas que son legalmente necesarias para ejercer ciertas profesiones como medicina, abogacía, etc. También en el siglo XXI, las asociaciones profesionales son las instituciones que definen el «conocimiento mínimo necesario» (cf. **SWEBOK**, *Software Engineering Body of Knowledge* — Cuerpo de Conocimientos de la Ingeniería del Software). Son las instituciones educativas las responsables de dirigir hacia el dominio de este conjunto de conocimientos. Con el apoyo de la iniciativa Softnet del Gobierno Federal de Suiza, las escuelas superiores de tecnología (*Fachhochschulen*) intentan ajustar conjuntamente sus curricula al SWEBOK.

3. El encanto del software

Norman (1999) clama en su libro por la ruptura con el ordenador universal, como el PC actual; éste puede hacer muchas cosas, pero no puede gestionar fácilmente nada. Debe ser reemplazado por «aparatos de información» dedicados, dispositivos que idealmente dan soporte a ciertas actividades humanas. «La gente debería aprender la tarea, no la tecnología. Deberían ser capaces de adaptar la herramienta a la tarea, no como en la actualidad, donde debemos adaptar la tarea a la herramientas».

Alguien que haya comprado un PC para usar el correo electrónico (es decir, escribir y mandar mensajes, y recibir y leer mensajes), y que haya hecho adecuadamente todos los ajustes de software, ha tenido que pasar por diferentes pasos que, excepto el primero, no están directamente asociados con el correo:
a. encender el ordenador
b. esperar a que el sistema operativo arranque
c. realizar un ejercicio de habilidad con tres dedos llamado «atajo» (*shorcut*)
d. establecer la conexión del MODEM
e. iniciar el programa de correo

Cuando mi coche arranca, los controles me informan inmediatamente de la presión de aceite, el combustible disponible en el depósito, la hora y muchos otros datos necesarios para conducir. No hay espera y no hay arranque de programas.

Para el diseño del aparato que podría lograr nuestro «futuro sin ordenadores», Norman propone los siguientes principios:
a) Observar a los usuarios, descubrir qué hacen.
b) Analizar el mercado, descubrir todo lo que podamos sobre él.
c) Formular y validar las necesidades de los usuarios y del mercado.
d) Construir, conjuntamente con el usuario, maquetas y prototipos del futuro producto.
e) Escribir las instrucciones de funcionamiento para el futuro producto.
f) Diseñar el producto sobre la base de las instrucciones de funcionamiento, de las maquetas y de los prototipos.
g) Comprobar el diseño y modificarlo continuamente.

Estos principios presumen ciertas cualificaciones de los trabajadores implicados:
1. Observador de comportamientos
2. Diseñador de comportamiento
3. Constructor de modelos
4. Realizador de pruebas de usuario
5. Diseñador gráfico e industrial
6. Autores/escritores técnicos

Con al excepción del diseñador industrial, sé que no existe ningún centro de formación en Suiza donde se puedan adquirir estas habilidades.

Para el diseño de los productos del futuro, los ingenieros de software deben ser capaces de cubrir el hueco entre el ordenador y los dominios de aplicación y tendrán que hacer que el ordenador desaparezca. El ingeniero deberá estar preparado para dicha tarea. En caso de que las instituciones educativas no proporcionen estas habilidades, las asociaciones profesionales tendrán que hacerlo.

4. El ingeniero de software en el entorno social

Hay un aspecto adicional que deseo tratar: la Ingeniería del Software es, ante todo, comunicación. Comunicación entre personas, directivos, diseñadores y vendedores. Esta comunicación comprende diferentes terminologías y, finalmente, el lenguaje natural debe unir todas ellas.

Beck (2000) promueve la comunicación verbal como la única realmente adecuada. La expresión verbal y escrita deben aprenderse y este aspecto debe constituir un punto básico en la formación en Ingeniería del Software. Junto a la capacidad de trabajo en equipo, el compartir tareas, la capacidad de buscar acuerdos, de dar confianza y de ganar respeto.

6. Conclusiones

El papel ecónomico del software podría cambiar en el futuro. La importancia de la Ingeniería del Software se incrementará significativamente y esto sólo será posible con la ayuda de medios eficientes. Este desafío será resuelto mediante un profundo cambio de contenidos y de tipos de formación a todos los niveles. No todo usuario de Informática es un informático. El conocimiento de un lenguaje de programación no es suficiente para ser un desarrollador de software. Estas obviedades jamás serán suficientemente repetidas.

Seguridad

De mí misma líbreme Dios, que del Sircam ya me libro yo (y II)

Mª del Carmen Ugarte García
Técnica de Sistemas en IBM S.A.E., Socia de ATI

<cugarte@ati.es>

La primera parte de este artículo se publicó en el número 153 de *Novática*, sep.-oct. 2001 (pp. 56-59)

Resumen: *Papel que cumple el usuario en la transmisión y propagación de los virus. De su credibilidad y de cómo las numerosas bromas y chistes que circulan por Internet preparan el camino para que los virus reales hagan su labor. Los futuros virus aprovecharán varias vías de infección y unos prepararán el camino a los otros.*

Palabras clave: *virus de la credibilidad, bulos (*hoaxes*), virus manuales, bromas, cotidianidad., retrovirus, virus polifacéticos.*

1. Introducción

He de comenzar esta segunda parte declarando que cuando decidí escribir estos artículos sobre el **Sircam**, mis primeros pasos estuvieron dirigidos a recopilar información técnica, cuanta más y minuciosa mejor sobre la forma en la que los virus han ido actuando o qué nueva virguería había introducido el nuevo ejemplar aparecido esa misma mañana ... Luego, al examinar con más calma toda esa documentación y ponerme manos a la obra, en el momento de pararme realmente a pensar en por qué un virus tiene éxito, no me fue difícil darme cuenta de que todos esos aspectos técnicos no suelen ser la mayor parte de las veces tan importantes como los aspectos humanos. Así que vuelta a empezar, los aspectos técnicos irán pasando a un segundo plano o a referencias al pie y volvemos a hacernos la gran pregunta: ¿Por qué nos dejamos engañar tan fácilmente?

2. ¿Por qué nos engañó el Sircam?

Aunque en los últimos tiempos han aparecido virus que no requerían para su activación e infección ninguna acción prácticamente por parte del usuario [1] sigue siendo cierto que hasta ahora la mayor parte de los virus, y el Sircam en particular al menos por una de sus vías, requieren que el usuario abra y ejecute voluntariamente un fichero que suele presentarse como adjunto en una nota de correo electrónico, aunque tampoco faltan en los últimos tiempos las invitaciones para visitar URL infectadas.

El cebo que acompaña a estos virus forma parte del virus mismo. Si muchos de estos mensajes no estuvieran tan bien pergeñados, los usuarios no picarían, sin embargo ...

Ya se sabe que hay usuarios que por ignorancia tienden a abrir, ejecutar o visitar, todo lo que se les envía; en muchos casos porque vienen de amigos o conocidos, en otros simplemente porque no saben distinguir las fronteras, cada vez más sutiles, entre lo que es texto y lo que es un adjunto. Bien, no es ocioso, aunque lo parezca, recordar que nuestros mejores amigos o los competentísimos colegas pueden enviarnos un virus en un momento determinado; nadie, ninguno, estamos fuera de peligro, pues la infección puede haber sido totalmente accidental y más en estos tiempos donde las últimas novedades emplean diversas vías de infección. Pero dejando aparte a los amigos, que nos inspiran una cierta confianza, ¿cuál puede ser la razón para que cualquier usuario, incluso los más conscientes, caigan en la trampa?

Demos un repaso a los cebos clásicos y recordemos que los atractivos temas pornográficos siguen surtiendo efecto, recordemos el viejo **Melissa**, pero también el famoso **Hybrids**, que promete una nueva versión del cuento de Blancanieves ... En otras ocasiones se emplean como cebos personajes infantiles como Pikachu, o famosos, como la tenista Anna Kournikova. Tampoco hay que echar en olvido los que se escudan en relaciones amables, el propio **I Love You**, o los que te embarcan en campañas pacifistas como el **Vote**... De todas estas triquiñuelas hemos aprendido a sospechar, en mayor o menor medida, a veces escarmentando en carne propia, así como a sospechar del viejo truco de las dobles extensiones que ocultan el verdadero carácter del fichero adjunto... Pero quizás todavía no hemos aprendido, y el Sircam hizo buen uso de ello, de la cotidianidad y familiaridad de ciertas notas.

El **Magistr**, como ya vimos en el artículo anterior, había ensayado el crear familiaridad componiendo el cuerpo del mensaje en base a textos ya escritos, pero este método introducía demasiada aleatoriedad; el Sircam volvió sobre sus pasos y se presentó mediante un texto muy simple, pero perfectamente estudiado y a fin de cuentas uno de tantos que circulan todos los días por Internet.

Los estudiosos de las relaciones interpersonales en la Red aseguran que el lenguaje en esta tiene que estar entre medias del formalismo y la espontaneidad, más cerca del segundo punto que del primero y el Sircam dio en el clavo al reproducir, incluso con alguna falta de ortografía, el nivel de lengua al que tan acostumbrados estamos los que hacemos gran uso del correo electrónico. Por otro lado, el hecho de que el texto estuviera en español, en su versión más difundida, contribuyó a dar una nota de familiaridad entre la población hispanohablante --un hecho que sirvió, como ya vimos, para que las casas de antivirus minusvaloraran en un primer momento su potencial de expansión. He aquí el texto en español y sus variantes entre corchetes:

```
Hola como estas?
Te mando este archivo para que me des tu punto de vista
[ Espero me puedas ayudar con el archivo que te mando]
[ Espero te guste este archivo que te mando]
[ Este es el archivo con la informacion que me pediste]
Nos vemos pronto, gracias.
```

Los siguientes casos son reales y están extraídos de comentarios hechos por los propios usuarios infectados, el anonimato lo guardamos, pero describimos sus circunstancias personales: F. es escritor y periodista por cuenta propia, colabora además con diferentes publicaciones como corrector, editor y coordinador de proyectos, el material para su trabajo le llega en formato DOC, suele actualizar el antivirus como mínimo una vez a la semana y suele ser precavido, pero recibió el virus demasiado pronto. «El nombre del remitente me sonaba a conocido, pero no era uno de mis corresponsales habituales, me llamó la atención la descuidada ortografía -- confiesa F.-- y me dije: "¡Bien empezamos!". Pensando que era un texto de los que me mandan para corregir o emitir mi opinión comencé a anotar los fallos ya desde la presentación, incluso sin avanzar mucho en el texto le envié una nota al emisor sobre ello ... demasiado tarde, me había infectado y había comenzado a enviar copias del virus a diestro y siniestro, sólo me quedó avisar a mis corresponsales habituales». N. está en el equipo de F. y también recibió una copia temprana del virus, pero N. es mucho más estricto que F. con la ortografía y además sospecha, por principio, de todo lo que le llega sin haberlo pedido (el gato escaldado...). «No conocía al remitente y además sospeché al ver la ortografía, no es normal presentar el trabajo de forma tan descuidada». No piensan lo mismo los hermanos G., editores de una revista de temas etnoculturales en la Web, que suelen recibir a diario varios artículos, no sólo de sus colaboradores habituales, para su evaluación: «No tenemos la seguridad de con qué fichero o mensaje nos infectamos, pero sospechamos de uno que parecía estar incompleto claramente y además nada tenía que ver con los temas de nuestras publicaciones, aunque a decir verdad, el recibir ofertas de todo tipo y que nada tienen que ver con nuestro trabajo es bastante habitual en nuestro caso. ¿Las faltas de ortografía? ... Bueno, teniendo en cuenta que a veces tenemos que corregir las de los catedráticos... :-). No, tampoco nos llamó la atención, la gente no suele poner mucho cuidado en las líneas de presentación que suelen ser bastante escuetas».

El segundo elemento que influía confianza y cotidianidad era el nombre de los ficheros que se adjuntaban. Extraídos de la carpeta de documentos, los ficheros infectados eran en muchos casos uno más de la serie intercambiada normalmente entre corresponsales habituales. Nos dice S., traductora mexicana que trabaja para agencias estadounidenses y alemanas: «Frecuentemente utilizamos alguna palabra clave del proyecto para nombrar y numerar los ficheros que nos intercambiamos, ese nombre también lo solemos poner en el asunto y no solemos poner una presentación muy extensa. Para mí era un fichero más y por lo tanto no sospeché nada»: y quizás en estas líneas estén estupendamente resumidas las costumbres de los usuarios que el Sircam supo estudiar y reproducir y que a su vez resumo en una palabra: **cotidianidad.**

A las infecciones a través del correo habría que sumar las producidas por compartir discos en red, como es el caso de M.A.., que trabaja para una compañía de seguros: «Me vi sorprendido porque un amigo me avisó de que le había mandado un virus y de que casi al mismo tiempo mi bandeja se llenara de rechazos raros... Pero yo no había abierto ningún archivo que me hubiera llegado por correo en los días anteriores, de hecho no suelo recibirlos, así que de haberme llegado alguno lo tendría que recordar. Puse el hecho en conocimiento del departamento informático y dos días después me enteré de que la infección había venido a través de un fichero de trabajo compartido con el departamento de contabilidad; al parecer todo un subdirectorio compartido, al que teníamos acceso dos departamentos, se infectó». MA. confiesa no saber cada cuánto se actualiza su antivirus porque todo eso se lo dieron hecho desde el departamento informático.

3. ¿Dónde empieza la historia?: Los «virus de la credibilidad»

Si yo dijera ahora que muchos de los usuarios infectados por el Sircam muy probablemente fueron atacados antes por uno o varios «virus de la credibilidad» más de uno pensará que de qué estoy hablando o en todo caso que soy un poco exagerada pues esos virus suelen ser meros bulos totalmente inocuos. ¿Lo son realmente?

