Revista Electrónica Conocimiento Libre y Licenciamiento (CLIC) Mérida – Venezuela
ISBN: Obra Independiente 978-+80-7154-09-3 - ISSN: 2244-7423 - Depósito Legal: PPI201002ME3476
Índice de Revistas Venezolanas de Ciencia y Tecnología: RVR065.

POTENCIANDO EL DESARROLLO COMUNITARIO DE SOFTWARE LIBRE CON ARQUITECTURAS BASADAS EN COMPONENTES

Juan Ernesto Vizcarrondo Rojas
CENDITEL[1]
E-mail: jvizcarrondo@cenditel.gob.ve

Recibido: 13/09/2010
Aceptado: 11/10/2010
Vol. 1 Nº 3 AÑO: 2 Páginas: 47 - 68

Palabras clave
Software libre, Decreto 3390, Plataforma para el Desarrollo de Software Libre, Desarrollo de Software, Reutilización del software.

Resumen

En el país se están desarrollando soluciones de software a problemas específicos, que no deberían estar desconectados entre sí, debido a que comparten funcionalidades comunes que podrían ser usadas entre ellos e incluso por otros sistemas. El desarrollo de software libre en nuestro país debe

1 FUNDACIÓN CENTRO NACIONAL DE DESARROLLO E INVESTIGACIÓN EN TECNOLOGÍAS LIBRES. Mérida, Venezuela.

Revista Electrónica Conocimiento Libre y Licenciamiento (CLIC) Mérida – Venezuela
ISBN: Obra Independiente 978-+80-7154-09-3 - ISSN: 2244-7423 - Depósito Legal: PPI201002ME3476
Índice de Revistas Venezolanas de Ciencia y Tecnología: RVR065.

ser compartido, comunitario. Los desarrolladores deberían aprovechar las producciones hechas por otros para, a partir de ellas, reutilizar, modificar, agregar, corregir o revisar. Debería poder pensarse en un trabajo conjunto en torno a un mismo objeto, de manera que las fuerzas se concentren en una dirección y no que se dispersen en una suerte de competencia individual.

Introducción

El software libre se caracteriza por su valor acumulativo (Gracia y de la Cueva, 2002). Las personas pueden tener acceso y distribuir libremente el código fuente de miles de aplicaciones y programas proporcionando la posibilidad de obtener, aprender y reutilizar miles de lineas de código escritas por otras personas. En el mundo del software libre es factible y habitual encontrar código escrito que cubra una necesidad para un programa especifico que se quiera desarrollar. La reutilización de código como práctica habitual entre la Comunidad de desarrolladores de Software Libre es una de las razones que explica el increíble avance producido en los más de 20 años de su existencia[2].

Luego de la publicación del decreto 3390 (Gaceta oficial Nº 38.095, 2004), la adopción de sistemas basados en software libre en nuestro país se han visto como un mero cambio de productos de software propietario a otros con estándares libres, en el mejor de los casos se ha comenzado el desarrollo de herramientas de software libre aisladas, empotradas en estructuras individuales complejas, de difícil adaptación, que dificultan la cooperación entre ellas, que siguen la idea de desarrollar productos de corta vida y el aumento de especialistas en cada uno de los desarrollos.

Las aplicaciones desarrolladas bajo este enfoque resultan ser buenas soluciones a los problemas para los que fueron diseñados, pero la principal desventaja con las que se encuentran es la imposibilidad de adaptar las soluciones a nuevos requisitos (inflexible) que no encajen bien en el patrón en el que fueron diseñadas y la imposibilidad de cooperar con otros desarrollos de software para generar nuevas soluciones.

Así, en el país se están desarrollando soluciones de software a problemas específicos, que no deberían estar desconectados entre sí, debido a que comparten funcionalidades comunes que podrían ser usadas entre ellos e incluso por otros sistemas. El desarrollo de software libre en nuestro país debe ser compartido, comunitario. Los desarrolladores deberían aprovechar las producciones hechas por otros para, a partir de ellas, reutilizar, modificar, agregar, corregir o revisar. Debería poder pensarse en un trabajo conjunto en torno a un mismo objeto, de manera que las fuerzas se concentren en una dirección y no que se dispersen en una suerte de competencia individual.

2 El sistemas desarrollados bajo estadares libres han crecido tanto ha nivel de calidad, cantidad y usuarios desde la creación del proyecto GNU en 1983 por Richard Stallman. Para más información consultar: http://www.gnu.org/gnu/gnu-history.es.html.

Revista Electrónica Conocimiento Libre y Licenciamiento (CLIC) Mérida – Venezuela
ISBN: Obra Independiente 978-+80-7154-09-3 - ISSN: 2244-7423 - Depósito Legal: PPI201002ME3476
Índice de Revistas Venezolanas de Ciencia y Tecnología: RVR065.

Justificación

Últimamente se ha hablado del desarrollo de una Plataforma para el Desarrollo de Software Libre (PDSL) (Plataforma para el Desarrollo de Software Libre, 2008), donde el sueño es que sea el resultado del consenso[3] y participación de todos los actores del país, concretamente se ha pensado en ampliar el sistema Gforge (GForge Collaborative Development Environment, 2009). Al hacer una revisión de este sistema se percibe como una estructura compleja, estática, propia de su desarrollo, difícilmente adaptable que entorpece la cooperación de los actores interesados en su evolución y la adición de nuevas funcionalidades. Adicionalmente, el Gforge fue concebido como una herramienta para gestionar el deposito y coordinación de proyectos de software, pero no provee herramientas que faciliten el quehacer de programar. La implementación de un PDSL bajo este enfoque, resulta verdaderamente una quimera difícil de realizar.

Es así, que surge la necesidad de una herramienta que permita una vinculación entre los distintos desarrollos de software en el país, agrupándolos en un marco que permita obtener beneficios al explotar espacios comunes, generando nuevas soluciones a partir de otras previamente alcanzadas, facilitando de esta forma el desarrollo colaborativo entre los distintos desarrollos de software en el país.

Objetivo

Atendiendo a estos problemas, el objetivo es crear sistemas que faciliten la evolución de los sistemas computacionales, permitiendo fácilmente a dichos sistemas modificarse para adaptarlos a nuevas metas funcionales o no funcionales y desafíos de su entorno. Para esto, han surgido sistemas en los que la estructura evoluciona a lo largo del tiempo, los cuales la concepción de su modelo arquitectónico son realizados de tal forma, que pueden ser usado para la evolución del software.

La complejidad y evolución de los sistemas de Software

El incremento en la complejidad y el tamaño de los sistemas de software[4], el buen uso de los recursos disponibles y la tarea de cumplir con los tiempos de entrega de los proyectos de software se

3 El proyecto PDSL propuesto por Cenditel busca el desarrollo de un sistema informático desde las bases de las comunidades de desarrolladores, en donde estas no sean impuestas sino consensuadas. Para más información consultar: http://plataforma.cenditel.gob.ve/

4 Sistemas de software son un conjunto de programas que han sido escritos para servir a otros programas (Lehman, 1997).

Revista Electrónica Conocimiento Libre y Licenciamiento (CLIC) Mérida – Venezuela
ISBN: Obra Independiente 978-+80-7154-09-3 - ISSN: 2244-7423 - Depósito Legal: PPI201002ME3476
Índice de Revistas Venezolanas de Ciencia y Tecnología: RVR065.

han incrementado de manera espectacular[5] en los últimos años (Pressman, 2003)(Pigoski, 1997), lo que ha traído como consecuencia la necesidad de desarrollar y utilizar arquitecturas de software que permitan gestionar la complejidad a distintos niveles de abstracción y faciliten la reutilización de los elementos de software que la componen.

