FACULTAD DE INFORMÁTICA
UNIVERSIDAD POLITÉCNICA DE MADRID

UNIVERSIDAD POLITÉCNICA DE MADRID
FACULTAD DE INFORMÁTICA

TRABAJO FIN DE CARRERA

GENERACIÓN, GESTIÓN Y DISTRIBUCIÓN DE ARTEFACTOS JAVA CON TÉCNICAS DE INTEGRACIÓN CONTINUA Y SOFTWARE LIBRE

AUTOR: Carlos González Sánchez
TUTOR: Francisco Gisbert Canto

Agradecimientos

A la Comunidad de Software Libre, porque sin ella este trabajo difícilmente podría haberse llevado a cabo.

A Javier Bezos por compartir generosamente su amplio conocimiento acerca de LaTeX, y a Ignacio Estirado por su inestimable ayuda en el trabajo de campo.

A Susana, a Martín y a mis padres, por todo.

Facultad de Informática - UPM. Julio 2008.

Carlos González Sánchez

Índice general

Resumen

El desarrollo de proyectos en grupo, independientemente del lenguaje de programación que se considere, tradicionalmente se ha sustentado en herramientas comerciales de elevado coste, tanto en lo referente a licencias, como en lo tocante a su complejidad de uso.

La ingeniería del software, gracias a los modelos de desarrollo del mundo del software libre, a las metodologías denominadas ágiles y a las buenas prácticas puestas a disposición en la red de redes, ha evolucionado a todos los niveles: documentación de análisis más útil y práctica, planificación más ajustada a la realidad, codificación de mejor calidad, pruebas más eficientes, implantación más ordenada, etc.

No obstante, la abundancia de alternativas puede suponer en muchos casos, un problema añadido, como se verá más adelante en algunos capítulos del proyecto. El punto de partida del estudio es el propio estudio que Martin Fowler hace de las prácticas de Integración Continua [Fow06].

Este proyecto descubre, evalúa y propone, de entre las existentes, las mejores opciones en el marco del Software Libre para resolver una parte crítica de las actividades ligadas al desarrollo en grupo como es la gestión de código fuente y los repositorios de versiones de los productos desarrollados que a lo largo del documento serán nombrados como "artefactos", por adoptar la traducción del inglés *artifact*.

También sirve de guía o tutorial para la configuración de un entorno, totalmente funcional e independiente de plataforma que permita, sin demasiado esfuerzo, disponer de un juego de herramientas destinadas a mejorar la productividad y facilitar el orden en modelos de desarrollo distribuidos.

Capítulo 1

INTRODUCCIÓN

1.1. Motivación

Existen muchas herramientas comerciales y libres, de control de versiones, de gestión de dependencias entre módulos, de aprovisionamiento de artefactos, de despliegue en entornos distribuidos, de gestión de errores o incidencias de desarrollo, de distintas problemáticas de la ingeniería del software en definitiva.

Para cada una de estas categorías es posible encontrar herramientas con distintas capacidades dependiendo del lenguaje de programación en el que estén desarrolladas, y en algunos casos, del lenguaje para el que fueron diseñadas.

Otra posible clasificación se puede realizar atendiendo a las funcionalidades que ofrece, o al enfoque práctico que presentan. Hay piezas de software que nacen simples y sus autores quieren que evolucione siguiendo esa máxima, y hay otras que arrancan con pretensiones más de tipo empresarial (*enterprise*), con funcionalidades multiplataforma, capacidad de integración, etc.

En muchos casos la sencillez no está reñida con la funcionalidad. De hecho estamos asistiendo a escenarios en los que precisamente las propuestas de diseño más sencillo son las que están teniendo un mayor recorrido y aceptación.

El presente estudio parte de la necesidad de encontrar una solución óptima, basada en Software Libre para la gestión de dependencias entre artefactos Java generados por el propio equipo de desarrollo, con respecto de los artefactos de terceras partes disponibles en Internet.

Prácticamente cada proyecto de desarrollo de software depende de código fuente que proviene de otros proyectos. Un equipo de desarrollo en un proyecto determinado no mantiene el código fuente de otros proyectos, sino que se apoya sobre sus APIs para conectar ambos códigos. A medida que la lista de proyectos externos, o lo que es lo mismo, la lista artefactos externos, crece en un proyecto propio, también crece la dependencia de los artefactos entre si, y consecuentemente el proceso de construcción de software adquiere una complejidad notable. Se hace preciso entonces contar con herramientas que faciliten la gestión de estas dependencias y de su evolución, de modo que el proyecto crezca de manera controlada.

Del proceso de construcción de software, la gestión de dependencias es por tanto una pieza clave. Sin embargo, en el caso del lenguaje Java, no se dispone de una solución estandarizada en comparación con otros lenguajes emergentes como Ruby o Python.

El presente estudio se justifica en el intento de reducir el espectro de posibilidades, aprovechando la carencia en esta parcela, para extender el estudio al resto de bloques pertenecientes al ciclo de construcción de software: gestión de código fuente, gestión de construcción propiamente dicha y distribución de los productos generados.

