novática
CEPIS UPGRADE
Revista de la Asociación de Técnicos de Informática

Novática, revista fundada en 1975 y decana de la prensa informática española, es el órgano oficial de expresión y formación continua de **ATI** (Asociación de Técnicos de Informática), organización que edita también la revista **REICIS** (Revista Española de Innovación, Calidad e Ingeniería del Software). ***Novática*** edita asimismo **UP**GRADE, revista digital de **CEPIS** (*Council of European Professional Informatics Societies*), en lengua inglesa, y es miembro fundador de **UP**ENET (**UP**GRADE ***E****uropean* ***NET****work*).

<http://www.ati.es/novatica/>
<http://www.ati.es/reicis/>
<http://www.upgrade-cepis.org/>

ATI es miembro fundador de **CEPIS** (*Council of European Professional Informatics Societies*) y es representante de España en **IFIP** (*International Federation for Information Processing*); tiene un acuerdo de colaboración con **ACM** (*Association for Computing Machinery*), así como acuerdos de vinculación o colaboración con **AdaSpain**, **AI2**, **ASTIC**, **RITSI** e **Hispalinux**, junto a la que participa en **ProInnova**.

Coordinación Editorial, Redacción Central y Redacción ATI Madrid
Padilla 66, 3º, dcha., 28006 Madrid
Tlfn.914029391; fax.913093685 <novatica@ati.es>
Composición, Edición y Redacción ATI Valencia
Av. del Reino de Valencia 23, 46005 Valencia
Tlfn./fax 963330992 <secreval@ati.es>
Administración y Redacción ATI Cataluña
Via Laietana 46, ppal. 1ª, 08003 Barcelona
Tlfn.934125235; fax 934127713 <secregen@ati.es>
Redacción ATI Andalucía
Isaac Newton, s/n, Ed. Sadiel,
Isla Cartuja 41092 Sevilla, Tlfn./fax 954460779 <secreand@ati.es>
Redacción ATI Aragón
Lagasca 9, 3-B, 50006 Zaragoza.
Tlfn./fax 976235181 <secreara@ati.es>
Redacción ATI Asturias-Cantabria <gp-astucant@ati.es>
Redacción ATI Castilla-La Mancha <gp-clmancha@ati.es>
Suscripción y Ventas <http://www.ati.es/novatica/interes.html>, ATI Cataluña, ATI Madrid
Publicidad
Padilla 66, 3º, dcha., 28006 Madrid
Tlnf.914029391; fax.913093685 <novatica@ati.es>
Imprenta: Derra S.A., Juan de Austria 66, 08005 Barcelona.
Depósito legal: B 15.154-1975 -- ISSN: 0211-2124; CODEN NOVAEC
Portada: "Salida de la habitación 101" - Concha Arias Pérez / © ATI
Diseño: Fernando Agresta / © ATI 2003

Nº 190, noviembre-diciembre 2007, año XXXIII

sumario

Monografía del próximo número: "*Gobierno de las TIC*"

Presentación. Sofware libre: innovación científica y tecnológica

Andrea Capiluppi[1], José Rafael Rodríguez Galván[2], Manuel Palomo Duarte[2], Israel Herraiz Tabernero[3]

[1]Universidad de Lincoln, Reino Unido, [2]Oficina de Software Libre de la Universidad de Cádiz (OSLUCA), [3]Universidad Rey Juan Carlos

<acapiluppi@lincoln.ac.uk>, <rafael.rodriguez@uca.es>, <manuel.palomo@uca.es>, <herraiz@gsyc.escert.urjc.es>

Editores invitados

Andrea Capiluppi es Doctor por el Politécnico de Turín (Italia). Ha sido investigador visitante en el Grupo de Sistemas y Comunicaciones de la Universidad Rey Juan Carlos (Madrid) en octubre de 2003. De enero de 2004 hasta la actualidad, ha sido investigador visitante en el Departamento de Matemáticas y Computación de la Open University (Reino Unido), trabajando junto con los doctores Juan Ramil, Neil Smith, Helen Sharp, Alvaro Faria y Sarah Beecham. Este compromiso ha sido renovado hasta diciembre de 2008. En enero de 2006, entró a formar parte de la Universidad de Lincoln como profesor senior.

José Rafael Rodríguez Galván es profesor del Departamento de Matemáticas de la Universidad de Cádiz (UCA) y, desde el año 2004, director de la OSLUCA. Dentro de ésta ha sido responsable de la organización de distintos proyectos, entre ellos las I, II y III Jornadas de Software Libre de la UCA y el FLOSSIC (Free/Libre Open Source Systems International Conference). Ha llevado a cabo distintas conferencias y ha participado en foros relacionados con el software libre y la universidad. Por otro lado, ha realizado distintos trabajos en el ámbito de la simulación numérica de ecuaciones en derivadas parciales, en el marco de la mecánica de fluidos, dentro del grupo de investigación FQM-315 de la UCA.

Manuel Palomo Duarte es Ingeniero en Informática por la Universidad de Sevilla (2001). En la actualidad trabaja como profesor del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Cádiz donde imparte asignaturas relacionadas con los sistemas operativos y el diseño de videojuegos (haciendo uso de software libre en ellas) y realiza labores de Coordinador Erasmus en la titulación de Ingeniería Técnica en Informática de Sistemas. Es miembro del grupo de investigación "Mejora del Proceso Software y Métodos Formales", donde está realizando su tesis doctoral sobre calidad de composición de composiciones de servicios web con BPEL. Desde su incorporación a la UCA ha colaborado con la OSLUCA, principalmente en la realización de las "III Jornadas de Software Libre de la Universidad de Cádiz" (JOSLUCA3) y el "I Congreso Científico Internacional de FLOSS" (FLOSSIC 2007).

Israel Herraiz Tabernero realiza estudios de doctorado en la Universidad Rey Juan Carlos. Su investigación está relacionada con la evolución de proyectos de software libre. En particular, está empleando análisis de series temporales y otras técnicas estadísticas para caracterizar y predecir la evolución de proyectos de software libre. Ha participado en diferentes proyectos financiados por el Programa Marco de la Comisión Europea (QUALOSS, FLOSSMetrics, Qualipso, CALIBRE). Además, ha colaborado también en otros proyectos financiados por empresas como Vodafone o Teléfonica. Ha participado en la redacción de manuales y documentos sobre cómo gestionar y poner en marcha proyectos de software libre. Por ejemplo, junto con Juan José Amor y Gregorio Robles escribió un manual para el Máster en Software Libre de la Universitat Oberta de Catalunya. En este momento disfruta de un contrato de Formación de Personal Investigador, concedido en 2005 por la Comunidad de Madrid para desarrollar la tesis doctoral en el estudio de proyectos de software libre. Ha sido revisor para, entre otras conferencias, la IEEE Africon 2007, y para la revista IEEE Transactions on Software Engineering. En estos momentos está coordinando el programa del Máster en Software Libre, título propio de la Universidad Rey Juan Carlos que se imparte en colaboración con la empresa Igalia y Caixa Nova, e imparte clases en diversas titulaciones en la citada Universidad.

En los últimos años hemos podido observar cómo el Software Libre ha pasado de ser un simple modelo de desarrollo de software (con todas sus implicaciones técnicas y éticas) a ser un elemento clave en las estrategias de desarrollo de empresas, instituciones, regiones e incluso países enteros. Ejemplos como el de gobierno de Brasil apoyando la implantación de software libre en el país [1][2] o la Junta de Andalucía liberando todos sus desarrollos [3][4][5] han servido para que cada vez más instituciones y foros estudien las implicaciones del modelo a largo plazo.

Entre los hitos más importantes podemos destacar el informe "*Study on the: Economic impact of open source software on innovation and the competitiveness of the Information and Communication Technologies (ICT) sector in the EU*" [6] desarrollado por la universidad de UNU-Merit para la Comisión Europea. En él se concluye que el Software Libre representa una de las mejores oportunidades que tiene el sector TIC europeo para aumentar su desarrollo económico y competitividad a escala mundial y estimular el tan de actualidad I+D+I.

En este marco centramos esta número especial sobre "Software libre: investigación y desarrollo" de ***Novática*** y **UP**GRADE, ya prácticamente una cita anual de la comunidad. Que, como viene siendo habitual en los números relacionados con el conocimiento libre, publica gran parte de su contenido bajo licencia libre.

Después de un artículo introductorio sobre el estado del arte escrito por los editores invitados, comenzamos la monografía con el artículo "*De la catedral al bazar: un estudio empírico del ciclo de vida de proyectos basados en comunidades de voluntarios*" que presenta un estudio comparativo entre las comunidades de desarrolladores de dos proyectos libres de reconocido prestigio: Wine y Arla. En concreto, se compara la cantidad de desarrolladores que han conseguido aglutinar a lo largo de su ciclo de vida. Un riguroso seguimiento de la información generada por cada proyecto (*changelogs*, cambios hechos por los desarrolladores, etc) permite afirmar que los modelos *catedral* y b*azar* tan tratados en la literatura del Software Libre no son excluyentes a lo largo del ciclo de vida de un proyecto libre. Y aunque el hecho de mantener un modelo catedral no implica el fracaso de un proyecto (porque este puede cumplir sus objetivos) sí es cierto que demuestra la pérdida de una oportunidad de aumentar la comunidad desarrolladora asociada al proyecto. Además, se demuestra que gran parte de la responsabilidad de que dicho cambio se produzca recae sobre la comunidad desarrolladora.

A continuación, uno de los artículos más destacados del pasado "Workshop on Emerging Trends in FLOSS Research and Development 2007" [7]: "*Los bienes comunes como nueva economía y lo que esto significa para la investigación*", un artículo que de una manera valiente pero sensata y justificada se adentra en el planteamiento de los cambios que en las TIC se producirían en

caso de una adopción masiva de FLOSS. Se estudian algunas consecuencias que tendría, como la drástica reducción del precio de las licencias de software o la disminución del coste de la experimentación con software. Estas consecuencias llevarían a un escenario muy interesante en el que se abrirían nuevas vías en la educación en TIC (al disponer de código fuente de software de última generación para estudiar y mejorar en clase), reinventando además el concepto de programación (migrando hacia un modelo en el que primaría la capacidad para buscar código que proporcione la funcionalidad deseada e integrarlo en un sistema antes que la habilidad para crear código propio desde cero) y reduciendo la complejidad en software y personal necesarios para desplegar sistemas a escala ultra grande (*Ultra-Large Scale Systems*), etc.

El trabajo "*Software Libre para la gestión de proyectos de investigación*", realizado por el Grupo de Sistemas y Comunicaciones de la Universidad Rey Juan Carlos nos muestra cómo la implantación de metodologías y procesos propios del desarrollo cooperativo de software libre puede ayudar al desarrollo de investigaciones. En concreto, su propuesta permite facilitar la comunicación entre miembros del proyecto que trabajen en sede y/o con horarios distintos así como la visibilidad de los informes de actividad, productos producidos y resultados intermedios que se consideren de interés. Y, por supuesto, implementando todo el entorno con herramientas libres.

Cambiando un poco de temática hacia el mundo de las telecomunicaciones tenemos "*Innovación tecnológica en comunicaciones móviles, desarrollada con Software Libre: Campus Ubicuo*", que presenta un proyecto de desarrollo realizado como fruto de un convenio entre el Grupo de Investigación GITACA de la Universidad de Extremadura y una empresa privada, con el apoyo de la Junta de Extremadura. Este proyecto da respuesta a las necesidades de los usuarios de las tecnologías de las comunicaciones en su demanda de sistemas telemáticos que faciliten su movilidad y ubicuidad mediante software libre.

Siguiendo el hilo de la implicación de instituciones públicas con el software libre se ha incluido también "*El modelo de la Oficina de Software Libre de la Universidad de Cádiz en la universidad española*". Este trabajo hace un repaso de las acciones que ha llevado a cabo la Oficina de Software Libre de la Universidad de Cádiz desde su creación allá por 2004 hasta la actualidad. Uno de los aspectos más destacados de una institución perteneciente al ámbito universitario es su amplio espectro de acción. Las actividades realizadas están orientadas a los campos de la docencia, investigación, gestión, apoyo al desarrollo de Software Libre, divulgación y relaciones con el exterior.

Otro artículo procedente también del "Workshop on Emerging Trends in FLOSS Research and Development 2007", de interés en la relación entre I+D+I y software libre es "*Aprendiendo a introducir una nueva tecnología en un proyecto basado en Software Libre*", que presenta una metodología para la introducción de resultados de I+D+I en proyectos FLOSS (Free/Libre and Open Source Software). Este trasvase de conocimiento no sólo beneficia al proyecto, sino también a los investigadores que puede evaluar "al pie del cañón" los resultados de aplicar las conclusiones de sus investigaciones (herramientas, protocolos de trabajo, etc.) a proyectos reales con la transparencia que suele caracterizar a los proyectos FLOSS. Aunque parece claro que el uso de resultados de I+D+I es necesario y positivo para el desarrollo de FLOSS, la metodología de trabajo y desarrollo del software libre presenta algunas peculiaridades. Factores como el tipo de innovación concreta o la estructura y tamaño de la comunidad desarrolladora pueden influir en el modo de presentar e implantar la innovación para que sea aceptada.

De otro congreso de software libre, el FLOSSIC 2007 se incorpora el trabajo, "*Optimización del proceso de render 3D distribuido con software libre*". Este artículo fue galardonado con el premio al mejor artículo del congreso. Refleja los resultados del trabajo conjunto de dos instituciones europeas: por un lado la Universidad de Castilla La Mancha (España) y por otro el Software Competence Center de Hagenberg (Austria). En él se aborda uno de los problema "clásicos" de la informática: la generación de imágenes por computador, en concreto la generación de imágenes 2D de alta calidad a partir de la definición abstracta de una escena en 3D. Para dicho trabajo se utilizan herramientas libres con técnicas y algoritmos distribuidos de última generación con objeto de reducir el coste computacional del proceso. En concreto, se describen las herramientas Yafrid y MagArRo, que permiten la optimización del *renderizado* distribuido.

Por último, no nos gustaría terminar esta introducción sin agradecer a la direcciones de ***Novática*** y **UP**GRADE el confiar en nosotros para editar este número especial. Número que no hubiera sido posible sin el esfuerzo de todos los autores de artículos, revisores, traductores y, en general, de toda la comunidad que hace posible que el Software y Conocimiento Libre sean una realidad.

Nota del Editor de Novática: por razones de espacio no se ha incluido en esta monografía el artículo "*Identificando el Éxito y la Tragedia de los Comunes en FLOSS: Una Clasificación Preliminar de Proyectos de Sourceforge.net*" de ***Robert English*** y ***Charles M. Schweik***, que sí aparecerá en cambio en el número 6/2007 de **UP**GRADE en inglés.

Referencias útiles sobre "Software Libre"

En esta sección se incluyen referencias relevantes relacionadas con el software libre y sus modelos avanzados de investigación y desarrollo que complementan las que aparecen en los artículos que componen la monografía.

Citas de esta presentación

[1] <http://www.nytimes.com/2005/03/29/technology/29computer.html>.

[2] <http://news.bbc.co.uk/1/hi/business/4602325.stm>.

[3] Decreto 72/2003 de Medidas de Impulso a la Sociedad del Conocimiento en Andalucía del 18 de marzo de 2003 (BOJA 55, 21 de marzo de 2003)

[4] <http://www.20minutos.es/noticia/91463/0/programas/ordenador/pueden/>.

[5] <http://www.juntadeandalucia.es/repositorio/>.

[6] <http://ec.europa.eu/enterprise/ict/policy/doc/2006-11-20-flossimpact.pdf>.

[7] <http://cross.lincoln.ac.uk/floss2007/>.

Instituciones que apoyan el software libre

- **Free Software Foundation** <ttp://fsf.org>.
- **Open Source Initiative** <ttp://opensource.org>.
- **Cenatic** <ttp://www.cenatic.es/>.
- **OSLUCA** <ttp://www.uca.es/software libre>

Webs de noticias

- **Slashdot** <ttp://slashdot.org>.
- **Digg** <ttp://digg.com>.
- **Blog de Ricardo Galli** <http://ricardogalli.com>.
- **Meneame** <http://meneame.net>.
- **Barrapunto** <http://barrapunto.com>.

Libros

- **Eric S. Raymond.** *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, O'Reilly, 2001, ISBN: 0596001088. Disponible en <http://www.catb.org/~esr/writings/cathedral-bazaar/> y en castellano en <http://biblioweb.sindominio.net/telematica/catedral.html>.
- **Richard M. Stallman, Lawrence Lessig, Joshua Gay (Editor).** *Free Software, Free Society: Selected Essays of Richard M. Stallman.* Free Software Foundation, 2002.ISBN: 1-882114-98-1. Disponible en <ttp://www.gnu.org/philosophy/fsfs/rms-essays.pdf> y en castellano en <http://biblioweb.sindominio.net/pensamiento/softlibre/softlibre.pdf>.
- **Lawrence Lessig.** *Code 2.0.* Basic Books, 2006. ISBN-13: 978–0–465–03914–2. <http://codev2.cc/>.
- **Eric Von Hippel.** *Democratizing Innovation.* MIT Press, 2006. ISBN-13: 9780262002745. <http://web.mit.edu/evhippel/www/democ1.htm>.
- **Ron Goldman, Richard P. Gabriel.** *Innovation Happens Elsewhere: Open Source as Business Strategy*, Morgan Kaufman/Elsevier, 2005, ISBN: 1-55860-889-3. <http://dreamsongs.com/IHE/>.
- **Peter Wayner.** *Free for All: How Linux and the Free Software Movement Undercut the High-Tech Titans.* Peter Wayner, 2000. ISBN 0-06-662050-3. <http://www.rau-tu.unicamp.br/nou-rau/softwarelivre/document/?code=138>.
- **Proyecto O'Reilly Open Books.** <http://www.oreilly.com/openbook/>.
- **Lawrence Rosen.** *Open Source Licensing: Software Freedom and Intellectual Property Law.* Prentice Hall, 2004. ISBN-13: 978-0131487871. <http://www.rosenlaw.com/oslbook.htm>.
- **Joseph Feller, Brian Fitzgerald, Scott A. Hissam, Karim R. Lakhani.** *Perspectives on Free and Open Source Software.* ISBN-13: 978-0-262-06246-6. MIT Press, 2006. <http://mitpress.mit.edu/catalog/item/default.asp?tid=10477&ttype=2>.
- **Karl Fogel.** *Producing Open Source Software: How to Run a Successful Free Software Project.* Karl Fogel, 2005. <http://producingoss.com/>.
- **Linux Torvalds, David Diamond.** *Just for Fun, The story of an accidental revolutionary.* HarperCollins, 2001. ISBN-13: 978-0066620725.
- **Glyn Moody.** *Rebel Code: Linux and the Open Source Revolution.* Perseus Books Group, 2002. ISBN-13: 978-0738206707
- **Abella, M. A. Segovia**. Libro blanco del software libre en España III. 2007. <http://www.libroblanco.com>.

Otros enlaces

- Economic and Game Theory: Against Intellectual Monopoly. <http://levine.sscnet.ucla.edu/general/intellectual/againstnew.htm>.
- Recopilación de documentación libre del FLOSSIC 2007. <http://flossic.loba.es/>.
- Recursos libres creados pos la UOC para sus cursos de posgrado sobre software libre. <http://www.uoc.edu/masters/esp/web/materiales_libres.html>.
- European Interoperability Framework for pan-European eGovernment Services. European Communities, 2004. ISBN 92-894-8389-X. <http://europa.eu.int/idabc/en/document/3761>.
- Como colaborar con el proyecto KDE. <http://www.kdehispano.org/colaborar_KDE>.
- Proyecto Debian. <http://www.debian.org>.
- Guiactiva: guía para la creación de empresas Software Libre. CEIN, S.A., 2005. Depósito legal: NA 1078-2005. <http://www.cein.es/web/es/documentacion/ideas/2005/7831.php>.
- Linux Knowledge Base and Tutorial. <http://sourceforge.net/projects/linkbat>.
- Mitsubishi Research Institute, Inc. An Introduction to Open Source Software. 2006. <http://oss.mri.co.jp/i2oss/download/en/text.pdf>.
- Alessio Damato. Why The Future Of Science Must Be In Free Software. <http://scientificcomputing.net/debian/why.pdf>.

La necesidad de investigar sobre software libre en Europa

Israel Herraiz Tabernero[1], Rafael Rodríguez Galván[2], Manuel Palomo Duarte[2]

[1]Grupo de Sistemas y Comunicaciones, Universidad Rey Juan Carlos, Madrid; [2]Oficina de Software Libre de la Universidad de Cádiz

<herraiz@gsyc.escert.urjc.es>, <rafael.rodriguez@uca.es>, <manuel.palomo@uca.es>

Resumen: *la Comisión Europea, a través del Programa Marco, está financiando diversos proyectos de investigación relacionados con el software libre. En la sexta edición de este programa, se han dedicado 25,13 millones de euros a financiar estos proyectos de investigación. ¿Merece la pena esta inversión? ¿Puede aportar algo el software libre al desarrollo de Europa? En este artículo de opinión exponemos las razones por las que se investiga sobre software libre, y qué pueden aportar estos proyectos al desarrollo económico y social de Europa. Por último, incluimos un resumen de los principales proyectos de investigación en los que, dentro del Programa Marco, se viene trabajando.*

Palabras clave: *investigación, programa marco.*

Autores

Israel Herraiz Tabernero realiza estudios de doctorado en la Universidad Rey Juan Carlos. Su investigación está relacionada con la evolución de proyectos de software libre. En particular, está empleando análisis de series temporales y otras técnicas estadísticas para caracterizar y predecir la evolución de proyectos de software libre. Ha participado en diferentes proyectos financiados por el Programa Marco de la Comisión Europea (QUALOSS, FLOSSMetrics, Qualipso, CALIBRE). Además, ha colaborado también en otros proyectos financiados por empresas como Vodafone o Teléfonica. Ha participado en la redacción de manuales y documentos sobre cómo gestionar y poner en marcha proyectos de software libre. Por ejemplo, junto con Juan José Amor y Gregorio Robles escribió un manual para el Máster en Software Libre de la Universitat Oberta de Catalunya. En este momento disfruta de un contrato de Formación de Personal Investigador, concedido en 2005 por la Comunidad de Madrid para desarrollar la tesis doctoral en el estudio de proyectos de software libre. Ha sido revisor para, entre otras conferencias, la IEEE Africon 2007, y para la revista IEEE Transactions on Software Engineering. En estos momentos está coordinando el programa del Máster en Software Libre, título propio de la Universidad Rey Juan Carlos que se imparte en colaboración con la empresa Igalia y Caixa Nova, e imparte clases en diversas titulaciones en la citada Universidad.

José Rafael Rodríguez Galván es profesor del Departamento de Matemáticas de la Universidad de Cádiz (UCA) y, desde el año 2004, director de la OSLUCA. Dentro de ésta ha sido responsable de la organización de distintos proyectos, entre ellos las I, II y III Jornadas de Software Libre de la UCA y el FLOSSIC (Free/Libre Open Source Systems International Conference). Ha llevado a cabo distintas conferencias y ha participado en foros relacionados con el software libre y la universidad. Por otro lado, ha realizado distintos trabajos en el ámbito de la simulación numérica de ecuaciones en derivadas parciales, en el marco de la mecánica de fluidos, dentro del grupo de investigación FQM-315 de la UCA.

Manuel Palomo Duarte es Ingeniero en Informática por la Universidad de Sevilla (2001). En la actualidad trabaja como profesor del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Cádiz donde imparte asignaturas relacionadas con los sistemas operativos y el diseño de videojuegos (haciendo uso de software libre en ellas) y realiza labores de Coordinador Erasmus en la titulación de Ingeniería Técnica en Informática de Sistemas. Es miembro del grupo de investigación "Mejora del Proceso Software y Métodos Formales", donde está realizando su tesis doctoral sobre calidad de composición de composiciones de servicios web con BPEL. Desde su incorporación a la UCA ha colaborado con la OSLUCA, principalmente en la realización de las "III Jornadas de Software Libre de la Universidad de Cádiz" (JOSLUCA3) y el "I Congreso Científico Internacional de FLOSS" (FLOSSIC 2007).

1. El software libre en Europa

El software libre nació como un fenómeno de comunidades de voluntarios, que se unían para desarrollar software, y donde las motivaciones principales para trabajar en los proyectos eran de tipo personal. Sin embargo, en la actualidad es un fenómeno económico de importancia indiscutible, hasta el punto de que los gigantes de la industria del software han estado, de una u otra forma, obligados a posicionarse al respecto: en unos casos reconociéndolo, como una amenaza competidora para su negocio, mientras que en otros como una oportunidad (para abrir nuevos mercados, reforzar su competitividad, etc.).

El software libre respeta los estándares, permite la interoperabilidad entre instituciones públicas o privadas, evita el monopolio en el acceso a la información y refuerza la educación neutral, con independencia de marcas comerciales. Así, universidades y centros de investigación, que estuvieron relacionados con el movimiento del software libre desde sus orígenes (en paralelo al desarrollo de Internet), lo sitúan cada vez más como centro de interés, bien sea a través de su promoción para las actividades cotidianas que éstas desempeñan (investigación, docencia, gestión, etc.) o bien como agentes dedicados al estudio científico de cada una de sus facetas.

Su estudio cobra mayor relevancia debido al hecho de que el software se ha convertido en un instrumento fundamental de la economía. Está presente en todas partes: en la oficina, en el teléfono móvil, en el coche, en los sistemas públicos que gestionan nuestros datos personales, etc. Sin embargo, todavía hoy día, no tenemos una verdadera Ingeniería que nos proporcione las herramientas para construirlo. Aunque es obvio que existe la Ingeniería de Software, todavía hoy sigue vigente el *Mytical Man Month* de *Brooks* [1]: *no podemos predecir cuánto tiempo durará un proyecto de software, qué características concretas tendrá finalmente, qué fallos tendrá, cuánto dinero habrá que invertir. Además, cuando los proyectos terminan dan lugar a resultados complejos, difícilmente mantenibles. En ocasiones incluso es mejor empezar uno nuevo que basarse en algo ya hecho previamente.*

El software, que hoy en día vertebra nuestra economía, se realiza en su mayoría fuera de nuestro ciclo económico: no lo escriben ingenieros europeos ni lo comercializan empresas europeas. Debido a esta circunstancia, se podría decir que una industria vital para el desarrollo económico de Europa está en manos externas a sí misma.

Por otro lado, todavía la Ingeniería del Software no es una disciplina verdaderamente científica. Muchos artículos tienen que firmar acuerdos de no divulgación para tener acceso a las fuentes de datos. En ocasiones ni siquiera se conocen qué casos de estudio se han considerado en un trabajo, y los casos de estudio se etiquetan bajo los enigmáticos nombres de A, B, C, X ó Y. Así, es imposible repetir y verificar esos estudios. Nunca lograremos avanzar como lo han hecho otras ciencias con estas trabas. Es más, nunca se podrán superar los problemas inherentes al desarrollo de software con una disciplina

científica que pone trabas la innovación por medio de acuerdos de no divulgación.

El software libre puede ayudar a superar estos inconvenientes. En primer lugar, respecto al impacto económico, el software libre no está controlado por nadie (o desde otro punto de vista, lo está por todos). Es más, en estos momentos el software libre ya tiene un impacto importante en Europa. El informe recientemente publicado *The impact of Free/Libre/Open Source Software on innovation and competitiveness of the European Union* [2], menciona que el 20% de la inversión europea en software se está realizando en software libre (esta cantidad es similar también para EEUU), y para el año 2010 el impacto global del software libre en Europa podría llegar a cuantificarse en el 4% del PIB europeo.

Desde el punto de vista de investigación, por sus características, el software libre no impone ninguna traba al estudio científico, puesto que todas las fuentes de datos son públicas. Es más, el informe mencionado en el párrafo anterior también incluye que muchos productos de software libre son líderes en sus respectivos mercados. En otras palabras, esos productos tienen una calidad suficiente como para imponerse sobre otras soluciones desarrolladas en el seno de empresas.

Esto quiere decir que, a pesar de la falta de un sustento científico en la manera en la que desarrollamos software, se han desarrollado productos de calidad. Y lo que es mejor, todas las trazas dejadas por el equipo de desarrollo están públicamente disponibles (documentación, código fuente, historial de cambios, incluso las comunicaciones por correo electrónico de desarrolladores, usuarios, etc). Por tanto, tenemos la "rata de laboratorio" perfecta para estudiar cómo superar los problemas que ya mencionaba Brooks y que todavía hoy están vigentes.

En este sentido, dentro de la sexta edición del Programa Marco (6PM) se lanzaron 11 proyectos de investigación, con un presupuesto total a cargos públicos de más de 25 millones de euros (**tabla 1**). Dentro del séptimo Programa Marco (7PM) también se están financiado nuevos proyectos relacionados con el tema. Centrándonos en el 6PM, la temática de los proyectos es muy variada:

- Qualipso

Este es el mayor proyecto de investigación relacionado con software libre jamás financiado por la Comisión Europea. El primer año de proyecto ha culminado en la celebración en enero de 2008 de la Qualipso Conference en Roma, a la que acudieron empresas de toda Europa.

El propósito general de Qualipso es definir e implementar tecnologías, procedimientos y políticas para que las prácticas de desarrollo de software libre estén al mismo nivel que prácticas industriales comúnmente aceptadas como buenas. En concreto, intenta desarrollar un modelo de certificación para procesos de desarrollo de software libre similar al CMM (*Capability Maturity Model*), que es bastante común en la industria de software.

- TOSSAD

Es una acción coordinada que intenta difundir el software libre en diferentes sectores (empresas, sector público), y pretende crear consorcios e identificar sinergias para fomentar la innovación mediante la adopción de software libre.

- SELF

Es similar a TOSSAD, pero está centrado en recursos educativos. La idea principal es aprovechar las ventajas que proporciona el software libre (disponibilidad de materiales, independencia frente a proveedor, bajo coste de las licencias, etc) en el sector educativo.

- FLOSSWorld

Este proyecto estudió la situación del software libre en las diferentes regiones del mundo (Europa, Asia, África, América del Norte y del Sur, etc). El propósito principal era asegurar que Europa liderara la investigación sobre software libre en el mundo, así como identificar la situación en cuanto a desarrollo de software libre, industria, estándares e interoperabilidad y gobierno electrónico en las diferentes regiones del mundo.

- FLOSSMetrics

Este proyecto está recogiendo métricas e información sobre una cantidad muy grande de proyectos de software libre (del orden de miles), con el fin de crear una base de datos que pueda usarse para estudios posteriores sobre software libre. Este proyecto está intentando que sus bases de datos se empleen en otros proyectos del 6PM que necesitan recoger esas métricas para sus objetivos (en particular, la colaboración con QUALOSS está siendo bastante estrecha).

- TEAM

Este proyecto no estudia el software libre en sí, sino que está desarrollando un sistema para compartir conocimiento. Este sistema se distribuirá como software libre, y es probable que se usen componentes libres en su desarrollo.

- EDOS

La creación de distribuciones de software libre (como por ejemplo, Red Hat, Ubuntu, Debian, Suse, etc.) supone algunos proble-

Proyecto	Fecha comienzo	Plazo (meses)	Presupuesto EC (m€)	Presupuesto total (m€)
Qualipso	Nov-06	48	10,42	17,29
TOSSAD	Feb-05	25	0,78	0,79
SELF	Jul-06	24	0,98	0,98
FLOSSWorld	May-05	26	0,66	0,67
FLOSSMetrics	Sep-06	30	0,58	0,58
TEAM	Sep-06	30	2,95	4,16
EDOS	Oct-04	33	2,22	3,45
CALIBRE	Jun-04	28	1,50	1,65
SQO-OSS	Sep-06	24	1,64	2,47
PYPY	Dic-04	28	1,35	2,29
QUALOSS	Sep-06	30	2,05	2,95
		Total:	**25,13**	**37,28**

Tabla 1. Proyectos financiados por el Sexto Programa Marco relacionados con software libre (fuente: <http://cordis.europa.eu/ist/st/projects.htm>).

mas, debido sobre a todo a las dependencias que existen entre paquetes. Todavía hoy día es relativamente común "estropear" un sistema debida a una mala actualización donde no se han podido satisfacer las diferentes dependencias. Además, el mantenimiento y desarrollo de la distribución es cada vez más compleja debido a que las interacciones entre paquetes crecen con el cuadrado del número de paquetes. Para solucionar estos problemas, este proyecto se dirigió a proporcionar herramientas de gestión de paquetes y distribuciones.

- CALIBRE

Este proyecto intentó servir de nexo entre las diferentes empresas europeas del sector secundario del software, esto es, aquellas empresas que no desarrollan software libre, pero que su negocio depende de manera fundamental del software. Como resultado del proyecto se creó un foro denominado CALIBRATION, del que forman parte varias empresas europeas (Philips, Telefónica y Vodafone son algunas de las empresas).

- SQO-OSS

Este proyecto está intentando desarrollar un modelo para evaluar la calidad del software libre, mediante métodos empíricos a partir de las fuentes de datos públicas de los proyectos (repositorios de código, correos electrónicos, sistemas de seguimiento de fallos, código fuente, etc). Su propósito es muy similar al de QUALOSS.

- PYPY

Con el fin de portar Python (un conocido lenguaje de programación) a más entornos, y que sea más flexible su adaptación a otros sistemas en los que no está ahora disponible, este proyecto está intentando crear una nueva implementación de Python. Un punto interesante del proyecto es que están empleando metodologías ágiles para el desarrollo, y que se organizan como una comunidad de software libre. Todo el desarrollo está siendo cuidadosamente monitorizado, con el fin de investigar también el impacto de las metodologías ágiles en el desarrollo de software.

- QUALOSS

El objetivo de QUALOSS es crear un modelo de evaluación de la calidad de un proyecto de software libre, para lo que se están usando las diferentes fuentes de datos públicas de los proyectos de software libre. Este proyecto estudiará en detalle 50 proyectos de software libre, e intentará reutilizar al máximo la información sobre esos proyectos que se encuentren en las bases de datos de FLOSSMetrics.

En resumen, el software libre tiene un impacto importante en la economía europea, y este impacto va a crecer en los próximos años. Asimismo, proporciona una oportunidad inmejorable para estudiar el proceso de desarrollo de software, y lograr de una vez una verdadera Ingeniería del Software. En la actualidad existen muchos proyectos de investigación que están tratando de resolver estos problemas, con un claro apoyo por parte de la Comisión Europea a través del Programa Marco de investigación. Este apoyo debe continuar en el futuro.

Referencias

[1] Fred Brooks. *The Mythical Man-Month: Essays on Software Engineering.* Addison-Wesley. 1995. ISBN 0-201-83595-9.

[2] Rishab A. Ghosh et al. *The impact of Free/Libre/Open Source Software on innovation and competitiveness of the European Union.* Comisión Europea, 2007. Disponible en <http://flossimpact.eu>.