José Luis López [2] en un artículo publicado originalmente el 22 de junio de 1998 [3], lanzó esta acertada denominación, **virus de la credibilidad**, para referirse en primer lugar a todos los bulos, timos y camelos que en forma de cartas en cadena circulan por Internet. Todos alguna vez habremos recibido alguna de estas cartas, que abarcan todo un abanico de posibilidades, y que una voz anónima nos pedía que distribuyéramos o que no cortáramos la cadena, porque de lo contrario nos acontecerían grandes males... Pero seguro que en ese espacio de tiempo no habremos recibido un solo ejemplar de una determinada carta, sino varios. Ello es una prueba de que aun en los casos en los que no hay un mecanismo automático, como en el caso de los gusanos, los propios usuarios ponen ese mecanismo y se convierten ellos mismos en «gusanos» ...

Si pensamos en los contenidos de las cartas en cadena y los bulos veremos que siguen explotando, de alguna forma, los mismos trucos que los virus, de hecho los hacedores de virus han visto que estos mecanismos funcionan. La única diferencia, quizás, es que las cartas en cadena, salvo las que explotan claramente el tema de la avaricia, se valen de los valores más nobles del ser humano: la compasión, la generosidad, la justicia social, cuando no los propios miedos. ¿Quién no recuerda el caso de una niña enferma de cáncer para la que se pedía un dólar que ni tan siquiera tenía que salir del propio bolsillo, bastaba con pulsar en un enlace? ¿Quién no recuerda la lista abierta en favor de las mujeres afganas a la que se iban añadiendo nombres prestigiosos y que teóricamente tendría que acabar en las Naciones Unidas? ¿Quién no recuerda el bulo de antes del verano que relacionaba a cierto grupo de pop con una banda terrorista?... Bulos y bulos que se suceden y en los que la gente sigue cayendo una y otra vez.

Los acontecimientos del 11 de septiembre nos han traído una nueva oleada de estos virus de la credibilidad. Mientras los centros de alerta aventuraban ataques de virus reales, lo cierto es que por Internet se extendían toda una serie de bulos de distinto tipo, no faltaba el virus verdadero que te pedía te adhirieras a una campaña por la paz ejecutando cierto mensaje o pidiendo que visitaras cierta página web de la que en determinadas condiciones te descargabas el virus **Nimda**. En los primeros días las noticias y contranoticias sobre la actualidad de festejos y alegrías por el atentado se sucedían y prodigaban por los foros; toda una cadena de notas hablaba sobre si Nostradamus había profetizado o no la catástrofe, a la gente le picaba la curiosidad y en algún lugar el autor del bulo, Neil Marshall, contemplaba risueño como por segunda vez en la historia su cuento producía los efectos esperados [4]. El carbunco *(anthrax*, en inglés), tan temido en Estados Unidos, nos llegaba también al software maligno *(malware)* en forma de un cierto virus cuyas *releases*, una tras otra iban fallando, a la vez que hacía ruido en la prensa y hasta en los vagones del metro en forma de avisos que alertaban contra él; pero también en forma de la puesta en escena de un viejo bulo: el Klingerman, ahora en versión **Antrax**, supuesto virus biológico contra el que la policía te prevenía.

Pero quizás los más habituales, y no por ello más fáciles de reconocer, sean los relacionados con los propios virus: viejos bulos siguen recorriendo las redes de vez en cuando en nuevas oleadas, a ellos se unen nuevos bulos con ligeras variantes en el modo de presentación, pero el mecanismo de control, el que sigue funcionando sigue siendo esa frase típica de todos ellos: «Manda esta información a toda tu libreta de direcciones», a veces repetida dentro de un texto de pocas líneas.

Y hay que hacer notar que por descabellado que parezca el texto de uno de estos virus de la credibilidad, no está descartado en ningún caso el que se produzca una cierta alarma. Es el caso del llamado virus del **Real Madrid** [5], bulo que comienza, como todos, anunciándote grandes males para tu computadora, pasa en seguida pasar a anunciarte males en la cuenta corriente para pasar de plano al círculo familiar y personal: la suegra que se viene a vivir contigo, el gato que se agarra unas pulgas... y finaliza, sin posibilidades de baja, por afiliarte al equipo de fútbol rival. Pues bien, este virus que desde las primeras líneas se ve que es una auténtica broma ha llegado a alarmar al director del centro de proceso de datos de una empresa cuando ha circulado entre sus empleados [comentado en el foro electrónico de ATI]. Dicho esto, no podemos decir que ningún bulo sea inocuo y no sólo ya por la carga de tráfico que producen, sino porque van creando un clima propicio, una psicosis contra los virus cuyos efectos en cualquier momento pueden ser los menos esperados, salvo, probablemente, para los que los han pergeñado.

Muy relacionados con los virus de la credibilidad y dentro de ellos dando un paso hacia la sofisticación están los **virus manuales**, aquellos en los que llegado el caso, el crédulo usuario puede llegar a formatear el disco duro de su computadora. Era lo que te pedía que hicieras el virus del gallego o del lepero, aquel en que literalmente te anunciaba que habías recibido dicho virus pero que como su autor no tenía ni idea de programación, se basaba en tu honor por el que procederías a borrarte el disco duro. Bien, el virus del gallego fue un chiste pero no lo fue en la primavera del 2001 el virus SULFNBK.EXE. Clasificado como un *hoax* por la mayoría de las casas de antivirus, resultó ser totalmente auténtico: en un texto, que algunos amigos llegaban a redactar con sus propias palabras, te anunciaban que se había descubierto un nuevo virus, un troyano que se activaba en una determinada fecha aún lejana, aunque no demasiado, por lo que los antivirus no podían detectarlo, tampoco podían eliminarlo, por alguna causa que se desconocía. Te pedían que investigaras la presencia en tu disco de cierto fichero, el SULFNBK.EXE, y de encontrarlo, por ser el virus, eliminarlo. Por supuesto debías enviar esta información a cuantos más amigos mejor. El fichero, en realidad un fichero del sistema, fue borrado por no pocos usuarios, algunos bastante expertos en temas de computación que se encontraron con un palmo de narices al ver que habían caído como unos pardillos en este truco. Quizás no se leyeron todo el texto y quizás no cayeron en la serie de incongruencias como que los antivirus sólo detectan los virus mientras están dormidos, quizás sólo leyeron las letras mayúsculas, viejo truco, de los bulos, pero lo cierto es que picaron y transmitieron sus experiencias a sus amigos.

Desde antiguo conocemos también las bromas *(jokes)*, pequeños programas que simulaban grandes males en nuestra computadora y que normalmente también se aprovechaban de nuestras debilidades y complejos. No faltan los que simulan haber pillado un virus de verdad como el **Joke-Orla**, que simula una infección por *I Love You* [6]. Aunque son tan fáciles de detectar como un virus por los antivirus, algunas casas (Norton entre ellas) han elegido no incluirlos para no alarmar innecesariamente al usuario y por ser inocuas; sin embargo, algunos servicios, como el de RedIris, los incluyen en sus servicios de interceptación. Realmente no son tan inocuas, pues cuanto menos aparte del consumo de recursos gratuito que realizan, crean un ambiente de relajación bastante abonado para que virus reales arraiguen en él, por no hablar de las acciones que el estupefacto usuario puede emprender para tratar de librarse del virus.

4. El truco del almendruco y un mundo de idiotas

El Sircam trajo algunos beneficios indirectos pero también sirvió para que volvieran a salir a la palestra los viejos trucos de listillo que no hacían sino aumentar la bola de nieve.

Entre los beneficios indirectos deberemos señalar que una vez más la auténtica solidaridad, gracias a la cual en algunos aspectos Internet es lo que es, y la libre y rápida circulación de la información volvieron a actuar: En foros de todo tipo comenzaron a cundir los consejos serios y realmente útiles que recordaban cómo revisar las opciones de las herramientas para estar más seguros o la utilización de filtros previos en la revisión del correo, tales como Magic Mail <http://www.geocities.com/SiliconValley/Vista/2576/magic.html>, o clientes de correo como Pegasus Mail <http://www.pmail.com/> que permiten una selección de las notas en el servidor. Pero junto a consejos de este tipo también proliferaron los «trucos del

almendruco» que aconsejaban cambiar la forma de la fecha, añadir ristras de ceros en algún lugar de la libreta de direcciones para engañar (sic) al Outlook, y otras números con las herramientas más propias de contorsionistas de circo que de técnicos informáticos... Y sigue siendo curioso ver cómo estos consejos, algunos auténticos bulos que han llegado a provocar alertas reales de los centros oficiales, se propagan con una facilidad asombrosa. Mucho más reacios parecen mostrarse los usuarios a actualizar el antivirus o a visitar y suscribirse a boletines fiables de seguridad.

Entre los beneficios que nos trajo el Sircam, no hay mal que por bien no venga, habría que destacar el servicio de alerta puesto en marcha por el Ministerio de Ciencia y Tecnología español <http://www.AlertaAntivirus.es/> con distintas posibilidades de recibir la información. Quizás la más recomendable para el usuario final, pues no inunda con notas, sea la de la suscripción semanal por la que se recibe un informe de alertas fácilmente identificable en el buzón. También se reciben dichas alertas en el momento en que se detectan virus peligrosos clasificados como de gran difusión.

Y no podemos por menos que mencionar en lengua castellana la consolidación de un sitio al que venimos citando a lo largo de este artículo: *VirusAttack* < http://www.virusattack.com.ar/>, donde de forma puntualísima y documentadísima facilitan en español *(no excuses)* todo tipo de información sobre los virus a través de boletines y la propia web.

Pero a pesar de todas estas facilidades,

... El día 16 de octubre se cumplió la fecha fatídica en la cual el Sircam tenía que proceder a borrar el disco duro. Análisis del código del Sircam realizado por las casas de seguridad aseguraban que, al menos en algunas de sus variantes, el borrado del disco no llegaría a producirse debido a un tonto error de programación... Al día de hoy (29 de octubre) no parece que la prensa se haya hecho eco de desastres en este sentido debidos al Sircam, sin embargo, copias del mismo siguen llegando... Hay quien piensa que este borrado del disco tendría que haberse producido, «aunque sólo fuera para darles una lección a todos los idiotas que tienen aún su máquina infectada».

5. Lo que se avista

¿Ha sido suficiente el escarmiento del Sircam? Puede que sea pronto para contestar, algo hemos aprendido.

El Sircam ha marcado un hito --dicen algunos-- , pero no porque iniciara una nueva etapa, sino porque la gente ha aprendido demasiado con él y ya estarán avisados. ¡Bien por los optimistas! pero no olvidemos que el hombre es un animal que tropieza varias veces en la misma piedra, y tampoco minimicemos el componente psicológico (la llamada «ingeniería social») que los nuevos virus llevarán también incorporado. Digamos que hoy por hoy no hay virus técnicamente «puros», aunque por arte de birlibirloque nos infecten la máquina sin que nosotros hayamos pensando en cómo pudieron hacerlo. Si recapitulamos encontraremos que en algún punto fallamos por descuido, negligencia o curiosidad ... Pero repasaremos para finalizar con qué cara se presentan al día de hoy algunas de las amenazas.

En primer lugar los virus y agujeros de seguridad que afectan a los servidores y ante los cuales lo único que podemos hacer los usuarios es exigir calidad a los proveedores de servicio. La exigencia a los proveedores de contenidos ya es ciertamente más difícil, para ello sólo queda contar con un buen antivirus que nos avise del peligro e impida la infección. Tampoco está de más el estar al tanto de los parches de seguridad, aunque sobre la aplicación loca e indiscriminada de parches también habrá que estar alerta, amén de recordar que hay virus que se esconden tras parches enviados aparentemente por la mismísima Microsoft.

En segundo lugar deberemos estar atentos a aquellos virus que presentan varias vías de infección, virus polifacéticos [7]. Ya no sólo se infecta uno por abrir un archivo, el código maligno puede estar oculto en el cuerpo del mensaje, agazapado en el IRC o esperándonos en una página web. Ante esto es conveniente utilizar las opciones de correo que realmente se necesiten y como medida de precaución acostumbrarse a leer el correo desconectados de la Red o utilizar un cliente de correo que opere de forma totalmente autónoma, es decir que la opción de visitar un cierto sitio sea un acto voluntario y consciente. Y si somos usuarios habituales de las charlas en línea deberemos tomar precauciones adicionales así como proveernos de un buen nivel de software.

Si hemos sufrido una infección la cura deberá seguir varios pasos y el asesoramiento de un experto o la visita a páginas especializadas para ver todos los entresijos se hará necesaria porque no es extraño que los nuevos virus dejen abiertas las puertas a otros, virus que actúan en dos momentos del tiempo por la combinación de acciones diferentes, virus que crean otros virus. La cura deberá también estar encaminada a cerrar los agujeros de seguridad de nuestros productos.

El argumento de «yo me siento seguro porque utilizo...» va también debilitándose. Aunque sólo sea para probar que todo sistema es vulnerable: los *háckeres* lo prueban todo [8]. Y no podemos olvidar de que si ahora la mayoría de virus explotan las debilidades de un sistema y de unos productos de una determinada casa, es por la gran difusión de los mismos. Pero, a medida que aumenta la difusión de cierto programa su vulnerabilidad aumenta, el hacer virus para ellos resulta rentable.

No olvidemos tampoco en que las conexiones fijas están proliferando en el ámbito doméstico lo cual establece una puerta abierta de forma permanente para la entrada de todo tipo de ataques. Sería altamente recomendable que los proveedores de estos servicios de conexión alertaran de la necesidad de instalar un cortafuegos, pero no lo suelen hacer, «no es su competencia», sin embargo, no dudan en atemorizar al confundido usuario cuando detectan amenazas fundadas o infundadas.

Una potencial amenaza, que tampoco hay que echar en saco

roto, proviene de los llamados retrovirus (virus creados para atacar a los virus) [9] y de los virus experimentales *(zoo* en inglés). De unos y otros pueden derivarse virus verdaderos, sin embargo, parece haber una diferencia notable entre ellos. La mayoría de los expertos en seguridad condenan los retrovirus, no es una buena forma de defenderse de los virus, ni tan siquiera de frenar su expansión, y en cuanto a los segundos no cabe duda de que la experimentación continua y la búsqueda de agujeros y posibilidades para productos considerados hasta la fecha seguros, aunque comporten un riesgo, deben practicarse y alentar a las casas de antivirus, que siempre parecen ir detrás en sus acciones, a renovar sus productos e introducir nuevas vías de detección. Sin embargo, toda prudencia es poca en la manipulación de estos nuevos virus que deben probarse en redes estancas y aisladas y nunca en la propia red Internet, los experimentos con gaseosa, que diría un castizo [10].