En este orden de ideas, una herramienta de software tarde o temprano tendrá que evolucionar, ya sea para adaptarse a cambios en el ambiente y/o implementar nuevas funcionalidades. Es así que el desarrollo de una herramienta de software de gran tamaño tenga que someterse a un constante cambio o se vera relegada a ser menos usada con el transcurrir del tiempo. El proceso de cambio continuara hasta que los desarrolladores y usuarios de la herramienta consideren más provechoso reemplazar el sistema por una nueva versión.

Según Lehman, *"Los grandes programas no llegan nunca a completarse y están en constante evolución"*.

Las actividades de evolución y mantenimiento están principalmente ligadas con el interés de operación del software después de que éste es puesto en funcionamiento. El objetivo de estas actividades es asegurar que el software puesto en funcionamiento continúa operando de forma satisfactoria, adaptándose a las nuevas situaciones y necesidades. Así, todo software evoluciona, ya sea para ampliar, mejorar o adaptar. Las actividades de evolución y mantenimiento resultan ser los de más trabajo durante el desarrollo del software (Pigoski, 1997). Manny Lehman y sus colegas analizaron la industria del software llegando a postular un conjunto de leyes que rigen a los sistemas computacionales (Lehman, 1997):

- Deben adaptarse continuamente (ley del cambio continuo),
- Sus funcionalidades se incrementan continuamente para satisfacer a los usuarios (ley del crecimiento continuo),
- Cuando evolucionan aumentan su complejidad (ley de la complejidad creciente)
- Disminuyen su calidad (ley de calidad decreciente) si no se hace nada para evitarlo.

Así, todo software evoluciona, ya sea para ampliar, mejorar o adaptar. Las actividades de evolución y mantenimiento son los más costosos durante el desarrollo del software, llegando a representar cerca del 60% a 80% de esta actividad (Nazim, 2006).

5 Los cambios en hardware y arquitecturas informaticas han llevado a sistemas más complejos y sofisticados (Lehman, 1997).

Revista Electrónica Conocimiento Libre y Licenciamiento (CLIC) Mérida – Venezuela
ISBN: Obra Independiente 978-+80-7154-09-3 - ISSN: 2244-7423 - Depósito Legal: PPI201002ME3476
Índice de Revistas Venezolanas de Ciencia y Tecnología: RVR065.

Estudio de Diferentes Arquitecturas Comúnmente Usadas en el Desarrollo de Herramientas de Software Libre (Casos de estudio)

En el proyecto de PDSL se analizaron algunas arquitecturas de componentes utilizadas por aplicaciones de software libre desarrolladas de manera comunitaria, con el objeto de encontrar debilidades y fortalezas de estás con las necesidades de desarrollo comunitario de software:

- **Said**

Es un sistema administrativo integral que permite la automatización de los procesos inherentes a la administración pública venezolana de entes descentralizados sin fines empresariales. Es uno de los pocos proyectos con cierto éxito a nivel nacional[6] en conseguir algunos colaboradores. Adicionalmente, se encuentra desarrollado principalmente en PHP y una base de datos[7] postgresql y es gestionado por Cenditel Nodo Mérida (Said, 2009). Al momento de este estudio se encuentra en la versión 0.2.

Para incorporar nuevas funcionalidades al Said es necesario copiar el contenido del componente en la raíz de la aplicación y realizar un conjunto de modificaciones en los archivos items_menu.php para que las nuevas funcionalidades sean visibles para el usuario. Adicionalmente, si el nuevo componente necesita almacenar opciones de configuración este debe ser almacenado en un nuevo archivo del componente en la carpeta configuración o modificando el contenido de conf_apps_varglobal.php (aunque no se tiene establecido una manera de como hacerlo). Al nuevo componente no le es posible intervenir en las funcionalidades existentes y no provee un estándar en la manera en que nuevos componentes podrían realizarlo. Así, si es necesario actualizar funcionalidades existentes es necesario modificar los ficheros del Said. No cuenta con mecanismos que le permitan gestionar la instalación, actualización y des-incorporación de componentes por lo que esta es realizada manualmente al incorporar carpetas y archivos en la base de la aplicación y ejecutando scripts en la base de datos.

- **PhpMyAdmin**

phpMyAdmin (The phpMyAdmin Project Effective MySQL Management, 2009) es una herramienta escrita en PHP que se encuentra disponible bajo la licencia GPL[8]. La herramienta permite

6 El proyecto Said cuenta con gran apoyo en la administración pública venezolana, por lo que es muy usado y desarrollado por los entes descentralizados del gobierno venezolano. Para más información consultar:
7 Recopilación de datos con un mismo contexto los cuales son almacenados para su posterior uso.
8 La Licencia Pública General de GNU (GNU GPL) posibilita la modificación y redistribución del software.

Revista Electrónica Conocimiento Libre y Licenciamiento (CLIC) Mérida – Venezuela
ISBN: Obra Independiente 978-+80-7154-09-3 - ISSN: 2244-7423 - Depósito Legal: PPI201002ME3476
Índice de Revistas Venezolanas de Ciencia y Tecnología: RVR065.

manejar la administración de MySQL a través de una interfaz web. Entre las tareas que realiza están: crear y eliminar Bases de Datos, crear, eliminar y alterar tablas, borrar, editar y añadir campos, ejecutar cualquier sentencia SQL, administrar claves en campos, administrar privilegios, exportar datos en varios formatos y está disponible en 55 idiomas.

Al igual que Said (Said, 2009), para añadir nuevas funcionalidades al sistema, es necesario la creación de ficheros en la raíz del aplicación que deben incluir los ficheros del núcleo (phpMyAdmin 3.0.0-dev Documentation, 2008): header.inc.php,common.lib.php y config.inc.php que se encuentran el directorio libraries. Para añadir nuevas funciones es necesario editar el fichero common.lib.php y colocarlas al final del contenido del fichero. Adicionalmente, para modificar el comportamiento de funcionalidades existentes, es necesario modificar el código del núcleo de la aplicación.

- **Deluge**

Deluge (Deluge, 2009) es un cliente bittorrent escrito en python + gtk, proporciona uno de los más modelos de arquitectura más simples de componentes (Pitivi, 2009):

Los componentes se almacenan en los subdirectorios de la carpeta ./ Plugin . Cada subdirectorio toma el nombre del Plugin. Cada plugin debe al menos ofrecer un archivo __init__.py que definen los espacios de nombres y proporciona los puntos de acceso para el componente.

Dentro del archivo __init__.py son almacenados:

1. Campos de descripción (nombre del componente, autor, versión y descripción).
2. La función deluge_init() que es llamada para inicializar el componente.
3. La función enable() que es llamada cuando el usuario deshabilita el componente.