Andrea Capiluppi[1], Martin Michlmayr[2]
[1]Universidad de Lincoln, Reino Unido; [2]Universidad de Cambridge, Reino Unido

<acapiluppi@lincoln.ac.uk
<martin@michlmayr.org>

De la catedral al bazar: un estudio empírico del ciclo de vida de los proyectos basados en comunidades de voluntarios

Traducción: Israel Herraiz Tabernero (Universidad Rey Juan Carlos)

Resumen: *en el pasado, algunos proyectos de software libre han tenido mucho éxito. A menudo, el éxito de un proyecto de software depende del número de desarrolladores que es capaz de atraer: una comunidad grande (el "bazar") encuentra y arregla más defectos en el software y añade más características nuevas mediante un proceso de revisión por pares. En este artículo estudiamos dos proyectos de software libre (Wine y Arla) desde un punto de vista empírico, con el fin de caracterizar el ciclo de vida del software, los procesos de desarrollo y la comunidad en la que se basa el proyecto.*

Palabras clave: *desarrolladores de software, evolución del software, fases, procesos de software, software libre.*

Autores

Andrea Capiluppi es Doctor en Informática por el Politécnico de Turín (Italia). Fue investigador visitante en el Grupo de Sistemas y Comunicaciones de la Universidad Rey Juan Carlos (Madrid) en octubre de 2003. Desde enero de 2004 hasta la actualidad ha sido investigador visitante en el Departamento de Matemáticas y Computación de la Open University (Reino Unido), trabajando junto con los doctores Juan Ramil, Neil Smith, Helen Sharp, Alvaro Faria y Sarah Beecham. Este compromiso ha sido renovado hasta diciembre de 2008. En enero de 2006, entró a formar parte de la Universidad de Lincoln como profesor senior.

Martin Michlmayr ha estado involucrado en varios proyectos de software libre desde hace más de 10 años. Fue coordinador de voluntarios para el proyecto GNUstep, y actuó como Director de Publicidad para Linux International. En el año 2000, se unió al proyecto Debian, y fue elegido más tarde *Debian Project Leader* (DPL), puesto que ocupó durante dos años. Martin posee títulos de Máster en Filosofía, Psicología e Ingeniería del Software, y un doctorado por la Universidad de Cambridge. En la actualidad trabaja para HP como experto en comunidades de software libre.

1. Introducción

Los proyectos de software libre más conocidos, tales como Linux [32], Apache [27] y FreeBSD [18] han tenido un éxito tremendo. Hasta ahora, las pruebas para caracterizar el éxito de un proyecto de software libre han sido anecdóticas: más usuarios y/o desarrolladores suponían más "ojos" para localizar fallos, los desarrolladores implementaban nuevas características de manera independiente, y los líderes de proyectos gestionaban una estructura prácticamente horizontal con los consecuentes costes de coordinación [28].

Estudios previos han proporcionado pruebas empíricas de los procesos de desarrollo empleados en proyectos de software libre que tienen éxito: por ejemplo, la definición de diferentes tipos de desarrolladores para los proyectos Apache y Mozilla, que justificaba diferentes niveles de esfuerzo [27], y concluía que el primer tipo, los desarrolladores principales (*core developers*), son los que contribuyen al éxito de un proyecto. También, el análisis de redes sociales ha mostrado cuáles son los costes de comunicación y coordinación en proyectos de software libre de éxito [21].

[1] **Nota del editor:** Como se cita posteriormente a lo largo del artículo, la distinción entre las fases de "catedral" y "bazar" en los proyectos de software libre proviene de Eric S. Raymond [28]. En Edukalibre <http://collab.edukalibre.org/docs/un_libro/sobre_gordo/ch07s02.html> nos cuentan que: "Dentro de lo que Raymond toma como el modelo de creación de catedrales no sólo tienen cabida los procesos pesados que podemos encontrar en la industria del software (el modelo en cascada clásico, las diferentes vertientes del Rational Unified Process, etc.), sino que también entran en él proyectos de software libre, como es el caso de GNU y NetBSD. Para Raymond, estos proyectos se encuentran fuertemente centralizados, ya que unas pocas personas son las que realizan el diseño e implementación del software [...].El modelo antagónico al de la catedral es el bazar. Según Raymond, algunos de los programas de software libre, en especial el núcleo Linux, se han desarrollado siguiendo un esquema similar al de un bazar oriental. En un bazar no existe una máxima autoridad que controle los procesos que se están desarrollando ni que planifique estrictamente lo que ha de suceder".

En todos estos casos, los proyectos que tienen éxito se estudian y se caracterizan, pero no se analiza cómo comenzaron. Por tanto, no se han realizado estudios empíricos que muestren si el proyecto se benefició siempre de la participación de un gran número de desarrolladores, o por el contrario la fase de bazar[1] se alcanzó tras años de desarrollo. Para cubrir este hueco, este artículo explora la evolución y los procesos de desarrollo de dos sistemas de software libre, el proyecto Wine (una implementación libre de Windows para Unix) y el sistema de ficheros Arla. El primero de ellos se ha extendido entre muchos desarrolladores, que también han contribuido extensamente. Arla, por el contrario, está todavía en la fase "catedral"[1] si lo comparamos con Wine: tiene menos desarrolladores que estén dirigiendo el desarrollo del proyecto.

El propósito de este artículo es detectar y caracterizar, empíricamente, las fases alcanzadas por los dos proyectos mencionados, con el fin de ilustrar si una de las fases aparece a continuación y como consecuencia de la otra, y de proponer una de esas fases como "éxito" para un proyecto de software libre. En tal caso, compartir la metodología empírica para llevar a cabo la transición entre fases podría ayudar a los desarrolladores a trabajar en los beneficios que presenta la fase de bazar.

A continuación, en la **sección 2,** se muestran los fundamentos teóricos, además de dos cuestiones a investigar, basados en comunidades de software libre. Además, se presenta una descripción del enfoque usado para adquirir y analizar los datos empleados en el estudio. Estos datos se emplean para com-

probar las cuestiones a investigar propuestas. En la **sección 3**, se describen las fases observadas en los dos sistemas desde el punto de vista de las actividades de los desarrolladores. Esta sección también muestra una descripción detallada de las actividades que sustentan el éxito de un proyecto de software libre, tal y como se han observado en los casos de estudio propuestos. La **sección 4** trata del trabajo relacionado en ésta y otras áreas, identificando cuáles son las contribuciones principales de este artículo, y discute algunas cuestiones adicionales que aparecen en el artículo y que necesitan más estudios futuros. Por último, la **sección 5** muestra las conclusiones sobre el proceso general y el ciclo de vida de un proyecto de software libre, además de algunas indicaciones para trabajos futuros.

2. Antecedentes teóricos

Uno de los autores de este artículo, en un trabajo previo [29], presentó una teoría para las diferentes actividades y fases del ciclo de vida de un proyecto de software libre. El objetivo era proporcionar un enfoque sistemático para el desarrollo de proyectos de software libre, es decir, incrementar la probabilidad de éxito en proyectos nuevos. En este artículo, el objetivo es evaluar de un modo empírico la teoría contenida en el mencionado trabajo, a través de dos casos de estudio, e informar de cuáles son las mejores prácticas de proyectos de software libre reales de éxito. Dado que algunos trabajos previos han mostrado que muchos proyectos de software libre deberían considerarse como fracasos [3][7], se muestra que estos proyectos no presentan algunas de las características mencionadas en [29], principalmente la transición entre el estilo cerrado (o "catedral") y abierto (o "bazar").

En su popular ensayo *The Cathedral and the Bazaar*, Eric S. Raymond [28] investiga las estructuras de desarrollo de proyectos de software libre, basándose en el éxito de Linux. La terminología "catedral" y "bazar" presenta tanto un enfoque cerrado, que se encuentra en la mayoría de entidades comerciales, donde las decisiones sobre un proyecto grande de software se toman mediante una gestión centralizada, como un enfoque abierto, donde una comunidad entera es la responsable de la gestión de todo el sistema.

En lugar de presentar estos dos enfoques como diametralmente opuestos (tal y como proponía originalmente Raymond), este artículo considera que son eventos complementarios dentro de un mismo proyecto de software libre. La **figura 1** muestra las tres fases básicas, las cuales están presentes en un proyecto de software libre con éxito, según la tesis defendida en este artículo. La fase inicial de un proyecto de software libre no opera en el contexto de una comunidad de voluntarios. Todas las características del estilo catedral (recogida de requisitos, diseño, implementación, *testing*) están presentes en esta fase, y también en el estilo de construcción típico de una catedral, es decir, el trabajo lo realiza un individuo o un pequeño grupo de desarrolladores aislados de la comunidad [5]. Este proceso de desarrollo muestra un control estricto y una planificación centralizada y realizada por el autor principal, que ha sido denominada "prototipado cerrado" por Johnson [17].

Según [29], un proyecto de software libre tiene que realizar una transición de la fase catedral a la fase bazar, con el fin de convertirse en un producto útil y de calidad (tal y como se muestra con la flecha en la **figura 1**). En esta fase, nuevos usuarios y desarrolladores se unen de manera continua al proyecto, escribiendo código, enviando parches y arreglando fallos. Esta transición se asocia con bastantes complicaciones: por ejemplo, algunos estudios [7] concluyen que la mayoría de proyectos de software libre nunca abandonan la fase de catedral y por tanto no acceden a la vasta cantidad de recursos que la comunidad de software libre ofrece en forma de mano de obra y habilidades.

2.1. Cuestiones a investigar

En este artículo, se analizan datos históricos sobre las modificaciones y adiciones de secciones a gran escala (subsistemas) o pequeña escala (módulos) a un sistema de software, con el fin de trazar cómo han evolucionado en el tiempo los casos de estudio. Se proponen dos cuestiones a investigar, que se validarán frente a los datos históricos. Esto se realiza en la siguiente sección, donde también se muestran los resultados. La primera cuestión se basa en la respuesta del proyecto a un estímulo, y la segunda se refiere al tipo de trabajo al que suelen dedicarse los desarrolladores cuando comienzan su colaboración con el proyecto. Estas cuestiones se formulan a continuación (se incluyen también las métricas necesarias para validarlas):

1) Cuestión 1: la fase bazar supone un crecimiento en el número de desarrolladores, que se unen en un ciclo auto-sostenido. El resultado obtenido en esta fase muestra un patrón de crecimiento similar. Los proyectos de software libre no se benefician de esta tendencia creciente durante la fase catedral.
2) Cuestión 2: Cuando un desarrollador nuevo se une al proyecto, tiende a trabajar primero en los módulos más nuevos, ya sea creando ellos mismos el módulo o contribuyendo a un módulo de reciente creación. Esto puede explicarse argumentando que un desarrollador no necesitaría conocer toda la funcionalidad ya existente en el sistema para desarrollar una parte nueva del sistema. Esta cuestión a investigar se emplea en este artículo para proponer cómo Wine pudo alcanzar la fase de bazar.

2.2. Metodología empírica

El enfoque empírico supone la extracción de todos los cambios que se encuentran tanto en la entrada (esfuerzo proporcionado por los desarrolladores) como en la salida (cambios y nuevo código en los subsistemas y módulos) del proyecto. A continuación, en lugar de analizar los repositorios CVS de los proyectos, se analiza el fichero de *ChangeLog*[2], que guarda toda la historia de cambios del proyecto. Estudios previos [10][22] muestran que diferentes prácticas

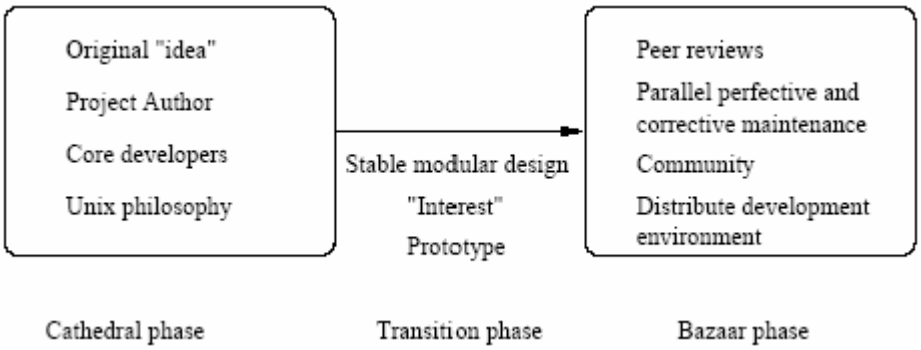

Figura 1. Ciclo de desarrollo de un proyecto de software libre.

de desarrollo tienen impacto en cuál es la mejor fuente de datos para analizar el proyecto; el fichero de *ChangeLog* proporciona más fiabilidad en los casos de estudio seleccionados [6][12][30].

Los pasos que hay que seguir para producir los datos necesarios para el estudio se resumen a continuación: se interpretan los datos de los ficheros de cambio y se extraen las métricas. Como parte del primer paso, se han escrito *scripts* en Perl para interpretar los datos contenidos en los ficheros de *ChangeLog*, y para extraer algunos campos de datos predefinidos. Los campos que se contemplan en este estudio son: el nombre del sistema, el nombre del módulo, el nombre del subsistema que contiene ese módulo, la fecha de creación o cambio y un identificador único (nombre y dirección de correo electrónico) del desarrollador responsable del cambio.

2.2.1. Extracción de datos

Los ficheros de *ChangeLog* analizados siguen un patrón muy regular, por lo que el análisis de los cambios producidos en la historia del proyecto puede realizarse muy fácilmente y de un modo casi automático. Para extraer los datos se han seguido los pasos siguientes:

1. Identificación de las fechas: en los casos de estudio se observó que cada cambio estaba delimitado por una fecha, usando el siguiente patrón o uno similar: AAAA-MM-DD, como en "2000-12-31". Cada cambio puede asociarse con uno o más desarrolladores; además, cada cambio puede asociarse con uno o más módulos. Sin embargo, sólo hay una fecha para cada cambio.
2. Módulos y subsistemas afectados: cada cambio afecta al menos a un fichero, y se guarda con una descripción en texto plano. En algunos casos, el mismo cambio afecta a varios ficheros: estas modificaciones presentan siempre la misma fecha. Los subsistemas se extraen como el directorio que contiene al fichero afectado por el cambio.
3. Detalles de los desarrolladores: Todos los cambios involucran al menos a un desarrollador, que se puede mostrar de varias maneras diferentes en la descripción del cambio. Si el cambio se debe a más de un desarrollador, todos los desarrolladores aparecen juntos en la descripción del cambio.
4. Cálculo de las métricas: Se calcularon tanto el esfuerzo de los desarrolladores como el trabajo producido creando nuevos módulos y corrigiendo módulos existentes.

2.2.2. Elección y descripción de las métricas

El análisis de los dos proyectos de software libre se realizó mediante tres tipos de métricas, que se usan de un modo diferente para discutir cada una de las cuestiones a investigar. La lista de métricas propuestas se muestra a continuación:

Métricas de entrada: el esfuerzo de los desarrolladores se evaluó contando el número de desarrolladores únicos (o distintos, por usar una terminología similar a SQL[3]) durante un intervalo específico de tiempo. La granularidad temporal que se ha seleccionado es la de meses: se pueden emplear diferentes enfoques, como contar los desarrolladores semanalmente o diariamente, pero creemos que un mes es una unidad temporal con una granularidad más indicada para extraer el número de desarrolladores activos. Estas métricas se han usado para evaluar la primera cuestión a investigar. Por ejemplo, durante febrero de 2006 el proyecto Wine tuvo 73 desarrolladores diferentes que escribieron código.

Métricas de salida: el trabajo producido se evaluó contando el número de cambios a los módulos o subsistemas durante el mismo intervalo de tiempo. No se han considerado métricas de granularidad fina, como por ejemplo líneas de código. Evaluar la producción de código por parte de los desarrolladores usando líneas de código hubiera supuesto importantes limitaciones al estudio[4]. En la siguiente sección se usará esta métrica como un indicador del trabajo de desarrollo en paralelo realizado en proyectos con éxito. Esta métrica se ha usado también para evaluar la primera cuestión a investigar. Siguiendo el ejemplo del párrafo anterior, en febrero de 2006 se detectó que en Wine había 820 módulos que habían sido modificados durante ese mes.

Métricas de nuevas entradas y salidas: el esfuerzo nuevo añadido al proyecto se midió como el número de desarrolladores que se unen al proyecto. Además, se aisló el trabajo que se debía a estos nuevos desarrolladores: el objetivo es determinar cuánto de este trabajo se centró en partes del sistema que existían previamente, y cuánto en partes nuevas. Estas métricas se han empleado para evaluar la segunda cuestión a investigar, esto es, para explorar si los desarrolladores nuevos tienden a trabajar en partes viejas o nuevas del sistema. Siguiendo el ejemplo de los párrafos anteriores, se detectó que durante febrero de 2006 se unieron 73 nuevos desarrolladores a Wine (esto es, los desarrolladores no fueron detectados en ningún cambio previo a esa fecha). Además, empíricamente se detectó que estos nuevos desarrolladores trabajan tanto en partes viejas como nuevas del sistema (añadidas ese mismo mes). Se observó que el 75% de su trabajo se producía en partes nuevas, y el 25% en partes que ya existían previamente.

2.3. Casos de estudio

La elección de los casos de estudio se realizó basándose en que uno (Wine) es un proyecto de éxito objetivo, mientras que el otro (Arla) parece haber sufrido problemas a la hora de reclutar a nuevos desarrolladores, alcanzando un tamaño mucho menor. Los dos proyectos se habían usado previamente para otros estudios empíricos, y se había estudiado ampliamente su estilo de desarrollo y patrón de crecimiento.

Los autores reconocen que los sistemas pertenecen a campos de aplicación muy diferentes: Wine es una herramienta para ejectuar aplicaciones Windows sobre Linux y otros sistemas operativos, mientras que Arla es un sistema de ficheros en red. El objetivo principal de estudio no era evaluar las razones exógenas detrás del éxito de estos proyectos

Atributo / Sistema	Arla	Wine
Entrada más antigua	Octubre 1997	Julio 1993
Última entrada	Marzo 2006	Marzo 2006
Número de cambios	7.000	88.000
Desarrolladores distintos (total)	83	880

Tabla 1. Resumen de la información relativa a los dos casos de estudio.

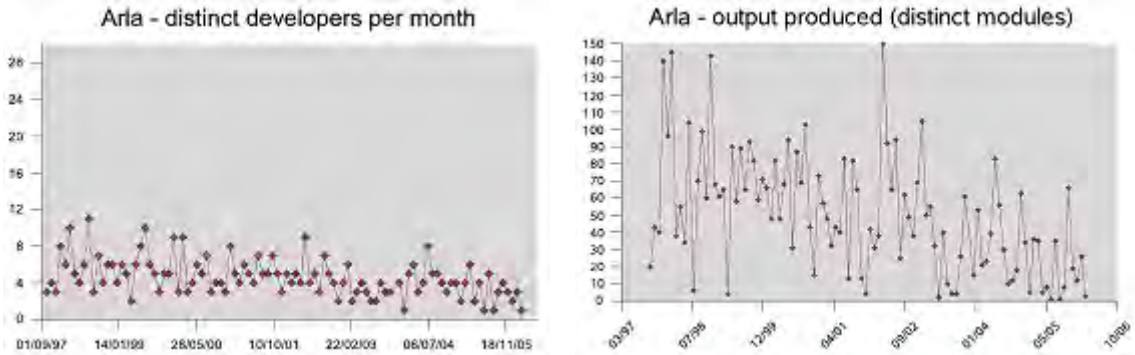

Figura 2. Número de desarrolladores (izquierda) y trabajo producido (derecha) en el caso de Arla.

a la hora de atraer desarrolladores (como por ejemplo la presencia de "gurús" en el proyecto, la buena reputación de la comunidad, etc. [9]). Al contrario, este estudio se centra en evaluar la presencia de tres etapas diferentes en proyectos que tienen éxito. El trabajo que se muestra en este artículo propone un marco teórico para proyectos de software libre, independientemente de su dominio de aplicación, y evalúa de manera empírica los mecanismos por los que se forman las comunidades alrededor de proyectos de software libre.

Se ha restringido la elección de las fuentes de información a dos tipos: los cambios realizados en el CVS y el fichero de *ChangeLog*. El repositorio CVS de Arla resultó estar *incompleto*, dado que no contenía la historia completa de la evolución del proyecto. Esto se debe con probabilidad al hecho de que el servidor CVS comenzó a usarse en algún momento posterior al comienzo del proyecto. Además, se observó que el repositorio CVS de Wine resultó ser *inexacto*: al consultar el número de desarrolladores en activo se obtuvo que había sólo 2 desarrolladores, mientras que los ficheros de *ChangeLog* contienen una cantidad mucho mayor de desarrolladores diferentes. Esto se debe probablemente a alguna restricción en el permiso de escritura en el repositorio. Debido a esto, era preferible usar los ficheros de *ChangeLog* que el registro de cambios del CVS.

La **tabla 1** muestra información acerca de los ficheros de *ChangeLog*, el tiempo de vida de los proyectos, y la cantidad de desarrolladores distintos, con el fin de caracterizar a los dos sistemas.

3. Resultados y discusión acerca de las fases

En esta sección, basándonos en datos empíricos de los dos casos de estudio, se discuten las dos cuestiones a investigar, y se evalúan las tres fases (catedral y bazar, separadas por una transición), tal y como se presentan en [29].

Además de esta evaluación, también se identifican algunas consideraciones prácticas para desarrolladores de software libre, de modo que se mejore el éxito evolutivo de sus proyectos, y se facilite la transición entre las fases de catedral y bazar.

3.1. Fase catedral

Una de las principales diferencias entre el software cerrado (tradicional) y el software libre es la propiedad del código. En entornos tradicionales, un grupo de individuos conduce el desarrollo, mientras que los usuarios ni contribuyen ni tienen acceso al código. En el software libre, potencialmente cualquiera tiene el derecho a acceder y modificar el código fuente de una aplicación. Creemos que un sistema libre típico presenta una fase de catedral en la primera parte de su historia evolutiva.

Sistema Arla – Entrada: La **figura 2** (izquierda) muestra la distribución de desarrolladores distintos por mes para el

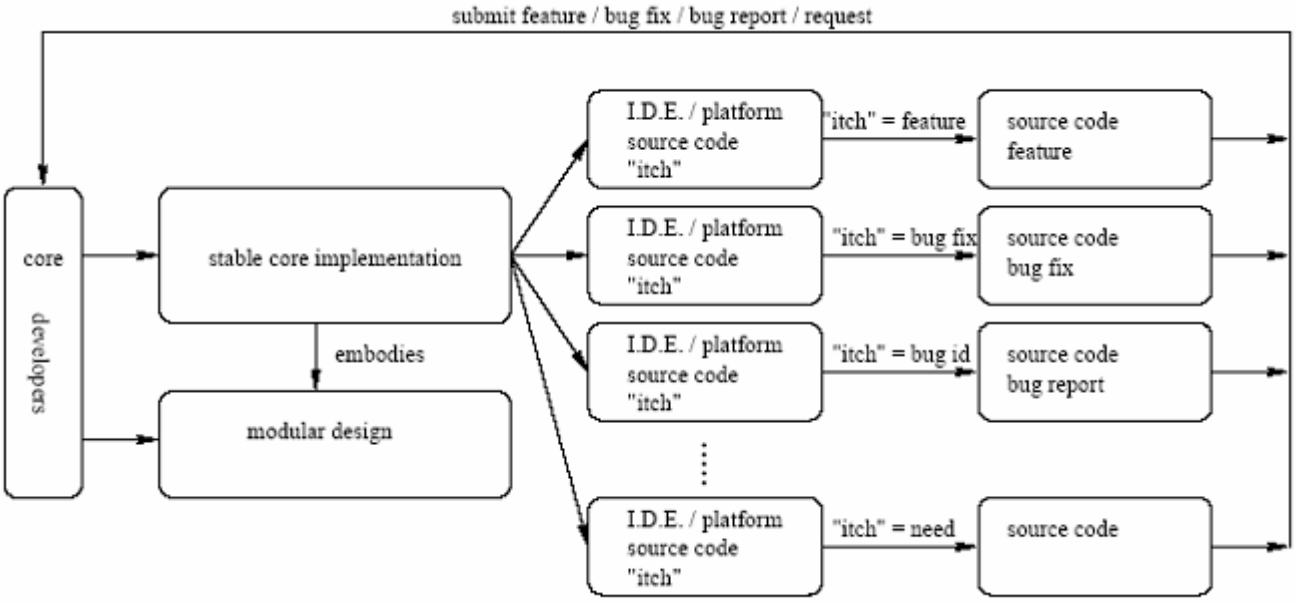

Figura 3. Fase de bazar (detalle).

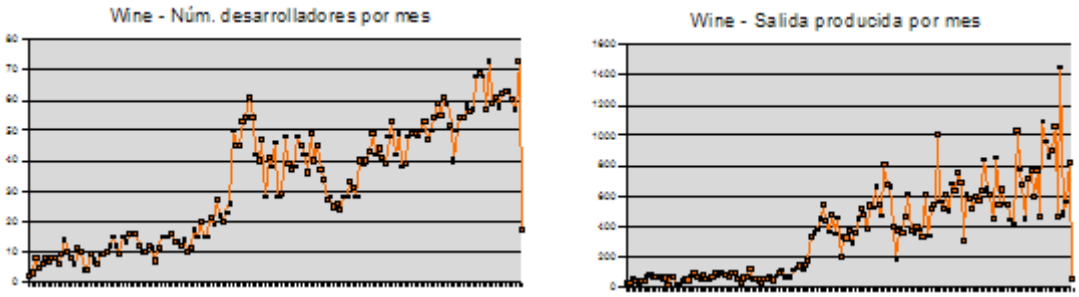

Figura 4. Número de desarrolladores (izquierda) y trabajo producido para el caso de Wine.

sistema Arla. A pesar de que más de 80 desarrolladores han contribuido con código, parches y arreglo de fallos al proyecto (véase **tabla 1**), el número de desarrolladores distintos por mes es mucho más bajo. De media, sólo 5 desarrolladores distintos trabajan cada mes en el proyecto. Como se dice en un párrafo anterior, los resultados empíricos no confirman la primera cuestión a investigar; la evolución de desarrolladores activos y distintos muestra un patrón regular y constante.

Sistema Arla – Salida: La **figura 2** (derecha), al contrario, muestra el número de módulos y subsistemas distintos en los que los desarrolladores de Arla han trabajado cada mes desde el comienzo del proyecto. La distribución es muy regular, lo que podría implicar que cuando nuevos desarrolladores se unen al proyecto no se expanden por áreas nuevas, sino que trabajan en funcionalidad ya existente, junto a los desarrolladores principales. Esto se comprobará en la sección dedicada a la fase de transición. Estos resultados, es decir, un patrón de la salida producida constante y que no crece, confirman que la primera cuestión a investigar no se verifica en el caso de Arla.

Aunque estos resultados no implican necesariamente que Arla sea un fracaso si lo comparamos con Wine (como se podría pensar al mirar el número total de desarrolladores en la **tabla 1**), sí que plantean algunas preguntas interesantes: por ejemplo, debería estudiarse por qué sólo un pequeño y constante grupo de desarrolladores está contribuyendo mediante código. Una explicación posible de este (reducido) éxito al reclutar a nuevos desarrolladores puede ser que los potenciales desarrolladores perciben el sistema como maduro [8], y por tanto se necesita poco trabajo en el proyecto. Se han encontrado problemas similares en el pasado en OpenOffice.org y Mozilla: estos sistemas representan dos aplicaciones extremadamente complejas y requerían una enorme inversión en su estudio antes de que los desarrolladores pudieran empezar a contribuir.

En las siguientes secciones se evalúan consejos prácticos sobre cómo un proyecto de software libre puede afrontar problemas como los que se están encontrando en Arla, y beneficiarse de los esfuerzos de un grupo de desarrolladores mayor.

3.2. Fase bazar

El objetivo de muchos proyectos de software libre es alcanzar una etapa en la que una comunidad de usuarios pueda contribuir de manera activa al desarrollo posterior del proyecto. Algunas de las características clave de la fase de bazar se muestran en la **figura 3** y pueden resumirse así:

- Contribuciones: el estilo bazar hace que el código fuente esté disponible públicamente, y las contribuciones se fomentan de manera activa, sobre todo de personas que sean usuarios del software. Las contribuciones pueden venir de muchas maneras diferentes y en momentos diferentes. Los usuarios sin perfil técnico pueden sugerir nuevos requisitos, escribir documentación y tutoriales, o poner de manifiesto problemas de usabilidad (que se representan como aportaciones de bajo nivel en la **figura 3**)
- Calidad del software: Las inspecciones exhaustivas y en paralelo del código proporcionan unos mayores niveles de calidad. Estas inspecciones las realiza una comunidad grande de usuarios y desarrolladores. Estos beneficios son consistentes con los principios de la Ingeniería del Software: el proceso de depuración de un proyecto de software libre es sinónimo de la fase de mantenimiento del ciclo de vida tradicional de un proyecto de software.
- Comunidad: una red de usuarios y desarrolladores revisa y modifica el código asociado con un sistema de software. El viejo dicho *"el trabajo compartido es más llevadero"*[5] describe las razones por las que algunos proyectos de software libre tienen éxito [27].

Sistema Wine – Entrada: La **figura 4** (izquierda) muestra la distribución de desarrolladores distintos por mes para el sistema Wine. En total, más de 800 desarrolladores han contribuido con código, parches y arreglando fallos (véase la **tabla 1**). Aunque el proyecto tiene un período de vida más largo, que podría haber facilitado el crecimiento del número de desarrolladores, mediante el número de desarrolladores se puede identificar una clara división entre la primera fase (catedral) y la última (bazar). Alrededor de julio de 1998, el sistema Wine experimentó una evolución masiva en el número de desarrolladores distintos involucrados en el proyecto. La sostenibilidad de esta nueva fase de bazar se demuestra por el incremento continuo de nuevos desarrolladores en el proyecto. Wine proporciona las evidencias empíricas para responder a la primera cuestión a investigar, esto es, un patrón creciente de desarrolladores activos señala la presencia de la fase de bazar. La sostenibilidad de la fase de bazar es visible en la cantidad de desarrolladores activos participando en la evolución del sistema, que cambia cada mes.

Sistema Wine – Salida: La fase de bazar se caracteriza por un proceso abierto en el que la entradas proporcionadas por voluntarios definen la dirección del proyecto, incluyendo la lista de requisitos. La implementación inicial se basa principalmente en los requisitos del autor del proyecto. En la fase de bazar, el proyecto se beneficia de la participación de un amplio espectro de usuarios (con diferentes requisitos), que trabajan juntos para incrementar la funcionalidad y el atractivo del software.

En el proyecto Wine se logra de una manera satisfactoria este proceso de desarrollo en paralelo. Durante la investigación de este sistema, se puso de manifiesto la evolución del alcance del proyecto, a través de la cantidad de módulos distintos en los que los desarrolladores trabajaban cada mes. En la **figura 4** (derecha) se muestra la cantidad de módulos y subsistemas distintos en los que los desarrolladores han trabajado desde el comienzo del proyecto: la distribución crece de manera brusca justo cuando se observa un crecimiento en el número de autores distintos. Esto significa que el proyecto se está expandiendo hacia nuevas áreas gracias a los nuevos desarrolladores que se unen de manera constante. El patrón de crecimiento de desarrolladores activos sostiene el crecimiento de la salida producida: como en el párrafo

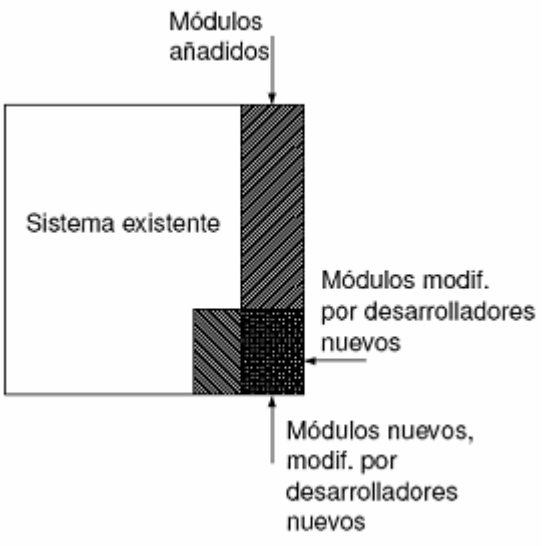

Figura 5. Diseño de la segunda cuestión a investigar.

anterior, la primera cuestión a investigar ayuda a señalar la presencia de la fase de bazar cuando ocurre ese patrón de crecimiento.

3.3. Fase de transición: nuevas vías de desarrollo

El marco teórico representado en la **figura 1** asigna un papel fundamental a la fase de transición, dado que requiere de una drástica reestructuración del proyecto, especialmente en la manera en la que es gestionado. Un aspecto importante es comenzar la fase de transición en el momento correcto. Este paso es crucial y muchos proyectos no logran superar este obstáculo [11]. Dado que durante la fase de transición es cuando hay que atraer a los voluntarios, el prototipo tiene que ser funcional pero a la vez todavía debe necesitar de desarrollo [17][28][2].

Si el prototipo no tiene suficiente estabilidad o funcionalidad, los voluntarios potenciales puede que no se unan al proyecto. Por otro lado, si el prototipo está demasiado avanzado, los nuevos voluntarios no tienen demasiados incentivos para unirse al proyecto porque el código ya existente es complejo o las características que estos desarrolladores requieren han sido implementadas ya. En los dos casos, añadir direcciones futuras de desarrollo al sistema puede proporcionar a los potenciales desarrolladores vías para el desarrollo del proyecto.

Basándonos en la segunda cuestión a investigar, cuando los nuevos desarrolladores se unen a un proyecto, tienden a trabajar en módulos nuevos más que en módulos viejos. Como consecuencia de esto, los desarrolladores principales deberían expandir el sistema original hacia nuevas direcciones para proporcionar nuevo código sobre el que trabajar: esto fomentaría el reclutamiento de desarrolladores nuevos y facilitaría la fase de transición.

Para evaluar esta cuestión, se diseñó un experimento: primero, se extraen los módulos nuevos añadidos cada mes. En paralelo, se extrae la cantidad de desarrolladores nuevos cada mes. Finalmente, las partes en las que los nuevos desarrolladores han trabajado se definen como el porcentaje de módulos nuevos que han sido tocados por estos desarrolladores. La **figura 5** muestra un resumen gráfico de este proceso. Se extrajeron los resultados empíricos para los dos sistemas, Arla y Wine. Se muestran en un diagrama de caja, que se extiende para todas las versiones de los dos sistemas. La **figura 6** describe, en porcentaje, la cantidad de módulos nuevos tocados por los desarrolladores nuevos.

Transición lograda – Wine: este sistema revela que cuando desarrolladores nuevos entran en el proyecto tienden a trabajar con menores dificultades en partes nuevas del código mejor que en partes antiguas. De hecho, más del 50% (en media) del trabajo que realizan los desarrolladores nuevos se produce en módulos añadidos el mismo mes, ya sea por los desarrolladores principales o por ellos mismos (**figura 6**, derecha). Además, la media era mayor cuando se consideraba sólo la fase de bazar en Wine.