Para finalizar quisiera dedicarle un parrafito a la razón del título de estos artículos: El peligro está en nosotros mismos, porque lo cierto es que, salvo invasiones masivas como la del Sircam, la mayor parte de los virus que me han llegado lo han hecho por mano de conocidos. Conocidos que antes se habían dedicado a mandarme toda clase de bulos, conocidos que no dudan en avisarte de que te cures antes de curarse ellos mismos, conocidos que disfrutan mandándote chistes y bromas, o conocidos que como yo misma sienten demasiada curiosidad por estas cosas y en un momento determinado la curiosidad les puede y dan un paso más allá... Porque yo soy uno de esos curiosos usuarios: de mí misma líbreme Dios, que del Sircam ya me sé librar yo.

Notas

[1] Dejando aparte las infecciones por visitar sitios web contaminados (Código Rojo, Nimda...) o por compartir recursos infectados, ateniéndonos al correo electrónico estamos hablando de virus como el W32Nimda <http://www.symantec.com/avcenter/venc/data/w32. nimda.a@mm.html> [Consultada: septiembre del 2001], el VBS. Cuerpo <http://www.symantec.com/avcenter/venc/data/vbs.cuerpo.a @mm.html> o el ya viejo BubbleBoy <http://www.symantec.com/avcenter/venc/data/vbs.bubbleboy.html>. [Consultada: septiembre del 2001]

[2] Periodista y consultor uruguayo. Mantiene una columna sobre temas víricos en *Las Noticias en la Red*, ha sido colaborador en varias publicaciones como *Virus en Internet* (Panda Software Anaya Multimedia) y mantiene una empresa de consultoría de seguridad, Video Soft <http://www.videosoft.net.uy>, que junto al web *VirusAttack* <http://www.virusattack.com.ar/>, ha recibido últimamente toda clase de reconocimientos por su labor dentro del ámbito de habla hispana.

[3] Publicado originalmente en *VSantivirus* núm. 53, año 2, puede leerse actualmente en línea en *VirusAttack* <http://virusattack.xnetwork. com.ar/articulos/VerArticulo.php3?idarticulo=5>. [Consultada: octubre del 2001].

[4] El nombre del equipo de fútbol varía según los lugares y las ciudades para hacerlo coincidir con el nombre del equipo rival, lo que supone una clara manipulación, una mutación, del mismo totalmente voluntaria.

[5] En 1998 este estudiante canadiense colgó en <http://www.ed.brocku.ca/%7Enmarshal/nostradamus.htm> una serie de cuartetas proféticas atribuidas a Nostradamus para demostrar lo fácil que es escribir profecías ambiguas e interpretarlas después como convenga. Ahora, esas cuartetas han sido convenientemente completadas asegurando el éxito. El número de visitas a esa página ha sido tal que la universidad de Neil se ha visto obligada a retirar la página, sustituyéndola por un aviso, para evitar un colapso del servicio. Más información en <http://videosoft.tripod.com/hoax-wtc-nostradamus.htm> [Consultada: octubre del 2001].

[6] Sus características pueden leerse en <http://www.alertaantivirus.es/detalles/virus.php?cod_virus=492> [Consultada: octubre del 2001] RedIris en su red académica de sensores interceptó 122 ejemplares de esta broma (un 0,09% del total de incidencias analizadas) durante el mes de octubre. Téngase en cuenta que sólo se propaga manual e intencionadamente.

[7] Ver, por ejemplo, «Girigat, el ataque del monstruo de 63 cabezas» en *Hispasec* <http://www.hispasec.com/unaaldia.asp?id=202> [Consultada: 2 de noviembre del 2001]. Aunque el virus y el artículo son de 1999, ilustra bastante bien las múltiples posibilidades de infección, así como otras vías de peligro.

[8] Un ejemplo interesante de virus en producto hasta ahora seguro: QUINTERO, Bernardo: «Primer gusano del mundo en PDF» en *Hispasec* <http://www.hispasec.com/unaaldia.asp?id=1017> [Consultada: 2 de noviembre del 2001]. En cuanto a virus para Linux, búsquedas en los sitios especializados nos darán algunos resultados; como ejemplo podríamos ver uno de los primeros, el Ramen <http://www.symantec. com/avcenter/venc/data/linux.ramen.worm.html> [Consultada: 2 de noviembre del 2001].

[9] Un ejemplo es el Code Green, gusano que limpia el Code Red. Ver noticia y comentarios en *BarraPunto* <http://barrapunto.com/article.pl?sid=01/09/02/0015243&mode=thread&threshold=> [Consultada: 2 de noviembre del 2001].

[10] Recomendable leer la entrevista al hacedor de virus Zulú en *Virusattack* <http://virusattack.xnetwork.com.ar/entrevistas/Ver Entrevista.php3?identrevista=1> [Consultada: 2 de noviembre del 2001].

Tecnología de Objetos

¿Es conveniente la Orientación a Objetos en un primer curso de programación?

Jesús J. García Molina
Departamento de Informática y Sistemas, Universidad de Murcia

<jmolina@um.es>

Resumen: *este trabajo defiende la tesis de la inconveniencia de elegir el paradigma orientado a objetos para un curso de introducción a la programación, en el ámbito de las titulaciones universitarias de Informática. Para justificar esta postura, primero se establece cuáles deberían ser los objetivos de un primer curso de programación y a continuación se analiza cómo la orientación a objetos dificulta la consecución de esos objetivos.*

Palabras clave: *programación orientada a objetos, enseñanza de la programación.*

1. Motivación

Desde principios de la década pasada, la Programación Orientada a Objetos (**POO**) es reconocida como el paradigma más adecuado para mejorar la calidad del software, ya que favorece dos aspectos importantes como son la reutilización y la extensibilidad del código. En los últimos años, la tecnología **OO** ha sido aceptada por la industria del software, como lo prueba el fenómeno Java. Así, el último informe curricular de **ACM/IEEE** [1] incluye la POO dentro del núcleo de conocimientos básicos que un estudiante de Informática debe conocer.

Las directrices generales propias de las tres titulaciones de Informática (B.O.E. de 20, Noviembre, 1990) no incluyen la POO como materia propia o parte de la materia *Metodología y Tecnología de la Programación*. Por tanto no debe extrañar que casi todos los planes de estudio de nuestras universidades, elaborados una vez publicadas las directrices, hayan ignorado la OO, e incluso entre los planes reformados recientemente son muy pocos los que incluyen una asignatura que enseñe los fundamentos de esta tecnología.

Se puede afirmar que en la actualidad existe un convencimiento en los departamentos encargados de la enseñanza de la programación de la necesidad de que un titulado de Informática haya recibido una formación en OO a lo largo de sus estudios. La cuestión es qué, cuándo y cómo. Por razones de espacio, en este trabajo no se presenta una propuesta completa sobre cómo abordar la enseñanza de la tecnología OO en un plan de estudios, sino que nos limitaremos a estudiar la conveniencia o no de elegir el paradigma orientado a objetos para un primer curso de programación. No obstante, al final se incluirá un esquema de los contenidos que se consideran básicos.

Hasta el auge de la OO, el mundo académico estaba convencido de que la programación estructurada era el paradigma adecuado para una introducción a la programación, utilizando Pascal, Modula o Ada como lenguajes de programación. Sin embargo, conforme maduraba la OO, han ido apareciendo trabajos como [2,7] que defienden que el alumno debe iniciar el aprendizaje de la programación con la POO y se han publicado libros tales como [3, 9] que ilustran este enfoque. En nuestro país también ha surgido el debate y actualmente muchos departamentos encargados de la docencia en programación están discutiendo si llevar o no a la práctica tales propuestas, siendo cada vez más los profesores que propugnan que los alumnos comiencen a programar con Java. De hecho, algunos centros ya han experimentado con la introducción en el primer curso de un lenguaje OO (por lo común Java, aunque también se ha probado con C++ y Delphi). En este trabajo defenderé mi convicción de que no es apropiado que los estudiantes de Informática aterricen en el mundo de la programación a través de la perspectiva que proporciona la OO. Mis argumentos son fruto de mi experiencia en la Facultad de Informatica de la Universidad de Murcia enseñando durante seis años programación desde un punto de vista tradicional (en concreto utilizando el enfoque presentado en [8]), diez años impartiendo una asignatura sobre los fundamentos de la OO y cuatro años enseñando métodos OO y técnicas de diseño OO.

El trabajo está organizado del siguiente modo: primero se exponen los objetivos de un primer curso de programación, a continuación se analiza cómo la POO dificulta la consecución de dichos objetivos y finalmente se presentan las conclusiones.

2. Un primer curso de programación

Todos los planes de estudio de Informática incluyen en primer curso o bien dos asignaturas cuatrimestrales o bien una asignatura anual (lo menos común) destinadas a introducir al alumno en el universo de la programación de ordenadores. Aunque no tiene que ver con el problema que estamos tratando, hago un inciso para señalar mi convencimiento de la necesidad de que esta asignatura sea anual, ya que de este modo los alumnos tienen todo un curso para madurar una materia que para muchos de ellos supone entrar en una nueva forma de resolución de problemas. Además se evita un examen final en febrero, cuando apenas han tenido tiempo de asimilar y comprender los primeros conceptos y algoritmos,

y también se permite que el profesor pueda organizar mejor las prácticas, ya que se elimina la interrupción entre el fin de una asignatura cuatrimestral y el comienzo de la siguiente.

Este primer curso (a partir de ahora supondré una asignatura anual) debe ser planificado con la suposición de que los alumnos no conocen absolutamente nada sobre programación, ya que en la enseñanza secundaria no existe ninguna materia obligatoria en la que se imparta una introducción a la programación. Desde mi punto de vista, el objetivo del primer curso de programación es *dotar a los alumnos de los mecanismos necesarios para enfrentarse a la creación de programas para problemas pequeños y elementales en el marco de expresividad de un lenguaje imperativo estructurado, enseñándoles métodos, técnicas y conceptos que les permitan resolver los problemas, llegando de un modo riguroso desde la especificación al programa correcto, y siendo conscientes del problema de la eficiencia*. A continuación expondré una organización de un primer curso de programación y en base a ella justificaré la conveniencia del uso de un lenguaje imperativo estructurado (también se podría denominar lenguaje procedural) frente a un lenguaje OO.

El primer curso de programación debería estar organizado en dos partes: una introducción a la programación (**IP**) y unos fundamentos de la programación (**FP**), cada una con una carga mínima de 7.5 créditos, 4.5 teóricos y 3 prácticos.

La parte IP estaría destinada a que el alumno escriba sus primeros algoritmos utilizando: i) las composiciones secuencial, condicional e iterativa, ii) acciones y funciones, iii) tipos de datos básicos y iv) algoritmos básicos de recorrido y búsqueda en tablas (*arrays*) y secuencias. En los tres primeros meses debería manejar una notación algorítmica en vez de un lenguaje de programación, para prestar atención al diseño de los algoritmos, acostumbrando al alumno a pensar bien la solución antes de escribir el programa y haciéndole consciente de que se puede trabajar en el desarrollo de un algoritmo sin necesidad de un ordenador, todo ello para evitar crear «programadores compulsivos».

Durante esta etapa puede seguirse el enfoque presentado en [8] (en nuestra facultad seguimos uno inspirado en dicho texto plasmado en [5]). En el último mes de IP, se introduce el concepto de lenguaje de programación, se establece la correspondencia entre la notación algorítmica y un lenguaje estructurado (Pascal o Modula), se enseña a los alumnos el manejo de un entorno de programación y entonces se les exige que traduzcan algunos de los algoritmos escritos en la notación al lenguaje de programación elegido, proceso que les resultará casi inmediato.

El objetivo fundamental de IP es que el alumno comprenda bien la construcción de iteraciones. Para ello, es conveniente trabajar con el razonamiento inductivo que se esconde detrás del concepto de invariante (cuya formalización no es necesaria), resolviendo ejercicios clásicos como el de «la meseta más larga», «la secuencia con mayor peso», etc. Estos algoritmos iterativos implicarán el manejo de dos estructuras de datos básicas en programación: secuencias y *arrays*.

La parte de FP tendría como objetivo mostrar un conjunto de conceptos teóricos cuya comprensión es fundamental para un buen programador, sobre los cuales se profundizará en cursos posteriores. Estos conceptos serían: i) abstracción operacional, ii) recursión, iii) eficiencia, iv) abstracción de datos, y v) estructuras de datos dinámicas. La introducción al problema de la eficiencia serviría para presentar los algoritmos básicos de ordenación y búsqueda. El otro objetivo importante de FP sería que los alumnos se enfrentasen al problema de construir programas pequeños, para lo cual utilizarían primero la técnica del refinamiento por pasos sucesivos y más adelante se les introduciría en la programación basada en módulos.

3. La OO en la introducción a la programación

Supuesto que aceptamos como razonable el planteamiento expuesto en el apartado anterior para un primer curso de programación, ahora analizaremos cómo influye la elección de la POO como primer paradigma en lugar del paradigma imperativo estructurado.

Antes de continuar conviene hacer algunas aclaraciones. Primera, la POO puede considerarse como un paradigma construido sobre la base del paradigma imperativo, en tanto en cuanto el modelo operacional está basado en la asignación, como mecanismo de cambio de estado, y en las estructuras de control secuencial, condicional e iterativa, como mecanismos de control de la ejecución. Sin embargo, la POO cambia la invocación a rutinas por mensajes, lo cual, combinado con la herencia, introduce los conceptos de polimorfismo y ligadura dinámica, dando lugar a un paradigma claramente diferenciado.