El gestor de componentes revisa el directorio plugin para buscar nuevos componentes y cuando encuentra alguno, este importa el espacio de nombres dentro de la lista de componentes y llama a la función deluge_init() para inicializar el componente. Esta arquitectura tiene la gran ventaja de ser bastante ligera, pero no crea una clase jerárquica de componentes, lo que podría ser a la vez una ventaja y una desventaja ya que los cambios en en el programa base podría romper la compatibilidad con el componente, pero con un padre en común puede simplificar el funcionamiento habitual mediante la definición de ellos en el interior del padre.

Esta arquitectura de componentes es útil para extender la aplicación con funcionalidades que no se hayan considerado en el núcleo, ya que el componente tiene un control total sobre la aplicación y por lo tanto, puede ampliar cualquier aspecto de ella.

Revista Electrónica Conocimiento Libre y Licenciamiento (CLIC) Mérida – Venezuela
ISBN: Obra Independiente 978-+80-7154-09-3 - ISSN: 2244-7423 - Depósito Legal: PPI201002ME3476
Índice de Revistas Venezolanas de Ciencia y Tecnología: RVR065.

- **Jokosher**

Jokosher (Jokosher ... audio production made easy, 2009) es un software para la producción de audio que usa python + gtk + gstreamer. Su arquitectura de componentes busca plugins dentro de la carpeta ./extensions, los cuales pueden contener archivos con extensiones py (python) o egg (huevos python) (The PEAK Developers Center, 2009). La interacción de la extensión es provista para insertar opciones del menú del Jokosher, así como funciones para controlar la reproducción, instrumentos y para añadir nuevos formatos de salida.

Las preferencias para el uso del componente son almacenadas en la carpeta ./extensions-config path con un archivo de configuración para cada componente llamado con el mismo nombre. El gestor del componente instala y quita plugins tomando cuidado de posibles conflictos, este también gestiona el proceso de carga y descarga de configuración para cada plugin, por lo que cada plugin se carga una sola vez (Patrón singletons).

Cada componente debería contener:

1. Algunos campos descriptivos que permitan conocer el nombre, descripción y versión del componente.
2. La función startup(api) se llama cuando el plugin se activa, pasando todo el conjunto de API para el plugin.
3. shutdown() es llamado cuando el complemento está desactivado, se encarga de la limpieza.

Este enfoque define un conjunto claro de interfaces que el componente puede utilizar para interactuar con la aplicación principal, ingresando opciones del menú, añadiendo nuevos formatos de salida, funcionalidades y generando las pautas a la aplicación principal para su gestión. Los componentes pueden interactuar con Jokosher de dos maneras distintas: Pueden registrarse para una llamada del núcleo (hook como Project, Instrument, Event, startup), obteniendo una notificación cuando algo ocurre o pueden ellos mismos integrarse dentro de Jokosher al agregar items al tope del menú de extensiones (Jokosher.Extension.add_menu_item("Hola", MuestraHola)).

La desventaja de esta arquitectura es que los componentes tiene una acción limitada en la interfaz del usuario porque sólo puede insertar elementos al menú bajo el nombre del "componente" (submenú), la creación de un archivo de preferencias para cada plugin disponible podría dar lugar a una contaminación de los ficheros de configuración, dejando al plugins el deber de eliminar de la interfaz de usuario insertado por él, que podrían dar lugar a pérdida de memoria si el plugin no hace un buen trabajo.

Revista Electrónica Conocimiento Libre y Licenciamiento (CLIC) Mérida – Venezuela
ISBN: Obra Independiente 978-+80-7154-09-3 - ISSN: 2244-7423 - Depósito Legal: PPI201002ME3476
Índice de Revistas Venezolanas de Ciencia y Tecnología: RVR065.

- **Gforge**

Gforge (GForge Collaborative Development Environment, 2009) es una Macro-herramienta web que facilita la colaboración de equipos para el desarrollo de proyectos, escrita en PHP y base de datos Postgresql o Mysql. Posee herramientas como foros de mensajes y listas de correo, gestión de tareas y de repositorios como CVS y Subversion. GForge crea automáticamente un repositorio y controla el acceso a ella, dependiendo de la función de la configuración del proyecto.

En Gforge los componentes son extensiones para el núcleo del Gforge (Roland, 2008), las cuales permiten agregar funcionalidades sin estar herméticamente integrado dentro del código del Gforge. Ellos son muy usados porque permiten el desarrollo independiente de funcionalidades y agregan flexibilidad para adaptar el núcleo hacia características deseables de instalación particular.

Se espera que un componente provea nuevas características a GForge, y no un cambio en el comportamiento de las actuales características de él. Un componente, por lo tanto, sólo debe agregar archivos, no cambiar los ya existentes. Un componente debería estar identificado principalmente por un manejador de cadena, el cual sera estático a través de todas instalaciones del componente. Este debe estar compuesta de solo letras minúsculas, porque van a ser usadas en tablas de nombres y no se quieren conflictos entre ellos.

El Gforge cuenta con tablas para la gestión de los componentes, que deben ser manejadas por estos para su inscripción y des-incorporación dentro del núcleo (tablas plugins, group_plugin y user_plugin). Adicionalmente, un componente podría crear sus propias tablas, secuencias, índices, vistas, entre otros, los cuales utilizara el componente para almacenar la información que necesite para su funcionamiento, pero sin modificar la estructura de la base de datos que impida a el funcionamiento de otros componentes.

Por otro lado, el Gforge provee una serie de puntos de montajes (hooks) que facilitan la cooperación entre los componentes incrustados en el núcleo, tales como: after_session_set, group_approved, session_before_login, entre otros que permiten tener el control de las actividades propias del núcleo como de los componentes base que este suministra. Para que un componente exponga nuevos puntos de montajes, es necesario comunicárselos a los mantenedores del código de Gforge para su posterior incorporación, pasando previamente por una fase de consideración. , Gforge no provee puntos de montaje para la instalación, actualización y des-incorporación de componentes, por lo que esta es realizada ejecutando scripts y moviendo carpetas entre www, lib, include, etc y bin.

Revista Electrónica Conocimiento Libre y Licenciamiento (CLIC) Mérida – Venezuela
ISBN: Obra Independiente 978-+80-7154-09-3 - ISSN: 2244-7423 - Depósito Legal: PPI201002ME3476
Índice de Revistas Venezolanas de Ciencia y Tecnología: RVR065.

- **WordPress**

WordPress (wordpress > Blog Tool and Publishing Platform, 2008) es una herramienta para la creación de bitácoras personales, esta escrito en php y permite su adaptación mediante la inclusión de componentes. Un componente en WordPress es un programa, o un conjunto de una o más funciones escritas en lenguaje php, que añaden un conjunto especifico de características y/o servicios a WordPress, los cuales pueden ser fácilmente integrados a la herramienta utilizando puntos de acceso y métodos proporcionados por él.