El primer resultado se confirma dibujando la cantidad de módulos nuevos creados por los desarrolladores (**figura 7**, derecha). Se detecta un patrón creciente, similar al patrón de evolución global del sistema (**figura 4**): cuando desarrolladores nuevos se unen al proyecto trabajan en las partes más nuevas del sistema, mientras que los desarrolladores principales sostienen la comunidad del proyecto añadiendo continuamente módulos nuevos.

Transición no lograda – Arla: este sistema proporciona un diagrama de caja mucho más interesante. La tendencia de los desarrolladores nuevos es claramente trabajar en algo nuevo mejor que en algo antiguo (**figura 6**, izquierda). La diferencia principal con Wine es que, para la mayoría de los períodos no hay desarrolladores nuevos que se unan al proyecto. Basándonos en las suposiciones de la segunda cuestión a investigar, los desarrolladores nuevos todavía prefieren comenzar partes nuevas, o trabajar en partes que se han añadido recientemente: en cualquier caso, este proyecto no puedo superar la fase de transición al no reclutar desarrolladores nuevos. Por tanto, podemos concluir que los desarrolladores de Arla fallaron al no crear nuevas direcciones para el proyecto mediante la creación de módulos o subsistemas nuevos (**figura 7**, izquierda). Se observa un patrón decreciente, que confirma que los desarrolladores nuevos (y la comunidad alrededor del proyecto), aunque querían participar en el proyecto, no fueron adecuadamente estimulados por los desarrolladores principales.

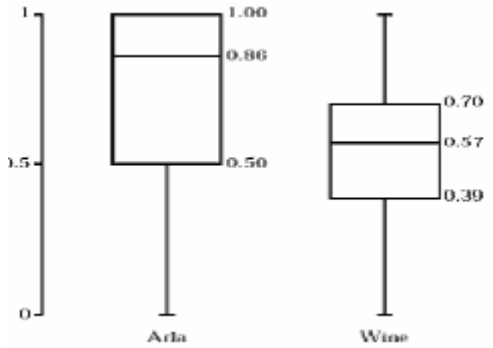

Figura 6. Descripción del esfuerzo dedicado por los desarrolladores nuevos.

En resumen, considerando la segunda cuestión de investigación planteada en los párrafos anteriores, encontramos evidencias similares para los dos proyectos: cuando un desarrollador se une a un proyecto de software libre, tiende a trabajar (añadir, modificar) en módulos nuevos más que en partes previamente existentes. Como conclusión para estos resultados, los desarrolladores principales del proyecto deberían perseguir de manera activa la transición a la fase de bazar: los desarrolladores nuevos tienen que ser estimulados añadiendo nuevas ideas o direcciones al proyecto.

4. Trabajo relacionado

En esta sección relacionamos este trabajo con otros en diferentes campos, específicamente en el estudio empírico de software y en la evaluación de esfuerzos. Dado que este trabajo se enmarca también en el campo más amplio del estudio de la evolución de proyectos de software libre, también son relevantes para este trabajo los estudios empíricos sobre el software libre.

Los estudios más tempranos sobre la evolución de software se realizaron sobre el sistema propietario OS/360 [4]. El estudio inicial se realizó sobre 20 versiones del OS/360, y los resultados de estas investigaciones y otras posteriores sobre software propietario, incluyen la clasificación SPE de programas y un conjunto de leyes de evolución de software [20].

Este trabajo se ha realizado de manera similar, pero evaluando tanto la entrada (esfuerzo) proporcionada al proyecto, como la salida (cambios en el código) lograda. Las cuestiones de investigación planteadas en este artículo derivan de [29], y se basan en la presencia de dos fases diferentes en el ciclo de vida de los proyectos de software libre, denominadas fases de catedral y de bazar [28]. Este hecho contrasta con la afirmación de Raymond de que la fase de bazar es típica de proyectos de software libre [15][28]: se realizó una evaluación empírica estudiando dos proyectos de software de tamaño grande, de los cuales sólo uno había realizado la transición a la fase de bazar y atraído a una comunidad grande de desarrolladores. Los autores piensan que tradicionalmente se ha puesto demasiado énfasis en proyectos de éxito que no necesariamente representan a una comunidad de software libre en su totalidad [13][15][16][26]. Pocos proyectos logran hacer la transición a la fase de bazar, atrayendo a una comunidad grande de desarrolladores activos en el proceso.

Tener un bazar grande alrededor del proyecto tiene muchas ventajas, como la capacidad de incorporar información de retorno de una base de usuarios y desarrolladores muy diversa. En cualquier caso, esto no implica que los proyectos que no han alcanzado la fase de bazar sean necesariamente fracasos: no quiere decir que no hayan tenido éxito o que sean de baja calidad. Es interesante señalar que al contrario de lo que afirma Raymond, existen algunas aplicaciones, como *GNU coreutils* y *tar*, que siguen claramente un modelo de catedral y forman parte de todos los sistemas Linux. De manera similar, existen muchos proyectos desarrollados por una única persona, con competencias excelentes, que muestran altos niveles de calidad. Debido a la falta de mejores teorías y de investigación empírica, la calidad de un proyecto de software se supone que se produce debido al proceso de revisión por pares en el bazar [1][26][28]. Sin embargo, no todos los proyectos de alta calidad presentan un bazar grande o un proceso de revisión por pares.

Un proyecto en la fase de catedral puede ser muy exitoso y tener mucha calidad [31]. Sin embargo, existen algunas restricciones que un proyecto en la fase de catedral debe afrontar, además de los problemas potenciales que serían menos graves si el proyecto tuviera una comunidad grande. Por ejemplo, aunque es posible para un único desarrollador escribir una aplicación de alcance limitado (como un cargador de arranque), sólo una comunidad puede completar el proyecto para llevarlo a entornos más amplios (como un entorno de escritorio). Además, un proyecto escrito por un único desarrollador puede ser de alta calidad pero también asume el riesgo de fracaso debido a que se sustenta en una única persona que trabaja como voluntario [23][25]. Tener una comunidad alrededor del proyecto lo hace más sostenible.

Estos argumentos muestran la falta de investigación en algunas áreas relacionadas con proyectos de software libre. Aunque se han asumido en el pasado algunos modelos para todos los proyectos de software libre, parece que está cada vez más claro que existe mucha variedad en los procesos de desarrollo [9][19][14]. Se necesitan mejores teorías acerca del éxito y la calidad de proyectos de software libre [24], además de comparaciones entre proyectos con diferentes grados de éxito y calidad. Finalmente, no debemos asumir que la fase de bazar es necesariamente la óptima para todos los proyectos de software libre, o que no está asociada con algunos problemas. Se acepta de manera general que es mejor que un proyecto de software libre sea abierto, pero si el proyecto es demasiado abierto puede estar demasiado expuesto a desarrolladores incompetentes o personas que desaniman a los que más contribuyen.

5. Conclusiones y trabajo futuro

Hasta este momento se han estudiado proyectos de software libre que tienen éxito, pero sin proporcionar pruebas empíricas de cómo han logrado este éxito. Para cubrir esta carencia, este artículo presenta el estudio empírico de dos proyectos de software libre, Arla y Wine, para ilustrar las diferentes fases en su ciclo de desarrollo y las comunidades que se forman alrededor de ellos. Se analizaron los ficheros de *ChangeLog*, de modo que se grabaron todos los cambios y nuevo código realizados por los desarrolladores durante varios años.

La principal hipótesis de este artículo es que las fases de catedral y bazar, tal y como las propuso y describió inicialmente Raymond [28], no son mutuamente excluyentes: los

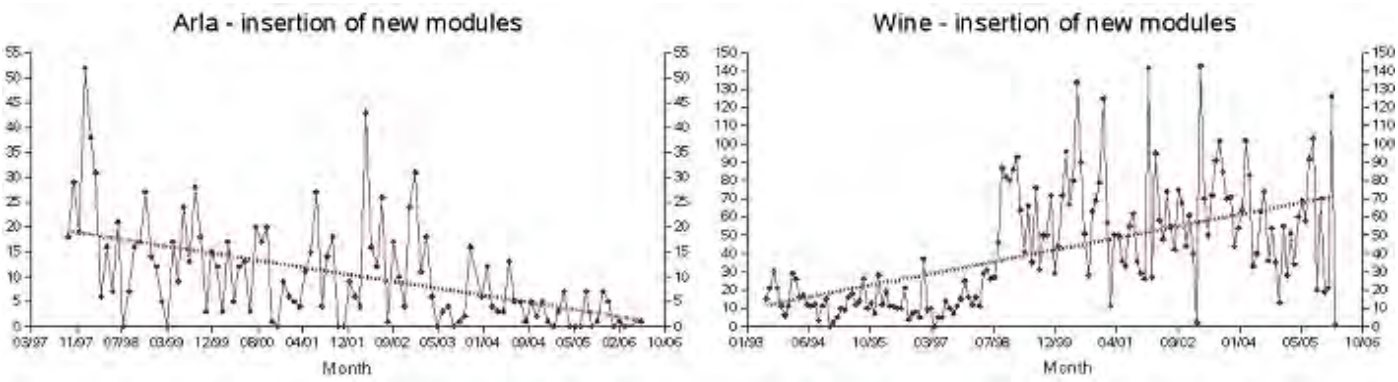

Figura 7. Creación de módulos nuevos en Arla y Wine.

proyectos de software libre comienzan en una fase de catedral, y de manera potencial migran a una fase de bazar. La fase de catedral se caracteriza por un desarrollo cerrado realizado por un grupo pequeño o un único desarrollador. La fase de bazar explota las ventajas de tener un número grande de voluntarios que contribuyen al proceso de desarrollo informando de defectos, solicitando nuevas características, arreglando fallos o proporcionando nuevas funcionalidades. La transición entre las dos fases es también una fase en sí misma, que necesita ser acondicionada mediante las acciones específicas del grupo de desarrolladores principales o del autor del proyecto. Esta fase de transición es fundamental para lograr un verdadero éxito y que el proyecto resulte popular.

Se propuso una cuestión a investigar para estudiar las diferencias entre las fases de catedral y bazar: el primer sistema (Arla) ha permanecido durante todo su ciclo de vida en una fase de catedral, debido a que sólo aportaba esfuerzo un limitado número de desarrolladores. Esto no debe entenderse como un signo de fracaso de un proyecto de software libre, sino como una oportunidad potencialmente perdida para establecer una comunidad próspera alrededor del proyecto. Por el contrario, el segundo sistema (Wine) sólo muestra una fase inicial similar a la de Arla: una segunda fase, más larga, presenta un número de desarrolladores activos creciente y una expansión continua del sistema.

Mediante la segunda cuestión a investigar, se centró el estudio en las preferencias de los desarrolladores nuevos que se unen al proyecto: los resultados de los dos proyectos muestran que los desarrolladores nuevos prefieren trabajar en módulos añadidos recientemente más que en módulos existentes previamente. En el caso del sistema Wine, los desarrolladores principales facilitaron la transición de fase añadiendo nuevos módulos en los que los desarrolladores pudieran trabajar. Por el contrario, los desarrolladores nuevos en el caso de Arla, aunque tuvieran ganas de trabajar en código nuevo, no encontraron nuevas direcciones en el proyecto, de modo que no se logró atraer a un número suficiente de nuevos desarrolladores.

Proponemos como trabajo futuro la replicación de este estudio en otros proyectos de software libre, especialmente en aquéllos que pertenezcan al mismo dominio de aplicación: los resultados tal y como se han obtenido en este estudio han analizado la comunidad desde un punto de vista neutral, esto es, sin considerar factores exógenos. El próximo paso será introducir estos factores en el estudio, y analizar proyectos grandes que están compitiendo en este momento por un recurso escaso, los desarrolladores.

Referencias

[1] A. Aoki, K. Hayashi, K. Kishida, K. Nakakoji, Y. Nishinaka, B. Reeves, A. Takashima, Y. Yamamoto. A case study of the evolution of jun: an object-oriented open-source 3d multimedia library. En actas de la *23rd International Conference on Software Engineering*, pp. 524-533, Toronto, Canada, 2001.

[2] B. Arief, C. Gacek, T. Lawrie. Software architectures and open source software – where can research leverage the most? En actas del *1 st Workshop on Open Source Software Engineering*, Toronto, Canada, 2001.

[3] R. Austen, G. Stephen. Evaluating the quality and quantity of data on open source software projects. En actas de la1st International Conference on Open Source Systems, Génova, Italia 2005.

[4] L. A. Belady, M. M. Lehman. A model of large program development. *IBM Systems Journal, 15(3)*:225-252, 1976.

[5] M. Bergquist, J. Ljungberg. The power of gifts: Organising social relationships in open source communities. *Information Systems Journal, 11(4)*:305-320, 2001.

[6] A. Capiluppi. Models for the evolution of OS projects. En actas de la *International Conference on Software Maintenance*, pp. 65-74, Amsterdam, Países Bajos, 2003.

[7] A. Capiluppi, P. Lago, M. Morisio. Evidences in the evolution of OS projects through changelog analyses. En actas del *3rd Workshop on Open Source Software Engineering*, Portland, OR, EEUU, 2003.

[8] A. Capiluppi, M. Morisio, J. F. Ramil. Structural evolution of an open source system: A case study. En actas del *12th International Workshop on Program Comprehension* (IWPC), pp. 172-182, Bari, Italia, 2004.

[9] K. Crowston, J. Howison. The social structure of free and open source software development. *First Monday, 10(2)*, 2005.

[10] M. Fischer, M. Pinzger, H. Gall. Populating a release history database from version control and bug tracking systems. En actas de la *International Conference on Software Maintenance*, pp. 23-32, Amsterdam, Países Bajos, 2003.

[11] K. F. Fogel. *Open Source Development with CVS*. The Coriolis Group, Scottsdale, Arizona, 1ª edición, 1999. ISBN 1-57610-490-7.

[12] D. M. German. An empirical study of fine-grained software modifications. pp. 316-325, Chicago, IL, EEUU, 2004.

[13] D. M. German. Using software trails to reconstruct the evolution of software. *Journal of Software Maintenance and Evolution:Research and Practice, 16(6)*:367-384, 2004.

[14] D. M. German, A. Mockus. Automating the measurement of open source projects. En actas del *3rd Workshop on Open Source Software Engineering*, Portland, OR, EEUU, 2003.

[15] M. W. Godfrey, Q. Tu. Evolution in open source software: A case study. En actas de la *International Conference on Software Maintenance*, pp. 131-142, San Jose, CA, EEUU, 2000.

[16] J. Howison, K. Crowston. The perils and pitfalls of mining SourceForge. En actas del *International Workshop on Mining Software Repositories* (MSR 2004), pp. 7-11, Edimburgo, UK, 2004.

[17] K. Johnson. *A descriptive process model for open-source software development*. Tesis de máster, Department of Computer Science, Universidad de Calgary, 2001. <http://sern.ucalgary.ca/students/theses/KimJohnson/thesis.htm>.

[18] N. Jorgensen. Putting it all in the trunk: Incremental software engineering in the FreeBSD open source project. *Information Systems Journal, 11(4)*:321-336, 2001.

[19] S. Koch, G. Schneider. Effort, cooperation and coordination in an open source software project: GNOME. *Information Systems Journal, 12(1)*:27-42, 2002.

[20] M. M. Lehman, L. A. Belady, editors. *Program evolution: Processes of software change*. Academic Press Professional, Inc., San Diego, CA, EEUU, 1985. ISBN:0-12-442440-6.

[21] L. Lopez, J. G. Barahona, I. Herraiz, G. Robles. Applying social network analysis techniques to community-driven libre software projects. *International Journal of Information Technology and Web Engineering, 11(4)*:321-336, 2006.

[22] T. Mens, J. F. Ramil, M. W. Godfrey. Analyzing the evolution of large-scale software: Guest editorial. *Journal of Software Maintenance and Evolution, 16(6)*:363-365, 2004.

[23] M. Michlmayr. Managing volunteer activity in free software projects. En actas del *2004 USENIX Annual Technical Conference, FREENIX Track*, pp. 93-102, Boston, EEUU, 2004.

[24] M. Michlmayr. Software process maturity and the success of free software projects. En K. Zielinski and T. Szmuc (editores), *Software Engineering: Evolution and Emerging Technologies*, páginas 3-14, Cracovia, Polonia, 2005. IOS Press. ISBN: 978-1-58603-559-4.

[25] M. Michlmayr, B. M. Hill. Quality and the reliance on individuals in free software projects. En actas del *3rd Workshop on Open Source Software Engineering*, pp. 105-109, Portland, OR, EEUU, 2003.

[26] M. Michlmayr, F. Hunt, D. Probert. Quality practices and problems in free software projects. En M. Scotto and G. Succi (editores), *Proceedings of the First International Conference on Open Source Systems*, pp. 24-28, Génova, Italia, 2005.

[27] A. Mockus, R. T. Fielding, J. D. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology, 11(3)*:309-346, 2002.

[28] E. S. Raymond. *The Cathedral and the Bazaar*. O'Reilly & Associates, Sebastopol, CA, EEUU, 1999. ISBN: 1565927249.

[29] A. Senyard, M. Michlmayr. How to have a successful free software project. En actas de la *11th Asia-Pacific Software Engineering Conference*, pp. 84-91, Busan, Corea del Sur, 2004. IEEE Computer Society.

[30] N. Smith, A. Capiluppi, J. F. Ramil. Agent-based simulation of open source evolution. *Software Process: Improvement and Practice, 11(4)*:423-434, 2006.

[31] I. Stamelos, L. Angelis, A. Oikonomou, G. L. Bleris. Code quality analysis in open-source software development. *Information Systems Journal, 12(1)*:43-60, 2002.

[32] L. Torvalds. The Linux edge. In C. DiBona, S. Ockman, and M. Stone, editors, Open Sources: *Voices from the Open Source Revolution*, pp. 101-111. O'Reilly & Associates, Sebastapol, CA, EEUU, 1999. ISBN: 1-56592-582-3.

Notas

[2] N. T.: En este fichero se escribe una entrada describiendo los cambios que se realizan en cada nueva versión del software.

[3] N.T.: *distinct* se usa en SQL para seleccionar registros que son distintos en su identificador.

[4] Las líneas de código producidas por un desarrollador están sesgadas por sus habilidades, por el lenguaje de programación y, en general, por el contexto de las modificaciones.

[5] N.T: *Many hands make light work* en el texto original.

Richard P. Gabriel
IBM Research

<rpg@{dreamsongs.com | us.ibm.com}>

Los bienes comunes como nueva economía y lo que esto significa para la investigación

Traducción: Alejandro Álvarez Ayllón (Oficina de Software Libre, Universidad de Cádiz).

Resumen: *suponga que toda la estructura social y comercial que está detrás de la creación de software esté cambiando, refiriéndonos a que se esté convirtiendo en bienes comunes[1], haciendo caer drásticamente sus costes. ¿Como podría el mundo cambiar y cómo podríamos reconocer esos cambios? El software no sería continuamente recreado por diferentes organizaciones, por lo que la "eficiencia" global de la producción de software se incrementaría espectacularmente; por lo tanto sería posible crear valor sin despilfarrar medios, la experimentación y el riesgo serían asumibles (y probablemente necesarios debido a que las compañías no podrían cobrar por duplicar su infraestructura), y el tamaño y complejidad de los sistemas existentes se incrementaría, quizá más allá de la comprensión humana. Tanto o más importante sería que las actividades de creación de software tendrían su origen en personas, organizaciones y disciplinas que, hoy en día, se consideran consumidores. Habría, en un sentido literal, un único sistema software existente, continuamente creciente y sería ésta una ecología gestionada por economistas, sociólogos, gobiernos, clubes, comunidades y multitud de disciplinas. Aparecerían nuevos modelos de negocio, quizá a una velocidad alarmante. ¿Cómo deberíamos diseñar nuestra investigación de forma que podamos observar y comprender estos cambios? Existen algunas evidencias de que el cambio está en proceso, como resultado de la adopción de software libre por parte de compañías que no son simplemente receptores agradecidos de los "regalos" procedentes de los evangelizadores del software libre, sino que son inteligentes ladrones que reenfocan las ideas y crean algunas nuevas de su propia cosecha.*

Palabras clave: *bienes comunes, código fuente, estrategia de negocio, licencia de software, nuevos modelos de negocio, producción de software, propiedad intelectual, software libre.*

Autor

Richard P. Gabriel se doctoró en Informática por la Universidad de Standford en 1981 y recibió un Master of Fine Arts en Poesía por el Warren Wilson College en 1998. Es un Ingeniero Distinguido en IBM Research, especializado en la arquitectura, diseño e implementación de sistemas ultragrandes y autosustentados, así como en las técnicas de desarrollo para construirlos. Fue Presidente y es actualmente miembro del Consejo de Dirección de Hillside Group, una organización sin ánimo de lucro que apoya a la comunidad de patrones de software mediante conferencias, publicación de libros y concesión de becas. Es autor de 4 libros y un libro de cordel de poesía. Ha ganado varios premios, incluyendo el AAAI/ACM Allen Newell. Es guitarrista líder en una banda de *rock and roll* y poeta.

1. Introducción

A veces aparecen novedades a una velocidad que ni investigadores ni profesionales son capaces, o están dispuestos, a observar. Un ejemplo ocurrido recientemente ha sido la *aparición de la aparición* [2] como un campo de estudio, en forma de ciencia de la complejidad. Durante siglos, fenómenos que actualmente se consideran como el centro de muchas disciplinas científicas no fueron observados o ni siquiera se consideraron algo serio.

Los investigadores y los profesionales del software libre están enamorados de las licencias, de las herramientas y su uso, del desarrollo comunitario, y de la eficacia y eficiencia con la que la metodología del software libre está produciendo software. Sin embargo, algo mucho mayor podría estar cambiando el panorama de la computación, y no sólo añadiendo algunos conocimientos a la ingeniería del software.

Durante los últimos 10 años, las empresas han estado contribuyendo con una considerable cantidad de software al mundo del software libre. Por ejemplo, Sun Microsystems ha calculado que, usando medios convencionales para la asignación de valor monetario al código, ha contribuido con más de mil millones de dólares en código. IBM y, posiblemente, otras grandes compañías no están muy atrás. Especialmente interesante es que Sun ha tomado la decisión de abrir todo el código de su software, y parece que sigue el camino correcto. Al mismo tiempo, no está depositando todas sus expectativas de beneficio en su hardware: también esperan ganar dinero con su software.

[1] **Nota del Editor:** la mejor traducción del término inglés *commons*, un término cada vez más en boga, de la mano del conocimiento abierto y del software libre, se halla todavía en discusión. La traducción más habitual es "bienes comunes", aunque hay quien opina que bastaría con traducirlo por "comunes", dada la primera acepción del término "común" en el diccionario de la Real Academia: *Dicho de una cosa: Que, no siendo privativamente de nadie, pertenece o se extiende a varios. Bienes, pastos comunes.*

[2] **Nota del Traductor:** "emergence of emergence", se trata de un juego de palabras.

2. Sun: Un caso de estudio (Breve visión general)

Sun comenzó en 1982 como una compañía basada en estándares abiertos y productos de consumo: BSD Unix, procesadores Motorola 68000, y TCP/IP. A finales de los 90 comenzó a experimentar con ideas relacionadas con el código abierto, y con verdadero código abierto: Jini (no es auténtico código abierto, pero es un experimento interesante sobre los conceptos y prácticas de código abierto, combinados con estrategias de mercado), Netbeans, Juxta, y OpenOffice fueron los primeros pasos, seguidos por Glassfish, Grid Engine, OpenSparc, OpenSolaris, Open Media Commons, y, más recientemente, Java. Si a esto añadimos Java.net se abren interesantes posibilidades. Sun está claramente experimentando con el concepto global de bienes comunes. OpenSparc es un diseño hardware que fue licenciado bajo una licencia libre, con el objetivo de crear mercado; Open Media Commons es ante todo un proyecto abierto de DRM, pero también enfoca la pregunta de qué significa la propiedad intelectual en el siglo XXI. Java.net es una especie de metacomunidad dirigida a la creación de mercados alrededor de Java. Se considera a Solaris y Java las joyas de la corona de Sun.

Durante esta etapa experimental en Sun (que todavía continúa) se puso especial énfasis en los modelos de negocio y de gobierno.

Sun está impulsando cuatro estrategias de negocio relacionadas con el software libre:

- Incrementar el volumen comprometiendo a los desarrolladores de software y reduciendo los obstáculos a su adopción.
- Compartir el desarrollo con desarrolladores externos y con proyectos libres ya establecidos que sean requeridos por los sistemas de Sun.
- Dirigirse a mercados emergentes cuyos gobiernos o intereses demanden software libre, como Brasil, partes de la Unión Europea, Rusia, India y China
- Irrumpir en mercados cerrados suministrando alternativas de software libre.

Sun ha realizado un interesante conjunto de observaciones sobre los cambios ocurridos desde que se monetizó el software. En los años 70, el software era fundamentalmente parte de un paquete hardware. Se compraban sistemas completos: hardware y software. En muchos casos, las compañías que vendían hardware suministraban el código de las aplicaciones a sus clientes para que lo adaptaran a sus necesidades, y no se consideraba extraño.

Entre las dos décadas entre los años 1980 y 2000, las compañías productoras de hardware comenzaron a desvincular su software, y empresas dedicadas a éste empezaron a vender software que realizara todo tipo de cosas, incluyendo sistemas operativos. Lo que estos dos periodos tienen en común es que el software era pagado cuando era adquirido. Y parecía no haber otra opción: si querías usar algo, primero tenías que comprarlo.

Con el software libre y la búsqueda de los modelos de negocio apropiados, esto puede cambiar, y ese cambio comenzó a principios de esta década de los 2000. El uso de software libre suele ser, por lo general, gratuito. No obstante, hay empresas y, a veces, individuos dispuestos o ansiosos por pagar a cambio de: soporte y mantenimiento, actualizaciones periódicas y solución de errores, indemnizaciones por responsabilidad y protección de patentes. En estos casos, la monetización puede ocurrir cuando se implanta el producto final. Esto implica que, en esos casos no cuesta nada explorar una idea para un producto concreto hasta que ésta sea completamente integrada para su venta o distribución. Entonces, si el productor lo desea, podrían comprarse uno o varios de estos servicios.

Aplazando algunos de los costes derivados de la aparición de nuevos productos y, posiblemente, nuevas empresas, pueden explorarse muchas más ideas considerando todo el mercado. Las barreras para la experimentación son muy bajas.

El repertorio completo de modelos de negocio que Sun ha identificado son:

- Suscripción (como ya hemos mencionado), incluyendo indemnizaciones y protección de patentes, extendiendo un paraguas de seguridad en lo referente a la propiedad intelectual que cubre a las partes que suscriben los servicios.
- Licencia dual, vendiendo las nuevas versiones del código, mientras que las antiguas son libres.
- Administración o custodia, esto es emplear un estándar para atraer a desarrolladores que lo empleen, y venderles otros productos y servicios a ellos mismos.
- Embebido, donde el código es parte de otra cosa, normalmente hardware, que es comercializada.
- Consultoría, donde se vende la experiencia de una persona o empresa en un determinado código, típicamente servicios de liderazgo de grupos de programación.
- Alojamiento, donde los servicios proporcionados por el software libre son ejecutados en servidores, vendiéndose el acceso a dichos servicios, o empleando otras maneras de obtener beneficio (como anuncios).
- Entrenamiento y formación sobre el código o sobre las metodologías de software libre.

Los teóricos del software libre de Sun ven en estas observaciones un ciclo virtuoso donde, encontrando un lugar para añadir valor en el código de estos bienes comunes, una compañía (o persona) puede crear un punto de monetización sin tener que invertir por sí sola en el código, produciendo de este modo un producto o servicio a un coste menor.

3. Lo que esto significa para el software

Suponga que Sun no está en una situación aislada, y que las compañías y otro tipo de organizaciones (incluyendo los individuos) se están preparando para cambiar sus modelos de negocio y de desarrollo de software basándose en el ciclo virtuoso descrito por Sun. ¿Cómo podrían las iniciativas completas de producción de software cambiar, y qué significaría esto para la ingeniería del software?

Analicemos el escenario. La gran mayoría del software sería un bien común, y estaría disponible para ser usado. No quedaría mucho de propietario. Habría presiones por parte de los clientes para que se realizaran unificaciones o simplificaciones. Por ejemplo, ¿por qué habría necesidad de tener múltiples sistemas operativos si no es por los diferentes requerimientos de escala, tiempo real, y distribución? Por otro lado, encontrar nuevo valor podría provocar que las empresas crearan bifurcaciones a partir del código base con el objetivo de crear plataformas o puntos de partida para categorías enteras de nuevas fuentes de valor. ¿Cómo podría romperse este equilibrio?

Como las barreras para emprender casi cualquier empresa serían tan bajas, habría muchos jugadores (incluyendo pequeñas empresas e individuos) capaces de ser un factor a tener en cuenta en cualquier área de negocio. Con más participantes, habría más oportunidades para que surgieran nuevas ideas e innovaciones. ¿Cómo irrumpirán estos en el mercado? ¿Intentarán quizás las empresas convertirse en repositorios de propiedad intelectual para ofrecer las mejores indemnizaciones? ¿Lograrán ser otras entidades, como universidades privadas o laboratorios de investigación, jugadores significados debido a que pueden ofrecer un potente porfolio de patentes que pueden usar para proteger a sus clientes? Teniendo en cuenta los enormes porfolios existentes, como los pertenecientes a IBM o Microsoft, parecería que éstos continuarían dominando; sin embargo, en nuevas áreas o nichos de mercado las pequeñas organizaciones, incluso individuos, podrían mantener las patentes clave.

Inmediatamente surgen algunas consideraciones obvias. ¿Qué pasa con las licencias? Actualmente, los grandes sistemas se obtienen a través de la unión de "subsistemas" (por expresarlo de alguna forma) publicados bajo distintas licencias. Lo que no se permite es mezclar piezas licenciadas con diferentes bases fuente. ¿Existirá una presión por poner todo el código bajo la misma licencia o la presión irá en otro sentido, es decir en crear nuevas licencias especializadas?

4. Lo que esto significa para la Ingeniería del Software

Como pocas compañías "poseerían" un sistema completo o aplicación sectorial, podría haber alguna presión en las bases de código para distanciar las APIs, protocolos, formatos de datos, etc. Y si esto ocurriera, ¿de donde vendría la presión opuesta? ¿Se encargarían las organizaciones de estandarización o se crearían estructuras de gobierno como la Fundación Apache o la IETF? ¿O surgirían empresas para definir las aplicaciones o estructuras de los sistemas tal como pasó con el ordenador personal a principios de los 80? En ese caso, IBM estableció una serie de normas de diseño determinando cuáles eran los componentes de un PC y cómo interactuaban [1].

Esto permitió al mercado formarse alrededor de los diversos componentes, lo que cambió la naturaleza del diseño de los sistemas informáticos. Hoy en día, esta forma de ver el diseño ha creado una nueva aproximación a los problemas de la ingeniería del software: *la ingeniería del software guiada por la economía.*

La enseñanza de la informática y del software cambiaría, ya que todo el código estaría disponible para su estudio (e, incluso, la

mejora formaría parte del proceso). En este sentido, los desarrolladores estarían mejor formados que nunca.

La programación dejaría de ser materia de inteligencia e invención para ser más un proceso de encontrar un código fuente existente que se aproxime y, o bien adaptarlo o adaptarse a él. Las licencias podrían ayudar o entorpecer. Con menos presión para crearlo todo a partir de cero, sería posible crear sistemas cada vez más grandes con un tamaño de equipo asequible. Esto afloraría las cuestiones y problemas asociados a los sistemas de muy gran escala (ULS, *Ultra Large Scale*). Citando una convocatoria de artículos para un taller sobre este tema [2][3]:

En pocas palabras, un incremento radical en la escala y complejidad demandará nuevas tecnologías y enfoques para todos los aspectos relacionados con la concepción, definición, desarrollo, implementación, uso, mantenimiento, evaluación, y regulación del sistema. Si los sistemas software en los que nos centramos hoy en día son comparados a edificios o a infraestructuras individuales, entonces los sistemas ULS son más parecidos a ciudades o redes de ciudades. Y, como éstas, tendrán nodos individuales complejos (similares a edificios o infraestructuras), así que deberemos continuar mejorando las tecnologías y métodos tradicionales; pero también mostrarán organización, y requerirán tecnologías y visiones fundamentalmente distintas a aquellas que son adecuadas a nivel de nodo. Los elementos software de los sistemas ULS presentan unos retos especialmente abrumadores. Desarrollar las tecnologías y enfoques necesarios a su vez requerirá investigaciones básicas y aplicadas diferentes a las que hemos realizado en el pasado. Facilitar el desarrollo de sistemas ULS (y, particularmente, sus elementos software) requerirá nuevas ideas en muchas disciplinas, incluyendo las ciencias de la computación y la ingeniería del software, pero también en otras como economía, urbanismo y antropología.

El cambio de software propietario a software basado en bienes comunes aceleraría la época de los sistemas ULS, lo que supondría un cambio cualitativo debido a su escala masiva. Si esto ocurre, se pondría de relieve la insuficiencia de nuestras herramientas, incluyendo metodologías y lenguajes de programación.

5. Lo que esto significa para la investigación

El hábito de la investigación en computación es profundizar mucho en las cuestiones planteadas. En cierto sentido, a los investigadores les encantan los rompecabezas. Gregory Treverton escribió esto acerca de los rompecabezas y los misterios en un artículo para y sobre los servicios de inteligencia [4].

Actualmente, los servicios de inteligencia se hallan en el negocio de la información, no solamente en el ámbito de los secretos, lo que supone un enorme cambio para la profesión. En las circunstancias de la era de la información, es hora para la comunidad de la inteligencia de distinguir entre rompecabezas y misterios. Los rompecabezas tienen soluciones concretas, sólo si tenemos acceso a la información (secreta) necesaria. Fueron la especialidad de los servicios de inteligencia durante la Guerra Fría: ¿cuántos misiles tiene la Unión Soviética? ¿cómo son de precisos? ¿cual es el potencial de guerra de Irak? Lo contrario a los rompecabezas son los "misterios", preguntas que no tienen una respuesta, en principio, definitiva. ¿Provocará Corea del Norte una nueva crisis nuclear? ¿Renunciará a su supremacía nacional el Partido Comunista de China? ¿Cuándo y dónde tendrá lugar el próximo ataque de Al Qaeda? Nadie conoce las respuestas a estas preguntas. Un misterio sólo puede ser "iluminado", pero no resuelto.

Encontrar evidencias del profundo cambio desde el software propietario al basado en bienes comunes en el mundo comercial es parte de un misterio, no es un rompecabezas, por lo que nuestros medios tradicionales podrían no ajustarse adecuadamente. Pero, desde luego, estudiar los métodos de ingeniería que los proyectos de software libre usan no iluminará el contexto general, refiriéndonos a cómo cambia la industria del software cuando las corporaciones cambian sus modelos de negocio para ajustarse a los bienes comunes. Las preocupaciones de las compañías no son las mismas que las de alguien que usa una herramienta de seguimiento de errores, edita el código con Emacs, y automatiza una parte complicada del proceso de prueba. Además, como existen ideas preestablecidas adheridas a ciertos ideales de la ingeniería, probablemente veamos nuevas ideas en las que los ingenieros del software no hemos pensado.