En segundo lugar, existen diferentes visiones sobre cómo iniciar la enseñanza de la programación con la POO. Una posible visión, que podríamos denominar *enfoque débil*, sería impartir un curso tradicional de programación pero utilizando un lenguaje OO para escribir los programas (no se explican con detalle los fundamentos de la OO). En el otro extremo tendríamos un *enfoque fuerte* en el que se enseñan desde el principio los fundamentos de la OO y el alumno llega a manejar la herencia. Este segundo enfoque es defendido por B. Meyer en [7] y también se justifica en [2]. En medio, tendríamos un *enfoque híbrido* como el reflejado en [3], donde se plantea definir los conceptos de clase y de diseño dirigido por responsabilidades, combinado con el resto de temas propios de un primer año (estructuras de control, iteración, *arrays*, ordenación, recursividad, estructuras de datos dinámicas,…) pero no se maneja la herencia y el polimorfismo.

La tercera aclaración es señalar que en la discusión que sigue supondremos que el lenguaje OO que se enseñaría en un primer curso es un lenguaje OO puro[1] (Eiffel, Smalltalk o Java) no un híbrido como C++, ya que existe un consenso sobre la conveniencia de iniciar la enseñanza de la OO con un lenguaje OO puro.

Si nos centramos en IP, nos encontramos que en esa etapa el alumno se debe preocupar principalmente de dominar algoritmos iterativos que manejan secuencias (simplemente

obtenidas del *buffer* del teclado o generadas por recurrencia), *strings* y *arrays* (posiblemente multidimensionales) de tipos básicos. La introducción del concepto de objeto complica innecesariamente estos algoritmos, al ser necesario incluir el concepto de clase y mensaje, en vez de escribir una simple secuencia de instrucciones imperativas como se haría en Pascal, por ejemplo. En el caso de Java, se da una consideración especial a los tipos básicos y a los *arrays*, de modo que se podrían escribir los algoritmos de una forma similar a Pascal, pero todavía sería necesario incluir el concepto de clase (el programa sería la rutina *main*) y el de mensaje para escribir las operaciones de entrada/salida (aunque el alumno puede escribirlas por acto de fe, sin preocuparse del concepto de mensaje). Para manejar *strings*, ya sería necesario introducir la clase *String,* la creación de objetos y el empleo de mensajes (por ejemplo para comprobar la igualdad). Si se examina el texto [3], se puede ver cómo en el segundo capítulo, en el que sólo se analiza cómo mostrar por pantalla un mensaje (un *string* almacenado en una variable), se deben introducir los conceptos de clase, método, mensaje, objeto, referencia y asignación con semántica de referencia. Todo esto parece una complicación evitable.

Es claro que en una asignatura con el enfoque de IP la OO no produce ningún beneficio, al contrario. ¿Pero es posible dar otro enfoque para IP? En mi opinión, no. Al principio de sus estudios de programación, el alumno debe pelearse con el diseño de algoritmos que le van a permitir dominar la iteración y manejar los tipos y estructuras de datos básicos. Y el diseño de este tipo de algoritmos se ajusta al paradigma imperativo, ya que el concepto de mensaje, sobre el que se articula el modelo computacional OO, no es tenido en cuenta, aunque se use POO, por ser innecesario.

Por ejemplo, supongamos un ejercicio clásico de un curso de introducción a la programación: Sea una secuencia de 0's y 1's de longitud n (≈1) almacenada en un array *x*, debemos contar el número de pares de índices *(i,j)* con *i<j* para los cuales *x[i]=0* y *x[j]=1*. Una solución expresada en Pascal (sólo mostramos la iteración) sería:

```
i:=1;
np:=0;
nc:= 1- x[ 1] ;
while i<>n do begin
     i:= i + 1;
     if x[ i] = 1 then np:= np + nc
     else nc:= nc + 1
end
writeln('Numero de pares=' ,np);
```

La comprensión del razonamiento que lleva a un algoritmo de este tipo es el principal objetivo de IP. Supuesto que en vez de una solución imperativa nos planteásemos una solución OO, ¿habría que cambiar la forma de razonar? Claramente no. No tiene sentido aplicar en este caso el modelo computacional OO caracterizado por el paso de mensajes entre objetos.

Los conceptos de la POO (clase, objeto, mensaje, herencia, polimorfismo y ligadura dinámica) y el modelo computacional subyacente son aplicables cuando es necesario estructurar programas en diferentes módulos. De hecho, en un curso de introducción a la POO, para que el alumno ponga a prueba su compresión del significado de la OO, es preciso que se enfrente a la construcción de programas en los que deba identificar las clases, las relaciones entre clases, la interacción entre objetos. El típico problema del «cajero automático» ilustra bien el tipo de ejercicio adecuado para ese propósito.

Conviene también señalar que aunque el algoritmo anterior se expresaría prácticamente igual en Java, en cambio el alumno debería incluirlo en la rutina *main* de una clase y tratar las complicaciones propias de la entrada de datos.

En cuanto a la parte de FP, vamos a analizar por separado los cuatro principales aspectos que se tratan en ella: refinamiento por pasos sucesivos, recursividad, abstracción de datos y estructuras de datos dinámicas. El refinamiento por pasos sucesivos es una técnica propia de la programación estructurada, no aplicable en el contexto OO. He sido testigo de alumnos que debían escribir programas Java para un primer curso de programación, que aplicaban el refinamiento por pasos sucesivos dentro de la rutina *main* de una clase creada con el único objetivo de incluir dicha rutina, pero que no representaba una abstracción bien definida. Este tipo de clases ilustra cómo el uso de un lenguaje OO en IP o FP puede crear malos hábitos de programación.

La recursividad, al igual que pasaba con el estudio de la iteración, se estudia mediante esquemas algorítmicos en los que no tiene sentido aplicar el modelo computacional OO y es un concepto que todo programador debe conocer, pero para cuya comprensión la OO no aporta nada. Además, explicar su funcionamiento interno es más natural hacerlo a través del concepto de rutina (y su invocación) que del concepto de método (y mensaje).

Existe un consenso en que el paradigma OO se fundamenta sobre el concepto de abstracción de datos. Una clase no es más que la implementación de un tipo abstracto de datos. Luego no admite discusión la conveniencia de estudiar este concepto y las estructuras de datos dinámicas introduciendo los conceptos de la OO. Pero entonces surge un grave problema como sería la necesidad del aprendizaje de un segundo lenguaje en un mismo curso: el alumno que comenzaba a defenderse con Pascal o Modula debería enfrentarse al aprendizaje de un lenguaje OO, como podría ser Java. ¿Es conveniente que un alumno de primer curso deba manejar dos lenguajes distintos, cada uno reflejando un paradigma diferente? Me parece excesivo ya que es necesaria una transición no trivial desde el paradigma estructurado al paradigma OO. Además de la transición costosa de un lenguaje a otro, surgen otros problemas.

Si se utiliza un lenguaje OO en FP entonces los alumnos no conocerán el tipo puntero ya que trabajarían con semántica de referencia, pero esto no tendría excesiva importancia desde el punto de vista de la implementación de las estructuras de datos, ya que la forma de razonar es la misma se utilicen punteros explícitos o implícitos (referencias).

Si el lenguaje OO proporcionase genericidad, como es el caso de Eiffel, la definición de clases (módulos) que representasen estructuras de datos genéricas capaces de almacenar objetos de cualquier tipo (clases *contenedores*), no plantearía problemas. Sin embargo, si se utilizase un lenguaje sin genericidad, como es el caso de Java, sería necesario usar la clase raíz *Object* como el tipo de objeto que se puede almacenar en el contenedor, siendo necesario manejar el concepto de polimorfismo. Este sería el mayor problema ya que el alumno tendría que manejar conceptos de la OO relacionados con la herencia: polimorfismo, clase abstracta, interfaz, regla de la asignación y necesidad de conversión de tipos (*narrowing* en Java), no siendo consciente de su significado.

En definitiva, puesto que no habría tiempo para impartir una buena introducción al paradigma OO al mismo tiempo que se enseñan los otros contenidos de FP, los alumnos deberían manejar los conceptos OO sin tener una clara comprensión de ellos. Además, insistir en el esfuerzo que supondría para el alumno aprender un nuevo paradigma y un nuevo lenguaje, lo que por otra parte dificultaría el estudio del resto de temas del curso. Por ello veo positivo que para la enseñanza de IP y FP se utilice Modula (también podría elegirse Pascal para IP y luego Modula para FP, ya que hay una continuidad entre ambos).

Nadie duda de la necesidad de que a lo largo de sus estudios universitarios, un estudiante de Informática deberá conocer el paradigma imperativo y manejar lenguajes procedurales, como es el caso de C. Esto por dos motivos. Primero porque en algunas asignaturas, como es el caso de aquellas relacionadas con sistemas operativos y compiladores, es normalmente necesario el conocimiento del mencionado lenguaje. En segundo lugar porque todavía existen y se desarrollan numerosas aplicaciones con lenguajes procedurales, especialmente C, y por tanto el mercado demanda profesionales con esta formación.

Aceptando esta consideración, cabe plantearse la cuestión sobre si es más apropiado que un alumno conozca primero la POO y luego conozca la programación imperativa o es mejor hacerlo al revés. Por mi experiencia, no tengo dudas en afirmar que es mejor pasar de la programación imperativa a la POO, ya que el alumno comprende con facilidad las ventajas que aportan los conceptos OO.

Una vez que el alumno ha recibido un curso de programación modular (por ejemplo con Modula) en el que ha visto que el concepto de módulo es diferente al de tipo abstracto de dato, es muy natural ver que una clase combina los conceptos de módulo y tipo. Del mismo modo, se puede apreciar mejor el significado de la semántica de referencia una vez que se conoce la semántica de almacenamiento y el tipo puntero. El concepto de mensaje también se presenta de modo más elegante como un tipo de invocación de rutina en la que un parámetro juega un papel especial (objeto receptor). La herencia aparece como una nueva propiedad de esa combinación módulo-tipo que es la clase: un módulo-tipo que se define a partir de otro. El polimorfismo se puede presentar como una extensión del caso en el que una variable sólo se puede ligar a valores del tipo asociado en la declaración (monomorfismo). Y una vez que se comprende el concepto de mensaje y polimorfismo surge de forma natural la necesidad de la ligadura dinámica, que se contrasta con la ligadura estática que conoce el alumno. En definitiva, parece que la POO se puede enseñar con más facilidad cuando se construye sobre los conocimientos y experiencia del paradigma imperativo.

Por otra parte, cuando el alumno ha diseñado/construido programas aplicando las técnicas estructuradas y modulares, comprende en toda su extensión los beneficios que aporta la OO en la mejora de la calidad del software (facilidad para el modelado, reutilización, extensibilidad,...).

En definitiva, creo que ninguno de los tres enfoques mencionados al principio de este apartado (*débil*, *fuerte* e *híbrido*) es adecuado para iniciar la enseñanza de la programación, sino que la POO debería ser incluida como una asignatura de tercer curso, una vez que el alumno ha recibido una formación en programación procedural a través de una asignatura que combinase IP con FP, junto con otra que proporcionase una formación en algoritmia y estructuras de datos en el segundo curso. Tendría sentido que dicho curso de estructuras de datos utilizase un lenguaje OO para expresar los tipos abstractos de datos, de modo que el alumno al llegar al curso de OO del tercer año ya estuviese familiarizado con el lenguaje OO que se emplease en la parte práctica.

En el caso de la titulación de Ingeniero en Informática, debería existir en el plan de estudios una asignatura dedicada al estudio de un proceso software OO y a los patrones de diseño OO, en el sentido expuesto en [6], (éste puede ser un buen libro de texto, combinado con [4] para el estudio riguroso de los patrones de diseño básicos). Éste es el enfoque que hemos seguido hasta el momento en nuestra facultad y que creo ha dado muy buenos resultados, así lo han entendido los alumnos en las evaluaciones realizadas sobre las asignaturas. Conviene señalar que en [1], el proceso OO y los patrones de diseño se consideran obligatorios dentro del curriculum.

En [7] se critica el enfoque que proponemos en este trabajo, argumentando que los profesores de programación creen que lo más adecuado para sus alumnos es que aprendan los lenguajes en el mismo orden en que los aprendieron ellos: primero procedural, luego los lenguajes OO. Sin embargo, afirma que un buen día todo el mundo académico coincidió al considerar Pascal como el lenguaje adecuado para iniciar la enseñanza de la programación y nadie se echó las manos a la cabeza, así que si ahora todo el mundo considera la OO como el camino acertado, pues habrá que cambiar los esquemas. Además, entiende que los lenguajes procedurales se asimilan mejor una vez se conoce el paradigma OO. Mi experiencia dice lo contrario.

Creo que el enfoque expuesto en este trabajo refleja la idea de ir de lo más elemental a lo más complejo, como es habitual en la enseñanza. Primero se enseñan lenguajes monomórficos, luego polimórficos; tipos abstractos aislados antes que tipos que se definen a partir de otros estableciéndose relaciones de herencia; ligadura estática antes que la dinámica. Parece

sensato ver la OO como una evolución del paradigma imperativo.

4. Conclusión

En este trabajo se ha intentado justificar la inconveniencia de utilizar un lenguaje OO en un primer curso de programación. Una vez se han establecido los objetivos de ese primer año se han planteado dos objeciones principales:

- los contenidos del primer curso se ajustan mejor a un lenguaje procedural, y si se utiliza un lenguaje OO el alumno usa los mecanismos OO sin una comprensión clara.
- durante sus estudios, los alumnos deben conocer tanto el paradigma procedural como el OO, y es más apropiado y natural pasar del paradigma imperativo al OO que al revés.

5. Referencias

[1] ACM/IEEE, *Computing Curricula 2001*, Draft, March, 2000.
[2] Joel C. Adams, *Object-oriented Design: a five phase introduction to OO programming in CS1-2,* ACM SIGCSE'96. pp. 78-82, 1996.
[3] D. Arnow y G. Weiss, *Introduction to Programming using Java*, 2ª Edición, Addison-Wesley, 2000.
[4] E. Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.
[5] J. García Molina et al., *Introducción a la Programación*, ICE, Universidad de Murcia, Ed. Diego Marín, 1999.
[6] C. Larman, *Applying UML and Patterns: An introduction to object-oriented analysis and design*, Prentice-Hall, 1998.
[7] B. Meyer. *Towards an object-oriented curriculum.* Journal Object-Oriented Programming, pp 76-81, March, 1994.
[8] P. C. Scholl y J. P. Peyrin, *Esquemas Algorítmicos Fundamentales*, Masson, 1989.
[9] R. S. Wiener, *An Object-Oriented Introduction to Computer Science using Eiffel*, Prentice-Hall, 1996.