Los componentes en WordPress logran sus objetivos mediante la conexión a uno o más componente existentes a través de hooks (ganchos). La forma en que trabajan los hooks es que en varias ocasiones mientras se está ejecutando WordPress, la herramienta revisa para ver si algún componente se ha inscrito para ejecutar funciones en ese momento y, si es así, las funciones son ejecutadas, permitiendo modificar el comportamiento por defecto de WordPress. En Wordpress existen dos tipos de hooks:

Actions: Las acciones (actions) son hooks que el núcleo de WordPress llama en puntos específicos durante su ejecución, o cuando se producen eventos específicos. Un componente puede especificar que uno o más de sus funciones de PHP se ejecuten en estos puntos, utilizando la API de acción. Así, las acciones se activan por eventos específicos que tienen lugar en WordPress, como la publicación de un envío, el cambio de temas, o muestra una página del panel de admin. Un componente puede responder al evento por la ejecución de una función de PHP, por lo que podría realizar una o más de las siguientes operaciones (wordpress Codex, 2008):

1. Modificar datos de bases de datos.
2. Enviar un mensaje de correo electrónico.
3. Modificar lo que se visualiza en la pantalla del navegador (administrador o usuario final).

Filters: Los filtros son hooks que WordPress llama para modificar el texto de diversos tipos antes de añadirlo a la base de datos o enviarlo a la pantalla del navegador. Su complemento puede especificar que uno o más de sus funciones PHP es ejecutado para modificar los tipos específicos de texto en estos momentos, utilizando la API filtros. WordPress hace algunos filtrado por defecto, y su plugin puede añadir sus propios filtros. Los pasos básicos para añadir sus propios filtros para WordPress son:

1. Crear la función de PHP que filtra los datos.
2. Hook para el filtro en WordPress, llamando add_filter
3. Ponga a su función de PHP en un plugin de archivo, y activarla.

Revista Electrónica Conocimiento Libre y Licenciamiento (CLIC) Mérida – Venezuela
ISBN: Obra Independiente 978-+80-7154-09-3 - ISSN: 2244-7423 - Depósito Legal: PPI201002ME3476
Índice de Revistas Venezolanas de Ciencia y Tecnología: RVR065.

- **Drupal**

Drupal (Drupal, Come for the software, stay for the community, 2008) es un programa de código abierto distribuido bajo la licencia GPL y desarrollado en php que permite a un individuo o a una comunidad de usuarios la facilidad de publicar, gestionar y organizar una amplia variedad de contenidos en un sitio web. Drupal es desarrollado y mantenido por una comunidad de miles de usuarios y desarrolladores.

El núcleo de Drupal se complementa con un importante número de extensiones que son las que ofrecen verdadera funcionalidad a este entorno. Podría verse como un enorme juego de Lego en el que los desarrolladores deciden cómo combinar y configurar los diferentes módulos para crear su nuevo sitio web.

Este sistema de extensiones hace uso de lo que se conoce como Inversion Control Design Pattern en el que la funcionalidad modular es llamada por el framework sólo en el momento de necesitarla (VanDyk, 2009). Este patrón de diseño es muy usado para hacer pruebas unitarias en orientación a objetos y Drupal lo integra gracias a los llamados hooks (eventos internos o callbacks). Un hook es una función que permite al modulo interactuar con el núcleo de Drupal, la cual es llamada de la forma foo_bar(), donde "foo" es el nombre del modulo y "bar" es el nombre del hook. Cada hook tiene definido un conjunto de parámetros y especifica un tipo de resultado.

Para extender Drupal, un módulo necesita simplemente implementar uno o varios hook. Cuando Drupal desea permitir la intervención desde **módulos**, este determina cuales módulos implementan un hook y llaman a este hook en todos los módulos activos. Los hooks en drupal le permiten interactuar con el core en todo momento, desde la instalación del plugin (hook_install, hook_settings), interactuando con la vista (hook_form, hook_view, hook_menu, hook_update) hasta su desinstalación (hook_uninstall)). La estructura de un componente en Drupal consta usualmente de 3 archivos indispensables para su funcionamiento, los cuales se encuentran contenidos en una carpeta con el nombre del componente: el NombreModulo.install que contiene los puntos de montaje (modulo_install y modulo_uninstall), NombreModulo.module el contenido del componente y Nombre_modulo.info con información acerca del componente como fecha de creación, dependencias, versión, entre otros.

- **Mozilla**

Mozilla (Home of the Mozilla Project, 2009) fue el nombre original dado por la fundación Mozilla para su Mozilla Application Suite, anteriormente conocido como SeaMonkey suite. Usualmente Mozilla es asociado a interfaces usadas para visualizar contenido html (firefox, SeaMonkey, Scintilla, OpenKomodo, entre otros), pero es mucho más que eso, Mozilla es también un

Revista Electrónica Conocimiento Libre y Licenciamiento (CLIC) Mérida – Venezuela
ISBN: Obra Independiente 978-+80-7154-09-3 - ISSN: 2244-7423 - Depósito Legal: PPI201002ME3476
Índice de Revistas Venezolanas de Ciencia y Tecnología: RVR065.

framework para construir aplicaciones multi-plataforma (Boswell, 2002). El gran éxito de Mozilla se debe a los miles de colaboradores ad-hoc que trabajan tanto para la versión actual como la beta a probar, informar y arreglar errores y añadirle funcionalidades a través de extensiones.

Como Mozilla fue considerado como una suite para el desarrollo de aplicaciones del estilo Front-end [38], cuenta con una infinidad de puntos de montaje que permiten a los componentes interactuar con la aplicación, desde la gestión del componente (Component Management, XUL Planet, 2009), creación de interfaces del usuario con hooks de XUL, hasta crear nuevas estructuras (widget) con XBL y nuevos puntos de montajes con XPCOM (Boswell, 2002). Dichos componentes pueden ser escritos en C++, Javascript, Python e incluso Ruby. Adicionalmente, un componente puede implementar múltiples interfaces de la interfaz principal (núcleo), como de los demás componentes ya implementados en la suite.

- **Trac**

Trac (Welcome to the Trac Project, 2009) es un sistema para el manejo de proyectos que permite la edición de tareas, wikis[9] y manejo de repositorios de versiones (SVN)[10]. El cuerpo de la aplicación contiene algunos puntos de extensión donde se conectan los componentes para incorporarse al sistema. Cada punto de entrada esta caracterizado por un contrato que deben suscribir tanto el componente con la aplicación principal para que puedan interactuar, dicho contrato toma la forma de una interfaz declarada por la aplicación principal y que sera implementada por el componente.

Cada componente puede implementar múltiples interfaces de la aplicación principal y a la vez un punto de entrada puede ser conectado por múltiples componentes. Además, cada componente crea todos los puntos de entradas que necesite para que otros componentes interactúen con él.

Los componentes en Trac se declaran por medio de la clase "Component" y son desplegados como **huevos de python** (python eggs) (The PEAK Developers Center, 2009). La aplicación completa fue diseñada para ser modular y los componentes son creados para completar o reemplazar los ya existentes. El trabajo principalmente es realizado por el core, el cual declara la siguiente estructura (Trac Component Architecture, 2009):

1. La clase Interface define el padre de todas las interfaces implementadas por los componentes al usar la función implements().

9 Programa informático que permite la edición de paginas web por un grupo de personas.
10 Herramienta informática que es capaz de administrar la cronología de cambios del código fuente en desarrollo de programas computacionales.