Pongamos un ejemplo, de nuevo extraído del caso de estudio Sun. Un fabricante japonés de automóviles contactó con el Open Source Group de Sun con el objetivo de aprender sobre el software libre. El grupo de la empresa japonesa, era el responsable de la creación del grueso de las aplicaciones de la compañía. Decían no tener ni un solo programador empleado directamente, sino que estaban subcontratados, principalmente en la India. Estaban preocupados de que las empresas indias con las que tenían contratos no eran tan expertas interpretando las especificaciones que se les había dado como sería de desear desde un punto financiero. Así que el vicepresidente del grupo estaba interesado en que el Open Source Group de Sun les ayudara a imponer una metodología (que no realidad) de software libre en las compañías subcontratadas, de forma que los encargados de las aplicaciones pudieran monitorizar el progreso, dirigir el trabajo nocturno, seguir las comunicaciones por e-mail, wikis, etc. para poder juzgar cómo progresaba el proyecto y poder tomar medidas sobre la marcha, tal vez empleando técnicas de software libre.

Ni una sola línea de código sería publicada; no habría licencia. Sería simplemente una herramienta de administración. Los investigadores que tendrían conocimiento y realizarían informes sobre dichas innovaciones y actividades vendrían de una escuela de negocios, o serían economistas. Puede que, incluso, antropólogos. Por tanto, lo que vemos es que se requería un punto de vista más amplio, más interdisciplinar, lo que se adecua a las conclusiones obtenidas por los autores del informe de los sistemas ULS.

Otra parte del cambio es que los investigadores en software podrían realizar "ciencia real", basada en la evolución del software, sistemas, marcos de trabajo, etc. Por ejemplo, empezaría a tener sentido controlar cuántas veces se codifica un conjunto de datos en su camino desde una base de datos hasta un cliente, un número que podría ser muy alto si el sistema que realiza la transmisión en general está hecho con un número determinado de marcos de trabajo desarrollados separadamente. Hoy en día, obtener esta información precisa de una relación especial con una corporación, una relación que, sospecho, es bastante rara de encontrar.

6. Conclusiones

Uno puede preguntarse si los movimientos de Sun son predictivos o iconoclastas. Si se trata de lo segundo, entonces Sun es simplemente pura curiosidad; pero si se trata de lo primero, nos compete a aquellos de nosotros que estamos en el límite entre la investigación y la práctica concebir una especie de programa de investigación que nos ayude a darnos cuenta de los cambios, de forma que podamos registrarlos y estudiarlos.

Referencias

[1] C. Baldwin, K. Clark. *Design Rules: The Power of Modularity*. MIT Press, 1999. ISBN: 0262024667.

[2] First ICSE Workshop on Software Technologies for Ultra-Large-Scale (ULS) Systems. <http://www.cs.virginia.edu/~sullivan/ULS1/>

[3] The Software Engineering Institute (SEI). *The Software Challenge of the Future: Ultra-Large-Scale Systems*, Junio de 2006. <http://www.sei.cmu.edu/uls/>.

[4] Gregory F. Treverto. Reshaping Intelligence to Share with "Ourselves". *Commentary No. 82, Canadian Security Intelligence Service*, Julio de 2003. <http://www.csis-scrs.gc.ca/en/publications/commentary/com82.asp>.

Israel Herraiz Tabernero, Juan José Amor Iglesias, Álvaro del Castillo San Félix
Grupo de Sistemas y Comunicaciones, Universidad Rey Juan Carlos, Madrid

<{herraiz, jjamor, acs}@gsyc.escert.urjc.es>

Software libre para la gestión de proyectos de investigación

Resumen: *tradicionalmente, los proyectos de investigación se gestionan de forma bastante opaca, siendo visibles desde fuera únicamente los documentos de resultados de carácter público. Tampoco se hace pública la información sobre la evolución del proyecto, excepto para su difusión en entornos limitados. Incluso, cuando un proyecto es gestionado entre varias entidades diferentes, los detalles sobre el desarrollo de las partes asignadas a cada entidad son privados y no conocidos por los demás socios participantes. En este sentido, los proyectos de investigación resultan similares a los modelos tradicionales cerrados de desarrollo. Los proyectos de investigación IST (Information Society Technologies - Tecnologías para la Sociedad de la Información) comparten ciertas características con los proyectos de software libre, como un modelo de desarrollo distribuido global y la posibilidad de teletrabajo. En este artículo presentamos una propuesta para gestionar proyectos de investigación de este tipo, adoptando métodos utilizados por la comunidad de software libre, utilizando para ello las herramientas libres usuales de esa comunidad. Nuestra metodología hace posible los flujos de comunicación entre los diferentes socios del proyecto, incluso si se encuentran geográficamente dispersos, y además permiten hacer públicos documentos seleccionados. Creemos ciertamente que esta nueva forma de gestionar proyectos tiene importantes ventajas sobre métodos tradicionales, y puede mejorar los procesos de gestión de los proyectos de investigación.*

Palabras clave: *gestión de la investigación, investigación, Programa Marco, software libre.*

Autores

Israel Herraiz Tabernero realiza estudios de doctorado en la Universidad Rey Juan Carlos. Su investigación está relacionada con la evolución de proyectos de software libre. En particular, está empleando análisis de series temporales y otras técnicas estadísticas para caracterizar y predecir la evolución de proyectos de software libre. Ha participado en diferentes proyectos financiados por el Programa Marco de la Comisión Europea (QUALOSS, FLOSSMetrics, Qualipso, CALIBRE). Además, ha colaborado también en otros proyectos financiados por empresas como Vodafone o Teléfonica. Ha participado en la redacción de manuales y documentos sobre cómo gestionar y poner en marcha proyectos de software libre. Por ejemplo, junto con Juan José Amor y Gregorio Robles escribió un manual para el Máster en Software Libre de la Universitat Oberta de Catalunya. En este momento disfruta de un contrato de Formación de Personal Investigador, concedido en 2005 por la Comunidad de Madrid para desarrollar la tesis doctoral en el estudio de proyectos de software libre. Ha sido revisor para, entre otras conferencias, la IEEE Africon 2007, y para la revista IEEE Transactions on Software Engineering. En estos momentos está coordinando el programa del Máster en Software Libre, título propio de la Universidad Rey Juan Carlos que se imparte en colaboración con la empresa Igalia y Caixa Nova, e imparte clases en diversas titulaciones en la citada Universidad.

Juan José Amor Iglesias es Licenciado en Informática por la Universidad Politécnica de Madrid y actualmente es jefe de proyectos en la Universidad Rey Juan Carlos, trabajo que compagina con sus estudios de doctorado en la misma universidad. Sus intereses investigadores abarcan diversos temas en la Ingeniería del Software Libre, centrándose en la estimación de actividad y de esfuerzo de los desarrolladores de proyectos de software libre. Su relación con la Comunidad de Software Libre ha sido intensa desde hace años: desde 1995 ha colaborado en diversas organizaciones relacionadas con el Software Libre, siendo cofundador de LuCAS, el portal de documentación libre más conocido e Hispalinux, la asociación de usuarios españoles de software libre. Colabora también con medios como Barrapunto.com y Linux+.

Álvaro del Castillo San Félix participa en el mundo del software libre desde hace ya más de 12 años donde tuvo sus primeras experiencias con el núcleo Linux. Desde entonces su vida laboral ha estado muy ligada a este mundo en el que ha participado en proyectos como la fundación de Barrapunto.com, GNOME Hispano o Mono Hispano. Es desarrollador de GNOME habiendo participando principalmente en el desarrollo de Planner y su integración con Evolution, y dentro de las empresas en las que ha trabajado, ha estado coordinando proyectos como la primera versión de LinEx o Compatiblelinux.org. Ha trabajado como director de Desarrollo y Arquitectura en la compañía LambdaUX Software Services que tenía entre sus objetivos llevar el software libre a los usuarios finales. Durante cuatro años fue profesor asociado de la Universidad Rey Juan Carlos. Actualmente, es coordinador de proyectos europeos dentro del grupo de investigación GSyC/LibreSoft de la Universidad Rey Juan Carlos.

1. Introducción

En el ámbito del Sexto Programa Marco de la Comisión Europea (6FP), el software libre ha empezado a causar interés, con el fin de ganar conocimientos y mejorar el desarrollo del software. Muchas de las buenas prácticas del software libre pueden adaptarse para la gestión de entornos complejos, aunque no tengan relación con este tipo de software. Uno de esos entornos sería el de los proyectos de investigación.

En un proyecto de investigación, personas de diferentes países trabajan de forma coordinada para conseguir llevar a cabo los objetivos del proyecto. Estas personas necesitan trabajar muchas veces en los mismos documentos o, si se trata de desarrollo de software, en los mismos ficheros de código fuente; estando estas personas geográficamente dispersas, siendo necesario que trabajen de forma coordinada con todos los demás socios, con el fin de hacer un desarrollo eficiente.

Sin embargo, los proyectos de investigación suelen llevarse de forma bastante opaca. Cada socio no está informado del trabajo que realizan los demás, y el público general accede solo a publicaciones seleccionadas, llamadas *entregables públicos* (del inglés *deliverable*); existiendo también una colección de estos entregables que no se ponen a disposición del público.

Esto es un problema importante. En primer lugar, los proyectos de investigación, al menos aquellos del Sexto Programa Marco, son financiados con fondos públicos. Los resultados, por tanto, deberían ser completamente públicos (y no solamente una selección de éstos).

Por otro lado, al menos en los proyectos del Sexto Programa Marco relacionados con el software libre, muchos comparten objetivos, y necesitan a veces las mismas fuentes de información. Estos proyectos podrían beneficiarse de otros similares si pudieran acceder a documentación interna y a otras informaciones generadas. Pensando en una analogía con el mundo del software libre, si un programador sabe que puede reutilizar un

trozo de código de otro proyecto, simplemente acudirá a este, copiará ese trozo y lo adaptará a sus propias necesidades.

Por todas estas razones, proponemos una metodología para adoptar las prácticas realizadas por la comunidad de desarrollo de software libre, en la gestión de proyectos de investigación. Nuestra metodología está pensada para ser adoptada por cada uno de los socios de estos proyectos.

El resto del artículo se organiza como sigue. En la **sección 2** se describen las características de un proyecto de desarrollo típico. En la **sección 3** se describen las necesidades de un proyecto de investigación y una propuesta sobre las herramientas que sirven para esas necesidades. En la **sección 4** se explica cómo organizar el trabajo y el entorno para hacer el trabajo, basado en la experiencia de nuestro grupo. Finalmente, en la **sección 5** se incluyen algunas conclusiones.

2. Estructura de un proyecto de investigación

En esta sección describiremos la estructura de un proyecto de investigación típico. Tomaremos como ejemplos algunos proyectos del Sexto Programa Marco en los que hemos participado.

Los proyectos de investigación son propuestos y desarrollados normalmente por una serie de socios localizados en diferentes países. Esto origina el primer problema: el idioma. Normalmente, se elige el inglés como idioma para la comunicación entre socios, y para escribir los documentos resultantes del proyecto (sean internos o públicos).

El trabajo se divide en *paquetes de trabajo*. Cada socio puede liderar uno o más paquetes de trabajo, y todos los socios participarán en al menos un paquete de trabajo. Dentro de estos paquetes de trabajo encontraremos tanto *hitos* como *entregables*. Los hitos son fechas clave, en las que cierto trabajo debe ser entregado. Los entregables son documentos (aunque también puede ser un software, una base de datos, etc) que son parte del producto final del proyecto. Algunos entregables son públicos, mientras que otros son internos de los socios y otros se destinan a ser entregados al *patrocinador* del proyecto (que, en el caso del Sexto Programa Marco, es la Comisión Europea).

El trabajo requerido para realizar los entregables debe ser ejecutado habitualmente por varios socios, de forma coordinada. Normalmente, uno de los socios actúa como coordinador, siendo el que administra la parte económica, y asegurándose de que todos los trabajos que deben realizar los socios son finalizados de acuerdo con el plan de trabajo y en su debida fecha.

La coordinación es clave en los proyectos de investigación: cada socio tiene que trabajar coordinadamente con los demás, y es muy importante que todos los socios estén al tanto del trabajo del resto. Por supuesto, cada uno de los socios también es responsable de su propio trabajo y de las fechas de entrega de éste.

3. Necesidades de un proyecto de investigación

Se necesita el apoyo de ciertas herramientas para realizar el trabajo descrito anteriormente. En primer lugar, expondremos ciertos conceptos sobre lo que necesita un proyecto de investigación, y luego daremos una lista de programas libres que pueden utilizarse para satisfacer estas necesidades.

■ Sitio web

En primer lugar, se necesita un sitio web para cubrir la necesidad de *diseminar* los resultados públicos del proyecto. Suele usarse un CMS (Sistema de Gestión de Contenidos), para facilitar la participación de todos los socios en la gestión del sitio web para publicar documentos.

Tomando la definición de Wikipedia sobre CMS [1]: *Un Sistema de gestión de contenidos (Content Management System, en inglés, abreviado CMS) permite la creación y administración de contenidos principalmente en páginas web. Consiste en una interfaz que controla una o varias bases de datos donde se aloja el contenido del sitio. El sistema permite manejar de manera independiente el contenido y el diseño. Así, es posible manejar el contenido y darle en cualquier momento un diseño distinto al sitio sin tener que darle formato al contenido de nuevo, además de permitir la fácil y controlada publicación en el sitio a varios editores. Un ejemplo clásico es el de editores que cargan el contenido al sistema y otro de nivel superior que permite que estos contenidos sean visibles a todo el público.*

El sitio web debe además distinguir entre documentos públicos y privados, haciendo que estos últimos estén accesibles únicamente a determinados grupos de usuarios (normalmente, los socios del proyecto).

■ Listas de correo electrónico

En segundo lugar, con el fin de facilitar la comunicación entre socios, se necesitan las listas de correo electrónico. A veces es buena idea tener varias listas para cada proyecto. Una de éstas debería incluir a todas las personas involucradas en el proyecto. Otra debe incluir solamente a los máximos responsables del mismo. En nuestra opinión, algunas decisiones estratégicas sobre el proyecto deben ser discutidas únicamente por los responsables y no por todo el grupo de investigadores involucrados.

Si el grupo de personas que trabajan conjuntamente es mayor de 4 ó 5, está plenamente justificado usar una lista de correo. Pero además, las listas proporcionan otras ventajas como el archivado de mensajes, que puede ayudar a nuevos miembros del proyecto a adquirir el contexto necesario. Normalmente, habrá una lista de correo para el grupo de responsables máximos y otra para todos los involucrados. Si el grupo de investigación es muy pequeño, con una sola lista podría ser suficiente.

■ Sistema de control de revisiones

Otro requisito es tener un repositorio de documentos de trabajo, programas o ficheros de cualquier clase en general, con capacidad para controlar revisiones. Esto facilita la recuperación de versiones antiguas de los documentos y trabajar varias personas a la vez en el mismo documento de forma coordinada. Es también un lugar donde cualquiera puede encontrar cualquier documento o fichero del proyecto. El repositorio no está pensado para la publicación de entregables, sino más bien para realizar el trabajo de desarrollo cooperativo. Las capacidades de control de revisiones son cruciales, ya que diferentes personas pueden trabajar en el mismo documento sin problemas; y también facilita la recuperación de versiones antiguas.

■ *Wiki*

Otra herramienta interesante son los *wikis*, que facilitan el trabajo con documentos utilizando un navegador web. La página de Wikipedia nos dice sobre los *wikis* [2]:

Un wiki (o una wiki) [...] es un sitio web colaborativo que puede ser editado por varios usuarios. Los usuarios de una wiki pueden así crear, modificar, borrar el contenido de una página web, de forma interactiva, fácil y rápida; dichas facilidades hacen de la wiki una herramienta efectiva para la escritura colaborativa.

Los *wikis* permiten trabajar en documentos a través de la web, utilizando únicamente un navegador. Está pensado para documentos breves. En nuestra opinión, no es una buena herramienta para escribir entregables, pero sí es apropiada para organizar una base de conocimiento acerca del proyecto para el equipo investigador.

■ Sistema de gestión de fallos

Finalmente, puede ser interesante utilizar un *sistema de gestión de fallos*. La Wikipedia nos da esta definición sobre estas herramientas [3]:

Un sistema de seguimiento de errores es una aplicación informática diseñada para asistir a los programadores y otras personas involucradas en el desarrollo y uso de siste-

mas informáticos en el seguimiento de los defectos de software. [...].

Uno de los componentes principales de un sistema de seguimiento de errores es la base de datos donde se almacenan los hechos e historia de un fallo de software. Los hechos pueden ser una descripción detallada del fallo, la severidad del evento, forma de reproducirlo y los programadores que intervienen en su solución easí como infromación relacionada al proceso de administración de la corrección del fallo como puede ser personal asignado, fecha probable de remedio y código que corrige el problema.

En los proyectos de investigación, el sistema de seguimiento de fallos puede ser usado por los gestores del proyecto para asignar tareas a las personas, y para seguir la evolución del trabajo. Esto hace la vida más fácil al jefe de proyecto, y permite a todo el mundo estar al tanto del día a día del trabajo de las demás personas del grupo.

En nuestra opinión, estas son las herramientas fundamentales que deben usar los grupos que desarrollan proyectos de investigación. Esto facilita la organización del trabajo y permite tener un seguimiento bastante preciso del día a día.

3.1. Herramientas que satisfacen estas necesidades

Sitio web

Para el primer requisito, un sitio web con funcionalidad CMS, hay muchas opciones disponibles basadas en software libre. Se puede encontrar una lista de soluciones en [5]. Muchas de ellas pueden satisfacer todas las necesidades de un grupo de investigación, como la existencia de perfiles de acceso a los documentos (públicos, privados, etc). Sin embargo, nuestra recomendación no está incluida en la lista anterior. Nosotros recomendamos el uso de Plone [9].

Listas de correo electrónico

Al respecto de las listas de correo, recomendamos *GNU Mailman*, que es un completo paquete de gestión de listas. Tiene una interfaz web para administración, y permite archivar los mensajes y acceder al archivo mediante una interfaz web. Se puede encontrar más información en su página de la Wikipedia [4].

Sistema de control de revisiones

Para construir un repositorio de ficheros con control de revisiones recomendamos *Subversion* [6] (también conocido como SVN). El principal motivo es que Subversion se integra bastante bien con otras herramientas, y puede utilizarse con clientes Webdav, disponibles en muchos navegadores de archivos de casi cualquier sistema operativo. En cualquier caso, para acceder al repositorio es mejor utilizar un cliente específico de Subversion, para poder aprovechar todas sus capacidades.

Wiki

Para las *wikis*, en nuestra opinión la solución a elegir es el popular MediaWiki, que es el sistema utilizado por ejemplo en Wikipedia [7].

Sistema de gestión de fallos

Finalmente, para el sistema de gestión de fallos recomendamos Trac [8]. Lo más interesante de Trac es que integra, además del sistema de gestión de fallos, una *wiki*, un repositorio Subversion, y una línea de tiempo para la planificación de proyectos. Por ejemplo, cuando se envía un informe de fallo (llamado *ticket* en Trac), se puede asociar a un hito de planificación, a un documento mantenido en su repositorio de Subversion o a las personas involucradas en él. La información está disponible en varios formatos accesibles por web: formato texto y RSS. En particular, con la traza basada en RSS se pueden realizar seguimientos e informes de actividad. Hay, sin embargo, muchas alternativas para sistemas de gestión de fallos. En [10] encontraremos una lista extensa de sistemas de este tipo, clasificados según diferentes categorías.

4. Organización del trabajo

En esta sección presentaremos nuestra experiencia de uso de las herramientas anteriores para satisfacer las necesidades de nuestra participación en algunos proyectos europeos. En primer lugar, estas son las herramientas que seleccionamos:

- Zope, para el sitio web.
- Mailman, para las listas de correo.
- Subversion, como sistema de control de versiones.
- Trac, para el *wiki* y el sistema de gestión de fallos. El repositorio de versiones Subversion está integrado con Trac.

Para el sitio web, hemos desarrollado nuestra propia solución, utilizando Zope como plataforma. El sitio web no cumple con todos los requisitos (repositorio de documentos, perfiles para diferentes clases de usuarios, etc.). Sin embargo, en el ámbito de cada proyecto, sí se han adoptado herramientas que cumplen los requisitos. Por ejemplo, en algunos proyectos se utiliza Plone (un CMS que se ejecuta sobre Zope).

Para la gestión de cada proyecto, hemos determinado usar tres listas de correo:

- Una lista donde todas las personas involucradas en el proyecto pueden participar.
- Una lista donde solo los gestores del proyecto pueden participar.
- Una lista donde se suscribe a todos los socios. Ésta es útil cuando el Trac da soporte al trabajo de un solo equipo, pero en el proyecto participan varios equipos de diferentes instituciones.
- Una lista para los seguidores de los *commits*. Cada vez que alguien envía una nueva versión de un fichero al sistema de control de revisiones, se envía automáticamente un mensaje con la descripción del envío a esta lista. Esto permite a quien se suscriba estar al tanto de todos los cambios realizados en el repositorio.

Para las listas de correo utilizamos Mailman. Las listas se configuran habitualmente como moderadas para personas no suscritas (es decir, los mensajes de personas no suscritas quedan retenidos) para evitar correo no deseado (*spam*). Algunas listas, como las destinadas a los gestores del proyecto, se pueden configurar además como privadas (es decir, nadie puede leer los mensajes ni suscribirse sin autorización).

Para la *wiki* y el sistema de seguimiento de fallos hemos adoptado Trac. También hemos integrado en ese Trac el repositorio de control de versiones Subversion. Utilizamos la *wiki* como base de conocimientos del proyecto, y el sistema de seguimiento de fallos para controlar, asignar y monitorizar el trabajo en el proyecto. Además, los mensajes de correo electrónico generados por esta herramienta (cuando se crean, modifican o cierran *tickets* del Trac) son enviados a la lista utilizada por el grupo de trabajo.

Cuando se gestionan varios proyectos simultáneamente, y cada uno cuenta con su propio Trac, es muy útil integrar el seguimiento de la actividad de todos los proyectos en un "*planeta*" (un programa de agregado de RSS[1]). El *planeta* es muy útil para ver la actividad más reciente de todos los proyectos, en una sola página web, para lo que se importan todos los RSS que representan la línea de tiempo de cada uno de los Trac.

Nuestro equipo investigador ha modificado el software Planet con el fin de incluir *indicadores de actividad*. Un *indicador de actividad* es una caricatura que representa la actividad reciente de un proyecto. Por ejemplo, si un proyecto ha registrado actividad en la última hora, la caricatura corresponde a una cara que ríe. Pero cuando la actividad más reciente es del día anterior, la expresión es más seria. Hay diferentes rostros hasta llegar al peor caso, que representa a cualquier proyecto que no haya tenido actividad en el último mes.

Un ejemplo de este sitio web se encuentra en la **figura 1**. En la parte izquierda vemos una lista de eventos recientes, clasificados por proyecto. En la parte derecha, vemos una lista de todos los proyectos, con la indica-

Figura 1. *Planeta*, con la actividad reciente de todos los proyectos, e indicadores de actividad.

ción de actividad reciente. También se incluye un resumen de los indicadores de actividad utilizados.

El fichero RSS de cada herramienta Trac es el que integramos en el sitio web llamado *planeta*. Este fichero contiene una entrada por cada evento sucedido en el Trac. Puede ser por creación o finalización de un *ticket*, por un cambio en la *wiki* (crear, modificar o borrar una página) o por un cambio registrado en el sistema Subversion asociado (adición, borrado o modificación de un fichero).

El *planeta* ha resultado muy útil para nuestro grupo. En primer lugar porque permite a cualquiera que trabaje en el grupo estar al tanto de toda la actividad reciente en todos los proyectos, y quién está participando. En segundo lugar, porque los indicadores de actividad actúan como "motivadores" de los diferentes equipos que trabajan en cada proyecto. Por ejemplo, si un grupo se pone a la cabeza de la actividad reciente, otro grupo se puede ver motivado a potenciar su actividad para recuperar el liderazgo.

En resumen, hemos puesto en producción todas las herramientas indicadas. Al integrarlas, hemos obtenido los mejores resultados de usarlas. Por ejemplo, nuestros sitios con Trac integran *wiki*, repositorios Subversion y sistemas de seguimientos de fallos. Además, tenemos listas de correo que reciben un mensaje cada vez que el Trac registra algún cambio. En un sitio web unificado podemos tener constancia de la actividad de todos los proyectos. Esto permite a cualquiera estar al tanto del trabajo reciente, realizado por el resto de grupo, independientemente de que trabajen en esos proyectos o no. Y además los indicadores de actividad resultan dinamizadores de la actividad de los proyectos, al comparar la actividad de unos proyectos con otros.

Sin embargo, hemos de admitir que debido a requisitos ajenos a nosotros, no podemos abrir el acceso a nuestras herramientas todavía. Por ahora estamos disfrutando de las ventajas de poder compartir el conocimiento con el resto de socios. Estamos haciendo esfuerzos para conseguir abrirlo a más usuarios.

5. Conclusiones

En este artículo, hemos presentado una metodología y una serie de herramientas, destinados a organizar el trabajo en proyectos de investigación así como a sus distintos grupos participantes. Nuestra metodología está basada en los métodos y herramientas utilizados para gestionar las comunidades de software libre.

Los proyectos de investigación deberían ser abiertos, como el software libre, por dos razones: están financiados con fondos públicos, por lo que los resultados deberían ser públicos; y porque algunos proyectos se pueden beneficiar de la colaboración con otros proyectos de investigación, haciendo más eficiente el gasto de fondos públicos.

La metodología propuesta permite que toda la información sea de libre disposición. No solamente los entregables finales sino también todo el trabajo desarrollado durante la vida del proyecto.

Las herramientas propuestas permiten hacer seguimiento de todo el trabajo realizado durante la vida del proyecto. Estos repositorios de información acerca del proyecto de investigación abren nuevas direcciones para mejorar la eficacia de este tipo de proyectos. Por ejemplo, para la vigilancia tecnológica automática de los proyectos de investigación, basada en las trazas disponibles en los repositorios (sitio web, listas de correo, sistema de control de versiones, gestor de fallos, etc).

Por otro lado, las herramientas propuestas y los métodos permiten filtrar parte de la información a determinados grupos de usuarios o al público en general, basándose en determinados perfiles de niveles de acceso a la información.

Otro punto interesante de esta metodología es que hace posible trabajar de forma remota, ya que toda la información gestionada por las herramientas, así como las propias herramientas están accesibles de manera remota. Esto permite que los participantes puedan continuar trabajando incluso cuando realizan viajes a reuniones de proyecto o estancias de investigación en otros centros. Estas herramientas también permiten la coordinación entre diferentes socios, normalmente localizados en diferentes países.

El único inconveniente de nuestra propuesta es que es solo válida para proyectos de tecnologías de información y comunicaciones. Por ejemplo, para proyectos sobre química o biología, es necesario que las personas trabajen en un lugar común. En cualquier caso, las herramientas podrían ser de utilidad para gestionar el control de los entregables, por

ejemplo, aunque el teletrabajo en determinados proyectos no sea posible.

En un trabajo futuro, utilizaremos las trazas de los repositorios de los proyectos en los que trabajamos para construir una plataforma de vigilancia tecnológica de la investigación en software libre. Estamos también planeando la construcción de herramientas para automatizar los informes de actividad y participación, basándonos en la información proporcionada por los repositorios. En un futuro próximo, consideraremos también abrir todos nuestros repositorios, para poner la información a disposición del público. Por ahora, al estar trabajando con otros socios, la decisión no está en nuestras manos. En cualquier caso, todos los resultados públicos de nuestros proyectos se ofrecen mediante licencias de uso no restrictivas, tanto en el caso del software como de los documentos.

Referencias

[1] <http://es.wikipedia.org/wiki/Cms>.
[2] <http://es.wikipedia.org/wiki/Wiki>.
[3] <http://es.wikipedia.org/wiki/Sistema_de_seguimiento_de_errores>.
[4] <http://en.wikipedia.org/wiki/GNU_ Mailman>.
[5] <http://www.opensourcecms.com>.
[6] <http://en.wikipedia.org/wiki/Subversion_(software)>.
[7] <http://en.wikipedia.org/wiki/Mediawiki>.
[8] <http://en.wikipedia.org/wiki/Trac>.
[9] <http://plone.org>.
[10] <http://en.wikipedia.org/wiki/Comparison_of_issue_tracking_systems>.

Nota

[1] El más utilizado, escrito en Python, se encuentra disponible en <http://www.planetplanet.org/>.

Javier Carmona Murillo[1], José Luis González Sánchez[1], Manuel Castro Ruiz[2]

[1]Departamento de Ingeniería de Sistemas Informáticos y Telemáticos, Escuela Politécnica de Cáceres, Universidad de Extremadura; [2]SADIEL, S. A. Mérida.

<{jcarmur, jlgs}@unex.es>, <mcruiz@sadiel.es>

Innovación tecnológica en comunicaciones móviles desarrollada con Software Libre: Campus Ubicuo

Resumen: *las redes móviles constituyen hoy en día uno de los sectores de mayor crecimiento en el campo de las comunicaciones. La creciente demanda de servicios y la necesidad de movilidad por parte de los usuarios ha modificado el modelo tradicional de conectividad a Internet, que ya no se basa únicamente en el acceso a través de redes fijas. Partiendo del auge de los nuevos dispositivos portátiles y de las actuales redes de acceso inalámbricas, proponemos un sistema enfocado a proporcionar movilidad y ubicuidad en el entorno de un campus universitario extensible a todo tipo de organizaciones. Este artículo presenta Campus Ubicuo, un proyecto de investigación, desarrollo e innovación tecnológica en el campo de las comunicaciones móviles. Los esfuerzos del proyecto se centran en ofrecer ubicuidad a los usuarios del sistema por medio de servicios telemáticos avanzados sobre tecnologías inalámbricas implementados con Software Libre. Además, el desarrollo de Campus Ubicuo ha permitido realizar tareas de investigación relacionadas con la movilidad en redes IP y con las interferencias experimentadas por las diferentes tecnologías inalámbricas.*

Palabras clave: *Mobile IP, movilidad, PDA, Software Libre, ubicuidad, 3G.*

Autores

Javier Carmona Murillo es estudiante de doctorado en el departamento de Ingeniería de Sistemas Informáticos y Telemáticos de la Universidad de Extremadura. Allí recibió el título de Ingeniero en Informática (2005). Sus principales temas de investigación son las comunicaciones en banda ancha, la provisión de QoS en redes móviles y el soporte de movilidad en IP. Simultáneamente a sus estudios de doctorado, está trabajando en el proyecto Campus Ubicuo y realiza tareas de apoyo a la investigación en el grupo de investigación de Ingeniería Telemática Aplicada y Comunicaciones Avanzadas (GÍTACA) <http://gitaca.unex.es>.

José Luis González Sánchez es Diplomado en Informática, Ingeniero en Informática y, desde 2001, Doctor Ingeniero en Informática por la Universidad Politécnica de Cataluña. Ha desarrollado su labor profesional en diversas empresas públicas y privadas y, desde 1995, pertenece al Personal Docente e Investigador de la Universidad de Extremadura, estando adscrito al área de Ingeniería Telemática del Departamento de Ingeniería de Sistemas Informáticos y Telemáticos. Es el investigador responsable del grupo de investigación GÍTACA (Grupo de Ingeniería Telemática Aplicada y Comunicaciones Avanzadas) de la Universidad de Extremadura y ha sido autor de múltiples libros, artículos, trabajos y proyectos de investigación relacionados con la informática y las comunicaciones. En la actualidad es el Presidente del Colegio Profesional de Ingenieros en Informática de Extremadura (CPIIEx).

Manuel Castro Ruiz es Ingeniero Técnico en Informática de Sistemas por la Universidad de Extremadura. Ha cursado el Máster en Dirección Estratégica y Gestión de la Innovación por la Universidad Autónoma de Barcelona y en la actualidad es estudiante de Ingeniería Informática por la UNED. En los últimos diez años viene desarrollando su carrera profesional en distintas consultoras TIC en Madrid, Andalucía y actualmente en Extremadura, dirigiendo proyectos de desarrollo de Sistemas de Información para grandes empresas y organismos públicos. Actualmente es el delegado de la consultora Sadiel para Extremadura.

1. Introducción

En los últimos años muchos investigadores han fijado su interés en el Software Libre como base para el desarrollo de sus propuestas y contribuciones. En el campo de la informática y las comunicaciones este interés es, particularmente, útil en proyectos de desarrollo e innovación tecnológica si tenemos en cuenta el modelo de desarrollo característico del Software Libre [1].

Por otra parte, la actual demanda de conectividad ubicua, independientemente del lugar, instante o medio de acceso utilizado, ha convertido a las comunicaciones móviles en una necesidad. La movilidad de los usuarios de las redes sin dependencias de hilos, la Calidad de Servicio (QoS, *Quality of Service*) obtenida en las comunicaciones y, por último, la seguridad de la información transmitida por dichas redes, son tres características que Campus Ubicuo[1] propone interrelacionar a través de un cuarto aspecto no menos importante, como es la aplicación de todas las ventajas que aporta el Software Libre.

El proyecto Campus Ubicuo es el resultado de la experiencia de varios años de investigación en comunicaciones, movilidad y Software Libre, y aparece con el objetivo de aprovechar las posibilidades de movilidad y portabilidad de dispositivos como los PDA (*Personal Digital Assistant*), teléfonos móviles y ordenadores portátiles, para ofrecer servicios a los usuarios del sistema a través de tecnologías de comunicaciones inalámbricas como GSM (*Global System for Mobile Communications*), GPRS (*General Packet Radio Service*), UMTS (*Universal Mobile Telecommunications System*), Bluetooth o Wi-Fi (*Wireless Fidelity*).

Este artículo presenta el trabajo desarrollado en el proyecto así como las tareas de investigación llevadas a cabo, y está organizado de la siguiente forma: En la **sección 2** se describen los desarrollos en los que se ha dividido el proyecto, mientras que las tareas de investigación relacionadas con Campus Ubicuo aparecen en la **sección 3**. Finalmente, las conclusiones y el trabajo futuro se presentan en la **sección 4**.

2. Desarrollo de Campus Ubicuo

2.1. Arquitectura del sistema

La **figura 1** presenta la arquitectura global del sistema propuesto. Una plataforma sobre la que ofrecer servicios de ubicuidad a los usuarios, a través de las actuales tecnologías de comunicaciones móviles.

En la imagen aparecen los cuatro pilares fundamentales sobre los que se asienta esta arquitectura: movilidad, QoS, seguridad y Software Libre.

En los apartados siguientes se describe cada uno de los subproyectos en los que se ha dividido el desarrollo del proyecto.