Nota

[1] Aunque consciente de las discusiones sobre la pureza de Java, comparado con C++ se puede considerar un lenguaje OO puro.

Referencias autorizadas

Las habituales referencias que desde 1999 nos ofrecen los coordinadores de las Secciones Técnicas de nuestra revista pueden consultarse en <http://www.ati.es/novatica/lecturas.html>

Sección Técnica «*Informática gráfica*» (Roberto Vivó)

Tema: libro de texto

Sergei Savchenko. *3D Graphics Programming. Games and Beyond*. Sams Publishing, 2000. ISBN 0-672-31929-2. El libro discute los fundamentos de los gráficos por computadora. Aunque su título promete introducirnos en el diseño de vídeo juegos y la problemática general de los gráficos en tiempo real, es un libro de texto más sobre fundamentos teóricos y técnicas básicas de *rendering*. La mayor ventaja reside en la posibilidad de construcción de una librería gráfica propia siguiendo las técnicas descritas y consultando en el CD, que acompaña al libro, el código desarrollado por el autor. El contenido del libro se divide en nueve capítulos y dos apéndices. El primer capítulo está dedicado a la interfaz hardware a modo de introducción. El segundo se refiere a fundamentos algebraicos y de representación numérica. Del tercero al octavo se tratan los problemas típicos del proceso de visualización; modelado geométrico y visual, recortado, visibilidad, iluminación, etc. En el último capítulo se habla someramente sobre el diseño de una aplicación. De los apéndices, el primero es el más interesante pues da una visión particular de una librería propia denominada **3Dgpl** que se desarrolla en el CD. En definitiva una obra bien escrita que puede servir de libro de texto para el conocimiento o enseñanza de los fundamentos de los gráficos por computadora.

Sección Técnica «*Ingeniería del Software*» (Luis Fernández Sanz)

Tema: *Extreme Programming* (XP) y otros métodos innovadores de desarrollo de software

- Centro de Recursos sobre Programación Extrema <http://www.xprogramming.com/> Se trata del clásico centro de recursos con artículos, información, descarga de recursos para la XP, enlaces, comunidad de interesados en XP, etc.
- Centro de recursos sobre Programación Extrema: *Extreme Programming: a gentle introduction* <http://www.extremeprogramming.org/index.html> Se trata de un centro de recursos sobre XP similar al anterior pero en el que la información trata de estar más estructurada y resulta más fácil de asimilar.
- Pruebas unitarias de clases para XP: JUnit <http://www.junit.org/> En este sitio se incluyen toda clase de ayudas para una realización tanto eficaz como eficiente de las pruebas de unidad de cada clase asociadas a la práctica de XP.
- Experimentación y datos de aplicación de XP y de *pair-programming* de Laurie Williams <http://collaboration.csc.ncsu.edu/laurie/>. En este sitio, se incluyen diversas publicaciones de la Dra. Williams sobre los efectos de XP y de la programación de pares o *pair-programming* (es decir, trabajo en equipos de dos desarrolladores) tomando algunos datos sobre su aplicación real (aunque en buena parte en desarrollos académicos).

Sección Técnica «*Internet*» (Alonso Álvarez García, Llorenç Pagés Casas)

Tema: Nuevas formas de atraer la atención del internauta: las empresas pasan, los conceptos permanecen

Hace ya tiempo que el *banner* tradicional está en entredicho como formato para la publicidad *online*. Es por ello que los publicistas del medio tratan de idear formas alternativas que eviten los recelos del internauta tanto hacia la interrupción de su navegación como hacia la dilapidación de su ancho de banda.

Para ello se apoyan en las capacidades cada vez mayores de generación y visualización de contenidos web multimedia: animaciones, audio, video,etc., concepto para el que en inglés se ha acuñado el término *rich media*.

Así por ejemplo, en **Webspot** <http://www.webspot.com> proponen la creación de anuncios interactivos para favorecer la «experiencia» de los usuarios (Podemos observar su galería de ejemplos en <http://www.webspot.com/pages/the_gallery.htm> o visitar la web de uno de sus clientes <http://www.warnerbross.com/>, mientras que en **OnFlow** <http://www.onflow.com> facilitan una tecnología para empaquetar anuncios interactivos y animados en solamente 10Kb.

La visualización de los anuncios de Webspot suele requirir el *plug-in* Shockwave <http://www.macromedia.com/shockwave> mientras que en el caso de OnFlow se requiere un pequeño *plug-in* propietario del que nos sorprenderá leer que fueron distribuidas 10 millones de copias en 5 meses <http://www.emerginginterest.com/news/onflow_052301.shtml> y que además se ocupa de enviar anónimamente al anunciante trazas de la navegación del usuario.

Pero quizás entre los nuevos conceptos manejados en la publicidad *online* el caso de los *interstitials* y sus precursores, los *zings*, es especialmente llamativo.

A mediados de 1998, Zing <http://www.zing.com> anunció una nueva idea, la **Zing Network**, una red de entretenimiento que pretendía aprovechar los tiempos muertos entre la carga de páginas para suministrar contenidos a los usuarios. Aún hoy podemos leer la presentación de la misma que hacía la revista **Wired** <http://www.wired.com/news/topstories/0,1287,13197,00.html> o bajarnos el *plug-in* que su uso requería desde la web de **PCWorld** <http://www.pcworld.com/downloads/browse/0,cat,548, sortIdx,1,00.asp>.

Pero Zing no logró nunca rentabilizar su idea quizás porque, aunque en su concepción inicial y consecuente oferta a sus usuarios, la publicidad no tenía un papel relevante, cualquier contenido preparado para ser visualizado en un tiempo limitado a unos pocos segundos no podía ser visto, interpretado o diseñado más que como mera publicidad.

Así, mientras Zing.com se centró en otros negocios <http://www.smartcomputing.com/editorial/article.asp?article=articles/archiv e/g0905/74g05/74g05.asp> y acabó en Julio de 2001 sucumbiendo a la crisis de las .com, el concepto de *zing* aplicado a la publicidad ha dejado huella en el campo del marketing digital. Basta consultar algunos diccionarios de términos especializados para comprobarlo <http://www.serprimeros.com/diccionario2.htm> o <http://www.altawebs.com/public/faq.asp?lngpos=3>.

Y lo que es más importante, ha dado lugar a un concepto algo más genérico, los *interstitials* <http://imarketingsolutions.com/interstitial.htm> que son páginas que se entremezclan de manera temporal (caducan y son eliminadas automáticamente) en la secuencia de navegación normal y que se pueden usar tanto para anuncios o promociones de la propia marca propietaria del web (*splash pages*) como para insertar publicidad ajena. Representan un eficaz antídoto a la capacidad de supresión automática de la atención hacia los *banners* que desarrollan los usuarios y figuran hoy en día en el

catálogo de opciones publicitarias aceptadas en la mayoría de los sitios más visitados de la Red (veamos por ejemplo las especificaciones de Lycos, <http://adreporting.lycos.com/specs.html>.

Como caso de éxito en esta tecnología podemos citar a **Unicast** y su desarrollo propietario los *superstitials* <http://www.unicast.com/superstitial> ampliamente aceptados en los sitios más importantes de la Red y visualizables en los clientes que acepten tecnología **Flash** (otra tecnología que se está convirtiendo en pseudo-estándar en la Red: <http://www.macromedia.com/software/flash>). Como muestra de campaña con *superstitials* < http://www.emerginginterest.com/partnerships/unicast/bmwclickstream/index.shtml>.

Destaquemos la rigidez de las normas que impone Unicast para el diseño de *superstitials* en cuanto a tamaño, caducidad, opciones del usuario para saltarlos, etc. (ver sus «Creative Specs») y su esfuerzo por mostrar que se cargan únicamente en los tiempos muertos de la navegación («How It Works»).

Cualquier cosa con tal de convencer a la cada vez más extendida militancia antipublicidad que viene dando lugar a negocios tan curiosos como el «Aniquilador de Ventanas Emergentes» (*Popup Killer*) de Meaya Software <http://www.popup-killer.com>.

Sección técnica «*Lengua e Informática*» (Mª Carmen Ugarte)

Tema: II Congreso Internacional de la Lengua Española

Las novedades de este número giran en torno al II Congreso Internacional de la Lengua Española celebrado recientemente en Valladolid: <http://cvc.cervantes.es/obref/congresos/valladolid/>

Desde este portal puede accederse a las distintas ponencias (en formato DOC) clasificadas por las cuatro áreas temáticas:
- El activo del español: <http://cvc.cervantes.es/obref/congresos/valladolid/activo/>
- El español en la sociedad de la información: <http://cvc.cervantes.es/obref/congresos/valladolid/informacion/>
- Nuevas fronteras del español: <http://cvc.cervantes.es/obref/congresos/valladolid/fronteras/>
- Universidad y diversidad del español: <http://cvc.cervantes.es/obref/congresos/valladolid/unidad/>

Desde estas secciones también se puede acceder a debates virtuales abiertos en torno a las ponencias.

Los documentos del I Congreso Internacional de la Lengua Española (Zacatecas, abril 1997) pueden consultarse en: <http://cvc.cervantes.es/obref/congresos/zacatecas/>, con especial atención al área de nuevas tecnologías: <http://cvc.cervantes.es/obref/congresos/zacatecas/tecnologias/>.

Entre las páginas privadas relacionadas con este evento destacamos la de uno de los ponentes, José Antonio Millán, <http://www.jamillan.com>, que presenta además del texto en HTML de su ponencia: <http://jamillan.com/propuesta.htm> su propia selección de ponencias: <http://www.jamillan.com/valladolid.htm>.

Diccionario de la Lengua Española (vigésima segunda edición). Coincidiendo con el II Congreso, la Real Academia Española <http://www.rae.es> ha publicado la vigésima segunda edición de su diccionario: *Diccionario de la Lengua Española* (2 vol.). Madrid: Editorial Espasa Calpe, 2001. ISBN: 84-239-6814-6. PVP: 6950 ptas. (41,77 EUR).

A la versión en línea, así como a la presentación, se puede acceder a través de: <http://buscon.rae.es/drae/drae.htm>. Está prevista la edición en CD-ROM pero de momento no está en el mercado.

La nueva web de la RAE. Finalmente la RAE ha hecho la presentación de su nueva web <http://www.rae.es>, donde pretende ampliar su gama de servicios. Lamentablemente el anuncio parece haberse realizado anticipadamente pues la mayoría de sus servicios o no están disponibles todavía (11 de noviembre del 2001) o presentan deficiencias. Seguiremos informando en próximos números.

Nota: los URL que se mencionan han sido consultados expresamente para estas referencias en la primera quincena de noviembre del 2001.

Sección técnica «*Libertades e Informática*» (Alberto Escolano)

Recomendamos la visita al centro de alertas de **EPIC** (*Electronic Privacy Information Center*) y su alerta <http://www.epic.org/alert/EPIC_Alert_8.21.html>, en la que se da información sobre legislación que nos va a afectar durante los próximos cuatro años. Un detalle más preciso se encuentra en: <http://www.epic.org/privacy/terrorism/s1510.html>.

Sección técnica «*Lingüística computacional*» (Xavier Gómez Guinovart)

Iwanska, Lucja M. y Stuart C. Shapiro (eds.), *Natural Language Processing and Knowledge Representation*. AAAI Press / The MIT Press, Menlo Park, 2000. ISBN 0-262-59021-2. Esta monografía académica de publicación reciente recopila en once capítulos algunos de los enfoques actuales de mayor relevancia en el campo de la representación y procesamiento computacional del significado del lenguaje natural. En la primera parte del libro, se discuten cinco importantes modelos teóricos para la representación del conocimiento y el razonamiento basados en el lenguaje natural: el sistema **UNO**, la sintaxis de Montague, **KRISP**, la lógica de episodios y **SNePS**. En la segunda parte del volumen, se exponen diversas aproximaciones a la representación del conocimiento aplicadas en diferentes sistemas de procesamiento del lenguaje natural, orientados a la traducción automática, al razonamiento espacio-temporal, al diálogo persona-ordenador y a la adquisición de conocimiento lingüístico. El libro concluye con un apéndice en el que se presentan los fundamentos matemáticos y computacionales de la representación del significado lingüístico mediante la lógica de primer orden, y con dos apéndices finales en los que se ilustran mediante ejemplos algunas de las dificultades clásicas de representación e inferencia características del lenguaje natural.

Martí Antonín, M. Antònia (coord.), *Les tecnologies del llenguatge*. Edicions de la Universitat Oberta de Catalunya, Barcelona, 2001. ISBN 84-8429-266-5. Manual universitario de lingüística computacional que presenta los fundamentos y aplicaciones de las tecnologías derivadas del procesamiento del lenguaje natural oral y escrito. Se trata de una obra colectiva, coordinada por la profesora M. Antònia Martí (de la Universidad de Barcelona), que cuenta con la participación de destacados especialistas pertenecientes a distintos centros de investigación y desarrollo del ámbito del Estado español. El volumen se orienta a la docencia universitaria de la materia y se estructura en ocho módulos de estudio, cada uno de ellos organizado en seis apartados: contenidos teóricos, resumen, actividades didácticas, ejercicios de autoevaluación (con solucionario), glosario y bibliografía. Los ocho temas desarrollados, junto a sus

respectivos autores y autoras, son los siguientes: correctores ortográficos, gramaticales y estilísticos (Xavier Gómez Guinovart); procesamiento de corpus lingüísticos (Joaquim Rafel y Joan Soler); hipertextos (Joan Campàs); traducción automática (Juan Alberto Alonso); interfaces en lenguaje natural (Horacio Rodríguez); recuperación y extracción de información (Julio Gonzalo y Felisa Verdejo); técnicas de representación y procesamiento del lenguaje (Toni Badia); y tecnologías del habla (Joaquim Llisterri). La publicación de la versión en castellano de esta obra se halla en fase avanzada de preparación y está prevista para fechas próximas.