Revista Electrónica Conocimiento Libre y Licenciamiento (CLIC) Mérida – Venezuela
ISBN: Obra Independiente 978-+80-7154-09-3 - ISSN: 2244-7423 - Depósito Legal: PPI201002ME3476
Índice de Revistas Venezolanas de Ciencia y Tecnología: RVR065.

2. La clase ExtensionPoint (puntos de extensión define cada punto de extensión, caracterizado por una interfaz que cada componente que busca conectarse con el debe conformar.
3. La clase Component define un componente genérico enlazado al gestor de componentes.
4. Finalmente la clase ComponentManager administra todos los componentes disponibles de la aplicación (encendiendo o apagando componentes tomando cuidado que cada componente sea uno solo (singleton).

- **Zope**

Desde la versión 3.0 de Zope (Welcome to Zope.org, 2009) ha introducido plenamente el concepto de interfaz en python, como núcleo del sistema para la interconexión de su arquitectura. En python el concepto de interfaz (Phillip 2008), esta relacionado con el polimorfismo y se presenta como una manera formal para definir relaciones entre elementos, que necesariamente no existan todavía. La manera en que se usa es: si para un objeto foo existe un atributo bar hasattr(foo, 'bar'), entonces es posible obtener foo.bar(), pero este enfoque por si solo resulta aveces insuficiente, pues no es posible determinar si un atributo puede ser invocado (callable) o tiene una restricción para sus valores posibles.

Así, Las interfaces de Zope son diseñadas para ser ampliadas no solo por clases, sino también por módulos, objetos y funciones, donde ellos siguen la regla de oro que las "especificaciones no deberían hacer asunciones acerca de la implementación" [22]; este enfoque deja completa libertad para que el desarrollador del componente pueda organizar su código en la forma que prefiera, tan largos y complejos como la interfaz se lo permita.

Las Interfaces apoyan plenamente la herencia desde otras interfaces (teniendo previamente la certeza que los métodos de los hijos forma parte de uno de los padres). Las interfaces son contratos que definen como los componentes trabajan juntos (Phillip 2008), definiendo que funcionalidad promete un componente proveer, para que sean usadas por el resto de componentes.

Otro punto a favor de Zope es que puede ser usado para crear lo que se conoce como adaptadores [44] en una dirección muy eficiente, proveyendo también un registro centralizado para estos.

Consideraciones de Aspectos y Selección de Arquitectura para el Proyecto PDSL

El desarrollo de Software basado en Componentes (DSBC) (Clements, 1996), proporciona una nueva forma de crear aplicaciones mediante la composición de componentes software independientes. Esta tecnología promueve la utilización de los componentes como elementos básicos en la

Revista Electrónica Conocimiento Libre y Licenciamiento (CLIC) Mérida – Venezuela
ISBN: Obra Independiente 978-+80-7154-09-3 - ISSN: 2244-7423 - Depósito Legal: PPI201002ME3476
Índice de Revistas Venezolanas de Ciencia y Tecnología: RVR065.

construcción de las aplicaciones y su reutilización en diferentes contextos.

Un programa basado en componentes es aquél cuya funcionalidad se logra a partir de de la unión distintos componentes acoplados por un código "enganche" y dentro de un mismo contexto. A diferencia del esquema basado en librerías, las dependencias del programa respecto a los componentes son mucho menores.

La idea es armar sistemas a partir de componentes probados y confiables (Heineman y Councill, 2001). Para ello, es preciso extender la tarea de comprensión: de comprensión de software a comprensión de componentes. Esto implica que el objetivo ya no es la comprensión del código, sino del funcionamiento del componente, sus aplicaciones y sus requerimientos de adaptación (Bertoa y Vallecillo, 2006). Esto quiere decir que además de tener un modelo que describa técnicamente al software, el componente debe entenderse pensando en su reutilización (funcionalidad, requerimientos y restricciones).

Desde una perspectiva lógica, los componentes son una manera de modelar conceptos del mundo real dentro del dominio de un sistema computarizado. Con esto, se vuelven más manejables los problemas complejos, des-agregándose en entidades, procesos, roles o transacciones, por ejemplo. Desde una perspectiva física (de software), los componentes son unidades independientes de software que implementan dichos conceptos lógicos.

En la literatura se pueden conseguir varias definiciones de componentes de software:

Clemens Szyperski (2000) define a un componente de software como una unidad de composición con interfaces especificadas contractualmente y solamente dependencias explícitas de contexto. Un componente de software puede ser desplegado independientemente y está sujeto a composición por terceros.

Igualmente Herzum y Sims (2002) considera a un componente como un artefacto de software auto-contenido y claramente identificable que describe ó ejecuta funciones específicas; que tiene, además, una interfaz claramente establecida, una documentación apropiada y un estado de uso recurrente bien definido.

Según (CDBI Forum, 1999), un componente es una pieza de software que describe y/o libera un conjunto de servicios que son usados sólo a través de interfaces bien definidas.

Así, se puede definir un componente de software como a una pieza de software auto-contenido, reusable, casi independiente y reemplazable, el cual solo es accesible haciendo uso de interfaces bien definidas, que expresan algunas funcionalidades especificas del sistema y que su integración mediante

Revista Electrónica Conocimiento Libre y Licenciamiento (CLIC) Mérida – Venezuela
ISBN: Obra Independiente 978-+80-7154-09-3 - ISSN: 2244-7423 - Depósito Legal: PPI201002ME3476
Índice de Revistas Venezolanas de Ciencia y Tecnología: RVR065.

mecanismos de composición e interacción con otras dan lugar al sistema como un todo.

El uso del paradigma (SBC) nos proporcionará las siguientes ventajas (Jameel, 2009)(Huizing, 2000) (Coffman y Stokes, 2002)(Pérez, 2002)(Viglas, Stamatis y Kouroupetroglou, 1998):

- **Reutilización del software:** Nos lleva a alcanzar un mayor nivel de reutilización de software.
- **Robustez:** Si algún componente falla, el programa principal puede seguir funcionando perdiendo tan sólo la funcionalidad que se obtenía de dicho componente, el cual incluso puede ser reiniciado si se detecta el error.
- **Simplifica las pruebas:** Permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.
- **Simplifica el mantenimiento del sistema:** Cuando existe un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema.
- **Mayor calidad:** Dado que un componente puede ser construido y luego mejorado continuamente, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.
- **División del trabajo:** El desarrollo de los componentes se puede realizar de forma totalmente separada de los programas que hagan uso de ellos facilitando de esta forma el trabajo en paralelo de equipos muy desacoplados.
- **Ciclos de desarrollo más cortos:** La adición de una pieza dada de funcionalidad tomará días en lugar de meses ó años.
- **Ciclos de Vida del Software más Largos:** Al permitir que las aplicaciones se mejoren continuamente permite prolongar su uso a través del tiempo.

Interfaces

Una interfaz de un componente puede ser definida como una especificación de sus puntos de enlace (hooks) (Crnkovi y Larsson, 2002). La arquitectura y otros componentes acceden a los servicios provistos por los distintos elementos que componen la herramienta de software haciendo uso de los puntos de extensión. Un componente puede implementar múltiples puntos de accesos, cada uno de los cuales, representa un servicio diferente ofrecido por el componente, entonces el componente espera tener múltiples interfaces.