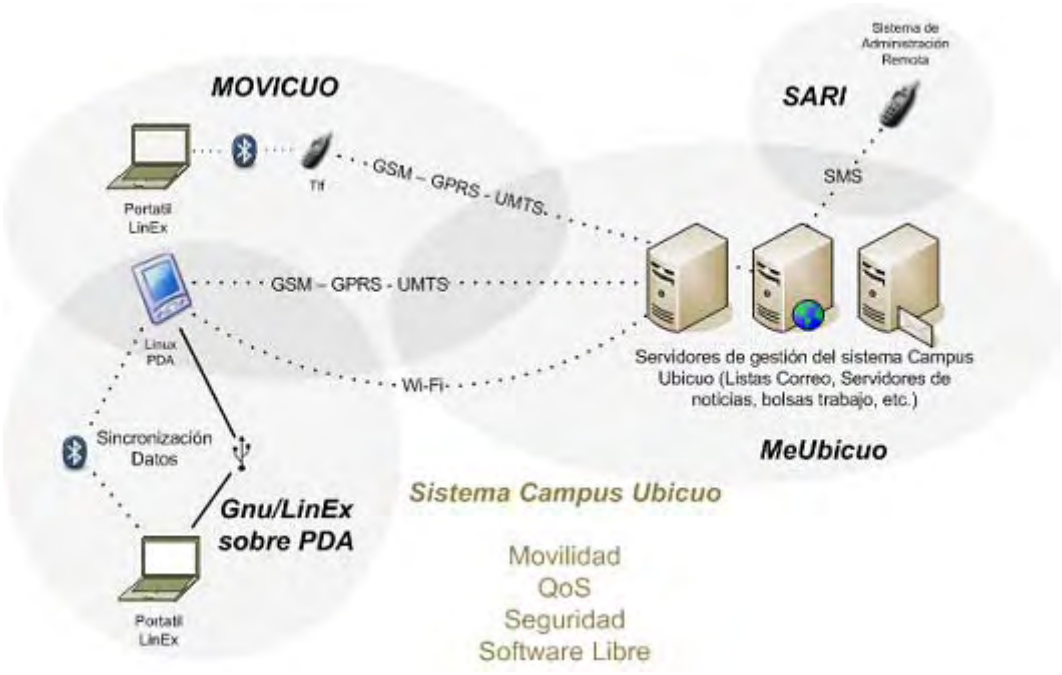

Figura 1. Arquitectura de Campus Ubicuo.

2.1.1. Movicuo

Campus Ubicuo dedica gran parte de sus esfuerzos al desarrollo de aplicaciones que aporten movilidad a los usuarios de sistemas operativos libres como GNU/Linux. Movicuo es el subproyecto dedicado a esta tarea.

Hoy en día, una de las tecnologías que tiene a la movilidad como una característica inherente es la de las redes de telefonía móvil. GPRS (2.5G) complementa el diseño de GSM (2G) con una red de conmutación de paquetes para el tráfico de datos y soporta la negociación de parámetros de QoS. Sin embargo, las tasas de velocidad alcanzadas en GPRS (171,2 kbps en situaciones ideales) [2] no son adecuadas para determinados servicios como el tráfico multimedia. UMTS (3G) utiliza un nuevo método de acceso al medio que añade complejidad pero permite velocidades más altas [3].

Para establecer una conexión 2.5G o 3G desde un sistema GNU/Linux, en primer lugar, deben conectarse los dispositivos de la capa física. En el caso de utilizar un ordenador portátil y el teléfono móvil, esta conexión será USB, serie o Bluetooth. Si se utiliza un PDA este enlace físico se realiza sin intervención del usuario. El siguiente paso consiste en establecer una conexión punto a punto (PPP, *Point to Point Protocol*) a nivel de enlace entre los dispositivos [4]. En la **figura 2**, aparece la pila de protocolos resultante tras la activación de PPP.

Una de las características más interesantes en este proceso de conexión es el acceso a las capas inferiores. Para proporcionar a los desarrolladores el acceso al hardware del terminal y a las propiedades de la red GPRS/UMTS directamente, el estándar define una serie de comandos AT+ que extiende el conjunto de comandos AT tradicionales de control de módem [5].

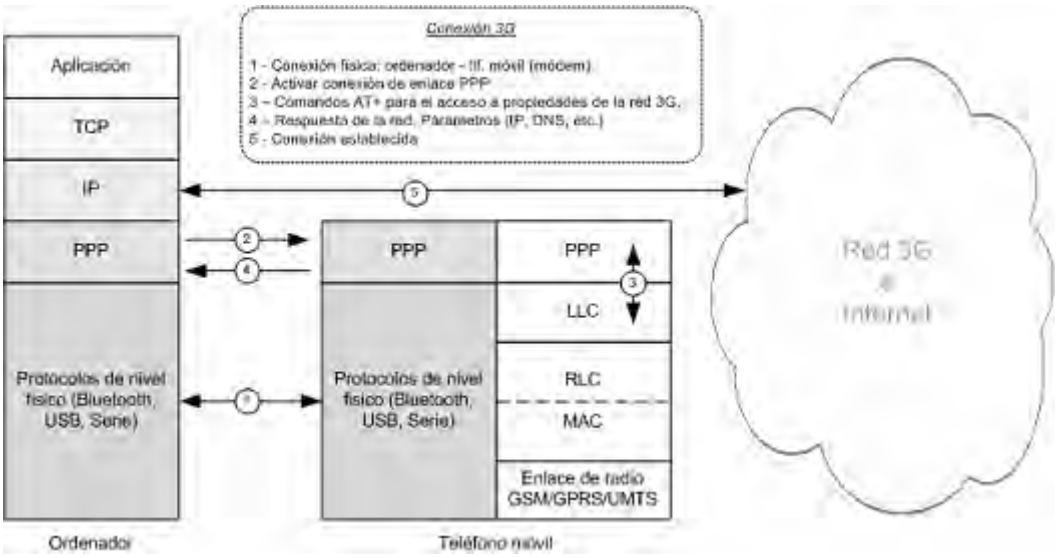

Figura 2. Protocolos para conexión 3G en el cliente.

La aplicación desarrollada en Movicuo simplifica este proceso, ofreciendo a los usuarios de Campus Ubicuo un mecanismo de conexión utilizando dispositivos portables y tecnologías móviles desde GNU/Linux. La herramienta se ha implementado utilizando programación GNOME con librerías GTK+. Además, permite la negociación de conexiones con distintos parámetros de QoS y ofrece información en tiempo real del estado de la conexión (ver **figura 3**). Particularmente útil es la gráfica velocidad/tiempo, para la cual se ha utilizado la herramienta *rrdtool* <http://oss.oetiker.ch/rrdtool>.

Desde el punto de vista investigador, gracias a esta herramienta, hemos podido analizar conexiones 2.5G y 3G, estudiando parámetros como el *throughput*, el *delay* y el *jitter*, además del comportamiento de la red en función del tipo de tráfico enviado o el nivel de QoS negociado.

Figura 3. Ventana de información de la conexión 3G.

2.1.2. GNU/LinEx sobre PDA

Este apartado está dedicado a la instalación del sistema GNU/LinEx en dispositivos PDA, sobre los que realizar desarrollos de software y de comunicaciones avanzadas.

El desarrollo de un sistema Linux empotrado para un PDA es una tarea compleja y que requiere conocimientos de sistemas operativos, del sistema Linux en particular y de la arquitectura propia del dispositivo. Estos PDA utilizan una familia de microprocesadores RISC denominados ARM [6], lo que significa que todo el software que se ejecute sobre él debe estar compilado específicamente para esa arquitectura. El proceso de compilación cruzada (*cross-compiling*) consiste en compilar un código fuente sobre una arquitectura para generar binarios en otra distinta. Así, para compilar una aplicación para el PDA o para construir la distribución hay que realizar compilación cruzada. En la **figura 4** se observa la diferencia entre un fichero compilado para las arquitecturas x86 y ARM, usando GCC (*GNU Compiler Collection*) como *cross-compiler*.

Actualmente existen proyectos centrados en el desarrollo de sistemas Linux para dispositivos empotrados. Podemos destacar EmDebian <http://www.emdebian.org> o Familiar <http://familiar.handhelds.org>, así como herramientas que facilitan la compilación cruzada como Scratchbox <http://www.scratchbox.org> y OpenEmbedded <http://www.openembedded.org>. Scratchbox ofrece un conjunto de herramientas diseñadas para facilitar la compilación cruzada, mientras que la segunda es un entorno de desarrollo que simplifica la tarea de construir distribuciones Linux completas para dispositivos empotrados.

En Campus Ubicuo uno de los PDA utilizados es el *HP iPAQ h6340*, que dispone de un microprocesador *ARM TI OMAP 1510*. Construir un sistema GNU/LinEx para este PDA supone construir cada una de las partes que lo componen: núcleo, sistema de ficheros y gestor de arranque.

Las fuentes del *kernel* utilizadas para este microprocesador se denominan `omap-linux`, (son además necesarios una serie de parches para el dispositivo concreto). La compilación cruzada de las fuentes del núcleo se realiza con el compilador GCC para ARM (*arm-linux*) y, como resultado, obtendremos una imagen que es la que se cargará durante el proceso de arranque. Además del núcleo, el SO requiere aplicaciones que aporten interactividad con el usuario y establezcan el árbol de directorios jerárquico (lo llamaremos *rootfs*). Compilar de forma cruzada cada librería y cada aplicación de un sistema Linux no es una tarea aceptable en términos de tiempo y de esfuerzo, por lo que utilizamos OpenEmbedded para esta tarea. Con respecto al cargador de arranque, usamos Uboot, ya que está soportado para la arquitectura utilizada.

Una vez que disponemos de la imagen del núcleo, el *rootfs* y el *bootloader*, podemos arrancar el sistema GNU/LinEx construido. Esta plataforma sobre el PDA, que puede ser extendida a cualquier distribución Linux, es la base de desarrollos y actividades de otras partes del proyecto.

COMPILACIÓN CRUZADA DE UNA APLICACIÓN

hola_mundo.c

```
#include <stdio.h>
int main (void){
  printf (";Hola Mundo!\n");
  return 0;
}
```

Compilación para la arquitectura x86

```
# gcc -Wall -o hola_x86 hola_mundo.c
# file hola_x86
hola_x86: ELF 32-bit LSB executable, Intel
80386, version 1 (SYSV), for GNU/Linux 2.2.0,
dynamically linked (uses shared libs), not
stripped
```

Compilación para la arquitectura ARM

```
# arm-linux-gcc -Wall -o hola_arm hola_mundo.c
# file hola_arm
hola_arm: ELF 32-bit LSB executable, ARM,
version 1 (ARM), for GNU/Linux 2.4.3,
dynamically linked (uses shared libs), not
stripped
```

Figura 4. Ficheros compilados para las arquitecturas x86 y ARM.

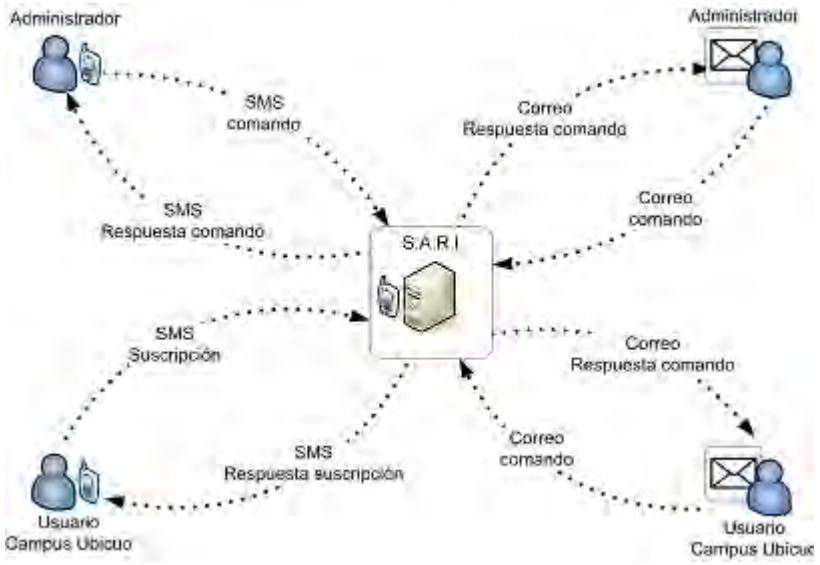

Figura 5. Interacción de los usuarios con SARI.

2.1.3. SARI (Sistema de Administración Remota Inalámbrico)

Volviendo a la arquitectura mostrada en la **figura 1**, Campus Ubicuo dispone de un sistema de administración remota al que hemos denominado SARI. Los sistemas informáticos a los que accedemos cada día requieren una administración particular, además de una disponibilidad de gestión absoluta. Por otra parte, la capacidad de la tecnología de mensajería instantánea (SMS, *Short Message Service*), la hace especialmente interesante en la manipulación de ciertos sistemas críticos que requieran un control desde cualquier localización y en cualquier instante.

Los trabajos y aplicaciones existentes para la administración vía SMS, tienen ciertas limitaciones, especialmente relativas a la seguridad. Por esto, se ha desarrollado un nuevo sistema robusto y extensible, capaz de soportar las necesidades de administración de un sistema informático. SARI permite la ejecución remota de comandos en sistemas Linux, además de la gestión de las suscrip-ciones a los distintos servicios de difusión de información de Campus Ubicuo. Esto aporta al administrador del sistema la ubicuidad requerida para labores administrativas que deben ser realizadas en cualquier momento y lugar. El diagrama de la **figura 5** representa la interacción de los distintos usuarios con SARI.

El programa principal, o sistema base, se ejecuta como un *daemon* al que se le pueden ir añadiendo *plugins* según el tipo de comunicación utilizada. Actualmente se han implementado dos *plugins*, uno encargado de la comunicación vía *bluetooth* [7] con un teléfono móvil para gestionar el envío y recepción de los mensajes SMS, y otro que gestiona la comunicación a través de mensajes de correo electrónico.

Así, el *daemon* ejecuta los procedimientos correspondientes a cada tipo de comunicación implementada en los *plugins* y cada *plugin* divide su ejecución en pasos, siendo posible mantener en una cola estos pasos de ejecución para cada uno de los *plugins*. Aunque la ejecución de cada paso no es paralela a la de los otros *plugins*, sí lo es en el proceso general, pudiendo funcionar al mismo tiempo distintos sistemas de comunicación.

Por otra parte, cada *plugin* se encarga de gestionar la comunicación para que sea no bloqueante, es decir, la implementación debe evitar que si en cualquier momento el dispositivo o servidor no contesta a una petición, el proceso quede parado a la espera de una respuesta que no llegará. Cada uno de los *plugins* desarrollados sigue un patrón en el que se distinguen las fases clásicas de una conexión: establecimiento, mantenimiento y cierre de la conexión.

El funcionamiento del *daemon* SARI es similar al resto de servicios de los sistemas Linux. El fichero de configuración se encuentra en `/etc/sari/sari.conf`, y la herramienta puede lanzarse desde `/etc/init.d/sari`. En ejecución SARI permite, tanto funciones administrativas, como funciones relacionadas con las suscripciones de los usuarios de Campus Ubicuo, basándose, para ello, en una sintaxis predefinida que puede ser configurada en uno de sus ficheros de configuración.

2.1.4. MeUbicuo

Campus Ubicuo es administrado por un conjunto de servidores centrales en los que se almacena la información que debe ser gestionada. Esta gestión ha sido separada del resto del sistema en un subproyecto independiente al que hemos denominado MeUbicuo, debido a que todos los servicios que se ofrecen son controlados desde estos servidores y puede aislarse del resto para facilitar un cambio en los requerimientos o una adaptación a otro tipo de organizaciones.

En estos servidores es donde se ejecuta SARI, permitiendo que los usuarios puedan realizar el envío y recepción de determinadas tareas relacionadas con las suscripciones a los servicios ofrecidos por Campus Ubicuo, así como ciertas tareas administrativas. También se encuentran las bases de datos y todas las estructuras de datos necesarias para su correcta gestión y la de su sistema de difusión. Es decir, si un usuario está dado de alta en el servicio de información de noticias universitarias a través de SMS, cuando llegue una noticia nueva de este tipo al sistema, ésta será difundida a él y al resto de usuarios suscritos.

Para que los usuarios puedan acceder a su información, además de la gestión de la suscripción mediante SMS, existe un portal web desarrollado con PHP y MySQL, a través del cual los usuarios realizan su registro en el sistema y pueden modificar o configuran sus preferencias como los servicios suscritos o el medio de acceso a esta información.

3. Tareas de investigación

Campus Ubicuo no es sólo un proyecto de desarrollo e innovación. También ha permitido realizar tareas de investigación relacionadas principalmente con la movilidad en redes IP y con las interferencias entre distintas tecnologías inalámbricas. Estos resultados son presentados brevemente en esta sección.

3.1. Movilidad IP: análisis y optimización del *handover*

Uno de los campos de investigación más interesante actualmente en las comunicaciones es el relacionado con el soporte de movilidad en redes IP. Aunque existen distintas soluciones para abordar el problema, la mayoría de las contribuciones giran en torno a *Mobile IP* [8].

Este protocolo, propuesto por el IETF (*Internet Engineering Task Force*), tiene como objetivo principal el permitir a los nodos móviles cambiar su punto de conexión a la red sin perdida de conectividad. Esto se consigue manteniendo una dirección IP fija en el nodo móvil (*Home Address*), y otra temporal (CoA, *Care-of Address*) a través de la cual es accesible cuando el nodo se mueve a otra red (*Foreign Network*).

Uno de los procesos más costosos de este protocolo es el *handover*, que se produce cuando un terminal realiza un movimiento y cambia su punto de conexión a la red. Exis-

ten soluciones que utilizan información de nivel 2 para minimizar el retardo que se produce al detectar el movimiento. Estas propuestas, aun siendo más rápidas que las de nivel 3, limitan la capacidad de movilidad de un nodo entre redes heterogéneas al ser dependientes de la tecnología de acceso. Podemos representar el tiempo de *handover* (T_H) de nivel de red de la siguiente forma:

$$T_H = T_{DM} + T_{REG}$$

donde T_{DM} es el tiempo que tarda el nivel de red en detectar que el nodo se ha movido y T_{REG} es el tiempo empleado en configurar la nueva dirección CoA y registrarla en su red origen.

El estudio del *handover* en *Mobile IPv6* nos ha permitido desarrollar un algoritmo denominado FDML3 (*Fast Detection Movement L3*) que, partiendo del trabajo realizado en [9], modifica el algoritmo de detección de movimiento a nivel de red propuesto por *Mobile IPv6*, basado en la pérdida de varios anuncios de *router* consecutivos como indicativo de que se ha producido un *handover*. Este método no depende de la capa de enlace, sino de la frecuencia con la que se envían los anuncios de *router* (RA, *Router Advertisement*) no solicitados. En FDML3, esta frecuencia tiene una influencia menor debido a que el aviso de *handover* se produce con la pérdida del primer RA.

Para analizar y validar los resultados del nuevo algoritmo, así como para compararlo con otros métodos de detección del movimiento, hemos utilizado el simulador OMNET++ <http://www.omnetpp.org>, donde se ha implementado la propuesta. En la **figura 6** se muestra la gráfica de los tiempos producidos por 8 movimientos.

Se han obtenido resultados que mejoran el tiempo de detección del movimiento hasta en un 25% con respecto al algoritmo propuesto por *Mobile IPv6*. Aún así, no en todos los casos consiguiendo una disminución del tiempo, ya que en configuraciones con valores muy pequeños para el intervalo de tiempo entre RAs no solicitados, los tiempos no mejoran.

3.2. Análisis de interferencias

Otra tarea de investigación que Campus Ubicuo nos ha permitido realizar es el estudio analítico y evaluación de las interferencias que experimentan las distintas tecnologías de comunicaciones inalámbricas.

En este trabajo se analizan las interferencias que unas tecnologías inalámbricas ejercen sobre otras en un entorno cercano, estudiando cómo afectan dichas interferencias en términos de rendimiento a cada una de las señales existentes. El estudio se lleva a cabo desde el punto de vista de un administrador de red, cuya función es la de mantener operativas distintas redes e intentar solucionar los problemas que se le presenten, algunos derivados de esta situación de interferencias.

Así, hemos podido comprobar a través de pruebas de laboratorio y de trabajo de campo, cómo pueden verse afectadas las redes inalámbricas por el funcionamiento de elementos tan comunes como microondas, teléfonos inalámbricos, mandos a distancia de radiofrecuencia o, en general, aquellos que emiten en la frecuencia ISM (*Industrial, Scientific and Medical*) de 2,4 GHz, reservada internacionalmente para uso no comercial y cuya banda está abierta para usos industriales, científicos y médicos sin necesidad de licencia. Es en este espacio donde se sitúan muchas tecnologías de comunicaciones inalámbricas como Bluetooth o Wi-Fi. De este modo, el administrador de red puede saber a qué se debe la caída de rendimiento de las redes inalámbricas a su cargo en un momento dado para actuar en consecuencia.

En este estudio se ha utilizado un analizador de espectros para conocer el número de dispositivos inalámbricos activos, el canal en el que emiten y otro tipo de información útil.

4. Conclusiones y trabajo futuro

En los últimos años estamos asistiendo a un cambio progresivo en las necesidades de los usuarios de las tecnologías de comunicaciones. La movilidad es un aspecto cada vez más demandado y las redes de datos de última generación son partícipes de este cambio. En esta situación el sistema desarrollado en Campus Ubicuo ofrece servicios telemáticos avanzados y de movilidad a sus usuarios.

Este proyecto tiene en el Software Libre uno de sus pilares principales. Un proyecto de este tipo, realizado en convenio entre universidad y empresa, permite, tanto al grupo GÍTACA como a SADIEL, S. A., la transferencia tecnológica hacia otros grupos y empresas que, gracias a las ventajas del Software Libre, podrán beneficiarse de la innovación generada al poner a disposición pública los resultados de nuestra investigación, difundida a través del portal web del proyecto <http://gitaca.unex.es/cubicuo>.

Si bien el proyecto está a punto de concluir, aún queda por acometer uno de las tareas más atractivas: la implantación de Campus Ubicuo en otras organizaciones distintas a las de una universidad. Campus Ubicuo ya está siendo analizado para su uso en tres entornos donde su aplicabilidad es directa; en hospitales y centros de educación secundaria donde las tecnologías utilizadas en este proyecto pueden agilizar y facilitar tanto los trámites administrativos como el control de los pacientes y estudiantes; y, en tercer lugar, como apoyo de los servicios turísticos en una administración local para ofrecer, como valor añadido, a los visitantes servicios telemáticos avanzados con tecnologías móviles sobre dispositivos como PDA o teléfonos móviles.

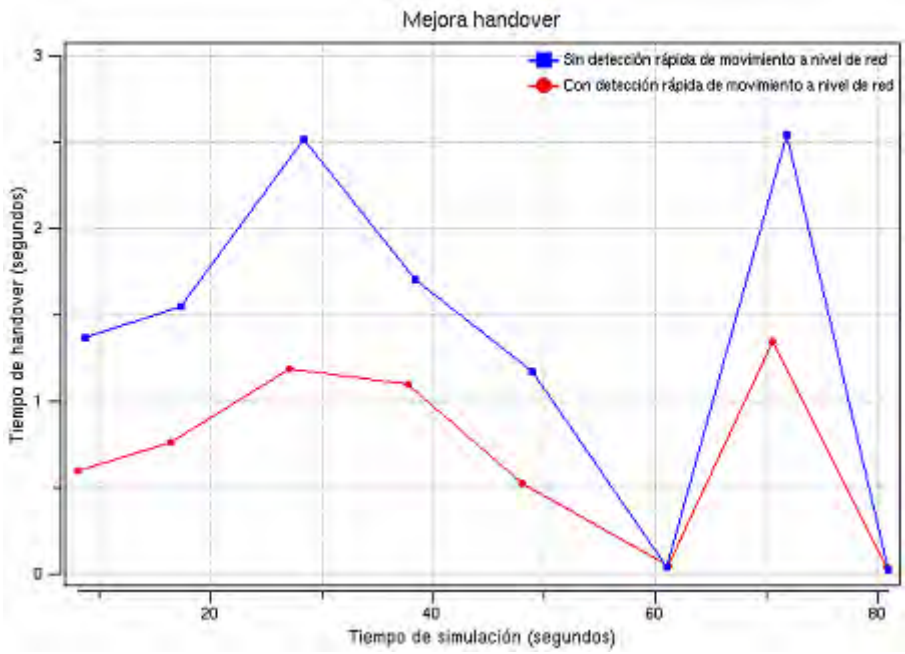

Figura 6. Comparativa del tiempo de *handover* con y sin FDML3.

Referencias

[1] I. Herraiz, J. J. Amor, A. del Castillo. Libre software for research. *FLOSS International Conference*. Jerez de la Frontera, Spain. March 2007, pp. 97-105.

[2] J. Carmona Murillo, J. L. González Sánchez, A. Gazo Cervero, L. Martínez Bravo. MOVICUO: Comunicaciones móviles y software libre para la ubicuidad. *III Jornadas de Software Libre de la Universidad de Cádiz*, Cádiz, Spain. April 2006, pp 45-58

[3] J. Perez-Romero, O. Sallent, R. Agustí, M. A. Díaz-Guerra. *Radio Resource Management Strategies in UMTS*, Wiley, 2005. ISBN: 0470022779.

[4] C. Andersson. *GPRS and 3G Wireless Applications*. Wiley. 2001. ISBN: 0471414050.

[5] ETSI TS 07.07. *Digital cellular telecommunications system (Phase 2+); AT Command set for GSM Mobile Equipment (ME)*. 2003.

[6] C. Hallinan. *Embedded Linux Primer*. Prentice Hall, 2006. ISBN: 0131679848.

[7] A. Huang. The use of Bluetooth in Linux and location aware computing. *Master thesis*, *Massachusetts Institute of Technology*. Cambridge, MA. 2005.

[8] D. Johnson, C. Perkins, J. Arkko. Mobility Support in IPv6. *IETF RFC 3775*, June 2004.

[9] N. Blefari-Melazzi, M. Femminella, F. Pugini. A layer 3 Movement Detection Algorithm Driving Handovers in Mobile IP. *Wireless Networks* 11 (3), pp. 223 – 233. 2005. Springer Netherlands.

Nota

[1]Campus Ubicuo <http://gitaca.unex.es/cubicuo> es un proyecto desarrollado en convenio por el Grupo de Investigación de Ingeniería Telemática Aplicada y Comunicaciones Avanzadas (GÍTACA) de la Universidad de Extremadura <http://gitaca.unex.es> y la empresa SADIEL, S. A. <http://www.sadiel.es>, con el apoyo de la Junta de Extremadura. (Expediente del proyecto: PDT05A041).

El modelo de la Oficina de Software Libre de la Universidad de Cádiz en la universidad española

José Rafael Rodríguez Galván, Manuel Palomo Duarte, Juan Carlos González Cerezo, Gerardo Aburruzaga García, Antonio García Domínguez, Alejandro Álvarez Ayllón
Oficina de Software Libre de la Universidad de Cádiz (OSLUCA)

<osl@uca.es>

Resumen: *desde los primeros años del siglo XXI, los órganos de gobierno de las universidades españolas están viviendo un creciente interés por el uso del software libre como vía para alcanzar sus principales objetivos. En este artículo se presenta el caso de la Oficina de Software Libre de la Universidad de Cádiz (OSLUCA), con la finalidad de ilustrar la forma en que ésta se articula y la problemática ante la que se encuentra una institución de este tipo. Se describen los principales proyectos en los que la OSLUCA se ha venido embarcando desde sus orígenes en el año 2003 y, en todos los casos, se intenta establecer el paralelismo con otras oficinas y secretariados similares existentes en España.*

Palabras clave: congresos de Software Libre, difusión del Software Libre, Software Libre, universidad.

Autores

José Rafael Rodríguez Galván es profesor del Departamento de Matemáticas de la Universidad de Cádiz (UCA) y, desde el año 2004, director de la OSLUCA. Dentro de ésta ha sido responsable de la organización de distintos proyectos, entre ellos las I, II y III Jornadas de Software Libre de la UCA y el FLOSSIC (Free/Libre Open Source Systems International Conference). Ha llevado a cabo distintas conferencias y ha participado en foros relacionados con el software libre y la universidad. Por otro lado, ha realizado distintos trabajos en el ámbito de la simulación numérica de ecuaciones en derivadas parciales, en el marco de la mecánica de fluidos, dentro del grupo de investigación FQM-315 de la UCA.

Manuel Palomo Duarte es Ingeniero en Informática por la Universidad de Sevilla (2001). En la actualidad trabaja como profesor del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Cádiz donde imparte asignaturas relacionadas con los sistemas operativos y el diseño de videojuegos (haciendo uso de software libre en ellas) y realiza labores de Coordinador Erasmus en la titulación de Ingeniería Técnica en Informática de Sistemas. Es miembro del grupo de investigación "Mejora del Proceso Software y Métodos Formales", donde está realizando su tesis doctoral sobre calidad de composición de composiciones de servicios web con BPEL. Desde su incorporación a la UCA ha colaborado con la OSLUCA, principalmente en la realización de las "III Jornadas de Software Libre de la Universidad de Cádiz" (JOSLUCA3) y el "I Congreso Científico Internacional de FLOSS" (FLOSSIC 2007).

Juan Carlos González Cerezo es Ingeniero Técnico en Informática de Gestión por la Universidad de Cádiz, Premio Extraordinario de Fin de Carrera de la Diplomatura de Informática (1990-91) otorgado por la Junta de Gobierno de la UCA y el Segundo Premio Nacional de Terminación de estudios de Diplomatura en Informática (octubre 1992) otorgado por el MEC. Programador de los servicios informáticos de dicha Universidad desde 1992. Actualmente, entre otras tareas, participa como técnico en la OSLUCA, a la vez que prosigue sus estudios como Ingeniero en Informática.

Gerardo Aburruzaga García es Coordinador del Área de Informática y Profesor asociado del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Cádiz desde 1989. Ha trabajado de programador y analista en los servicios informáticos de la UCA, donde entró en el año 1988. Actualmente, aparte de otras tareas, es responsable técnico de la OSLUCA. Como profesor ha dado clase de Soportes Lógicos, Programación II, Metodología y Tecnología de la Programación II, Administración de Sistemas Operativos y Programación Orientada a Objetos.

Antonio García Domínguez es Ingeniero Técnico en Informática de Gestión por la Universidad de Cádiz (2006), y prosigue en ella sus estudios como Ingeniero en Informática. Ha recibido una Mención Especial en el Premio Nacional de Fin de Carrera del curso 2005/2006. Presentó el PFC "Post-procesador y visor de demostraciones para el sistema de demostraciones ACL2" y en el verano de 2007 participó en el proyecto "Grid-based financial software" del programa Openlab del CERN. Actualmente colabora con una beca de la OSLUCA en la elaboración de un marco de pruebas de composiciones de servicios web en BPEL.

Alejandro Álvarez Ayllón es Ingeniero Técnico en Informática de Gestión por la Universidad de Cádiz (2007), y continúa en ella con sus estudios como Ingeniero en Informática. Su PFC consistió en una forja de software construida sobre Zope/Plone, orientada especialmente a la comunidad universitaria y que actualmente se utiliza como repositorio oficial de proyectos de la OSLUCA. Es becario de la OSLUCA, colaborador activo del proyecto QtOctave, y el principal desarrollador y mantenedor del sitio Hispaciencia.com.

1. El software libre en la universidad española

Los orígenes del software libre están enraizados en el mundo universitario. Incluso antes de que existiera conciencia del significado de este concepto, científicos e ingenieros de universidades (en especial estadounidenses) trabajaban según un modelo de intercambio de conocimientos que hoy podría concebirse como comunidad de desarrolladores de software libre. Luego, cuando a mediados de la década de 1970 se hizo común la situación que es todavía habitual en el mundo del software, las iniciativas aisladas que podían ser etiquetadas como libres o de código abierto provenían del mundo universitario y de los centros de investigación. Este es el caso, por ejemplo, de T_EX y, en cierto modo, de UNIX, dotado de una licencia que, en la práctica, permitía su libre acceso para fines académicos y que, en la Universidad de Berkeley fue el origen de los sistemas BSD (*Berkeley Software Distribution*). La propia génesis del concepto del software libre y del sistema GNU/Linux se debe a personas que, como Richard Stallman o Linus Torvalds, habían estado vinculados al mundo universitario.

Aunque la universidad española hubiera permanecido, hasta la década de 1980, ajena a este proceso, durante la última parte del siglo XX vivió una paulatina extensión del software libre y, en concreto, de las herramientas GNU (*GNU is Not Unix*). La contrastada calidad de estas herramientas, así como su disponibilidad de forma gratuita en servidores FTP (como los conocidos *sunsites*) hicieron que los servicios de informática de universidades españolas las adoptaran de forma paulatina. De igual modo, determinados grupos de profesores comenzaban a familiarizarse con el software libre y a utilizarlo como apoyo a su docencia e investigación. La calidad de los sistemas GNU/Linux, así como su progresiva madurez y popularización, de la mano de la facilidad de instalación y mantenimiento que proporcionaban las distintas distribuciones existentes, hicieron que comenzaran a sustituir a los sistemas UNIX comerciales, que eran habitualmente utilizados en los servidores de centros de cálculo y gestión universitarios.

A principios del siglo XXI, existía una consolidada comunidad de software libre en España, buena parte de la cual estaba formada por personas y grupos relacionados con la Universidad, aislados o vinculados a asociaciones, aunque sin ningún tipo de iniciativa institucional que los respaldara. Por entonces, ya era factible concebir el salto del software libre desde los servidores hasta el escritorio de los usuarios, contando con entornos de ofimática que eran realmente competitivos. Sobre esta base, pudieron nacer en España importantes iniciativas políticas dedicadas a utilizar software libre y sistemas GNU/Linux en administraciones públicas, como la Junta de Extremadura y, posteriormente, la Junta de Andalucía.

Los órganos de gobierno de las universidades públicas, siendo consecuentes con los fines de esta (entre ellos, la responsabilidad de transmisión a la sociedad del conocimiento superior del que es depositaria y de valores éticos como la solidaridad y la libertad), no podían permanecer ajenos a este proceso. Por supuesto, la posibilidad de racionalizar los costes de las (con frecuencia abusivas) licencias de software suponía un aliciente. Y, de esta forma, florecieron las primeras declaraciones institucionales en favor del software libre, así como las primeras entidades dedicadas, oficialmente a su promoción y desarrollo en la universidad. Este fue el caso, por ejemplo, de la moción de apoyo al software libre en la UAM y del nacimiento de la Oficina de Software Libre de la Universidad de Las Palmas de Gran Canaria.

En este contexto se enmarca el nacimiento de la Oficina de Software Libre de la Universidad de Cádiz (OSLUCA) y de las diferentes entidades universitarias que, dotadas de un mayor o menor carácter institucional, fueron apareciendo en los años siguientes (ver **figura 1**). El caso de la OSLUCA será expuesto en las siguientes secciones como paradigma de la labor que han venido realizando estas entidades, haciendo énfasis en aquellas actividades más comunes (realización de jornadas, elaboración de distribuciones GNU/Linux locales, etc), así como en algunas de sus aportaciones más específicas (como el marco de interoperabilidad sobre estándares abiertos que fue aprobado por consejo de gobierno).