Sección técnica «*Seguridad*» (Javier Areitio Bertolín)

Aiwald, E. *Network Security: A Beginner's Guide*. McGraw-Hill Professional. 2001.
Anderson, R.J. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons. 2001.
Chirillo, J. *Hack Attacks Revealed: A Complete Reference with Custom Security Hacking Toolkit*. John Wiley & Sons Inc. 2001.
Cole, E. and Riley, J. *Hackers Beware: Defending Your Network from the Wiley Hacker*. New Riders Publishing. 2001.
Hatch, B., Kurtz, G. and Lee, J. *Hacking Linux Exposed*. McGraw-Hill Professional Publish. 2001.
Kaaranem, H. and Niemi, V. *UMTS Networks: Architectures Mobility and Services*. John Wiley & Sons. 2001.
Mandia, K. and Prosise, C. *Incident Response: Investigating Computer Crime*. McGraw-Hill Professional. 2001.
McCarthy, M.P. and Campbell, S. *Security Transformation: Digital Defense Strategies to Protect Your Company's Reputation and Market Share*. McGraw-Hill Professional. 2001.
Northcutt, S., Cooper, M., Fearnow, M. and Frederick, K. *Intrusion Signatures and Analysis*. New Riders Publishing. 2001.
Scambray, J. and McClure, S. *Hacking Exposed Windows 2000: Network Security Secrets and Solutions*. McGraw-Hill Professional. 2001.
Schneier, B. and Spitzner, L. *Know Your Enemy: Revealing the Security Tools, Tactics and Motives of the Black-Hat Community*. Addison-Wesley Longman Inc. 2001.
Skouds, E. and Perlman, R. *Counter Hack: A Step-by-Step Guide to Computer Attacks and Effective Defenses*. Prentice-Hall PTR. 2001.
Tipton, H.F. and Krause, M. «Information Security Management Handbook». Vol 3. *CRC Press*. 2001.
Tudor, J.K. «Information Security Architecture: An Integrate Approach to Security in the Organization». *CRC Press*. 2000.
Welch, D. *Essential Check Point Firewall-1: An Installation, Configuration and Troubleshooting Guide*. Addison-Wesley Longman Inc. 2001.

Sección técnica «*Tecnología de Objetos*» (Esperanza Marcos, Gustavo Rossi)

Tema: Programación Orientada a Aspectos

En los últimos años ha existido un debate creciente acerca de la imposibilidad de alcanzar mayores grados de modularidad en orientación a objetos, particularmente en aspectos vinculados a separación de *concerns* de diseño. Por ejemplo, si construimos una aplicación que requiere chequeos de seguridad, o satisfacción de ciertas restricciones de tiempo real, el código que realiza estos chequeos o satisfacciones suele estar «mezclado» con el código de la aplicación. Para resolver este y otros problemas semejantes surgió hacia fines de los 90 un nuevo paradigma de programación, conocido como **programación orientada a aspectos**, donde el desarrollador se concentra en aspectos ortogonales por separado; los módulos que implementan dichos aspectos luego se integran en el software (usando un mecanismo conocido como *weaving*).

Para quien esté interesado en conocer el estado del arte de esta nueva área que extiende las fronteras de la orientación a objetos, recomendamos diversas fuentes: por un lado el último número de la revista *Communications of the ACM* dedicado especialmente al tema pero fundamentalmente en el sitio web de *Aspect Oriented Software Development* en <http://www.aosd.net>.

Este sitio es muy interesante pues contiene «todo lo que Vd. quiso saber y nunca se animó a preguntar» sobre orientación a aspectos. El sitio está muy bien organizado, es simple y tiene acceso al material de todos los *workshops* sobre orientación a aspectos desde 1997. De paso, informar de que la primer conferencia internacional sobre el tema se realizará en Holanda en 2002.

El sitio y las discusiones que allí se encuentran son altamente recomendables para todo aquel desarrollador de software que sienta que aun puede construirse software de mayor calidad sin necesidad de mezclar «aspectos» diferentes en el mismo módulo.

Sección Técnica «*TIC para la Sanidad*» (Valentín Masero Vargas)

Tema: Libros de lectura recomendada y sitios web de interés

Nerlich, M. Y Kretschmer, R. *The impact of telemedicine on health care management.* 1999, 290 Págs. Precio aprox.: 61.000 Ptas. Este libro ofrece información sobre aspectos legales, desarrollo y evaluación de aplicaciones de Telemedicina. Presenta también numerosos proyectos que cubren los campos clínicos de la medicina de urgencias, cirugía, oncología, cardiología, endocrinología, oftalmología, radiología y otras especialidades clínicas. Por último, proporciona una selección de enlaces a los sitios web más relevantes en Internet.

L.M. Friedman, C.D. Furberg, D.L. DeMets. *Fundamentals of Clinical Trials* (3rd Edition). Springer Verlag, New York, 1999. Los ensayos clínicos constituyen uno de los mejores métodos científicos estándares para la evaluación de pruebas diagnósticas, procedimentales y de dispositivos electrónicos para la medicina. Esta referencia clásica, adaptada ahora con los resultados y las aplicaciones informáticas más novedosas, es una referencia muy útil para aquellos investigadores que deban desarrollar aplicaciones informáticas para ensayos clínicos o para automatizar la evaluación de los resultados de dichos ensayos.

Aplicaciones TIC para la medicina gratuitas y disponibles en la red:
<http://www.connmed.com.ar/doctorsoft.com.ar/>
<http://www.medal.org.ar/>

Programar es crear

Ancho de banda en Internet

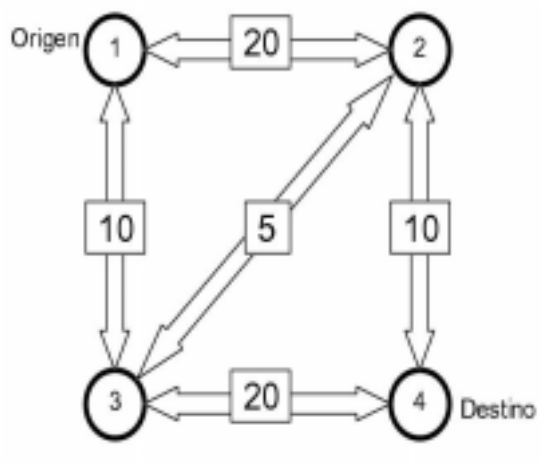
Traducción: Juan Céspedes

<cespedes@acm.org>

Éste es el programa E de los planteados en el 24º Concurso Internacional de Programación ACM (2000)

En Internet existen muchas conexiones entre distintos ordenadores (nodos) y puede haber muchos caminos distintos ntre un determinado par de nodos. La capacidad total para el envío de mensajes (ancho de banda) entre dos nodos es la máxima cantidad de datos por unidad de tiempo que se puede transmitir de un nodo a otro. Usando una técnica llamada conmutación de paquetes, estos datos se pueden transmitir usando varios caminos diferentes al mismo tiempo.

Por ejemplo, la siguiente figura muestra una red con cuatro nodos (mostrados como círculos) con un total de cinco conexiones entre ellos. Cada una de las conexiones está etiquetada con su ancho de banda, que representa la cantidad máxima de datos por unidad de tiempo que se pueden enviar por ese enlace.



En nuestro ejemplo, el ancho de banda entre el nodo 1 y el nodo 4 es 25, que se puede expresar como la suma de los anchos de banda 10 a través del camino 1-2-4, 10 por el camino 1-3-4, y 5 por el 1-2-3-4. Ninguna otra combinación de caminos entre los nodos 1 y 4 da como resultado un ancho de banda total mayor a éste.

Tu tarea es escribir un programa que calcule el ancho de banda entre dos determinados nodos de una red, teniendo en cuenta los anchos de banda de todas las conexiones en esta red. En este problema se puede Dar por supuesto que el ancho de banda de una conexión es siempre el mismo en ambos sentidos (lo que no es necesariamente cierto en el mundo real).

Entrada

La entrada contiene descripciones de distintas redes. Cada descripción comienza con una línea que contiene un entero n (2 <= n <= 100), que indica el número de nodos en la red. Los nodos están numerados desde 1 hasta n. La siguiente línea contiene tres enteros s, t y c. Los números s y t indican los nodos de origen y destino, y c es el número total de conexiones en la red. Seguidamente hay c líneas que describen las conexiones. Cada una de estas líneas contiene tres enteros: los dos primeros son los números de los nodos conectados y el tercero es el ancho de banda de la conexión. El ancho de banda es un número entre 0 y 1.000.

Puede haber más de una conexión entre un par de nodos, pero un nodo no puede estar conectado consigo mismo. Todas las conexiones son bidireccionales, esto es, los datos pueden transmitirse en ambos sentidos de una conexión, pero la suma de los datos transmitidos en ambos sentidos no puede ser mayor que el ancho de banda.

Después de la última descripción de red habrá una línea con el número 0, que indica el final de la entrada.

Salida

Por cada una de las descripciones de red se ha de escribir el número de la red en una línea, seguida de otra línea con el ancho de banda total entre los nodos s y t, tal como se indica en el ejemplo. Después de cada red se ha de escribir una línea en blanco.

Ejemplo de entrada

```
4
1 4 5
1 2 20
1 3 10
2 3 5
2 4 10
3 4 20
0
```

Salida del ejemplo de entrada

```
Network 1
The bandwidth is 25.
```

La solución comentada de este problema la encontrarán en el próximo número de Novática.

Programar es crear

«Fila y asociados»: solución

Álvaro Martínez Echevarría

<ame@acm.org>

El enunciado de este problema apareció en el número 153 de Novática (septiembre-octubre 2001, p. 70). Es el programa G de los planteados en el 24º Concurso Internacional de Programación de la ACM (2000)

La solución es muy sencilla: simplemente una simulación. Puesto que los parámetros propuestos son razonablemente pequeños (un cuanto de tiempo de un minuto durante un día, con un máximo de 20 temas y 5 personas), podemos permitirnos la fuerza bruta y avanzar de minuto en minuto. El código es algo aparatoso, como en toda simulación, pero el concepto es muy sencillo: describir el estado del sistema en cada momento e ir incrementando el tiempo en minutos.

```
#include  <stdio.h>
#include  <stdlib.h>
#include  <string.h>
#include  <vector>
#include  <hash_map>

struct  topic  {
      /*  Peticiones  que  aún  quedan  por  llegar  */
      int  remaining;
      /*  Peticiones  a  la  espera  de  atenderse  */
      int  waiting;
      /*  Tiempo  de  llegada  de  la  próxima  petición  */
     int  next;
      /*  Tiempo  de  servicio  */
      int  duration;
      /*  Tiempo  entre  peticiones  */
      int  interval;
} ;

struct  person  {
      /*  Identificación  */
     int  id;
      /*  Temas  bajo  su  responsabilidad  */
      vector<int>  topics;
      /*  Tema  del  que  se  ocupa  en  este  momento  */
     int  now;
      /*  Tiempo  de  la  última  asignación  */
     int  last;
      /*  Tiempo  a  partir  del  cual  queda  disponible  */
      int  available;
} ;


struct  state  {
     int  t;
       hash_map<int,topic>  requests;
       vector<person>  personnel;

      /*  Peticiones  totales  que  quedan  */
      int  pending;
      /*  Número  de  personas  ocupadas  */
      int  busy;
} ;


void
simulate(state  *st)  {
       vector<int>  available;

     /*
        *  Inicialmente  todos  están  disponibles.
```

```
 */
  for (unsigned int i=0;i<st->personnel.size();i++) {
      available.push_back(i);
}

 while (1) {

     /*
       * Recibir las peticiones.
      */
      if (st->pending>0) {
          for (hash_map<int,topic>::iterator it=st->requests.begin();it!=st->requests.end();it++) {
                if (it->second.remaining>0 && st->t==it->second.next) {
                   it->second.remaining--;
                   it->second.waiting++;
                    it->second.next+=it->second.interval;
             }
         }
     }

     /*
       * Liberar y ordenar el personal.
      */
        for (unsigned int i=0;i<st->personnel.size();i++) {
             if (st->personnel[i].now>=0 && st->t==st->personnel[i].available) {
               st->personnel[i].now=-1;
              st->busy--;

             /*
                  * Insertamos 10*(1+última_asignación)+(orden), para que
                 * al ordenar el vector el resultado sea el que queremos;
                 * es decir, que el personal quede ordenado por:
                 * (1) petición anterior menos reciente
                * (2) identificador
              */
                  available.push_back(i+10*(st->personnel[i].last+1));
         }
     }
         sort(available.begin(),available.end());

     /*
        * Asignar las peticiones entre las personas disponibles.
      */
      if (st->pending>0) {
            hash_map<int,topic>::iterator topic;
              for (vector<int>::iterator it=available.begin();it<available.end();it++) {
               int who=(*it)%10;
                  for (unsigned int i=0;i<st->personnel[who].topics.size();i++) {
                  /*
                      * ¿Existe ese tema? ¿Hay alguna pregunta esperando?
                    * Si es así, asignarla.
                   */
                      if ((topic=st->requests.find(st->personnel[who].topics[i]))==
                     st->requests.end() || topic->second.waiting==0) {
                        continue;
                  }
                     st->personnel[who].last=st->t;
                        st->personnel[who].available=st->t+topic->second.duration;
                      st->personnel[who].now=topic->first;
                   st->busy++;
                   st->pending--;
                    topic->second.waiting--;
                    available.erase(it);
                  break;
             }
         }
     }

       if (st->pending==0 && st->busy==0) {
          break;
```

```
        }

         st->t++;

    }

}



int
main(int argc, char *argv[]) {
     int scenario=1,a,b;
     state *st=new state();

    while (1) {
         if (scanf("%d",&a)!=1 || a==0) {
            break;
        } else if (a<0) {
            abort();
        }

         st->t=0;
         st->busy=0;
         st->pending=0;

          st->requests.clear();
         for (int i=0;i<a;i++) {
              if (scanf("%d",&b)!=1) {
                 abort();
            }
               st->requests[b].waiting=0;
              if (scanf("%d %d %d %d",
                          &st->requests[b].remaining,
                          &st->requests[b].next,
                          &st->requests[b].duration,
                           &st->requests[b].interval)!=4) {
                 abort();
            }
                st->pending+=st->requests[b].remaining;
        }

          if (scanf("%d",&a)!=1 || a<=0) {
             abort();
        }

           st->personnel.resize(a);
          for (int i=0;i<a;i++) {
                if (scanf("%d %d",&st->personnel[i].id,&b)!=2) {
                  abort();
            }
               st->personnel[i].topics.resize(b);
              for (int j=0;j<b;j++) {
                     if (scanf("%d",&st->personnel[i].topics[j])!=1) {
                       abort();
                    }
            }
               st->personnel[i].last=0;
                st->personnel[i].available=0;
               st->personnel[i].now=-1;
        }

          simulate(st);
            printf("Scenario %d: All requests are serviced within %d minutes.\n",scenario++,st->t);

    }

     exit(0);

}
```

Coordinación editorial

Edición digital de *Novática* en la Intranet de ATI

Recordamos a los socios de ATI que las páginas de ***Novática*** en la IntrATInet, <http://intranet.ati.es/novatica/>, siguen actualizándose con las versiones digitales completas de nuestra revista, en formato PDF. Actualmente contiene todos los números publicados desde principios del año 2000 en adelante.