Una interfaz no ofrece implementación de cualquiera de las operaciones del componente. En vez de eso, simplemente presenta una colección de nombres de operaciones y provee solo

Revista Electrónica Conocimiento Libre y Licenciamiento (CLIC) Mérida – Venezuela
ISBN: Obra Independiente 978-+80-7154-09-3 - ISSN: 2244-7423 - Depósito Legal: PPI201002ME3476
Índice de Revistas Venezolanas de Ciencia y Tecnología: RVR065.

descripciones y el protocolo de esa operaciones. Dicha separación hace posible que los SBC puedan (Crnkovi y Larsson, 2002) reemplazar parte de la implementación sin cambiar la interfaz, y en esta vía mejorar el desempeño del sistema sin reconstruirlo; y (2) agregar nuevas interfaces sin cambiar las implementación existente, y en esta vía mejorar la adaptabilidad del componente. Así, Un componente puede importar y exportar interfaces para y desde el ambiente que pueden ser implementados por otros componentes.

Contratos entre Componentes

Los contratos son usados para especificar el conjunto de restricciones que normalizarán las interacciones entre un grupo de componentes, las cuales son definidas en términos de (Crnkovi y Larsson, 2002):

- El conjunto de componentes que participan.
- El rol de cada componente a través de sus obligaciones contractuales.
- La invariante a ser mantenida por los componentes.
- La especificación de los métodos que instanciara el contrato.

Un componente no solo provee servicios a otros componentes sino que también requiere los servicios de otros componentes. Un contrato contiene la lista de los servicios que deberán ser provistos por los demás componentes participantes. Así, los componentes trabajan en conjunto en base a un contrato para lograr un objetivo particular, donde cada componente, proporciona algunas de las funcionalidades requeridas y se comunica con los demás miembros del grupo.

Estilos de Programación

Los lenguajes de programación cumplen un doble rol. Por un lado sirven para comunicar a los humanos con las computadoras. Son la forma de transformar una serie de algoritmos, módulos, tipos de datos y sistemas en algo que una computadora sea capaz de comprender y ejecutar. El segundo rol y a mi punto de ver de gran importancia, es que un lenguaje de programación sirve para comunicar humanos con humanos. Es una poderosa herramienta para transmitir conocimiento. Por ejemplo, una

Revista Electrónica Conocimiento Libre y Licenciamiento (CLIC) Mérida – Venezuela
ISBN: Obra Independiente 978-+80-7154-09-3 - ISSN: 2244-7423 - Depósito Legal: PPI201002ME3476
Índice de Revistas Venezolanas de Ciencia y Tecnología: RVR065.

persona puede revisar el código de fuente de una aplicación y aprender el conocimiento necesario para realizar una determinada funcionalidad, o mejor aun mejorar el conocimiento previamente adquirido y transmitirlo a las demás personas con la publicación de estas (gracias a las libertades del software libre).

Dado lo anterior, es clara la motivación de poner gran atención en el estilo de programación usado en el desarrollo de una herramienta computacional, ya que permiten la comprensión, aprendizaje y mejoramiento por parte de las personas (apropiación) y que usualmente no influyen en la comunicación humano-máquina. Usualmente las reglas para implementar los estilos de programación son flexibles, lo cual no significa que van cambiando, sino que dentro de un proyecto se deben mantener siempre las mismas reglas rígidas, aunque estas sean distintas a las que uno usa en otros proyectos. Incluso, cuando se trabaja sobre un proyecto escrito por otro, es mejor adaptarse al estilo en que está escrito en vez de mezclarlos.

Como se mostró anteriormente no es suficiente que un programa funcione para lo objetivos que fue creado, sino debe estar bien escrito a nivel de estilos. El problema del estilo es muy recurrente en el desarrollo de software. Muchas veces se escribe los programas pensando que las únicas personas que lo revisaran o modificarán son lo programadores dentro del proyecto y no se piensa que constituyen una poderosa manera para facilitar la apropiación del conocimiento.

Selección de la Arquitectura para el Proyecto PDSL

Venezuela como el resto de países de latinoamerica se encuentran empotrada en un estilo y aparato científico que sigue las tendencias de los países llamados desarrollados, por lo que en el mejor de los casos, desarrollamos soluciones a problemas exógenos de poco valor para nosotros, fomentando a la vez el uso de herramientas que pocas personas dentro del país entienden, lo cual contribuye a que seamos cada vez más dependientes de los países desarrollados y ayudemos en el desarrollo de su tecnología.

Lo que se investiga en una sociedad es lo que ella considera importante. Dicha importancia, no tiene nada que ver con la verdad de sus posibles respuestas sino de los valores predominantes que se le hayan establecido. En Cenditel estamos interesados en el desarrollo de tecnologías que estén en sintonía con nuestros problemas y fomenten la participación de la sociedad venezolana en las formas de crear y hacer tecnología. Para esto, es necesario la selección de las herramientas que favorezcan el trabajo cooperativo y la participación entre actores y que eviten el desarrollo de pequeños grupos privilegiados de especialistas en torno a ellas, facilitando su apropiación por todos los miembros de la sociedad y permitan su adaptación hacia nuestros objetivos nacionales. En este orden de ideas, en los casos estudiados para la PDSL se presentan tres formas distintas para el desarrollo de aplicaciones a

Revista Electrónica Conocimiento Libre y Licenciamiento (CLIC) Mérida – Venezuela
ISBN: Obra Independiente 978-+80-7154-09-3 - ISSN: 2244-7423 - Depósito Legal: PPI201002ME3476
Índice de Revistas Venezolanas de Ciencia y Tecnología: RVR065.

través de componentes:

En la primera, Said y PhpMyAdmin se muestran como una estructura estática que no provee maneras en que nuevos componentes puedan cooperar con los instalados previamente e incluso presenta algunas dificultades para acoplarse con el núcleo, solo permite añadir funcionalidades a la aplicación, pero no es posible modificar las ya existentes, sin realizar una modificación a los componentes base provistos por el núcleo de la aplicación. Adicionalmente, no cuenta con un gestor de componentes que permita encenderlos o apagarlos y su instalación o desinstalación con el núcleo es realizada de manera manual y riesgosa, al tener que modificar el código del núcleo para que el nuevo componente sea reconocido dentro de la aplicación. Este enfoque tiene la ventaja de no contar con una estructura predefinida para que los componentes interactúen con el núcleo dejándolo realizar las operaciones que consideren, llegando a desarrollar aspectos de la aplicación que ni se habían pensado en su concepción, pero a expensas de una difícil gestión del componente dentro del núcleo y de una casi nula cooperación con los otros componentes.