En los últimos años, el movimiento universitario de software libre ha comenzado a vivir un interesante proceso de extensión, penetrando en los foros que han venido utilizando, tradicionalmente, las universidades para el intercambio y la colaboración. Este es el caso del grupo de trabajo Iris-Libre[1], dentro de la comunidad de RedIris, así como del Subgrupo de Trabajo en Software Abierto CRUE-TIC SL[2], dentro de la CRUE (*Conferencia de Rectores de las Universidades Españolas*). La coordinación entre universidades puede ser vital para no recaer en errores y permitir abordar objetivos que, de forma aislada, serían inalcanzables.

- Universidad de Alicante: **COPLA** http://copla.ua.es/
- Universidad Autónoma de Barcelona: **GNUAB** http://www.gnuab.org/
- Universidad de Barcelona: **gclUB** http://gclub.ub.es/
- Universidad de Cádiz **OSLUCA** http://softwarelibre.uca.es/
- Universidad Carlos III de Madrid **LUC3M** http://luc3m.uc3m.es/
- Universidad de Castilla-La Mancha: **CRySoL** http://crysol.inf-cr.uclm.es/
- Universidad de Deusto: http://softwarelibre.deusto.es/
- Euskal Herriko Unibertsitatea **itsas** http://itsas.ehu.es/
- Universidad Europea de Madrid: **GLUEM** http://www.gluem.net
- Universidad de Huelva: **OSLUHU** http://cibercomunidades.net/uhu/osluhu/
- Universidad Jaime I: **Software Libre UJI** http://www.swlibre.uji.es
- Universidad de A Coruña: **OSL-UDC** http://softwarelibre.udc.es/
- Universidad de La Laguna **SSL-ULL** http://ssl.ull.es/
- Universidad de Murcia: **SOFTLA** http://www.um.es/atica/softla/
- Universidad de las Palmas de Gran Canaria: **OSL** http://www.softwarelibre.ulpgc.es/
- Universidad Politécnica de Cataluña: **CPL** http://www.cpl.upc.edu
- Universidad Politécnica de Valencia: **poLinux** http://www.polinux.upv.es
- Universidad Pontificia de Comillas: **linuxec** http://linuxec.upcomillas.es/
- Universidad de Sevilla: **SOFLA-US** http://solfa. us.es/
- Universidad de Valencia: **LinUV** http://www.uv.es/LinUV/
- Universidad de Valladolid: **SOLEUP** http://soleup.eup.uva.es

Figura 1. Oficinas, secretariados y asociaciones de software libre en universidades españolas. (fuente: lugar de encuentro "Iris Libre", noviembre 2007[3]).

2. El modelo de la OSLUCA

En el año 2003, en la Universidad de Cádiz (UCA) había algunos núcleos de profesores que, siendo usuarios de software libre, habían llevado a cabo distintas iniciativas, como la realización de cursos de verano sobre este asunto, y se encontraban relacionados con la comunidad de software libre española. Por otro lado, técnicos del Área de Informática habían hecho que una gran parte de los servicios vitales de la universidad (correo electrónico, servidores web, etc.) estuvieran consolidados en software libre, y defendían con entusiasmo su validez.

Asesorados por algunas de estas personas y siendo conscientes de la situación favorable en Andalucía y de las primeras iniciativas que estaban naciendo en universidades españolas, el equipo de gobierno entrante en la Universidad en el año 2003 se comprometió a apoyar el software libre en la UCA. La Dirección General de TIC plasmó este compromiso como una línea de acción prioritaria, en la que se involucraron algunos de los profesores y técnicos más activos en torno al software libre, comenzando a realizar las primeras actividades. Éstas fueron:

- Creación de un boletín informativo que fue distribuido, de forma interna, en la Universidad.
- Primeras actividades de promoción del software libre, entre las que destaca la visita de Richard Stallman a la UCA en julio de 2003).
- Puesta a punto de una página web, foros, listas de correo, etc.

El 15 de marzo de 2004, el Consejo de Gobierno de la Universidad de Cádiz, órgano máximo encargado de establecer sus directrices estratégicas y programáticas, aprobó de forma institucional una declaración de apoyo al software libre[4] (aparecida en el Boletín Oficial de la UCA Nº 9) e hizo oficial a la *Oficina de Software Libre de la Universidad de Cádiz.*

Esta ambiciosa declaración reconoce que el software libre es un medio para alcanzar los objetivos de la UCA y define la función de la OSLUCA como "*promover el uso de las aplicaciones y recursos informáticos basados en software libre en la comunidad universitaria*" y, en concreto, tomar las medidas necesarias para "garantizar la no discriminación" de usuarios de aplicaciones y sistemas libres, así como "fomentar el uso y desarrollo de software libre" en la Universidad, promoviendo la formación en herramientas libres, su uso docente en las aulas informáticas, en la gestión y en la investigación, la publicación con licencia libre del material producido, etc.

Para su articulación, en la OSLUCA se integraron distintos profesores, técnicos del Área

de Informática y alumnos becados, con la siguiente estructura:

- El *Director de la Oficina de Software Libre* es un profesor, responsable de ella. Aunque desde el principio ha contado, como contraprestación, con una reducción en su carga docente, recientemente este cargo se ha consolidado con la categoría de Dirección de Secretariado (del mismo modo que en algunos secretariados de software libre existentes en las universidades españolas).
- El *Director Técnico* es un miembro del Área de Informática de la UCA con gran experiencia en la utilización de software libre que, entre sus tareas habituales, tiene encomendada esta labor. Además, existe un segundo técnico del Área de Informática que realiza importantes tareas de apoyo a la OSLUCA.
- A partir del año 2005, la OSLUCA contó con una sede física. Desde entonces, se creó una beca específica para la Oficina. Además, han existido distintas becas de tipo más concreto (por ejemplo, para colaborar en desarrollo de proyectos que fueran de interés para la Universidad).
- Entre el profesorado de la UCA, la implicación de algunas personas ha sido muy alta, ocupándose de tareas de gran importancia dentro de la OSLUCA.

3. Actividades realizadas

Con el respaldo del equipo de gobierno, la OSLUCA tenía ante sí un horizonte ilusionante, pero también una enorme responsabilidad y un camino de dificultades, la mayor parte de las cuales se han demostrado comunes a todas las universidades españolas.

Ante ello, las acciones que se han realizado, en general, atendían a las siguientes líneas de actuación:

1. *No discriminación de las personas de la universidad que desearan utilizar software libre.* Una universidad ofrece servicios (desde correo electrónico hasta consulta de actas o nóminas) a muchos miles de usuarios a través de la red. Durante mucho tiempo, se ha asumido que todos ellos utilizarían el entorno predominante (sistemas Windows, navegador web Explorer, etc.), lo que, con frecuencia condujo a problemas para los usuarios de software libre que deseaban acceder a estos servicios. De la misma forma, la falta de sensibilidad por el uso de estándares abiertos suponía una dificultad añadida para la lectura de los documentos intercambiados en la universidad. Este es el caso de los documentos de ofimática, codificados utilizando los formatos opacos de MS-Office.
2. *Difusión y formación de herramientas de software libre y en la filosofía de la colaboración y la publicación de contenidos libres.* En la universidad española, como en el resto de la sociedad, se utiliza mayoritariamente software propietario para las tareas habituales. La introducción de software libre significa un cambio que conlleva un rechazo inicial, el cual solamente se salvará a través de un esfuerzo para la formación, facilitando la migración y resaltando sus ventajas. Es aquí donde juega un papel importante la realización de conferencias, jornadas, cursos y todo tipo de actividades que amplíen, progresivamente, el conocimiento del software libre, las herramientas recomendadas, su filosofía y sus ventajas.
3. *Soporte técnico e instalación de software de código abierto en puestos de trabajo y apoyo a su uso en docencia, investigación y gestión universitarias.* Inicialmente, las personas que se aventuran a realizar una migración a software libre, tienen que realizar un esfuerzo adicional que está poco recompensado. Es fundamental, para paliar la discriminación comentada anteriormente, el garantizarles soporte técnico, como mínimo, en las mismas condiciones que a los usuarios de sistemas no libres. Del mismo modo, es necesario apoyar a todos aquellos proyectos que estén encaminados a utilizar software libre en tareas cotidianas de la universidad, así como promover el nacimiento de este tipo de iniciativas.
4. *Desarrollo de proyectos de interés estratégico.* En la línea de fomentar el uso de software libre en las tareas cotidianas de la universidad, con frecuencia resulta necesaria la implicación (a nivel de desarrollo, documentación, etc.) en proyectos de gran interés que necesiten un impulso para cristalizar. Este sería, por ejemplo, el caso de alternativas libres a programas muy utilizados en docencia, gestión o investigación. Paradójicamente, las políticas de licencias de software privativo en la universidad, con frecuencia abusivas, suponen una perfecta justificación de la necesidad de desarrollar alternativas.

A continuación, relacionaremos algunas de las acciones más representativas que se han realizado en la OSLUCA. Muchas de ellas han sido comunes a distintas universidades y así se destacará en cada uno de los casos.

3.1. Apoyo al uso de software libre en distintos servicios de la UCA

La Oficina de Software Libre ha impulsado la utilización de software libre en servicios de la UCA. Su elevada calidad ha motivado que, para muchos de estos servicios (servidores web, proxy, correo electrónico, etc), la UCA haya estado utilizando software libre desde hace mucho tiempo, como en la mayor parte de las universidades de nuestro entorno.

En otros casos la migración a software libre ha sido más reciente (y en ellos la OSLUCA ha intentado realizar tareas de apoyo, en la medida de sus posibilidades). Este es el caso del portal de la UCA (que actualmente utiliza Zope + Plone como gestor de contenidos) y del campus virtual, que utiliza Moodle. Utilizado como entorno del campus virtual de nuestra universidad, fue introducido en el año 2005 como alternativa al programa privativo que estaba siendo utilizado hasta entonces (y que suponía un alto coste) e ilustra perfectamente el interés económico de encontrar alternativas libres.

Algunas otras veces, los técnicos de la oficina se han encargado, directamente, de realizar la instalación y puesta a punto de los servicios. Este es el caso de los puestos de libre acceso a Internet (que utilizan sistemas GNU/Linux), portátiles de préstamo en biblioteca (se ha hecho un despliegue de 240 portátiles, que usan exclusivamente GNU/Linux), aumento del número de salas de ordenadores con doble arranque (Windows y GNU/Linux), etc.

3.2. La normativa de intercambio de información institucional en la UCA

El 27 de septiembre de 2004, el Consejo de Gobierno de la UCA aprobó, a instancias de la OSLUCA, la "Normativa de intercambio de información institucional en la Universidad de Cádiz"[5] (Boletín Oficial de la UCA Nº 15, pág. 63).

Esta normativa establece que "cualquier documento emitido por un organismo de la UCA que esté dirigido oficialmente a miembros de ella deberá estar codificado en un formato abierto, siempre que exista alguno adecuado para el tipo de documento del que se trate, o al menos deberá acompañarse de una versión en formato abierto, independientemente del medio empleado para su emisión: correo electrónico, página web, etc.".

Por último, señala que será tarea de los servicios de informática el mantener una lista actualizada de formatos abiertos para diferentes tipos de documentos, a los cuales deberán atenerse las publicaciones de documentos institucionales.

La normativa de la UCA es el primer documento de este tipo que fue publicado en universidades y administraciones públicas españolas, e incluso fue prácticamente pionero a nivel mundial. De hecho, casi cuatro años después, son pocas las iniciativas parecidas existentes en nuestro entorno.

Durante el año 2005, el Área de Informática y la OSLUCA debieron ocuparse de la tarea de realizar una taxonomía de formatos abiertos, estudiando para cuáles de ellos existían aplicaciones adecuadas para ser usadas en la UCA. La publicación, en septiembre de 2005, del estándar PDF/A (ISO 19005-1), y en noviembre de 2006 la de OpenDocument (estándar ISO 26300), nos permitió contar

con estándares adecuados para su uso en ofimática. Por lo tanto, entre los años 2006 y 2007 se puso en marcha un programa de formación para el personal de administración de la UCA que desembocó en una instrucción de servicio, de obligada asistencia, que debería suponer el pistoletazo definitivo para exigir el cumplimiento de la normativa. Aun así, una iniciativa tan revolucionaria como esta tiene que romper con la tremenda inercia que significa el uso de documentos de MS Office en el conjunto de la universidad y será necesario un esfuerzo suplementario hasta llegar a su cumplimiento.

3.3. Guadalinex_UCA

Varias universidades españolas han desarrollado distribuciones de GNU/Linux propias. De hecho, el proyecto *Metadistros* (del que se derivaron numerosas distribuciones, entre ellas las de algunas regiones como Andalucía) nació en la Oficina de Software Libre de la Universidad de Las Palmas de Gran Canaria, dentro de un proyecto destinado a facilitar la creación de sistemas universitarios.

Dentro de la OSLUCA hubo bastante controversia sobre este asunto. Entre sus ventajas, se encuentra:

1. El contar con un mecanismo más para promocionar el software libre entre los integrantes de nuestra universidad. En especial, si la distribución puede arrancarse en modo "live", experimentar con ella e instalarla, si se desea.
2. Posibilidad de incluir en esta distribución una serie de programas recomendados, así como material asociado (documentación, etc.) para que en nuestra universidad se experimente con ellos. También configurar la distribución para facilitar al usuario el acceso a los recursos de la UCA.
3. Definir un entorno claro sobre el cual ofrecer soporte técnico y adquirir conocimientos en nuestra universidad sobre creación de distribuciones GNU/Linux que podrían ser interesantes en otras circunstancias.

Pero en contra se podían aducir algunas otras razones:

1. La creación de una distribución supone un esfuerzo importante que es necesario no abandonar para mantenerla actualizada, creando sucesivas versiones.
2. Ante la gran variedad de distribuciones de GNU/Linux existentes, crear una nueva podría ser una fuente de confusión.

Finalmente, se decidió crear una adaptación de la distribución Guadalinex, (la primera y única variante no realizada por la Junta de Andalucía, aunque con su completo apoyo) a la que se denominó Guadalinex_UCA, tratando de subrayar el que no se trataba de ningún sistema nuevo, sino la particularización de uno existente, a la vez que, como universidad andaluza, dar un espaldarazo a la Junta de Andalucía ante su apuesta de apoyo al software libre.

Guadalinex_UCA consiguió despertar el interés de bastantes personas, se distribuyeron miles de copias a los alumnos y demostró que es posible, con los limitados recursos de una oficina de software libre, desarrollar estas actividades con garantías. Sin embargo, su incidencia en el aumento de uso de software libre no fue tan alta como habríamos deseado y, por otro lado, el mantenerla actualizada en ulteriores versiones significaba un gran esfuerzo. En la actualidad, estamos invirtiendo este esfuerzo en realizar una *selección del software recomendado en nuestra universidad*. Nuestro siguiente objetivo: dar la mayor difusión posible a estos programas en nuestra universidad y quizás, facilitar su distribución, por ejemplo a través de meta-paquetes y quizás CD recopilatorios. Aquí podríamos incluir CD supletorios para Guadalinex o recopilatorios para cualquier otro sistema operativo, incluyendo MS Windows. La edición de CD de aplicaciones libres en entornos Windows, para facilitar su difusión, es una idea que también ha estado presente en oficinas de software libre de distintas universidades.

3.4. Software libre en la docencia

Con algunas excepciones, el uso de software libre en las aulas de la UCA está poco extendido. En algunas ocasiones, el motivo es la inexistencia de alternativas a las aplicaciones utilizadas por los profesores para su docencia, pero en la mayor parte de los casos se trata simplemente del desconocimiento, por parte de los mismos, de las alternativas existentes, o bien la falta de motivación para realizar una migración.

Por este motivo, desde la OSLUCA se ha trabajado, en colaboración con el profesorado, en la catalogación y difusión del software libre existente, incluyendo en programas de formación aquellas alternativas más apropiadas y contribuyendo, cuando ha sido conveniente, a su mejora[6].

En este sentido, se pueden incluir:

- La colaboración, a través de un contrato-programa, en la adopción, por parte de algunos profesores del Departamento de Matemáticas, del programa *Maxima*, incluyendo la organización de un curso de formación al Personal Docente e Investigador en el año 2006.
- La colaboración (a partir de 2007) en el desarrollo de *QtOctave*, una interfaz gráfica para Octave, uno de los programas que podrían ser una alternativa para su uso en numerosas asignaturas de distintos departamentos de nuestra universidad.
- El proyecto *R-UCA*[7] (año 2007), que persigue facilitar la implantación del paquete estadístico R como estándar para la actividad docente e investigadora dentro de esta área. El proyecto se concreta en varias líneas de actuación, entre las cuales se encuentra el desarrollo de material docente (con licencia libre), la formación y apoyo al PDI, la contribución a la traducción y desarrollo de interfaces existentes, etc.

Este proyecto tiene además una gran importancia estratégica, pues el uso de un paquete estadístico como R, libre y de enorme calidad, permitirá a la UCA evitar la abusiva política de precios de distribuidores de software privativo.

3.5. Otras actividades

Entre el resto de las actividades realizadas, cabe destacar el esfuerzo por potenciar el desarrollo de proyectos de fin de carrera con licencia libre. En este sentido, se ha habilitado una forja de desarrollo interna a la UCA, que contiene una bolsa de proyectos. A la vez, desde su primera edición, se colabora con el Concurso Universitario de Software Libre[8]. Este concurso, es una interesante iniciativa que fue creada en el año 2006 por parte de alumnos, docentes y entidades vinculadas a la Universidad de Sevilla. Está dirigido a estudiantes de universidades españolas para estimular su participación en la creación de Software Libre y ha tenido un considerable éxito. A partir del Curso 2007/2008, la Universidad de Cádiz organiza un concurso local, en el que se otorgará un premio al mejor proyecto presentado por alumnos de la Universidad de Cádiz. El objetivo es fomentar su participación en el concurso.

4. Jornadas y congresos

Como la práctica totalidad de las oficinas de software libre españolas, la OSLUCA ha dedicado un gran esfuerzo a potenciar como línea estratégica la difusión del conocimiento y avances del software libre, a través de numerosas jornadas, conferencias y seminarios. Entre ellas destacan las siguientes:

4.1. I Jornadas de Software Libre de la Universidad de Cádiz

Estas jornadas se realizaron los días 14 y 15 de abril de 2004[9], y sirvieron como "puesta de largo" de la OSLUCA ante el entorno universitario.

Entre los ponentes se encontraron Jesús Mª. González Barahona (grupo Libresoft de la Universidad Rey Juan Carlos), José María Rodríguez Sánchez (Director General de Sistemas de Información y Telecomunicaciones, Sociedad de la Información. Junta de Andalucía) y Roberto Santos (Vicepresidente de Hispalinux). Otro aspecto a destacar fue la dedicación del segundo día de las Jornadas a la plataforma Zope, herramienta seleccionada para la web institucional de la Universidad.

4.2. IV Jornadas Andaluzas de Software Libre

Los días 5 y 6 de noviembre de 2004[10] se celebraron en la Escuela Politécnica Superior de Algeciras estas jornadas. Su organización corrió a cargo de la OSLUCA junto con las asociaciones ADALA y CAGESOL. Entre las charlas destacaron las de Álvaro López Ortega (desarrollador del proyecto GNU), Antonio Larrosa (desarrollador de KDE), o Juan Conde (Consejería de Innovación, Ciencia y Empresa de la Junta de Andalucía).

4.3. II Jornadas de Software Libre de la Universidad de Cádiz

Estas jornadas tuvieron lugar entre el 6 y el 8 de abril de 2005[11]. Las jornadas se vertebraron en torno a dos ejes: software libre y ciencia, por un lado, y software libre en la empresa por otro.

Dentro del primer apartado podemos destacar la charla impartida por Ricardo Galli (profesor de la *Universitat de les Illes Balears* y conferenciante oficial de la *Free Software Foundation* en España) sobre los aspectos éticos del software libre, o la de Antonio Fuentes Bermejo (IRISGrid, Iniciativa Nacional de GRID. RedIris) sobre supercomputación. Juan José Hierro (Proyecto Morfeo Telefónica I+D), Isidro Cano (director de supercomputación de HP) y Javier Viñuales Gutiérrez (Yaco Sistemas) participaron en la segunda temática.

Asimismo, las jornadas incluyeron por primera vez la realización de talleres técnicos en aula de ordenadores, uno de ellos centrado en la creación de distribuciones de GNU/Linux derivadas de Guadalinex, y otro en Blender, la excelente aplicación libre de modelado 3D, animación, *renderizado* y post-producción.

4.4 III Jornadas de Software Libre de la Universidad de Cádiz

Estas jornadas, celebradas el 20 y 21 de abril de 2006[12], significaron un salto de calidad importante. Por primera vez se siguió un proceso científico de petición pública de trabajos y revisión por pares para participar en las Jornadas, siendo todos los trabajos publicados posteriormente en el libro de actas de las Jornadas con ISBN. Además, para la organización del evento se contó con la colaboración del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Cádiz. Se distribuyeron más de 200 CD conteniendo las actas de las jornadas junto a más de 100 documentos libres (manuales, libros, cursos, etc.).

Entre los ponentes, podemos nombrar a Ismael Olea (Proyecto de Documentación de Linux en Castellano), Juan M. Rocha Ramos (administrador de la forja de Conocimiento Libre del Centro Informático Científico de Andalucía) o Juan Jesús Ojeda (desarrollador de Guadalinex). Como prólogo, se realizó una jornada temática sobre "Propiedad intelectual, software libre y universidad". En ella, entre otros, participaron César Iglesias (Díaz - Bastien & Truan, Abogados) y Malcolm Bain (Cátedra de Software Libre de la UPC).

4.5. FLOSS International Conference

La FLOSS International Conference (Congreso Científico Internacional de FLOSS) se organizó de manera conjunta con el grupo de investigación "Mejora del Proceso Software y Métodos Formales" y el Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Cádiz. Además se contó con la colaboración de la Escuela de Negocios de Jerez, perteneciente a la Delegación de Formación y Empleo del Ayuntamiento de Jerez de la Frontera (Cádiz).

Este congreso se celebró en la Facultad de Ciencias Sociales y de la Comunicación de Jerez de la Frontera (perteneciente a la Universidad de Cádiz). Las fechas elegidas fueron los días 7, 8 y 9 de marzo de 2007[13].

El congreso fue un éxito de asistencia, con más de 30 ponentes de varios países y superando el centenar de asistentes, incluyendo investigadores, docentes, profesionales de las tecnologías y estudiantes universitarios.

Las comunicaciones tuvieron un marcado carácter científico, trataron temas tan diversos como el *e-learning*, accesibilidad web, traducción automática, desarrollo de software, modelado 3D profesional, supercom-putación o inteligencia artificial. Algunos de los ponentes más destacados fueron Rüdiger Glott (de la Universidad UNU-Merit), Alberto Barrionuevo (Vicepresidente de FFII), Juan José Domínguez Jiménez (grupo SPI&FM) o Juan José Amor (grupo Libresoft).

Las ponencias fueron incluidas en el libro de actas publicado, con licencia libre, por el Servicio de Publicaciones de la Universidad de Cádiz, con ISBN. Dicho libro se distribuyó en un CD recopilatorio de documentación libre con más de 200 libros, manuales y cursos sobre sistemas libres que se repartieron entre todos los asistentes al evento y que está disponible para descarga gratuita en la web del congreso.

Paralelamente al evento se organizó junto con la Escuela de Negocios de Jerez una jornada destinada a divulgar las oportunidades que ofrece el software libre para la empresa. En ella varias empresas TIC presentaron las diferentes soluciones que ofrecen a sus clientes desde ofimática hasta clústeres y computación distribuida, pasando por ERP, *groupware*, CRM, etc.

5. Conclusiones

Aunque joven todavía, la Oficina de Software Libre de la Universidad de Cádiz ha conseguido logros interesantes dentro del software libre, erigiéndose como una de las instituciones de referencia en este marco a nivel regional.

De cara al futuro los objetivos que se plantean pasa por una cada vez mayor integración del Software Libre en todos los ámbitos de la vida universitaria: gestión, docencia, investigación y transferencia de conocimiento.

Notas

[1]Grupo de trabajo "IRIS-Libre" de RedIris, <http://www.rediris.es/gt/iris-libre/>.
[2]Grupo de trabajo "CRUE-TIC SL" de la CRUE, <http://cruesl.um.es/>.
[3]Lugar de encuentro "IRIS-Libre", <https://forja.rediris.es/projects/encuentro/>.
[4]Declaración institucional de apoyo al software libre en la UCA, <http://www.uca.es/softwarelibre/declaracion-uca-sl>.
[5]Normativa de intercambio de información institucional en la UCA, <http://softwarelibre.uca.es/normativa_estandares>.
[6]Programas recomendados para su uso en la UCA, <http://www.uca.es/softwarelibre/programas/>.
[7]Proyecto R-UCA, <http://knuth.uca.es/R/>.
[8]Concurso Universitario de Software Libre, <http://concursosoftwarelibre.org/>.
[9]I Jornadas de Software Libre de la UCA, <http://softwarelibre.uca.es/jornadas_1>.
[10]IV Jornadas Andaluzas de Software Libre, <http://softwarelibre.uca.es/node/260>.
[11]II Jornadas de Software Libre de la UCA, <http://softwarelibre.uca.es/jornadas_2>.
[12]III Jornadas de Software Libre de la UCA, <http://softwarelibre.uca.es/jornadas/josluca_06/>.
[13]FLOSSIC, <http://softwarelibre.uca.es/jornadas/fic/spa/index.php>.

Christopher Oezbek, Lutz Prechelt
Institut für Informatik, Freie Universität Berlin, Alemania

<{oezbek, prechelt}@inf.fu-berlin.de>

Aprendiendo a introducir una innovación en un proyecto basado en Software Libre

Traducción: Ignacio Palomo Duarte (Escuela Superior de Ingeniería, Universidad de Cádiz).

Resumen: *nuestra propuesta trata de investigar la introducción de nuevas tecnologías o innovaciones en el campo de la ingeniería del software en proyectos basados en software libre, (1) para ayudar a los investigadores a evaluar sus herramientas, metodología y diseño de procesos en el mundo real, y (2) para contribuir a que los proyectos basados en Software Libre mejoren sus métodos de trabajo gracias a conocimientos más modernos e innovadores. Esta investigación tratará de ir más allá de la simple difusión de la innovación, adentrándonos en una introducción activa, para aumentar las posibilidades de que el proyecto adopte la nueva tecnología. También hablaremos de la metodología seguida en nuestra investigación, nuestros primeros resultados, las limitaciones de nuestra metodología y por qué los investigadores interesados en evaluar sus propias innovaciones deberían leer este estudio.*

Palabras clave*: adopción de innovaciones, comunidad de software libre, decisión de innovación, invención en Ingeniería de Software, proyecto anfitrión, proyecto de software libre.*

Autores

Christopher Oezbek es diplomado en Informática por la Universidad de Karlsruhe (2002), Master en Informática por el Georgia Institute of Technology (2004) y está cursando el doctorado en la Freie Universität de Berlín. Sus intereses investigadores incluyen los procesos de desarrollo de software libre, documentación de código fuente y los API de usabilidad. Es miembro de ACM y de la asociación de informáticos alemana GI (*Gesellschaft für Informatik*).

Lutz Prechelt es profesor de Informática de la Freie Universität de Berlín desde 2003. Hasta el año 2000, trabajó como investigador senior en la Escuela de Informática de la Universidad de Karlsruhe donde se doctoró en 1995. Después, trabajó en abaXX Technology, Stuttgart, primero como jefe de varios departamentos y después como Director de Tecnología. Sus intereses investigadores incluyen la Ingeniería del Software (usando enfoques empíricos), problemas de medida y comparación, y metodología de la investigación. Actualmente está investigando en desarrollo de software libre, métodos ágiles y plataformas de desarrollo web. Es miembro de IEEE CS, ACM y GI (*Gesellschaft für Informatik*). Asimismo, es editor del *Forum for Negative Results* (FNR) del Journal of Universal Computer Science (J.UCS).

1. Introducción

La mayor parte de las investigaciones en ingeniería del software da como resultado tecnología en forma de herramientas, metodologías o procesos que se aplican en la creación de sistemas software. Poco a poco se va teniendo conciencia de que es necesario evaluar empíricamente estas invenciones para conocer con exactitud los avances conseguidos y obtener reconocimiento fuera del ámbito académico [25][28].

Hay dos métodos clásicos para desarrollar una evaluación empírica: El primero de ellos consiste en pruebas de laboratorio; normalmente se trata de experimentos muy concretos desarrollados por estudiantes. El principal problema es que es muy difícil que estas pruebas sean lo suficientemente imparciales, realistas (sobre todo en las tareas a realizar) y objetivas como para que resulten creíbles, cuando su objetivo precisamente es que lo sean [19]. Los experimentos controlados en temas profesionales son más difíciles de abordar, pero a menudo difícilmente más creíbles. El segundo método consiste en pruebas en el ámbito empresarial, llevadas a cabo como casos de estudio en cooperación con una empresa. Este tipo de estudios gozan de un alto nivel de realismo, pero no están exentos de problemas: el coste y posible riesgo para la empresa hacen difícil encontrar empresas que deseen colaborar. Los acuerdos de confidencialidad hacen difícil poder describir de forma anticipada las hipótesis iniciales y los resultados, y además, la idiosincrasia propia de las empresas dificulta aceptar la generalización de los resultados.

Para muchos propósitos de evaluación, algunos investigadores opinan que realizar observaciones en el contexto de proyectos de software libre (OSS en su acrónimo en lengua inglesa[1]) es un tercer método de evaluación que proporciona unas condiciones ideales de estudio en muchos sentidos: su credibilidad es alta, son fáciles de observar, casi nunca existen restricciones en cuanto a divulgación de la información, las consideraciones sobre el riesgo son más relajadas y las objeciones de coste empresarial se sustituyen por meros obstáculos de voluntad de grupo.

Desafortunadamente, los proyectos OSS no están interesados en estudios, sino en desarrollar software. Así que si deseamos realizar un estudio sobre una nueva tecnología debemos conseguir que el proyecto acepte esa nueva tecnología y la incluya en su trabajo cotidiano. Sin embargo, cualquiera que haya intentado que un grupo de personas adopte una nueva invención (es decir, presentar una invención como una innovación) habrá descubierto que es una tarea realmente difícil.

Así que, en lugar de permitir que un gran número de investigadores prueben, fallen, vuelvan a intentar y al fallar de nuevo se frustren y abandonen la investigación, nuestra sugerencia es hacer el proceso de adopción el centro de nuestro estudio, para proporcionar a los investigadores una metodología probada para introducir una innovación en un proyecto de software libre.

En este estudio se utilizará el término *introducir* con el significado de comenzar un proceso de adopción planificado en una organización o sistema social. Se puede entender que la *adopción* es el punto de inflexión en el que las tecnologías se convierten en

[1] **Nota del Editor:** el artículo se refiere a Open Source Software (OSS) en su sentido más amplio, es decir software abierto y libre para ser modificado. De ahí que traduzcamos ese término por "software libre". En inglés existe el problema de que la palabra "free" es polisémica y puede significar tanto "libre" como "gratuito". De modo que muchos autores de habla inglesa prefieren referirse al software libre como "Open Source Software" en lugar de "Free Software" para que no se confunda con el software gratuito.

innovaciones que se utilizan de una forma común [7]. El término *introducción* contrasta con el término *difusión*, que conlleva connotaciones pasivas, y el verbo *diseminar*, que simplemente se refiere a la distribución de información o recursos.

Desde el punto de vista del investigador, unir la introducción de innovaciones y los proyectos OSS tiene muchas ventajas. Al contrario de lo que ocurre en el ámbito industrial, la visibilidad pública del proceso de creación, sus utilidades auxiliares y su comunicación, así como su apertura a contribuciones de personas externas permiten al desarrollador recoger más información y tener una mayor influencia. Al contrario que con la difusión y diseminación, el investigador puede (1) observar la adopción y utilización de la nueva tecnología en el momento en el que se produce, en lugar de hacer análisis a posteriori, (2) adaptar la tecnología a las particularidades del proyecto para solucionar problemas que a menudo se presentan en las primeras fases de la invención a estudiar y (3) elegir el proyecto para maximizar la perspectiva obtenida.

Por otro lado, desde el punto de vista de la comunidad OSS, este tipo de investigaciones incrementa sus posibilidades de beneficiarse de mejoras en la Ingeniería del Software, dado que los enfoques convencionales en la gestión de la mejora de procesos software como CMMI [5], o incluso enfoques especializados en OSS [8] no precisan cómo se debe realizar la introducción real de mejoras, y los mecanismos de éxito tradicionales como el compromiso y soporte de la dirección [24], es poco probable que funcionen.

El resto del artículo presenta nuestro enfoque investigador para obtener una mejor perspectiva en la introducción de nuevas tecnologías en proyectos OSS, así como nuestros resultados preliminares sobre las siguientes problemáticas de investigación:
1. Cómo seleccionar proyectos apropiados para introducir invenciones en Ingeniería del Software.
2. Cómo acceder a un proyecto para ofrecer una tecnología nueva.
3. Cómo interpretar las reacciones y tomar decisiones estratégicas y tácticas basadas en ellas en el transcurso del proceso de adopción.
4. Cómo reducir la implicación y abandonar el proyecto.
5. Cómo obtener los resultados de la evaluación durante y después de la introducción.

2. Enfoque de la investigación

Para comprender la problemática de la introducción de innovaciones llevaremos a cabo una serie de casos de estudio iterativos [27], utilizando una metodología acción-investigación [2], es decir, un proceso repetitivo y colaborativo de planificación, ejecución y reflexión. Este estudio se aplicará a tres tipos distintos de invenciones y a un gran abanico de proyectos de software libre. Evitaremos la introducción de varias mejoras en un mismo proyecto [9], con la finalidad de evitar posibles sinergias o canibalización entre mejoras [11].

Dentro de cada caso de estudio recabaremos una gran cantidad de datos cualitativos en relaciones causa-efecto y patrones recurrentes (utilizando la metodología de análisis comparativo *Grounded Theory* [6]) para obtener y comprender las claves de las interacciones que se dan a la hora de introducir una nueva tecnología.

Procuraremos minimizar los riesgos que puedan producirse en el proyecto, así como proteger la autonomía y objetividad de los sujetos del estudio [4]. Esto se conseguirá creando un buen ambiente de colaboración, implicación y participación entre el proyecto y el investigador, y garantizando la privacidad y la confidencialidad [3][13]. Aunque se sabe que los proyectos OSS son muy robustos frente a influencias negativas externas, los investigadores que evalúan sus invenciones en un proyecto han de tomar precauciones similares para asegurar que se produzca un comportamiento ético.

3. Cómo elegir el proyecto anfitrión

A la hora de elegir el proyecto OSS adecuado para desarrollar un estudio de este tipo es importante optar por un proyecto que sea (a) lo suficientemente típico como para que se pueda extrapolar la información a otros proyectos, (b) adecuado para la tecnología a implantar y (c) que tenga potencial para la creación de interacciones de interés alrededor de la introducción.