Recordamos también que para acceder a la IntrATInet es necesario estar abonado al servicio ATInet y contar con el correspondiente usuario y contraseña. Los socios que aún no estén abonados a ATInet pueden informarse sobre cómo hacerlo visitando <http://www.ati.es/ATInet/> o dirigiéndose a la Secretaría de ATI <secregen@ati.es>

Reunión del Grupo de Trabajo de *Upgrade*

El pasado día 8 de noviembre, con ocasión de la celebración del Consejo de **CEPIS** (*Council of European Professional Informatics Societies*), se celebró en Zürich una reunión del Grupo de Trabajo de ***Upgrade***, la revista digital de CEPIS, proyecto del que actualmente forman parte ***Informatik/ Informatique***, revista bimestral de **SVI/FSI** (Suiza), y ***Novática***. A la reunión asistieron Wolffried Stucky y Walter Grafendorfer, Presidente y Tesorero de CEPIS, respectivamente, así como los directores de dichas revistas, François Louis Nicolet y Rafael Fernández Calvo. Durante la primera parte de la reunión estuvo también presente Hermann Engesser, editor ejecutivo del área de Informática de la importante editorial alemana Springer-Verlag, que mostró su interés en conocer este proyecto.

Durante la reunión se analizó tanto lo conseguido en el primer año de vida de la revista, como las insuficiencias de ésta. En el capítulo de logros se destacó que desde octubre del año 2000 ***Upgrade***, con una inversión económica mínima por parte de CEPIS, ha aparecido puntualmente cada dos meses en el sitio web <http://www.upgrade-cepis.org>, según lo programado, que el nivel de calidad de lo publicado es muy satisfactorio y que las revistas de algunas sociedades miembro de CEPIS, como la irlandesa **ICS** y la austriaca **OCG**, han reproducido artículos publicados en ***Upgrade***.

Se señaló como principal insuficiencia la falta de incorporación efectiva al proyecto de las revistas de otras sociedades miembro de CEPIS, en buena parte debida a que ***Upgrade*** no se ha difundido adecuadamente entre ellas, lo que ha llevado también a que el número de accesos al sitio web que alberga la revista no haya alcanzado la magnitud deseada.

Para superar ese importante problema se acordó proponer al Comité Ejecutivo de CEPIS la adopción de medidas que popularicen ***Upgrade*** tanto entre dichas sociedades miembro como entre los profesionales informáticos europeos. Dichas medidas son las siguientes: informar de forma periódica sobre ***Upgrade*** a las sociedades miembro, ofrecer de forma gratuita durante 2002 a todas las revistas de las sociedades miembro de CEPIS el derecho a la reproducción gratuita de los artículos que aparecen en ***Upgrade***, crear una lista de distribución de los directores de las revistas de dichas sociedades miembro y estudiar la posibilidad de realizar peticiones públicas de artículos para las monografías que se publiquen en 2002.

Programación de Novática

Monografías para el año 2002

Las monografías que, durante el año 2002, publicarán ***Novática***, ***Informatik/Informatique*** y ***Upgrade***, serán las siguientes, salvo causas de fuerza mayor:

Enero-febrero (nº 155)
Monografía: «**Gestión del Conocimiento y TIC - *Knowledge Management and Information Technology***».
Fecha de publicación: segunda quincena de febrero
Coordinador: Prof. Xavier Alamán (Universidad Autónoma de Madrid), Prof. Christopher Lueg (University of Technology, Sydney, Australia).

Marzo-abril (nº 156)
Monografía: «***eXtreme Programming***».
Fecha de publicación: segunda quincena de abril
Coordinador: Prof. Luis Fernández Sanz (Universidad Europea CEES, Madrid).

Mayo-junio (nº 157)
Monografía: «**Recuperación de la información y la Web - *Information Retrieval and the Web***».
Fecha de publicación: segunda quincena de junio
Coordinadores: Prof. Ricardo Baeza Yates (Universidad de Chile), Prof. Peter Schauble (Eurospider, Zürich, Suiza).

Julio-agosto (nº 158)
Monografía: «***XML, eXtended Mark-up Language***».
Fecha de publicación: primera quincena de septiembre
Coordinador: Prof. Luis Sánchez Fernández (Universidad Carlos III, Madrid)

Septiembre-octubre (nº 159)
Monografía: «**Inteligencia Artificial - *Artificial Intelligence***».
Fecha de publicación: segunda quincena de octubre
Coordinador: Prof. Federico Barber (Universidad Politécnica de Valencia), Prof. Vicente Botti (Universidad Politécnica de Valencia)

Noviembre-diciembre (nº 160)
Monografía: «**Seguridad en Comercio/Negocio Electrónico - *Security in E-Commerce/Business***».
Fecha de publicación: segunda quincena de diciembre
Coordinadores: Prof. Javier Areitio (Redes y Sistemas), Prof. Javier López (Universidad de Málaga), Prof. José A. Mañas (ETSIT-UPM), con la colaboración del Grupo de Trabajo de Seguridad de ATI.

Normas de publicación para autores

Novática agradece su contribución desinteresada a los miles de autores que han elegido y elegirán sus páginas para presentar sus aportaciones al avance profesional y tecnológico de la Informática.

Periodicidad: Novática tiene periodicidad bimestral y aparece los meses de Febrero, Abril, Junio, Septiembre, Octubre y Diciembre, salvo retrasos debidos a causas de fuerza mayor. El cierre de la edición es habitualmente un mes antes de la fecha de distribución (dos meses para los artículos del bloque monográfico).

Normas de revisión: Todos los artículos serán sometidos a revisión excepto los expresamente solicitados por *Novática* a sus autores. En el caso de las monografías serán los Coordinadores de éstas los que realicen la revisión y decidan sobre su publicación o no.

Excepto en el caso de las monografías, los artículos deberán ser enviados a la oficina de Coordinación Editorial (Novática-ATI. Calle Padilla 66, 3º dcha., 28006 Madrid, <*novatica@ati.es*> (ver **Soporte** más abajo). Una vez aprobados por el revisor(es), serán publicados tan pronto como sea posible, si bien la publicación no está garantizada pues razones de exceso de material pueden hacerla imposible. Los autores serán informados del resultado de la revisión y de la publicación o no de los artículos remitidos.

Tamaño y formato de los artículos: Los artículos deberán tener un máximo de 3.000 palabras, lo que equivale a entre 8 y 10 páginas DIN A4 a doble espacio (fuente Times, tamaño 12), incluyendo resumen, figuras, bibliografía y notas. Sólo en casos excepcionales se aceptarán artículos superiores a dicho tamaño. Salvo excepciones, los artículos no deberán incluir más de cinco ecuaciones ni más de doce referencias bibliográficas o notas, y deberán incorporar (en español e inglés) título, resumen (máximo 20 líneas), nombre y afiliación del autor/a (es/as), así como su dirección postal y electrónica, y números de teléfono y fax. **Nota importante**: título, resumen y palabras clave deberán enviarse en español e inglés.

Soportes: Los artículos deberán ser enviados a Novática en formato digital, bien a través de la red bien en soporte magnético (disquete) mediante correo postal. En caso de envío por correo electrónico, si el fichero tiene un tamaño superior a los 150.000 bytes, es preciso enviar el fichero comprimido con el programa xZIP e indicando qué procesador de texto entre los citados a continuación se ha utilizado. En ambos casos (correo electrónico o disquete) el artículo debe llegar en uno de los procesadores de texto más habituales para PC (Word, Word Perfect, Write, etc.), en HTML, RTF o, en último caso, en PDF o ASCII si se teme una difícil lectura o no se dispone de un procesador de texto estándar.

También en ambos casos (correo electrónico o disquete) es preciso enviar una edición completa del artículo en papel para el cotejo de texto y gráficos. Estos se admitirán solamente en blanco y negro y con una buena resolución; además cada uno de los gráficos deberá enviarse en hoja aparte, tamaño DIN A4.

Lengua: Aunque Novática admite artículos escritos en todas las lenguas reconocidas por la Constitución española y los Estatutos de las diferentes Comunidades Autonómas, dado que el ámbito de difusión de la revista conlleva su publicación en castellano, como lengua oficial común, los autores deberán presentar sus artículos en castellano y, si así lo desean, en otra lengua oficial de su elección. Novática enviará a los socios y suscriptores que lo soliciten una copia de la versión original de aquellos artículos que hayan sido escritos en una lengua oficial que no sea el castellano.

Copyright: Novática da por supuesto que un autor acepta las presentes normas al enviar su original y que, en caso de que esté destinado a ser publicado en otro medio ajeno a ATI (o ya haya sido publicado) debe de aportar la autorización del editor del mismo para su reproducción por Novática (incluida la autorización para realizar traducciones). Novática por tanto no asume ninguna responsabilidad sobre derechos de propiedad intelectual si un texto se ha publicado en otro medio de comunicación, sea inadvertidamente o no, por parte del autor. Todo autor que publique un artículo en Novática debe saber que autoriza su reproducción, citando la procedencia, salvo que el autor declare explícitamente que desea proteger sus derechos con © o *copyright*. Asimismo, se entiende que el autor acepta que, además de en Novática, su artículo podrá ser también publicado y distribuido electrónicamente, mediante los medios habituales de difusión de ATI (servidor WWW, listas de distribución Internet, etc.) en su totalidad o parcialmente.

Estilo: Si bien *Novática* respeta totalmente el estilo y contenido de cada artículo, da por supuesta la autorización del autor para retocar su ortografía, léxico, sintaxis, titulación y paginación, a fin de facilitar su comprensión por el lector y de subsanar posibles errores. Cualquier cambio que afecte al contenido será consultado con el autor.

SOCIOS INSTITUCIONALES DE ATI

Según los Estatutos de ATI, pueden ser socios institucionales de nuestra asociación «las personas jurídicas, públicas y privadas, que lo soliciten a la Junta Directiva General y sean aceptados como tales por la misma».

Mediante esta figura, todos los profesionales y directivos informáticos de los socios institucionales pueden gozar de los beneficios de participar en las actividades de ATI, en especial congresos, jornadas, cursos, conferencias, charlas, etc. Asimismo los socios institucionales pueden acceder en condiciones especiales a servicios ofrecidos por la asociación tales como Bolsa de Trabajo, cursos a medida, *mailings*, publicidad en Novática, servicio ATInet, etc.

Para más información dirigirse a <*info@ati.es*> o a cualquiera de las sedes de ATI.

En la actualidad son socios institucionales de ATI las siguientes empresas y entidades:

AGBAR GLOBAL MARKET, S.A. (GRUPO AGM)
AIGÜES DEL TER LLOBREGAT
AIS - Aplicaciones de Inteligencia Artificial
ALEPH SOFTWARE, S.A.
ALMIRALL PRODESFARMA, S.A.
ARCISA, S.A.
ATOS ODS, S.A.
AUBAY SDS
BARCELONESA DE DROGAS DE PRODUCTOS QUÍMICOS
BITWARE, S. L.
BBR INGENIERIA DE SERVICIOS, S.L.
BRAY'S Idiomas y Nuevas Tecnologías
BT TELECOMUNICACIONES, S.A.
CCS PROFESIONALES, S.L.
CESISA
CINDOC (Centro de Información y Documentación Científica)
CLASE 10 SISTEMAS, S.L.
CLINICA PLATÓ FUNDACION PRIVADA
CONSULTORES SAYMA, S.A.
CONSEJO GENERAL DEL NOTARIADO
COOPERS & LYBRAND AUDITORÍA Y CONSULTORÍA
DEBIS - Centre Informàtic General de Catalunya, S.A.
DISTRIBUCIÓN QUELLE LA SOURCE, S. A.
DOXA CONSULTORES, S.L.
ECONOCOM, S.A.
EDS España, S.A.
EPISER, S.L.
ESPECIALIDADES ELÉCTRICAS S.A. (ESPELSA)
ESRI ESPAÑA GEOSISTEMAS, S.A.
ESTEVE QUÍMICA, S.A.
FUNDACIÓN SAN VALERO
GESTEVISIÓN TELECINCO
GETRONICS GRUPO CP,S.L.
GRUPO INFORMÁTICO ITEM, S.A.
GS y C, Gabinete Sistemas y Consultoría, S.L.
INFORMATION BUILDERS IBÉRICA, S.A.
INSERT SISTEMAS, S. A.
INSTITUT D'ESTUDIS CATALANS
INVERAMA, S.A.
IN2 Ingeniería de la Información IRATI Consulting, S.L.L.
I.S.C. Ingeniería de Sistemas y Comunicaciones
J.J. SOFTWARE DE MEDICINA
JUNTA DE CASTILLA-LA MANCHA (Consejería de Administraciones Públicas)
LABORATORIOS SERONO, S.A.
M. SOFT, S.A.
META 4 SPAIN, S.A.
NEKKAR HEYDE ESPAÑA, S.A.
NORSISTEMAS, S.A.
OCCIDENTAL HOTELES MANAGEMENT, S.A.
RÁPIDA SISTEMAS INTEGRALES, S.A.
RD SISTEMAS, S.A.
SADIEL, S.A.
SARA LEE DE ESPAÑA, S.A.
SAS INSTITUTE, S.A.
SEMA GROUP, SAE
S.G. SOFTWARE DE GESTIÓN, S.L.
SYSDATA, S.L.
TATUM SISTEMAS
TCP SISTEMAS DE INGENIERÍA, S.L.
TRANSBAIX LLOBREGAT, S. A. (Grupo Seur)
TRW ISCS, S.L.
UNIVERSIDAD DE EXTREMADURA (Dpto. de Informática)
UNIVERSITAT OBERTA DE CATALUNYA
WAPETON NUEVAS TECNOLOGÍAS, S.A.
ZUGARTO EDICIONES, S.A.
3D DESIGN BOARD, S.L.
3M España, S.A.

novática

Revista de la Asociación de Técnicos de Informática

HOJA DE SUSCRIPCIÓN
Rellene esta hoja y envíela a:
Novática (Suscripciones)
Vía Laietana 41, lº, 1ª
08003 Barcelona, España
Tlfno.: 93 412 52 35 Fax: 93 412 77 13
Correo elec.: <*novatica@ati.es*>

Apellidos .... Nombre ....
Empresa/Organismo .... CIF/NIF ....
Domicilio ....
Ciudad .... Provincia ....
Código Postal .... País ....
Teléfono .... Fax .... Correo elec. ....