Como segunda forma tenemos a Deluge y Jokosher, las cuales ofrecen excesivas libertad a su componentes y solo definen un gestor que los enciende y los apaga. Cuando algún componente es encendido, este puede obtener acceso al núcleo de la aplicación y modificarla a su antojo, dejando espacio libre a la anarquía. Así, la arquitectura de Deluge resulta útil para extender la aplicación con funcionalidades que no se hayan considerado en el núcleo, ya que el componente tiene un control total sobre la aplicación y por lo tanto, puede ampliar cualquier aspecto de ella. En cambio Jokosher, define un conjunto claro de interfaces que el componente puede utilizar para interactuar con la aplicación principal, pero siempre dejando libertad para realizar las tareas que considere necesario cuando se sale de estas interfaces. A diferencia de el enfoque anterior, esta opción provee una manera de como gestionar los componentes dentro del núcleo, pero dejando las mismas libertades y el casi completo desacoplamiento con el resto de componentes.

Por ultimo, el tercer enfoque es el que provee una relación más clara de cooperación entre el núcleo y los componentes que forman parte de la aplicación. En este enfoque, se definen una serie de interfaces que definen un contrato entre componentes y el núcleo, los cuales permiten definir las reglas que restringirán las relaciones entre los mismos componentes y con el núcleo de la aplicación. Sin embargo, en este enfoque se nos presentan dos variantes: el primero en el que se encuentra Wordpress y Gforge se presentan una serie de puntos de montajes (hooks) para que los componentes cooperen entre ellos y a la vez con el núcleo, pero no es posible el ingreso de nuevos puntos de montajes sin manipular el núcleo de la aplicación. En la segunda variante tenemos al resto Drupal, Zope, Mozilla y Trac, que además de proveer un conjunto de puntos de montaje predefinidos para las interacciones entre los componentes y el núcleo, permiten definir nuevos contratos para que los componentes puedan definir nuevos puntos de montaje sin necesidad de modificar el núcleo del sistema ni de realizar una petición a los desarrolladores del proyecto para su posterior incorporación.

Revista Electrónica Conocimiento Libre y Licenciamiento (CLIC) Mérida – Venezuela
ISBN: Obra Independiente 978-+80-7154-09-3 - ISSN: 2244-7423 - Depósito Legal: PPI201002ME3476
Índice de Revistas Venezolanas de Ciencia y Tecnología: RVR065.

Así, con la presentación de estos tres enfoques, hemos visto cuatro maneras diferentes de realizar el desarrollo de software, dejando nuevamente en evidencia que la tecnología no es neutra[11] y que en ellas existen elementos que expresan valores e intereses de las personas que las crean:

En el primer enfoque, se trata de una estructura egoísta que dificulta la cooperación entre actores externos y fomenta la creación de expertos y proyectos de corta de vida. Al ser una estructura compleja, propia de su desarrollo y difícil de adaptar, es necesario realizar un estudio exhaustivo del núcleo de la aplicación, para poder entender su funcionamiento y realizar su adaptación, dejando solo a un numero reducido de personas, los que puedan colaborar con esta arquitectura. Adicionalmente, como la arquitectura niega la colaboración, es frecuente encontrar que en el momento en que personas realizan adaptaciones, se generen nuevas aplicaciones que no sean posible incluir en la aplicación original, generando incluso competencia entre aplicaciones hermanas y que persiguen los mismos fines. Un caso patente de esto, es la generación de un nuevo sistema Administrativo basado en Said denominado Said_Carabobo [45], que en esencia corresponde a la misma aplicación base con algunas pequeñas adaptaciones en su núcleo, pero que son imposibles incluir en el sistema original sin desvirtuarlo.

En cambio, en el segundo enfoque se da demasiada libertad a los componentes, llegando al extremo de imposibilitar la cooperación por falta de acuerdos entre ellos. Este enfoque permite alcanzar funcionalidades que nunca se pensaron en la aplicación, proveyendo un alto nivel adaptativo a costa de imposibilitar la interoperatividad de componentes o dejándolo solo a un reducido grupo que pueda entender y ponerse de acuerdo en la manera de hacerlo, imposibilitando en la mayoría de los casos, la reutilización de código, al no contar con reglas claras para el concierto. Esta arquitectura, puede generar en muchos componentes islas que siguen el primer enfoque.

Por otro lado, el tercero presenta un buen enfoque que permite la colaboración estableciendo una serie de acuerdos previos que un componente debe cumplir para poder cooperar con otro y/o con el núcleo, pero en esta se muestran dos vertientes: en la primera proveen una serie de contratos pero no es posible que nuevos componentes suscriban nuevos acuerdos, dejando total control a los creadores de la aplicación sobre el rumbo que esta debe seguir, dificultando la adición de nuevas funcionalidades e impidiéndoles a los usuarios adaptar la aplicación a sus verdaderas necesidades. Por otro lado, la ultima vertiente es la que aprovecha al máximo la cooperación entre los actores: Esta arquitectura permite la continua integración de componentes y su continua evolución en el tiempo, aprovechando los distintos aportes realizados por la comunidad y por ende proveyéndole una larga vida al permitir su continua adaptación.

11 La tecnología no es neutra solamente en razón de sus consecuencias, sino que su misma concepción no es neutra (Mendialdua, Aguilar y Terán, 2008).

Revista Electrónica Conocimiento Libre y Licenciamiento (CLIC) Mérida – Venezuela
ISBN: Obra Independiente 978-+80-7154-09-3 - ISSN: 2244-7423 - Depósito Legal: PPI201002ME3476
Índice de Revistas Venezolanas de Ciencia y Tecnología: RVR065.

Así, los objetivos de Cenditel son los de facilitar la apropiación de las tecnologías, favoreciendo de esta manera su adaptación hacia nuestros verdaderos problemas y fomentando la participación de la sociedad venezolana en las formas de crear y hacer tecnologías. Para esto, se busca que los componentes que se vayan generados para la PDSL, sean continuamente transformados y ensamblados con otros nuevos, a fin de multiplicar las funcionalidades cumplidas por estos dentro de las arquitectura, empleando un modelo simple que permita la apropiación y participación de la sociedad. Así, Cenditel no busca imponer las funcionalidades que tendrá la PDSL, sino que sean continuamente adaptadas por los usuarios de estas, modificando funcionalidades existentes y creando unas nuevas, todo esto en sintonía de resolver sus problemas y facilitando su quehacer diario, estrechando de esta manera, la brecha entre el generador de tecnología y el usuario de estas, favoreciendo de esta manera valores distintos a los de ganancia y eficiencia.

En este orden de ideas, los dos primeros enfoques contienen valores que dificultan la apropiación, y la cooperación entre actores, trayendo como consecuencia la generación de pequeñas elites en el desarrollo de los componentes para la arquitectura. Por otro lado, el tercer enfoque esta más alineado con los objetivos que busca Cenditel, al proveer mecanismos que facilitan la apropiación y cooperación de los actores, promoviendo un conjunto de reglas que estos deben seguir para su integración dentro de la PDSL. Sin embargo, la primera versión de este enfoque busca que los desarrolladores del proyecto puedan contener todo el control del rumbo por el que dirigirá este, lo cual es desfavorable para lo que se busca en la PSDL, al coartar la libertad de los desarrolladores ha concebir las herramientas que en verdad necesitan.

En vista de lo expuesto anteriormente, en Cenditel optamos por definir una arquitectura de componentes basada en el tercer enfoque, que facilite la cooperación entre componentes y que conceda a cada uno de ellos la potestad de definir los puntos de montaje que considere necesario para que otros nuevos puedan cooperar con él, sin la necesidad de reportarlos a un ente centralizado. que interfiera con la creatividad de los actores.