En concreto, el proyecto debe ser libre no sólo en cuanto a licencias de distribución, sino también por estilo de desarrollo: los participantes en el proyecto deberían estar geográficamente distribuidos en lugar de estar ubicados en una misma sede empresarial, la comunicación entre los participantes debe ser pública (y es recomendable que se almacene), debe permitirse la entrada a nuevos desarrolladores, y los procesos y herramientas básicos de trabajo (procesos de publicación, seguimiento de problemas y repositorio de versiones) deben estar bien definidos.

La distribución, observabilidad y apertura del proyecto aseguran que el investigador pueda estudiar la implantación de la tecnología objeto de estudio, mientras que la presencia de procesos y herramientas bien definidos indica que probablemente el proyecto cumple con los estándares profesionales básicos de la Ingeniería del Software, de modo que se puedan generalizar los resultados a otros proyectos de desarrollo de software. Afortunadamente, gracias a la existencia de proyectos anfitriones como SourceForge este tipo de procesos y herramientas son ahora estándar.

Respecto al tamaño del proyecto, éstos deben situarse en un tamaño medio, ni muy grandes ni muy pequeños. Los proyectos pequeños, de menos de tres o cuatro desarrolladores, suelen tener poca interacción, comunicación pobre entre miembros, herramientas y procesos ineficientes, o a veces no disfrutan de una metodología de trabajo bien definida, con lo cual no son recomendables a la hora de realizar un estudio de este tipo (excepto para invenciones muy básicas de ingeniería del software).

En cuanto a los proyectos grandes, de más de cincuenta desarrolladores, presentan el problema contrario: gozan de metodologías de desarrollo bien definidas, de modo que el síndrome del "*Not Invented Here*", el rechazo explícito, el lento proceso de búsqueda de consenso, la escasa percepción de los beneficios frente a los procesos establecidos y la saturación de la comunicación entre los miembros, pueden impedir que se preste atención a un investigador único. Por lo tanto, recomendamos que se opte por un proyecto de tamaño medio: entre cinco y cincuenta desarrolladores, de los cuales como mínimo cinco hayan estado desarrollando activamente en los últimos meses.

Como último requisito, estimamos oportuno elegir un proyecto que se haya mostrado receptivo al cambio (o que, al menos, no haya presentado rechazos de este tipo en el pasado). En muchos casos existe correlación entre esta propiedad y la capacidad del equipo de aceptar nuevos miembros, por lo que se recomienda estudiar los cambios e innovaciones adoptadas por el proyecto en el pasado, como por ejemplo la transición desde el sistema de control de versiones CVS al nuevo (y claramente superior) sistema SVN.

Para elegir un proyecto que cumpla estas características al tiempo que se mantiene cierta aleatoriedad en la elección, se recomienda visitar sitios web de noticias sobre proyectos como Freshmeat ,que agrega proyectos independientemente de su alojamiento, o SWiK, un directorio de proyectos. Ambos sitios permiten visitar aleatoriamente un proyecto de su lista. Mientras que SWiK muestra todo tipo de proyectos de Software Libre, la notable limitación de Freshmeat es que únicamente muestra proyectos OSS que se ejecuten bajo sistemas libres, por lo que no mostrará ningún proyecto que corra exclusivamente bajo plataforma Windows.

4. Cómo acceder a proyectos basados en Software Libre

Existe cierto conocimiento en la literatura sobre cómo acceder a un proyecto OSS [10][26]. Por un lado existe el concepto de "*gift culture*" [21] (cultura del altruismo), que explica que la influencia y el respecto que pueda tener un nuevo miembro del equipo de desarrollo es mayor cuanto más aporte esa persona al proyecto. Esto plantea la pregunta de si la invención en sí misma será vista como un regalo al diseminarse en el proyecto. Un caso de estudio al respecto, en el que se estudió la donación de una tecnología que requería cierto esfuerzo para ser integrada en el código base del proyecto, llegó a la siguiente conclusión: a menos que la donación tenga utilidad inmediata para el proyecto y sea comprensible inmediatamente por parte de los desarrolladores, las probabilidades de éxito serán realmente bajas [20]. Así que podemos suponer que el investigador debe plantearse emplear una considerable cantidad de esfuerzo generando estos beneficios hasta conseguir que la invención sea aceptada y adoptada.

En segundo lugar, el investigador necesita decidir si es recomendable entrar en contacto principalmente con los jefes de proyecto y desarrolladores principales, o es mejor tratar con la comunidad en general. Nuestra hipótesis sugiere que el tipo de acercamiento al proyecto depende de (a) el grado de independencia en la decisión de adopción de cada miembro del equipo, y (b) el nivel de beneficio que represente la innovación al proyecto. A continuación explicamos estos factores.

En "*Diffusion of Innovations*", Rogers distingue tres tipos de decisiones de innovación: *opcional*, en la que cada miembro del equipo de desarrollo es completamente independiente, *colectiva*, en la que se necesita cierto consenso, y *autoritaria*, que es tomada por un pequeño grupo de influencia dentro del proyecto [22].

Como ejemplo contémplese la adopción de un procedimiento en un equipo de desarrollo. La nueva norma que se debe aceptar es la siguiente: "es obligatorio realizar una revisión entre pares antes de enviar cambios (parches) al sistema de control de versiones". Este tipo de cambios suele empezar por una decisión de innovación colectiva para mejorar la calidad del código, dado que se necesita un consenso general para que cada miembro del proyecto envíe sus parches a una lista de e-mail general, y así la comunidad entera debe ser requerida para que promueva la adopción. Además, también implica una decisión de tipo opcional, ya que los miembros del equipo pueden participar y revisar los cambios enviados por los demás, y así puede ser también auspiciada por el investigador hablando individualmente con miembros del grupo. Como ejemplo del tercer tipo de decisión, y las implicaciones que conlleva para un investigador, consideremos el siguiente caso: dos semanas antes del lanzamiento de una nueva versión del proyecto se decide congelarla, es decir, se decide que a partir de este momento no se van a seguir integrando nuevas características, sino simplemente resolviendo fallos y depurando código. Esta decisión la pueden tomar los líderes y mantenedores del proyecto de una forma autoritaria, y soportada técnicamente mediante la creación de una rama local de la versión en el sistema de control de versiones. Los demás miembros del proyecto pueden discrepar de la decisión, pero realmente no pueden realizar ninguna acción al respecto. En este caso, el investigador debe comunicarse directamente con los jefes de proyecto.

La segunda gran característica de la nueva tecnología que puede afectar al acercamiento al proyecto es la estructura del beneficio que ofrece dicha innovación, es decir, la rentabilidad de la inversión o ventaja relativa [22] para cada participante del proyecto en contraste con la rentabilidad de la inversión para el proyecto completo. La documentación del proyecto, por ejemplo, no resulta muy rentable para el desarrollador experimentado que la escribe; en cambio, esa información puede ser tremendamente útil para los nuevos desarrolladores (lo cual puede generar grandes beneficios para el proyecto). Los inventores a menudo comprenden los retornos crecientes [1] prometidos por sus innovaciones, pero suelen pasar por alto que (a) los desarrolladores que conduzcan su introducción para ser compensados por el esfuerzo que realizan y que (b) puede ser realmente difícil medir el beneficio que supone, o que quizás ese beneficio solamente se presente a largo plazo.

Nuestra hipótesis es que el investigador debería empezar el acercamiento al proyecto a través de los miembros que más se puedan beneficiar a corto plazo de la implantación de la nueva tecnología. Así que en lugar de pedir a los miembros del proyecto que realicen tareas con una expectativa negativa a nivel personal, se recomienda que las lleve a cabo inicialmente el mismo investigador. Más tarde, una vez que estas tareas comiencen a generar beneficios visibles que afecten a otros individuos, el investigador tendrá muchas más posibilidades de implicar a otros desarrolladores y dejar de ocuparse de dichas tareas.

5. Cómo interpretar las reacciones y tomar decisiones tácticas y estratégicas

Cuando se introducen invenciones y novedades de cualquier tipo en un grupo social, el investigador debería esperar los siguientes tipos de reacción, tanto a nivel individual como colectivo: *rechazo*, *adopción* y *reinvención* [22].

El rechazo consiste en la decisión de no aceptar la innovación. Este rechazo puede ser activo (después de un estudio o toma de contacto) o pasivo, es decir, rechazar la innovación sin ningún motivo [22]. El rechazo pasivo, es decir la no obtención de respuesta, no es raro que se produzca, incluso en casos en los que el investigador expresa interés explícito en incorporarse al proyecto [26].

La reinvención ocurre cuando miembros del equipo de desarrollo utilizan la nueva tecnología de una forma totalmente inesperada por el investigador. La reinvención es tremendamente beneficiosa para el investigador, ya que puede abrir nuevas puertas a diferentes aplicaciones de la tecnología.

Por supuesto, todavía hay muchos aspectos y consideraciones referentes a la interacción entre desarrolladores, investigadores y tecnología hasta que se producen estos últimos pasos o reacciones. Existe mucha documentación en cuanto a ingeniería social, describiendo modelos que pueden encajar en estos desarrollos, como la teoría de campos (*theory of fields*) [12] o la teoría red-actor (*network-actor*) [17]. En nuestro caso hemos elegido el modelo de innovación desarrollado por Denning y Dunham [7]. En este modelo, el proceso de innovación comienza (1) detectando las posibilidades de cambio y (2) intentando saber qué se puede obtener de ese cambio. (3) ofreciendo esta visión a las personas afectadas (u otras unidades de adopción) y recibiendo su retroalimentación, permitiendo así transformar la idea inicial en algo que puede ser (4) derivando un producto, proceso o mejora social a partir de la ejecución e implementación de la idea. Solamente después de que la innovación haya sido (5) adoptada por la población objetivo y (6) se mantenga como una novedad positiva podremos decir que la introducción exitosa de la innovación ha sucedido. En el caso concreto que presenta este estudio, las dos primeras fases se centrarán en confeccionar y desarrollar el problema, la visión y la invención más que en intentar generar nuevas ideas e implementarlas.

6. Cuándo y cómo abandonar el proyecto

El momento de abandonar el proyecto será cuando se dé uno de los siguientes casos: el investigador ha tenido éxito implantando la innovación en un grupo de desarrollo determinado, y la tecnología se mantiene sin ayuda del investigador, o por el contrario la implantación ha fallado y no queda nada que arreglar o solucionar. En el caso de que hayamos tenido éxito, la retirada ha de producirse de forma gradual y no abrupta, so

pena de poner en peligro ese éxito y causar daño al proyecto. Por otra parte, después de una introducción fallida, el abandono del proyecto obliga al investigador a "limpiar", es decir revertir los cambios en el código base y realizar una retirada "ordenada".

7. Cómo obtener los resultados de la evaluación

Los resultados que se puedan obtener de este tipo de investigaciones dependen mucho de la naturaleza de la innovación a implantar y de cuál sea la meta final de la investigación. Para algunas innovaciones, la adopción con éxito puede ser suficiente. Para otras, quizás necesitemos comparar productos, procesos o métricas de uso con los valores previos a la introducción. Otro tipo de innovación puede que necesite que los desarrolladores rellenen una encuesta sobre su experiencia con la nueva tecnología.

Independientemente de estos tres enfoques, la forma con la que probablemente el investigador obtendrá más información útil, de calidad y práctica para mejorar la innovación será la comunicación directa con el equipo de desarrollo del proyecto. Un investigador que utilice la técnica de investigación-acción podría considerar esta información como su principal resultado.

8. Oportunidades, limitaciones y conclusión

Finalmente se nos plantea la pregunta de si la experiencia obtenida al introducir una nueva tecnología en un proyecto OSS se puede extrapolar a otros casos (validez externa). Por ejemplo, en proyectos de diferentes tamaños, dominios de aplicación, arquitecturas de software, con personal no voluntario, distintas configuraciones de gestión, distribución, otros ambientes de trabajo, experiencia previa en ingeniería del software, etc. El caso más típico es el estudio de un entorno corporativo dependiente de los ingresos. Los siguientes argumentos abogan porque la evaluación de resultados de proyectos OSS pueda transferirse a tales entornos: (1) Los desarrolladores de software libre se caracterizan por ser críticos ante los resultados académicos, (2) La existencia de liderazgo en la dirección y otras motivaciones externas (como compensaciones económicas) pueden a menudo impulsar la adopción y uso, y (3) los empleados a tiempo completo se beneficiarán más de las economías de escala y de los efectos del aprendizaje que desarrolladores de software libre trabajando a tiempo parcial.

La limitación más importante de nuestra investigación son las restricciones de las características concretas que debe presentar la nueva tecnología para que sea adecuada para la investigación. La literatura sobre difusión de la innovación señala varios atributos de la invención que afectarán a su tasa de éxito al ser introducida: (1) la compatibilidad de la invención con tecnologías, valores y creencias preexistentes (por ejemplo, los equipos de desarrollo de software libre pueden rechazar trabajar con herramientas que no estén a su vez licenciadas como software libre), (2) su complejidad técnica e intelectual, (3) la facilidad con que se puedan observar sus resultados, (4) la posibilidad de experimentar con la tecnología antes de ponerse a trabajar con ella, y (5) la incertidumbre sobre la innovación [22].

Halloran y Scherlis mantienen que los proyectos OSS diferencian mucho entre contribuciones de confianza y contribuciones que no generan confianza (metáfora del servidor amurallado, en inglés *"walled server"*), y que las invenciones necesitan preservar esta distinción para ser aplicables a un proyecto OSS [15]. Esto se puede interpretar de la siguiente manera: Una introducción exitosa de una tecnología puede significar que la tecnología es realmente valiosa; en cambio, una introducción sin éxito puede indicar que el proyecto OSS no presentaba las características idóneas (por ej., "el servidor amurallado"), y no que la tecnología no sea interesante.

Una segunda limitación que queremos remarcar es que, al contrario de otros trabajos de campo o estudios etnográficos llevados a cabo en grandes empresas (ver como ejemplo [18]), aquí será difícil estudiar las prácticas y procesos reales de cada participante en el proyecto, ya que sólo los resultados intermedios y los del proceso, tales como informes de errores, envíos a CVS y discusiones en listas de correo, son visibles al investigador. Para obtener información sobre cuánto se utiliza cada herramienta en cada ordenador de cada desarrollador es necesario instrumentarlo de forma adecuada [16][23].

Una tercera limitación del estudio se refiere a la velocidad de adopción de la tecnología. Los proyectos de software libre suelen ser desarrollados por personal voluntario, que suelen trabajar menos de 10 horas semanales en el proyecto, y se coordinan de forma asíncrona entre diferentes zonas horarias [14]. Por lo tanto, la velocidad de cambio es mucho más lenta que en un ambiente comercial, en el que los desarrolladores trabajan un número regular de horas y se comunican frecuentemente de forma síncrona.

Resumiendo, nuestra intención ha sido estudiar la introducción de innovaciones en el campo de la Ingeniería del Software, para ayudar a otros investigadores a evaluar herramientas, métodos y procesos desarrollados en un ambiente académico, y hemos ofrecido unos resultados preliminares.

Mientras que la comunidad investigadora se puede beneficiar del acceso a entornos de trabajo real y de la posibilidad de interactuar con la comunidad del Software Libre, dicha comunidad recibe tecnología punta ajustada a sus problemas específicos por parte de los inventores.

Referencias

[1] W. B. Arthur. *Increasing Returns and Path Dependence in the Economy*. University of Michigan Press, 1994. ISBN: 0472064967.

[2] D. E. Avison, F. Lau, M. D. Myers, P. A. Nielsen. Action research. *Commun. ACM, 42(1):94–97*, 1999.

[3] M. Bakardjieva, A. Feenberg. Involving the virtual subject. *Ethics and Information Technology, 2(4):233–240*, 2001.

[4] J. Cassell. Ethical principles for conducting fieldwork. *American Anthropologist, 82(1):28–41*, Marzo 1980.

[5] CMMI Product Team. CMMI for development, version 1.2. Technical Report CMU/SEI-2006-TR-008, Software Engineering Institute, 2006.

[6] J. M. Corbin, A. Strauss. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative Sociology, 13(1):3–21*, Mar. 1990.

[7] P. J. Denning, R. Dunham. Innovation as language action. *Commun. ACM, 49(5):47–52*, 2006.

[8] S. Dietze. Modell und Optimierungsansatz für Open Source Softwareentwicklungsprozesse. Tesis doctoral, Universität Potsdam, 2004.

[9] G.W. Downs, L. B. Mohr. Conceptual issues in study of innovation. *Administrative Science Quarterly, 21(4):700–714*, 1976.

[10] N. Ducheneaut. Socialization in an open source software community: A socio-technical analysis. *Computer Supported Cooperative Work (CSCW), V14(4):323–368*, Agosto 2005.

[11] M. L. Fennell. Synergy, influence, and information in the adoption of administrative innovations. *Academy Of Management Journal, 27(1):113–129*, 1984.

[12] N. Fligstein. Social skill and the theory of fields. *Sociological Theory, 19(2):105–125*, Julio 2001.

[13] M. S. Frankel, S. Siang. Ethical and legal aspects of human subjects research on the internet. Published by AAAS online , June 1999.

[14] R. A. Ghosh, B. Krieger, R. Glott, G. Robles, T. Wichmann. Free/Libre and Open Source Software: Survey and Study – FLOSS. Final Report, International Institute of Infonomics University of Maastricht, The Netherlands; Berlecon Research GmbH Berlin, Germany, Junio 2002.

[15] T. J. Halloran, W. L. Scherlis. High Quality and Open Source Software Practices. En J. Feller, B. Fitzgerald, F. Hecker, S. Hissam, K. Lakhani, and A. van der Hoek, editors, *Meeting Challenges and Surviving Success: The 2nd Workshop on Open Source Software Engineering, pages 26– 28*. ACM, 2002.

[16] P. M. Johnson, H. Kou, J. Agustin, C. Chan, C. Moore, J. Miglani, S. Zhen, W. E. J. Doane. Beyond the personal software process: metrics collection and analysis for the differently disciplined. En ICSE '03: *Proceedings of the 25th International Conference on Software Engineering, pages 641–*

646, Washington, DC, USA, 2003. IEEE Computer Society.

[17] J. Law. Notes on the theory of the actor-network: Ordering, strategy and heterogeneity. *Systems Practice, 5(4):379–393*, 1992.

[18] T. C. Lethbridge, J. Singer. Experiences conducting studies of the work practices of software engineers. En H. Erdogmus and O. Tanir, editors, *Advances in Software Engineering: Comprehension, Evaluation, and Evolution, pages 53–76*. Springer, 2001.

[19] D. E. Perry, A. A. Porter, L. G. Votta. Empirical studies of software engineering: a roadmap. In *Proceedings of the conference on The future of Software engineering, pages 345–355*. ACM Press, 2000.

[20] L. Quintela García. Die Kontaktaufnahme mit Open Source Software-Projekten. Eine Fallstudie. Bachelor thesis, Freie Universität Berlin, 2006.

[21] E. S. Raymond. *The Cathedral and the Bazaar*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1999. ISBN: 0596001088.

[22] E. M. Rogers. Diffusion of Innovations. Free Press, New York, 5th edition, August 2003. ISBN: 0743222091.

[23] F. Schlesinger, S. Jekutsch. ElectroCodeoGram: An environment for studying programming. En Workshop on Ethnographies of Code, Infolab21, Lancaster University, UK, Marzo 2006.

[24] D. Stelzer, W. Mellis. Success factors of organizacional change in software process improvement. *Software Process: Improvement and Practice, 4(4):227–250*, 1998.

[25] W. F. Tichy, P. Lukowicz, L. Prechelt, E. A. Heinz. Experimental evaluation in computer science: A quantitative study. *Journal of Systems and Software, 28(1):9–18*, Enero 1995.

[26] G. von Krogh, S. Spaeth, K. Lakhani. Community, joining, and specialization in open source software innovation: A case study. *Research Policy, 32:1217–1241(25)*, Julio 2003.

[27] R. K. Yin. *Case Study Research: Design and Methods*. Applied Social Research Methods. Sage Publications, Inc., 1988.

[28] M. V. Zelkowitz, D. R. Wallace. Experimental models for validating technology. *Computer, 31(5):23–31*, 1998.

Carlos González Morcillo[1,2] Gerhard Weiss[2] David Vallejo Fernández[1] Luis Jiménez Linares[1] Javier Albusac Jiménez[1]

[1] Escuela Superior de Informática, Universidad de Castilla-La Mancha; [2] Software Competence Center, Hagenberg (Austria)

<{carlos.gonzalez, david.vallejo, luis.jimenez, javieralonso.albusac}@uclm.es>, <{carlos.morcillo, gerhard.weiss}@scch.at>

Optimización del proceso de *render* 3D distribuido con software libre

Resumen: *en la última década las empresas que trabajan con contenido multimedia demandan imágenes de alta definición para sus proyectos de síntesis 3D. El proceso de renderizado consiste básicamente en la obtención de una imagen 2D a partir de la definición abstracta de una escena 3D. A pesar de la aparición de nuevas técnicas y algoritmos, el coste computacional de este proceso es elevado ya que necesita gran cantidad de tiempo para ser realizado, especialmente cuando la escena es compleja o bien cuando es necesario obtener imágenes fotorrealistas. En este artículo se describe Yafrid (Yeah! a Free Render grID) y MAgArRo (MultiAgent AppRoach to Rendering Optimization) ambas arquitecturas han sido desarrolladas en la Universidad de Castilla-La Mancha para la optimización del renderizado distribuido.*

Palabras clave: *agentes inteligentes, inteligencia artificial, optimización, renderizado.*

Autores

Carlos Gonzalez Morcillo es profesor ayudante y estudiante de doctorado en el grupo de investigación ORETO de la Universidad de Castilla-La Mancha. Sus temas de investigación recientes son los sistemas multiagente, *rendering* distribuido y lógica difusa. Obtuvo su licenciatura y un Máster en Informática por la Universidad de Castilla-La Mancha en 2002 y 2004, respectivamente.

.

Gerhard Weiss es el director científico de SCCH (*Software Competence Center Hagenberg GmbH*), uno de los centros de investigación independientes más importantes de Austria. Sus intereses principales son la inteligencia computacional y los sistemas autónomos en general, así como los fundamentos y aplicaciones de la tecnología de agentes y multiagentes en particular. Ha sido coeditor del libro de referencia en esta área "Multiagent Systems" (MIT Press).

David Vallejo Fernandez es profesor ayudante y estudiante de doctorado en el grupo de investigación ORETO de la Universidad de Castilla-La Mancha. Sus temas de investigación recientes son los sistemas multiagente, arquitecturas de inspiración cognitiva y rendering distribuido. Se licenció en Informática por la Universidad de Castilla-La Mancha en 2006.

Luis Jimenez Linares es profesor asociado de Informática en la Universidad de Castilla-La Mancha. Sus temas de investigación más recientes son los sistemas multiagente, representación del conocimiento, diseño de ontologías y lógica difusa. Obtuvo un Máster y un Doctorado en Informática por la Universidad de Granada en 1991 y 1997, respectivamente. Es miembro de la *European Society of Fuzzy Logic and Technology*.

Javier Albusac Jimenez es investigador y estudiante de doctorado en el grupo de investigación ORETO de la Universidad de Castilla-La Mancha. Sus temas de investigación más recientes son los sistemas multiagente, la vigilancia cognitiva y la visión cognitiva. Se licenció en Informática por la Universidad de Castilla-La Mancha en 2006.

1. Introducción

El proceso de *renderizado*[1] consiste en la generación de una imagen 2D a partir de la descripción abstracta de una escena 3D. La construcción de una imagen bidimensional requiere varias fases tales como el modelado, la configuración de materiales y texturas, la ubicación de las fuentes de luz virtuales, y finalmente, el *renderizado*. Los algoritmos utilizados en el proceso de *renderizado* utilizan como entrada información relacionada con la geometría de cada objeto, materiales, texturas, fuentes de luz y cámaras virtuales, y a partir de ésta se obtiene como salida una imagen, o una secuencia de imágenes si se trata de una animación. El *renderizado* de imágenes fotorrealistas de alta calidad es uno de los objetivos principales de la informática gráfica. Desafortunadamente, este proceso es computacionalmente muy costoso y necesita gran cantidad de tiempo cuando las técnicas de *renderizado* empleadas simulan la iluminación global de la escena. Dependiendo del método de *renderizado* y las características de la escena, la generación de una imagen en alta calidad podría suponer varias horas (¡o incluso días!) de cálculo.

En la última década se han propuesto varias soluciones a este problema basadas en el procesamiento paralelo y distribuido. Una de las más populares es las *granjas de render* formadas por ordenadores de una misma organización donde cada uno de los fotogramas de una animación se calcula con un procesador diferente. A las anteriores hay que añadir las propuestas basadas en sistemas *GRID*[2] donde se utiliza Internet para compartir ciclos de CPU.

En este artículo se describe el flujo de trabajo y las herramientas de software libre utilizadas en la Universidad de Castilla-La Mancha (UCLM) en varios proyectos para el *renderizado* de escenas 3D, así como algunas herramientas desarrolladas en la UCLM y distribuidas bajo licencia GPL (*General Public License*). En las próximas secciones se describirá con mayor detalle la arquitectura general de Yafrid y el sistema de optimización del proceso de *render* llamado MAgArRo. Finalmente, se muestran los resultados experimentales y los beneficios que se obtienen de la utilización de un sistema de estas características. El artículo está organizado de la siguiente forma: la siguiente sección ofrece una visión general sobre el estado del arte y algunos de los principales trabajos relacionados con la optimización del proceso de *renderizado*; principalmente, los que están basados en el proceso de *renderizado* en paralelo y distribuido. Las siguientes secciones describen la arquitectura general de un *cluster* basado en Oscar, el sistema *GRID* de *renderizado* llamado Yafrid

y la arquitectura para la optimización inteligente y distribuida del proceso de *renderizado*, llamada MAgArRO. Posteriormente, en la siguiente sección se muestran los resultados que han sido obtenidos empíricamente a partir de los sistemas citados anteriormente. Para finalizar, se expondrán las conclusiones y las propuestas de trabajo futuro.

1.1. Trabajos relacionados

Existen numerosos métodos y algoritmos con diferentes características y propiedades para el proceso de *renderizado* [11][6][10]. Sin embargo, según Kajiya [6] todos los algoritmos de *render* se centran principalmente en el modelo de comportamiento de la luz sobre varios tipos de superficie, y cómo intentan resolver la llamada *ecuación de render*, la cual constituye la base matemáticas de cualquier algoritmo de *render*. Para cualquiera de estos algoritmos, los niveles de realismo alcanzados están relacionados con la complejidad de la escena y el coste computacional requerido. Chalmers et al. exponen en [3] varias líneas de investigación relacionadas con la optimización del *renderizado* distribuido.

Optimización mediante hardware. Una alternativa para reducir el tiempo de *renderizado* consiste en realizar optimizaciones particulares mediante hardware. En esta línea de investigación existen diferentes aproximaciones; algunos métodos utilizan GPUs (*Graphics Processing Units*) programables como sistemas de paralelización masiva. De esta forma se dispone de potentes procesadores para la ejecución en paralelo de diferentes fragmentos de código de un trazador de rayos. El uso de GPUs programables superan a las CPUs (*Central Processing Unit*) de una estación de trabajo estándar en un factor de siete [2]. El uso de la CPU en conjunción con la GPU requiere nuevos paradigmas y alternativas a las arquitecturas tradicionales. Por ejemplo, la arquitectura propuesta por Rajagopalan et al. [8] muestra el uso de un GPU para *renderizar* imágenes en tiempo real a partir de conjuntos de datos complejos, los cuales requieren tareas de cómputo de gran complejidad. Existen algunos motores de *render* diseñados específicamente para ser utilizados con aceleración GPU, tales como Parthenon Renderer [5] el cual utiliza el punto flotante de la GPU, o el motor de *render* Gelato que trabaja con tarjetas gráficas Nvidia.

Optimización utilizando computación distribuida. Si dividimos el problema en otros más pequeños (y cada uno de ellos es resuelto en un procesador diferente), el tiempo necesario para resolver el problema completo debería disminuir. Para obtener una solución óptima al problema todos los ordenadores que forman el sistema deberán mantenerse ocupados. Por tanto, es necesario elegir una buena estrategia de distribución de tareas para conseguir este objetivo.

De acuerdo a la forma en la que se ha realizado la división de tareas, podemos distinguir sistemas de **granularidad fina**, si una imagen se ha dividido en fragmentos más pequeños y éstos han sido enviados a distintos procesadores para que sean ejecutados de forma independiente, o bien, sistemas de **granularidad gruesa** (en el caso de las animaciones) si cada fotograma de la animación sin fragmentar es enviado a una unidad de procesamiento. Dr. Queue [17] es una herramienta de código abierto diseñada para distribuir fotogramas a través de una granja de computadores interconectados en una red; en concreto, este software multi-plataforma trabaja en un nivel de división de granularidad gruesa. En la **sección 2**, la solución que proponemos está basada en Oscar [18], una herramienta de código abierto para la construcción de *clusters*. Muchas de las nuevas propuestas que se hacen en la literatura para el *rendering* distribuido están basadas en sistemas *GRID*, donde las tareas se distribuyen entre un número amplio de ordenadores conectados a Internet con características heterogéneas; estos sistemas se basan principalmente en el aprovechamiento de los ciclos de CPU de aquellos procesadores que se encuentran ociosos [1]. Esta tecnología emergente se denomina *Volunteer Computing* o *Peer-to-peer Computing*, y actualmente, se utiliza en algunos proyectos basados en la tecnología BOINC (*Berkeley Open Infraestructure for Network Computing*: por ejemplo, BURP [16] *Big and Ugly Rendering Project*). En la **sección 3**, se describirá la arquitectura general de Yafrid y las ventajas clave derivadas de su utilización.

Predicción del coste. El conocimiento relativo a la distribución de costes en la escena (es decir, en los diferentes fragmentos de la escena dividida) puede facilitar significativamente la localización de recursos cuando se está haciendo uso de una aproximación distribuida. De hecho, esta estimación cobra mayor importancia cuando hablamos de producción comercial de *renderizado* (una predicción fiable del coste podría ayudar a cumplir los plazos establecidos y a poder elaborar los presupuestos con mayor precisión). Para solucionar el problema anterior, existen numerosas propuestas basadas en el conocimiento sobre la distribución de costes; un buen ejemplo es [9]. En la **sección 4** exponemos el mecanismo para la predicción de costes utilizado en MAgArRO.

Optimización con sistemas multi-agente distribuidos. Los sistemas multi-agente distribuidos nos permiten obtener una posible alternativa a las propuestas anteriores para la optimización del proceso de *renderizado*. El trabajo de Rangel-Kuoppa [7] utiliza una plataforma multi-agente, basada en JADE, para la distribución interactiva de tareas de *rendering* en un red. Aunque este trabajo emplea la metáfora de sistema multi-agente, en realidad no hace uso de una tecnología multi-agente como tal. De hecho, la plataforma JADE se utiliza únicamente con el propósito de comunicar los nodos, sin utilizar ningún conocimiento experto ni negociación. La arquitectura de MagArRo propuesta en la **sección 4** es un ejemplo de una arquitectura multi-agente distribuida, que emplea el conocimiento proveniente de los expertos para optimizar los parámetros de *renderizado*.

2. *Cluster* basado en Oscar

Hoy en día, las Universidades disponen de laboratorios acondicionados con gran cantidad de ordenadores para las clases prácticas. Sin embargo, la mayoría de ellos se encuentran inactivos durante el periodo vacacional, fines de semana y durante las noches. Esta infraestructura hardware podría ser utilizada y coordinada durante los intervalos de tiempo en los que no es necesario el uso de los ordenadores [14]. Oscar [18] es una plataforma software que permite la programación de *clusters* con máquinas basadas en GNU/Linux. A continuación, se explicará en detalle la arquitectura general del sistema basada en Oscar, y que ha sido utilizada por la Universidad de Castilla-La Mancha para el *renderizado* de algunos proyectos 3D [20][22].

2.1. Esquema general de la arquitectura

En nuestro entorno de ejecución, el sistema está compuesto por 130 estaciones de trabajo localizadas en diferentes aulas. Estos computadores son PCs con características diferentes. Cada una de las aulas tiene un tipo de hardware específico (todos ellos basados en arquitectura x86). Los requerimientos mínimos que debe tener cada computador para pertenecer al sistema son una memoria RAM de 500 MB, una partición SWAP de 1 GB, y tener una conexión de al menos 100Mbits/s (todos los ordenadores están conectados a la red utilizando switches de 100Mbits/s). En la **figura 1** se muestran estos requisitos.

Las aulas donde se utiliza el *cluster* basado en Oscar están dedicadas a una actividad educacional. Por esta razón, no es apropiado instalar software en explotación (de forma permanente) en las máquinas. El subproyecto Thin-Oscar [19] nos permite utilizar máquinas sin disco duro local o partición dedicada para instalar el sistema operativo como miembro del *cluster* Oscar. La

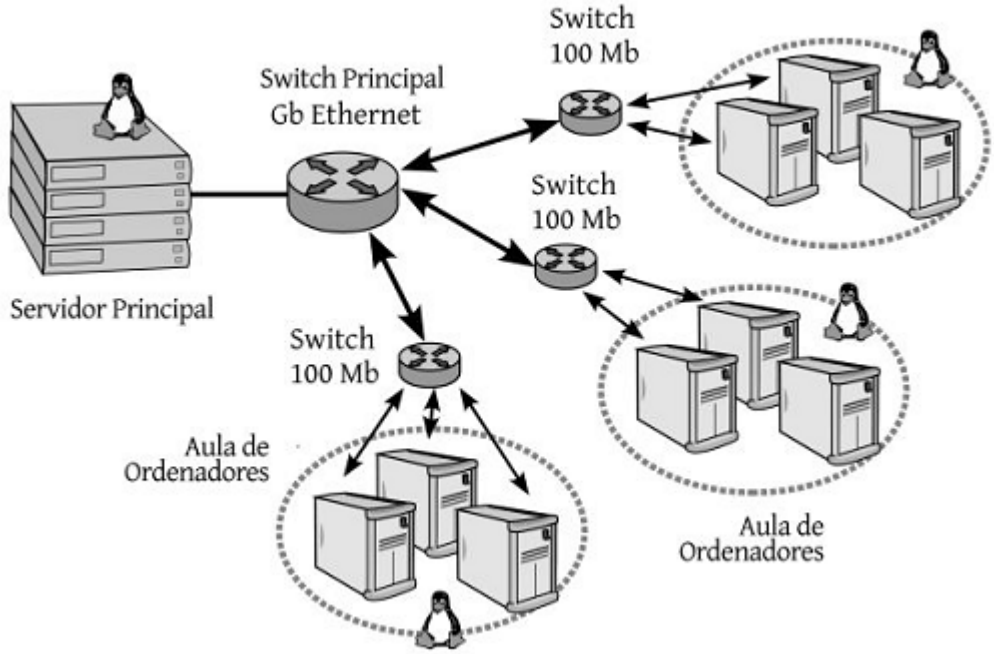

Figura 1. Granja de *rendering* basada en Oscar ubicada en ESI-UCLM.

partición SWAP se utiliza para gestionar los archivos temporales (utilizados en el proceso de *renderizado* en cada proyecto).