Nota: Rellenar los siguientes datos solamente si la dirección de envío es diferente de la anterior.
Domicilio para envíos ....
Ciudad .... Provincia ....
Código Postal .... País ....

Deseo suscribirme a Novática (6 numeros al año) en las siguientes condiciones (marcar con X la opción deseada y, en su caso, la cantidad de suscripciones solicitadas):

España
- ❒ 1 suscripción: 54,10 € (9.000 pts.) (+4% IVA)
- ❒ _ suscripciones: 49,60 € (8.250 pts.) cada una (+4% IVA)

Otros países de la Unión Europea y Marruecos
- ❒ 1 suscripción: 66,11 € (11.000pts.)
- ❒ _ suscripciones: 63,10 € (10.500 pts.) cada una

Resto del mundo
- ❒ 1 suscripción: 95 dolares USA
- ❒ _ suscripciones: 90 dolares USA cada una

Abonaré el importe:
- ❒ Con domiciliación de cobro por entidad bancaria (deberá rellenar los datos bancarios abajo solicitados)
- ❒ Talón adjunto
- ❒ Transferencia bancaria a la cta. 3025-0004-30-1500001500, Caja de Ingenieros, Calle Buen Pastor 5, 08018, Barcelona, (España)

Fecha .... Firma

DATOS BANCARIOS PARA DOMICILIACIÓN

Banco/Caja...

CÓDIGO CUENTA CLIENTE			
ENTIDAD	OFICINA	D.C.	NÚMERO DE CUENTA

nv 154

✂- -

AUTORIZACIÓN DE COBRO
Le rogamos repita los datos bancarios otra vez. ATI se encarga de su envío al Banco o Caja.

Banco/Caja...

CÓDIGO CUENTA CLIENTE			
ENTIDAD	OFICINA	D.C.	NÚMERO DE CUENTA

Ruego a Uds. se sirvan tomar nota de que, hasta nueva orden mía en contra, deberán adeudar en mi cuenta arriba indicada los recibos que a nombre de D./Dª... le sean presentados por la Asociación de Técnicos de Informática (ATI), en concepto de suscripción a la revista Novática.

Fecha .... Firma

Una Asociación abierta a todos los informáticos

Una Asociación útil a sus socios, útil a la Sociedad

¿Qué es ATI?

✓ ATI es una asociación *abierta a todos los técnicos y profesionales informáticos y que está implantada en todo el país a través de los Capítulos Territoriales existentes en diversas Comunidades Autónomas.* Creada en 1967, es en la actualidad la asociación más dinámica y más numerosa (5.000 socios a finales de 2000) de las existentes en el Sector Informático español, con sedes en Barcelona (sede general), Madrid, Sevilla, Silleda (Pontevedra), Valencia y Zaragoza.

✓ ATI es miembro de *CEPIS (Council for European Professional Informatic Societies)* y tiene un acuerdo de colaboración con *ACM (Association for Computing Machinery)*. En el plano interno tiene establecidos acuerdos de colaboración o vinculación con Ada Spain, AII y ASTIC.

¿Cuales son los objetivos de ATI?

Se resumen en uno esencialmente:

SER UTIL A SUS SOCIOS Y A LA SOCIEDAD

Más concretamente, ATI se propone:

✓Defender, promover y mejorar el desarrollo de la actividad de quienes ejercen como profesionales y técnicos en el campo de las Tecnologías de la Información.
✓Facilitar a sus socios el intercambio de experiencias, la formación y la información sobre dichas tecnologías.
✓Contribuir a la promoción y desarrollo de las Tecnologías de la Información.
✓Mantener relaciones con el entorno social y económico en que la Asociación se mueve.
✓Fomentar la difusión de las Tecnologías de la Información y estudiar su impacto sobre la sociedad y sobre los ciudadanos.
✓Colaborar con otras entidades profesionales informáticas implantadas tanto en nuestro país como fuera de él, especialmente en Europa y en la América Latina.

¿Cómo está organizada ATI?

✓La *Asamblea General* de socios y la *Junta Directiva General* son los órganos máximos de dirección para el conjunto de la asociación.
✓Los *Capítulos Territoriales*, con sus *Asambleas Territoriales* y sus J*untas Directivas Territoriales*, estructuran la asociación en las Comunidades Autónomas mediante una organización de orientación federal.
✓Las *Secciones Técnicas* y los *Grupos de Trabajo* sobre diversos temas facilitan la participación de los socios en las actividades de la Asociación.

¿Qué ofrece ATI?

Mediante el pago de una cuota anual, los socios de ATI pueden disfrutar de la siguiente gama de servicios:

✓ *Formación Permanente*
- Cursos, Jornadas Técnicas, Mesas Redondas, Seminarios, Conferencias, Congresos
- Secciones Técnicas y Grupos de Trabajo sobre diversos temas
- Intercambios con Asociaciones Profesionales de todo el mundo

✓ *Servicios Profesionales*
- Asesoramiento profesional y legal
- Peritajes, diagnósticos y certificaciones
- Bolsa de Trabajo
- Emisión en España del ECDL (European Computer Driving License)

✓ *Servicios de Información*
- Revista Novática (bimestral)
- Boletín Informativo ATI Informa
- Servidor Web
- Red asociativa ATInet (acceso básico gratuito a Internet, correo electrónico, IntrATInet, listas de distribución generales y especializadas, dirección permanente)
- Biblioteca

✓ *Actividades Sociales*
- Promociones y ofertas comerciales
- Intercambios internacionales

Juntas Directivas

Junta Directiva General

Presidente: Josep Molas i Bertrán.
Vicepresidente Primero: Fernando Piera Gómez.
Vicepresidente Segundo: Celestino Martín Alonso.
Secretario: Miquel Sàrries Griñó.
Interventor-Tesorero: Antoni Carbonell Nogueras.
Vocales: Asunción Yturbe Herranz, Gloria Nistal Rosique, Julián Marcelo Cocho, Didac López Viñas, Roberto Moya Quiles, César Pérez Chirinos.
Suplentes: Francisco López Crespo, Carmen Ugarte García, Mario Piattini Velthuis.

Capítulos Territoriales (Presidentes)

Andalucía (Fernando Sanjuán de la Rocha); Aragón (Manuel Solans); Catalunya (Pere Lluís Barbarà Butifull); Galicia (José Gómez); Madrid (Mª Carmen Ugarte García); Valencia (Manuel Ortí)

¿Dónde está ATI?

Servidor Web: *http://www.ati.es*

Sede General y Capítulo de Catalunya
Via Laietana 41, 1º, 1ª, 08003 Barcelona
Tlf. 93 4125235; fax 93 4127713 / <*secregen@ati.es*>
Capítulo de Andalucía
Isaac Newton, s/n, Ed. Sadiel (Isla Cartuja), 41092 Sevilla
Tlf./fax 95 4460779 / <*secreand@ati.es*>
Capítulo de Aragón
Lagasca 9, 3-B, 50006 Zaragoza
Tlf./fax 976 235181 / <*secreara@ati.es*>
Capítulo de Galicia
Recinto Ferial s/n, 36540 Silleda (Pontevedra)
Tlf.986 581413; fax 986 580162 / <*secregal@ati.es*>
Capítulo de Madrid
Padilla 66, 3º, dcha., 28006 Madrid
Tlf. 91 4029391; fax. 91 3093685 / <*secremdr@ati.es*>
Capítulo de Valencia
Palomino 14, 2ª, 46003 Valencia
Tlf./fax 96 3918531 / <*secreval@ati.es*>
Grupo Promotor Asturias-Cantabria <*gp-astucant@ati.es*>
Grupo Promotor Castilla-La Mancha <*gp-clmancha@ati.es*>
Revista Novática
Padilla 66, 3º, dcha., 28006 Madrid
Tlf.91 4029391; fax. 91 3093685 / <*novatica@ati.es*>

¿ Quién puede ser socio de ATI ?

- ***Socios de número***
  - Deben acreditar un mínimo de tres años de experiencia profesional informática, o dos años, si se posee un título de grado superior o medio, ***o bien,***
  - poseer un título de grado superior o medio relacionado con las Tecnologías de Información, ***o bien,***
  - haber desarrollado estudios, trabajos, o investigaciones relevantes sobre dichas tecnologías.

- ***Socios Estudiantes***
  - Deben acreditar estar matriculados en un Centro Docente cuya titulación dé acceso a la condición de socio de número.

- ***Socios Adheridos***
  - Profesionales informáticos que no cumplan las condiciones para ser Socios de Número o Socios Estudiantes, o también personas que, no siendo profesionales informáticos, quieran participar en las actividades de ATI.

- ***Socios Institucionales***
  - Personas jurídicas, de carácter público o privado, que quieran participar en las actividades de ATI.

Cuota anual socio (2002):

de Número ..... 57.10 €(9.500 pts.)
Estudiante ..... 24,04 €(4.000 pts.)
Adherido ..... 48.08 €(8.000 pts.)
Institucional (consultar con Secretaría)

(Enviar a cualquiera de las oficinas de ATI indicadas en el reverso de esta hoja)

Hoja de inscripción a ATI

NV 154

Deseo sólo información sobre ATI ☐

Tipo de socio: De Número ☐ Estudiante ☐ Adherido ☐

Apellidos .....
Nombre .....
Domicilio ..... Nº ..... Piso .....
Localidad ..... Tel. .....
Provincia ..... C.P. .....
Fecha de nacimiento ..../..../.....

Empresa/Entidad .....
..... Ramo .....
Puesto actual en ella .....
Dirección ..... Nº .....
Localidad ..... Tel. .....
Provincia ..... C.P. .....
E- Mail ..... Fax .....

Firma ..../..../20....

Banco/Caja .....

CODIGO CUENTA CLIENTE			
Entidad	Oficina	D.C.	Núm. de Cuenta

Dirección ..... Nº .....
Localidad .....
Provincia ..... C.P. .....

Autorización de cobro

Le rogamos repita los datos bancarios otra vez. ATI se encarga de su envío al Banco / Caja.

Banco/Caja .....

CODIGO CUENTA CLIENTE			
Entidad	Oficina	D.C.	Núm. de Cuenta

Dirección ..... Nº .....
Localidad .....
Provincia ..... C.P. .....

Ruego a Uds. se sirvan tomar nota de que, hasta nueva orden mía en contra, deberán adeudar en mi cuenta arriba indicada los recibos que a nombre de D. .....
.....
les sean presentados por la ASOCIACIÓN DE TÉCNICOS DE INFORMÁTICA (ATI)

Firma ..../..../20....

Datos complementarios obligatorios

Títulos Superiores o Medios que posee y centros otorgantes:
.....
.....
.....

Resumen experiencias .....
.....
.....
.....

Número de años experiencia informática .....

Presentado por los socios

Apellidos y Nombre .....
Número de socio ..... Fecha ../../.....

Firma

Apellidos y Nombre .....
Número de socio ..... Fecha ../../.....

Firma

ATI envía a sus socios información procedente de otras entidades, fundamentalmente sobre productos y servicios relacionados con la informática. Si no desea recibir dicha información, indíquelo a continuación. En cualquier caso ATI cumple escrupulosamente la Ley de Datos Personales.

- Anuncios de ATI solamente: ☐
- Anuncios de ATI y otras entidades: ☐

http://www.ati.es

Asociación de Técnicos de Informática

Vía Laietana, 41, 1º, 1ª
08003 BARCELONA
Voz: 93 412 52 35
Fax: 93 412 77 10
E-mail: secregen@ati.es

c/ Padilla, 66 3º dcha
28006 MADRID
Voz: 91 402 99 91
Fax: 91 309 36 65
E-mail: secretadr@ati.es

Isaac Newton, s/n
Ed. Sadel (Isla Cartuja)
41092 SEVILLA
Voz/Fax: 95 446 07 78
E-mail: secretand@ati.es

c/ Palomino, 14, 2º
46003 VALENCIA
Voz/Fax: 96 391 85 31
E-mail: secretval@ati.es

c/ San Miguel, 2, 9ºB
50001 ZARAGOZA
Voz: 976 23 51 81
E-mail: secretara@ati.es

Recinto Ferial, s/n
08540 SALLENT (Pralormir)
Voz/Fax: 938 68 14 13
Fax: 938 68 01 62
E-mail: secregal@ati.es