Referencias bibliográficas

Bertoa, Manuel F., & Vallecillo, Antonio (2006). Medidas de Usabilidad de Componentes Software. IEEE Latin America Transactions, vol. 4, Nº. 2, Abril 2006.

Boswell, David (2002). Creating Applications with Mozilla. O'Reilly. ISBN: 978-0-596-00052-3 | ISBN 10: 0-596-00052-9, Septiembre 2002.

ISBN: Obra Independiente 978-+80-7154-09-3 - ISSN: 2244-7423 - Depósito Legal: PPI201002ME3476
Índice de Revistas Venezolanas de Ciencia y Tecnología: RVR065.

CDBI Forum (1999). Component based development: Using componentised software. Disponible en: http://www.cdbiforum.com.

Crnkovi, Ivica, & Larsson Magnus (2002). Building Reliable Component-based Software Systems. Artech House.

Clements, Paul C. (1996). From Subroutines to Subsystems: Component-Based Software Development. Component-Based Software Engineering: Selected Papers from the Software Engineering Institute. Los Alamitos, CA: IEEE Computer Society Press.

Coffman, David, & Stokes, Jess (2002). A Coordinated Component-Based Development Process. Disponible en: http://dirm.state.nc.us/coolgen/pdf/cbd/services%20based%20architecture%.

Component Management, XUL Planet (2009). Disponible en: http://www.xulplanet.com/references/xpcomref/group_ComponentManagement.html

Deluge (2009). Disponible en: http://www.deluge-torrent.org.

Front-end and back-end (2009). Wikipedia la enciclopedía libre. Disponible en: http://en.wikipedia.org/wiki/Front-end_and_back-end.

Drupal, Come for the software, stay for the community. Disponible en: http://drupal.org/

Gaceta oficial Nº 38.095 (2004). Decreto Nº 3390. Disponible en: http://www.gobiernoenlinea.ve/docMgr/sharedfiles/Decreto3390.pdf.

GForge Collaborative Development Environment (2009). Disponible en: http://gforge.org.

Gracia, Pedro, & de la Cueva, Zuzen (2002). Construyendo la Administración electrónica con Software Libre, gnuINE una aplicación libre para la administración local. Disponible en: http://es.tldp.org/Presentaciones/200211hispalinux/gracia/gnuine.html.

Heineman, George T., & Councill, William T (2001). Component-Based Software Engineering: putting the pieces together. Addison-Wesley

Herzum, P., Sims, O. Business component factory : A comprehensive overview of component-based development for the enterprise (2000). John Wiley & Sons.

Huizing M (1999). Component Based Development. Xootic Magazine.

Revista Electrónica Conocimiento Libre y Licenciamiento (CLIC) Mérida – Venezuela
ISBN: Obra Independiente 978-+80-7154-09-3 - ISSN: 2244-7423 - Depósito Legal: PPI201002ME3476
Índice de Revistas Venezolanas de Ciencia y Tecnología: RVR065.

Home of the Mozilla Project (2009). Disponible en: http://www.mozilla.org/.

Jameel Qureshi, M. Rizwan (2009). Reuse And Component Based Development (CBD). Disponible en: http://ww1.ucmss.com/books/LFS/CSREA2006/SER3070.pdf.

Jokosher ... audio production made easy (2009). Disponible en: http://www.jokosher.org/

Lehman, Meir M. Lehman, & Ramil, Juan F., & Wernick, Paul, & Wladyslaw MDewayne E. Perry (1997). Metrics and Laws of Software Evolution - The Nineties View. IEEE METRICS 1997.

Mendialdua, & Aguilar, José, Terán, Oswaldo. Reflexiones desde Cenditel: Sentido de Cenditel. Cenditel Volumen 1, ISBN: 978-980-7154-00-0, 2008.

Nazim H. Madhavji, & Fernandez-Ramil, Juan, & Perry, Dewayne (2006). Software Evolution and Feedback: Theory and Practice. Wiley Junio del 2006.

Pérez, Melvin (2002). Component-Based Development in Geographically Dispersed Teams. Disponible en: http://chapters.computer.org/dominicana/contribuciones/Comp_Based_Dev.pd

Phillip J. Eby (2008). Web Component Development with Zope 3 (tercera edición). Springer.

PhpMyAdmin 3.0.0-dev Documentation (2009). Disponible en: http://www.phpmyadmin.net/documentation/

Pigoski, John (1997). Practical Software Maintenance - Best Practices for Managing Your Software Investment, Nueva York NY, 1997. Wiley & Sons.

Pitivi (2009). Design Docs Plugins. Disponible en: http://www.pitivi.org/wiki/Design_Docs_Plugins

Plataforma para el Desarrollo de Software Libre (2009). Disponible en: http://www.cenditel.gob.ve/wikicenditel/doku.php?id=pdsl.

Pressman, R. S (1993). Ingeniería del software. Un enfoque práctico. (3ª Edición), McGrawHill.

Roland, Mas (2008). Debian Gforge Plugins HowTo. Disponible en: http://synchroforge.com/plugins/scmsvn/viewcvs.php/trunk/docs/README.Plugins?root=synchroforge&sortby=rev&view=markup

Revista Electrónica Conocimiento Libre y Licenciamiento (CLIC) Mérida – Venezuela
ISBN: Obra Independiente 978-+80-7154-09-3 - ISSN: 2244-7423 - Depósito Legal: PPI201002ME3476
Índice de Revistas Venezolanas de Ciencia y Tecnología: RVR065.

Said (2009). Sistema Administrativo Integrado Descentralizado SAID. Disponible en: http://www.cenditel.gob.ve/wikicenditel/doku.php?id=said.

Szyperski, C. (2002). Component software: Beyond object-oriented programming (2da edición). Addison-Wesley Pub Co.

The phpMyAdmin Project Effective MySQL Management (2009). Disponible en: http://www.phpmyadmin.net/home_page/index.php

The PEAK Developers Center (2009). Disponible en: http://peak.telecommunity.com/DevCenter/PythonEggs.

Tech spec for Jokosher extensions and extension API (2009). Disponible en: http://jokosher.python-hosting.com/wiki/TemporaryExtensionAPIDocumentation.

Trac Component Architecture (2009). Trac Project. Disponible en: http://trac.edgewall.org/wiki/TracDev/ComponentArchitecture.

VanDyk, John, & Westgate Matt (2007). Pro Drupal Development. Appress.

Viglas, C., & Stamatis, C., & Kouroupetroglou, G. (1998). Remote Assistive Interpersonal Communication Exploiting Component Based Development. Proceeding of the XV IFIP World Computer Congress, Vienna, 1198.

Welcome to the Trac Project (2009). Trac Project. Disponible en: http://trac.edgewall.org/.

Welcome to Zope.org (2009). Disponible en: http://www.zope.org/

wordpress > Blog Tool and Publishing Platform (2008). Disponible en: http://www.wordpress.org.

Wordpress Codex (2008). Disponible en: http://codex.wordpress.org/Plugins.