Cada nodo del sistema distribuido es configurado para obtener los parámetros de configuración de la red; para ello se utiliza la extensión PXE de la BIOS (*Pre eXecution Environment*). En nuestro caso estos datos se encuentran en la imagen del sistema operativo que será ejecutado. De esta forma, sin necesidad de instalar ningún tipo de software, se carga la imagen del sistema operativo (que está formada únicamente por los servicios básicos y el motor de *render*).

El servidor tiene dos procesos clave para atender las peticiones PXE:

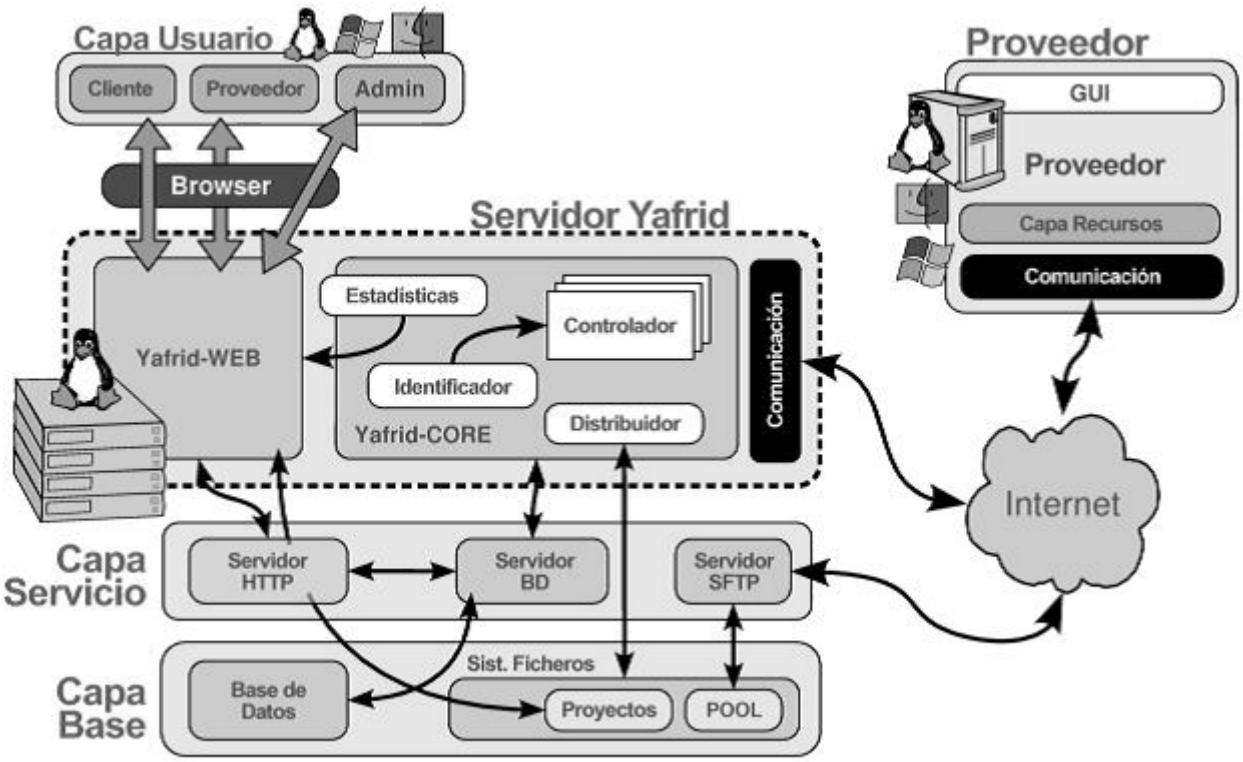

Figura 2. Arquitectura general de Yafrid.

■ **DHCPD** (*Dynamic Host Configuration Protocol Daemon*): este protocolo se utiliza para informar a cada cliente de la IP que le corresponde y la imagen del sistema operativo que debe cargar.
■ **TFTPD** (*Trivial Transfer Protocol Daemon*): Cuando el servidor recibe una petición de archivo, éste consulta las tablas de configuración para enviarlo al cliente.

Para activar o desactivar los ordenadores del sistema, se hace uso de una extensión de las BIOS actuales llamada WOL (*Wake On Lan*). Esta extensión se utiliza con la ayuda de la placa madre y el paquete software Ether-Wake (desarrollado por Donal Becker). Incluso si un ordenador se encuentra apagado, la tarjeta de red continúa escuchando posibles peticiones. Cuando el paquete generado por Ether-Wake llega a la tarjeta, el ordenador se enciende y por medio de una petición PXE se carga la imagen del sistema operativo. Finalmente, para apagar por completo los ordenadores es necesario configurar correctamente la interfaz ACPI (*Advanced Configuration and Power Interface*). Para poder hacer uso de esta funcionalidad se implementó un sencillo script en el lado del servidor el cual establece una conexión ssh con cada nodo y les envía el comando *shutdown* para apagarlos.

3. Yafrid: sistema de *renderizado* basado en *Grid*

Yafrid es un sistema que aprovecha las ventajas de los sistemas de computación *Grid*, permitiendo la distribución de una escena a través de Internet para su *renderizado*. Además, el sistema soporta otras tareas importantes como son la gestión de las unidades de trabajo y el uso controlado del *grid*.

3.1. Esquema general de la arquitectura

Los componentes de Yafrid que se encuentran en la capa de más alto nivel de la arquitectura son:
■ **Servidor**. Es el núcleo de Yafrid. Su componente básico es el *Distribuidor*, encargado de recoger los trabajos de una cola y enviarlos a los proveedores.
■ **Proveedor** . Esta entidad se encarga de procesar las peticiones de los clientes.
■ **Cliente**. Un cliente es una entidad externa que no pertenece al sistema en un sentido estricto. La función principal del cliente es añadir trabajos al sistema *grid*, para que posteriormente sean completados por los proveedores.

A continuación se describen los tres roles de usuario utilizados en Yafrid:
■ **Cliente**. Este rol permite a un usuario añadir nuevos trabajos al *grid*. Un cliente también puede crear y gestionar grupos asociados a los proyectos (clientes y proveedores pueden suscribirse a estos grupos). Únicamente los proveedores que pertenecen al mismo grupo pueden participar en el proceso de *renderizado* del proyecto.
■ **Administrador**. Este usuario es necesario para operar con cualquiera de los componentes que constituye el sistema, y además, posee todos los privilegios para acceder a la información relacionada con cualquier usuario del sistema.
■ **Proveedor.** El proveedor es un usuario que tiene instalado el software necesario para que el *grid* pueda enviarle trabajos.

Servidor Yafrid. El servidor es el nodo fundamental ya que todo el sistema gira en torno a él. Cada uno de los proveedores se conecta a este nodo para ceder al *grid* sus ciclos de CPU, con el objetivo de *renderizar* las escenas que los clientes han añadido.

En la **figura 2** se muestran las cuatro capas que forman la arquitectura del servidor Yafrid. Este diseño está basado en la arquitectura de [4]. Las capas mencionadas anteriormente son *Capa Base (o de Recursos*, *Capa de Servicio*, *Servidor Yafrid* y *Capa de usuario*.

Capa de recursos. Ésta es la capa de menor nivel de abstracción. La capa de recursos tiene los siguientes componentes:
■ **Sistema de base de datos**. En las bases de datos de esta capa se encuentran las tablas donde se almacena la información necesaria para el correcto funcionamiento del sistema. Algunas de estas tablas se utilizan para obtener estadísticas de la ejecución del sistema, mientras que otras almacenan datos asociados a usuarios, grupos, proyectos, etc. Los niveles que se encuentran sobre esta capa en la jerarquía pueden acceder a las bases de datos a través del servidor de bases de datos. La implementación actual del sistema utiliza MySQL como servidor para las bases de datos.
■ **Sistema de archivos**. En algunas ocasiones es necesario acceder directamente al sistema de archivos desde las capas superiores. Básicamente el sistema distingue dos tipos de directorios: uno de ellos se utiliza para almacenar las unidades de trabajo de los proyectos creados, y el segundo tipo contiene información relativa a los usuarios y sus proyectos.
■ **Sistema de red**. El módulo dedicado a las comunicaciones, que pertenece a la capa principal, oculta la utilización de los recursos de red de los ordenadores mediante el uso de un *middleware* (la implementación actual utiliza ZeroC ICE).

Capa de Servicio. Esta capa contiene los servidores que permiten a los módulos de las capas superiores acceder a los recursos que pertenecen a las capas inferiores. Los servidores que actúan en esta capa son los siguientes:
■ **Servidor HTTP.** El módulo web de Yafrid trabaja sobre este servidor. Como este módulo ha sido implementado con lenguajes de programación que permiten la construcción de páginas dinámicas (la implementación actual está escrita en lenguaje PHP), el servidor web debe proporcionar soporte para este tipo de lenguajes. También, es necesario tener soporte para la composición dinámica de gráficos y el acceso a la base de datos.
■ **Servidor de bases de datos**. Este servidor es utilizado por los módulos de Yafrid para acceder a los datos que son indispensables para el correcto funcionamiento del sistema.
■ **Servidor SFTP**. Los proveedores utilizan este servicio para obtener los archivos necesarios para el *renderizado* de las unidades de trabajo. Una vez que el proceso de *renderizado* ha finalizado, se utilizará el servidor SFTP (*Simple File Transfer Protocol*) para enviar al servidor de Yafrid la imagen resultante.

Capa Yafrid. Esta es la capa principal del servidor y está compuesta de dos módulos diferentes (Yafrid-WEB y Yafrid-CORE) que trabajan independientemente el uno del otro. **Yafrid-WEB** es el módulo interactivo del servidor y ha sido implementado mediante un conjunto de páginas dinámicas. **Yafrid-CORE** es la parte no interactiva del servidor.

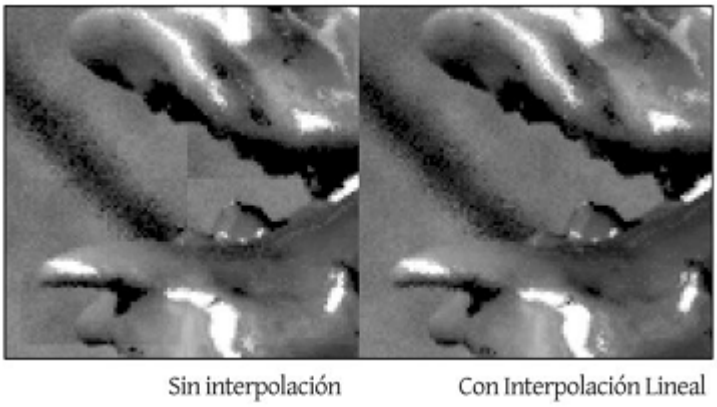

Figura 3. Artefactos sin interpolación entre unidades de trabajo.

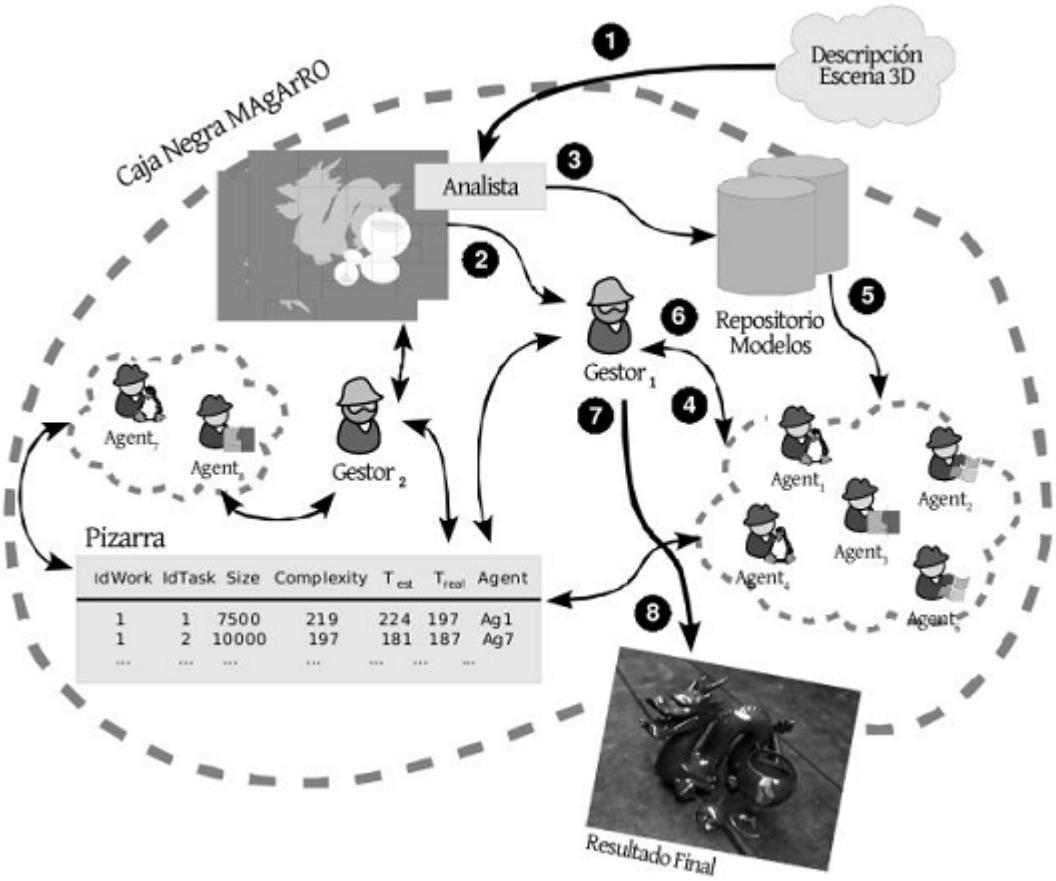

Figura 4. Flujo de trabajo y principales roles en la arquitectura.

Este módulo ha sido principalmente desarrollado utilizando el lenguaje Python. Además, Yafrid-CORE está compuesto por tres submódulos; Distribuidor, Identificador y Estadísticas.

■ **El Distribuidor** es la parte activa del servidor encargado de realizar las siguientes tareas indispensables: generación de unidades de trabajo, asignación de las unidades de trabajo a los proveedores, control del *Timeout*, finalización de proyectos y composición de resultados. A partir de los resultados generados por los proveedores, el distribuidor compone la imagen final. Este proceso no es trivial ya que se pueden distinguir pequeñas diferencias entre los fragmentos obtenidos de diferentes computadores, debido a la componente aleatoria de los métodos basados en Monte Carlo. Por esta razón es necesario suavizar la zona de unión entre los fragmentos mediante una mascara de interpolación lineal. En la **figura 3** (izquierda) podemos apreciar los problemas que surgen al unir dos fragmentos si no se utiliza algún método de combinación de los mismos.

■ La parte pasiva de Yafrid-CORE se denomina módulo **Identificador** cuya misión principal consiste en establecer las comunicaciones de los proveedores. La primera vez que el proveedor intenta conectar con el servidor Yafrid, el Identificador genera un objeto (controlador del proveedor) y le devuelve un *proxy*. Es importante destacar que cada proveedor tiene su propio objeto controlador.

■ **Proveedor**. El proveedor es el software que un usuario debe instalar en su ordenador para que el sistema *grid* pueda utilizar sus ciclos de CPU ociosos. La primera tarea que debe llevar a cabo un proveedor es establecer la conexión con el *grid*. Una vez activado, el proveedor espera hasta que el servidor le envía una unidad de trabajo para procesarla. Cuando finaliza el proceso de *renderizado*, el proveedor envía el archivo vía SFTP e informa al controlador que el trabajo ha finalizado.

4. MagArRO: optimización inteligente y distribuida del proceso de renderizado

De acuerdo a [12] , un agente es un sistema que se encuentra situado en un entorno determinado y es capaz de actuar de forma autónoma en dicho entorno para conseguir los objetivos para los que fue diseñado. MagArRO utiliza los principios, técnicas y conceptos del área a la que pertenecen los sistemas multi-agente, y además, está basado en los principios de diseño de los estándares de FIPA (*Foundation for Intelligent Physical Agents*) [21].

Para el desarrollo de MagArRO también se ha utilizado el *middleware* ICE [25]. IceGrid es el módulo de ICE encargado de la localización de servicios y se utiliza para indicar donde residen los servicios proporcionados por cada ordenador del *grid*. Por otra parte, Glacier2 se utiliza para resolver las dificultades relacionadas con los entornos de red hostiles, haciendo posible que los agentes puedan conectarse a través de un *router* y un *firewall*.

4.1. Esquema general de la arquitectura

En la **figura 4**, se muestra el flujo de trabajo y los principales roles que participan en la arquitectura. Aparte de los servicios básicos de FIPA, MAgArRO incluye servicios específicos relacionados con la optimización del proceso de *renderizado*. En concreto, un servicio denominado *Analista* estudia la escena para dividirla en varios fragmentos. Finalmente, el servicio llamado *Gestor (o Master)* maneja grupos de agentes dinámicos que cooperan para satisfacer las subtareas.

En la **figura 4**, también podemos ver el flujo

de trabajo básico en MagArRO (los círculos numerados lo definen).

1.- El primer paso es la suscripción de los agentes al sistema. Esta suscripción puede hacerse en cualquier momento; los agentes disponibles son gestionados dinámicamente. Cuando el sistema recibe un nuevo archivo para ser *renderizado*, éste es atendido por el Analista. 2.- El Analista analiza la escena, haciendo algunas particiones del trabajo y extrayendo un conjunto de tareas. 3.- Se informa al Gestor de que una nueva escena ha sido creada en el sistema, y además es enviada al Repositorio de Modelos. 4.- Algunos de los agentes disponibles en ese momento son gestionados por el Gestor y se les informa sobre la nueva escena. 5.- Cada uno de los agentes obtiene el modelo 3D del repositorio y comienza un proceso de subasta. 6.- Los agentes ejecutan las (sub) tareas y los resultados obtenidos son enviados al Gestor. 7.- El Gestor compone el resultado final utilizando la salida de las tareas que han sido resueltas previamente. 8.- El Gestor envía la imagen *renderizada* al usuario.

Análisis de la escena utilizando mapas de importancia. MAgArRO utiliza la idea de estimar la complejidad de las diferentes tareas para conseguir el particionamiento de carga balanceada. El agente Analista analiza la complejidad previa e independientemente al resto de pasos pertenecientes al proceso de *renderizado*.

El principal objetivo en este proceso de fragmentación es obtener tareas con complejidad similar para evitar la demora en el tiempo final causada por tareas demasiado complejas. Este análisis se realiza rápidamente y de forma independiente del proceso final de *render*. Una vez que el mapa de importancia es generado, se construye una partición para obtener un conjunto final de tareas. Estas particiones constan de diferentes niveles organizados jerárquicamente, donde en cada nivel se utilizan los resultados del proceso de fragmentación obtenidos del nivel anterior.

En el primer nivel, la partición está hecha siendo cuidadosos con el tamaño mínimo y la complejidad máxima de cada zona. Con estos dos parámetros, el *Analista* realiza una división recursiva de las zonas (ver **figura 5**). En el segundo nivel, se unen las zonas vecinas con complejidad similar. Finalmente, en el tercer nivel el *Analista* intenta obtener una división equilibrada en la que cada zona tenga casi el mismo ratio complejidad/tamaño. La idea que subyace en la división es la de obtener tareas que requieran aproximadamente el mismo tiempo de *renderizado*. Como se muestra abajo en los resultados experimentales, la calidad de este particionamiento está directamente relacionada con el tiempo final de *renderizado*.

Utilizando el Conocimiento Experto. Cuando una tarea es asignada a un agente, se utiliza un conjunto de reglas difusas para modelar el conocimiento experto y optimizar los parámetros del proceso de *renderizado* para esta tarea. Los conjuntos de reglas difusas son considerados apropiados para el modelado del conocimiento experto debido a su poder descriptivo y facilidad de extensión [13]. Los parámetros de salida (la parte consecuente de las reglas) son configurados de modo que el tiempo requerido para completar el *renderizado* se reduzca y la pérdida de calidad esté minimizada. Cada agente puede modelar diferente conocimiento experto con un conjunto distinto de reglas difusas. Por ejemplo, la siguiente regla se usa (en un conjunto de 28 reglas) para describir los parámetros de *renderizado* del método Pathtracing:

R_1: **If** C is {B, VB} $\not\subset$ S is B,N $\not\subset$ Op is VB **then** Ls is VS $\not\subset$ Rl is VS

El significado de esta regla es "Si la complejidad es grande o muy grande y el tamaño es grande o normal y el nivel de optimización es muy grande entonces el número de muestras de luz es muy pequeño y el nivel de recursión es muy pequeño". El parámetro de complejidad representa el ratio complejidad/tamaño de una tarea, el tamaño es el de una tarea en píxeles y el nivel de optimización es seleccionado por el usuario. El parámetro de salida *nivel de recursión*, define el nivel global de recursión en el trazado de rayos (número de rebotes de luz) y las muestras de luz definen el número de muestras por luz en la escena (si es más grande, tendremos mayor calidad y mayor tiempo de *renderizado*).

5. Resultados experimentales

Para analizar el comportamiento de los sistemas se conectaron a Yafrid y MagArRO 8 ordenadores con las mismas características. Estos nodos (Intel Pentium Centrino 2 Ghz, 1GB RAM) se usaron en ambos sistemas durante la ejecución de todas las pruebas. La escena a analizar contenía más de 100.000 caras, 5 niveles de recursión de trazado de rayos en superficies espejo (dragón), 6 niveles en superficies transparentes (copa), y 128 muestras por fuentes de luz fueron *renderizadas* usando el motor libre de *renderizado* Yafray [23]. Además , 200.000 fotones fueron lanzados para construir la estructura del mapa de fotones. Con esta configuración, el *renderizado* en una sola máquina sin ningún tipo de optimización necesitó 121:17 (121 minutos y 17 segundos).

En el caso de Yafrid, como se puede obervar en la **figura 6** (*izquierda*), el tiempo de *renderizado* usando la rejilla es, en el mejor de los casos casi siete veces mejor y dos veces menor en el peor de los casos. Con estos resultados queda clara la importancia de elegir un apropiado tamaño de unidad de trabajo. Esto ocurre debido a que existen tareas complejas que ralentizan el proceso completo de *renderizado* incluso si se aumenta el número de nodos.

Como se ha mencionado anteriormente, MAgArRO usa Mapas de Importancia para estimar la complejidad de las diferentes tareas. La **figura 6** (*derecha*) muestra el tiempo requerido usando diferentes niveles de partición. Utilizando un simple particionamiento de primer nivel (similar al enfoque de Yafrid), puede obtenerse un buen nivel de *renderizado* con pocos agentes. Sin embargo, cuando el número de agentes (nodos de procesamiento) crece, el rendimiento total del sistema se incrementa porque las diferencias en la complejidad de las tareas son relativamente pequeñas.

Figura 5. Mapas de importancia. *Izquierda*: Particionamiento a ciegas (Primer Nivel). *Centro:* Zonas de unión con complejidad similar (Segundo Nivel). *Derecha:* Equilibrio ratio complejidad/tamaño (Tercer Nivel).

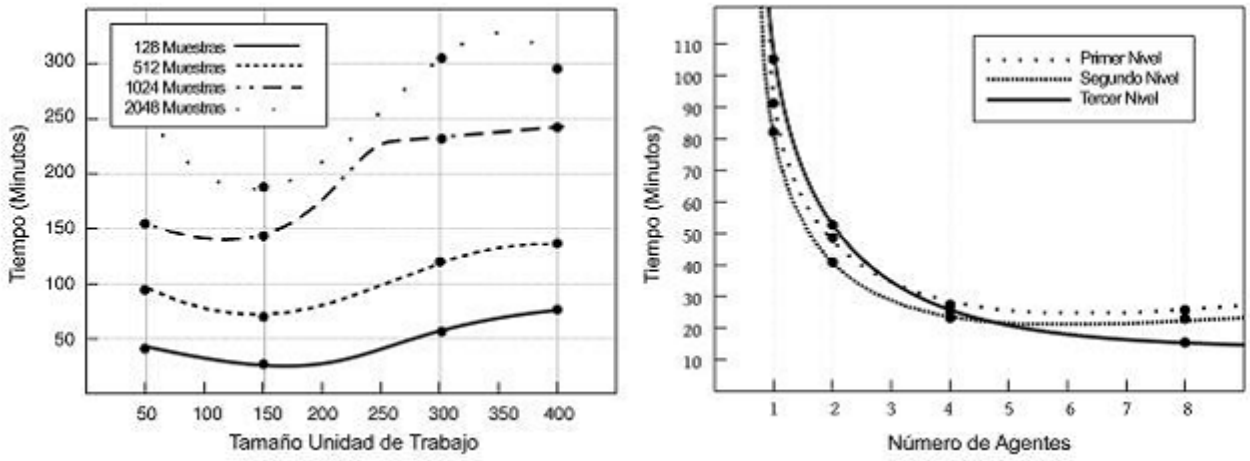

Figura 6. *Izquierda*: Yafrid. Tiempo de *renderizado* relacionado con el tamaño de una unidad de trabajo. *Derecha*: MagArRO. Diferentes niveles de particionamiento con un nivel normal de optimización.

Como observación final, apuntar que el proceso de optimización inteligente puede dar lugar a diferentes niveles de calidad para diferentes áreas en la escena total. Esto es debido a que niveles más agresivos de optimización (Grande o Muy Grande) pueden provocar pérdida del detalle.

Por ejemplo, en la Figura 7.e, los reflejos en el vaso no están tan definidos como en la figura 7.a. La diferencia entre el *renderizado* óptimo y el nivel más agresivo de optimización (Figura 7.f) es mínimo.

6. Discusión y conclusiones finales

Los requisitos computacionales que son necesarios para el *renderizado* de calidad fotorrealista son enormes y, por tanto, obtener resultados en un tiempo razonable y en un sólo ordenador es prácticamente imposible (incluso existen más dificultades en el caso de las animaciones). Muchas propuestas, basadas en diferentes tecnologías, han sido expuestas en este artículo.

Nuestro *cluster* basado en el sistema **Oscar** presenta algunas características interesantes:

- Muy buen rendimiento en el *renderizado* de animaciones. El sistema divide cada fotograma de la animación en diferentes nodos del *cluster*. El enfoque de granularidad fina requiere la programación de nuevas características en el servidor principal.
- El procesamiento de nodos es usado durante tiempos ociosos (por ejemplo, durante la noche).
- La latencia debida a la transferencia de archivos es mínima (gracias al uso de una red Ethernet de gran velocidad).

Por lo demás, el *cluster* sólo puede utilizarse añadiendo tareas al servidor principal en la misma organización. Para resolver algunos de estos problemas, se diseñó **Yafrid**. Este sistema *grid* tiene algunas ventajas importantes:

- No es un *cluster*, los proveedores pueden ser heterogéneos (software y hardware) y pueden estas distribuidos geográficamente.
- Con la aproximación de granularidad

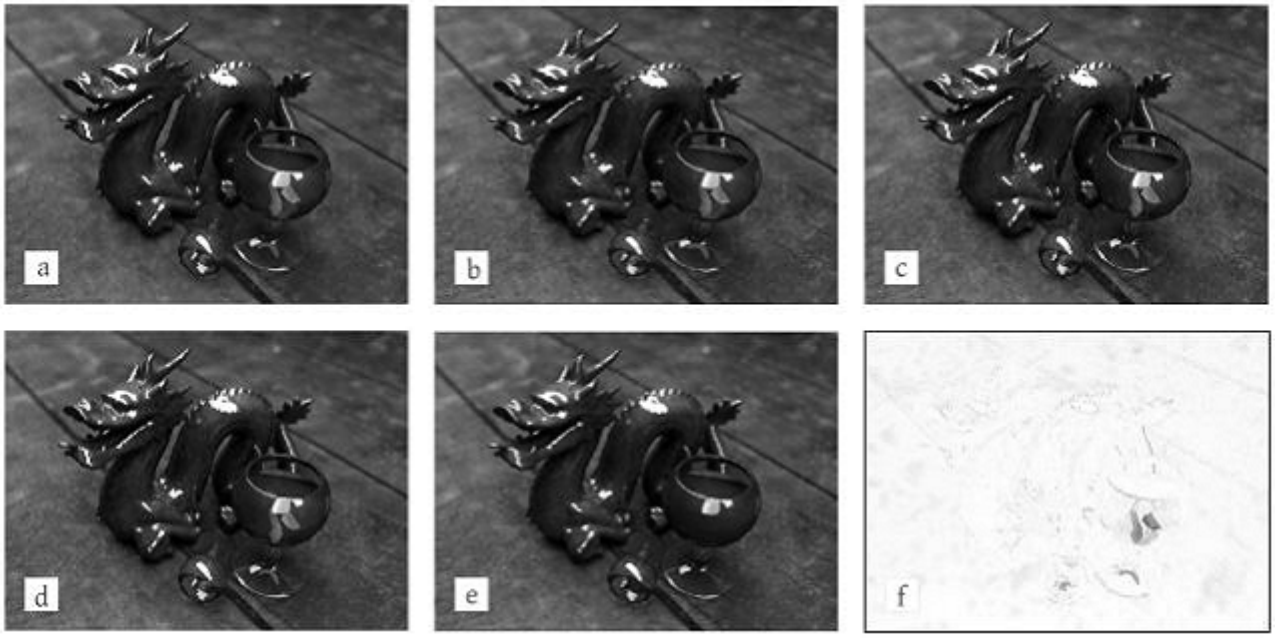

Figura 7. Resultado del *renderizado* utilizando diferentes niveles de optimización. (a) Sin optimización y *renderizado* en una máquina. (b) Muy Pequeño (c) Pequeño (d) Normal (e) Muy Grande (f) Diferencias entre (a) y (e) (la claridad del color, la diferencia más pequeña).

fina, se pueden realizar optimizaciones locales en cada fotograma.

- Una de las principales ventajas de este sistema distribuido es la escalabilidad. La ejecución percibida por el usuario depende del número de proveedores subscritos.

Algunas mejoras podrían realizarse perfeccionando la ejecución de Yafrid. Algunas de ellas se han añadido a **MAgArRO**:

- MAgArRO permite dirigir la importancia del *renderizado* a través del uso de mapas de importancia.
- Permite la aplicación de conocimiento experto mediante el empleo flexible de reglas difusas.
- Aplica los principios de control descentralizado y optimización local. Los servicios son fácilmente replicables, de manera que los cuellos de botella pueden ser minimizados en el despliegue final.

Existen muchas líneas de investigación futura. En nuestro trabajo actual, nos concentramos en la combinación de las mejores características de Yafrid y MAgArRO para integrar el nuevo sistema (llamado YafridNG) en el repositorio oficial de Blender [15]. El código fuente de estos sistemas, distribuidos bajo licencia GPL, pueden ser descargados desde [24].

Agradecimientos

Este trabajo ha sido financiado por la *Consejería de Ciencia y Tecnología* y la *Junta de Comunidades de Castilla la Mancha* bajo los proyectos de investigación PAC06-0141 y PBC06-0064. Agradecimientos especiales a Javier Ayllon por su ayuda al Servicio Supercomputacional (Universidad de Castilla-La Mancha).

Referencias

[1] D.P. Anderson, G. Fedak. The Computational and Storage Potential of Volunteer Computing. *Sixth IEEE International Symposium on Cluster Computer and the Grid (CCGRID '06)*. 7380. Mayo 2006.

[2] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, P. Hanrahan. Brook for GPUs: Stream Computing on Graphics Hardware. *Proceedings of SIGGRAPH '04*, 777786.

[3] A. Chalmers, T. Davis, E. Reinhard. *Practical Parallel Rendering*. Ed. A. K. Peters, 2002. ISBN: 1-56881-179-9.

[4] I. Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputing Applications 15, 3*(2002).

[5] T. Hachisuka. High-Quality Global Illumination Rendering using Rasterization. *GPU Gems 2: Programming Techniques for High Performance Graphics and General-Purpose Computation*. Addison-Wesley Professional, 2005.

[6] J.T. Ka jiya. The rendering equation. *Computer Graphics 20(4): 143150*. Proceedings of SIGGRAPH '86.

[7] R.R. Kuoppa, C.A. Cruz, D. Mould. Distributed 3D Rendering System in a Multi-Agent Platform. *Proceedings of the Fourth Mexican International Conference on Computer Science, 8, 2003*.

[8] R. Ra jagopalan, D. Goswami, S.P. Mudur. Functionality Distribution for Parallel Rendering. *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, 1828, April 2005.

[9] E. Reinhard, A.J. Kok, F.W. Jansen. Cost Prediction in Ray Tracing. *Rendering Techniques '96, 4150*. Springer-Verlag, June 1996. ISBN 3-211-82883-4.

[10] E. Veach, L.J. Guibas. Metropolis light transport. *Proceedings of SIGGRAPH'97, 6576*. New York, USA: ACM Press - Addison Wesley Publishing Co.

[11] T. Whitted. An improved illumination model for shaded display. *Proceedings of SIGGRAPH '79, 14*. New York, USA: ACM Press.

[12] M.J. Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2002. ISBN: 0-471-49691-X

[13] L.A. Zadeh. The concept of a linguistic variable and its applications to approximate reasoning. *Information Science*, 1975.

[14] Beowulf. Open Scalable Performance Clusters. <http://www.beowulf.org>.

[15] Blender. Free 3D content creation suite. <http://www.blender.org>.

[16] BURP. Big Ugly Rendering Pro ject. <http://burp.boinc.dk/>.

[17] Dr. Queue. OS Software for Distributed Rendering. <http://www.drqueue.org/>.

[18] OSCAR. Open Cluster Group. <http://www.openclustergroup.org/>.

[19] Thin-Oscar. <http://thin-osccar.sourceforge.net/>.

[20] Virtual Tour ESI UCLM. <http://www.inf-cr.uclm.es/virtual/index_en.html>.

[21] FIPA. Foundation for Intelligent Physical Agents. <http://www.pa.org>.

[22] Virtual Visit - Hospital Ciudad Real. <http://dev.oreto.inf-cr.uclm.es/www/vvhosp>.

[23] Yafray. Yet Another Free Raytracer <http://www.yafray.org>.

[24] Yafrid Next Generation. <http://www.yafridng.org>.

[25]. ZeroC ICE Middleware. <http://www.zeroc.com>.

Notas

[1] "La renderización es el proceso de generar una imagen desde un modelo. Los medios por los que se puede hacer un renderizado van desde lápiz, pluma, plumones o pastel, hasta medios digitales en dos y tres dimensiones. La palabra renderización proviene del inglés render, y no existe un verbo con el mismo significado en español, por lo que es frecuente usar las expresiones renderizar o renderear" <http://es.wikipedia.org/wiki/Renderizar>.

[2] "Grid computing es una tecnología innovadora que permite utilizar de forma coordinada todo tipo de recursos (entre ellos cómputo, almacenamiento y aplicaciones específicas) que no están sujetos a un control centralizado" <http://es.wikipedia.org/wiki/Grid_computing>.