Se buscará dentro del conjunto de herramientas de Software Libre, las mejores combinaciones posibles, teniendo en cuenta factores como capacidad de integración, extensibilidad, nivel de compromiso de sus respectivas comunidades de usuarios/desarrolladores, disponibilidad de documentación, etc. Se planteará un modelo de arquitectura que intentará probarse como viable y se documentará todo el proceso seguido en el estudio así como las razones que motivaron el camino elegido en cada caso.

Por último, se documentarán todas los resultados obtenidos, las impresiones, los problemas que se puedan presentar, etc. con formato similar al de un tutorial, para que pueda servir de guía al lector.

1.2. Objetivos

Se pretende dar solución a los siguientes problemas:

- Configuración de repositorio de fuentes.
- Incluir en repositorio no solo los fuentes de programas sino también la información necesaria para la generación de los artefactos
- Generar automáticamente y con periodicidad parametrizable los productos para informar al equipo de desarrollo de posibles desviaciones no controladas en el proyecto.
- Gestionar las dependencias entre los distintos proyectos y sus productos.
- Generar informes y estadísticas útiles para el desarrollo y la gestión de los proyectos.
- Volcar en espacio común las versiones estables de los productos generados para evitar su pérdida y favorecer la gestión de recursos compartidos: repositorio de artefactos.
- Gestionar mediante ficheros de configuración los destinos finales de los productos para su distribución a entornos de pruebas y producción: distribución o despliegue.

1.3. Alcance

Este estudio intenta servir de ayuda a los equipos de desarrollo a la hora de decidir cómo acometer su proceso de construcción de software. A menudo esta decisión se pospone para las fases previas a su entrada en producción, pero para entonces la adopción de un modelo ordenado puede tener un impacto en la hoja de ruta dificil de asumir.

No se trata aquí tanto de hacer un desarrollo específico como de mostrar las bondades de un modelo, de unas herramientas y de unos métodos de trabajo.

Dado que en la mayoría de los casos cada herramienta aporta documentación acerca de su instalación y su configuración, se intentará abordar otros aspectos como la integración, la facilidad de uso y adopción, así como las experiencias más reseñables.

Capítulo 2

MATERIAL DE ESTUDIO

2.1. Ciclo de vida del software

Existe mucha literatura acerca del ciclo de vida del software y no es objeto de este trabajo ahondar en lo conocido. No obstante se incluye a continuación un extracto de [GP05] a modo de punto de partida para el desarrollo de los apartados siguientes:

> Se llama ciclo de vida del software a las fases por las que pasa un proyecto software desde que es concebido, hasta que está listo para usarse. Típicamente, incluye las siguientes actividades: toma de requisitos, análisis, diseño, desarrollo, pruebas (validación, aseguramiento de la calidad), instalación (implantación), uso, mantenimiento y obsolescencia.
>
> El proyecto tiende a pasar iterativamente por estas fases, en lugar de hacerlo de forma lineal. Así pues, se han propuesto varios modelos (en cascada, incremental, evolutivo, en espiral, o concurrente, por citar algunos) para describir el progreso real del proyecto.
>
> El modelo en cascada es el más simple de todos ellos y sirve de base para el resto. Simplemente asigna unas actividades a cada fase, que servirán para completarla y para proporcionar los requisitos de la siguiente. Así, el proyecto no se diseña hasta que ha sido analizado, o se desarrolla hasta que ha sido diseñado, o se prueba hasta que ha sido desarrollado, etc.
>
> Los modelos incremental y evolutivo son una variación del modelo en cascada en la que éste se aplica a subconjuntos del proyecto. Dependiendo de si los subconjuntos son partes del total (modelo incremental) o bien versiones completas pero con menos prestaciones (modelo evolutivo) estaremos aplicando uno u otro.
>
> El modelo en espiral se basa en la creación de prototipos del proyecto, que pasan por las fases anteriores, y que van acercándose sucesiva-

mente a los objetivos finales. Así pues, nos permite examinar y validar repetidamente los requisitos y diseños del proyecto antes de acometer nuevas fases de desarrollo.

Finalmente, el modelo iterativo o incremental es el que permite que las fases de análisis, diseño, desarrollo y pruebas se retroalimenten continuamente, y que empiecen lo antes posible. Permitirá atender a posibles cambios en las necesidades del usuario o a nuevas herramientas o componentes que los desarrolladores descubran y que faciliten el diseño o proporcionen nuevas funcionalidades.

Se trata de obtener de manera rápida una versión funcional del software, y añadirle prestaciones a partir de lo que se ha aprendido en la versión anterior. El aprendizaje proviene tanto del desarrollo anterior, como del uso del software, si es posible. En este tipo de desarrollo es imprescindible establecer una lista de control del proyecto, donde iremos registrando las funcionalidades que faltan por implementar, las reacciones de los usuarios, etc. y que nos proporcionará las bases para cada nueva iteración.

Este último método es el más usado (aunque sea involuntariamente) en proyectos de software libre, por la propia naturaleza cambiante de los requisitos o la aportación constante de nuevos colaboradores.

2.2. Control de versiones

Los sistemas de control de versiones permiten a grupos de personas trabajar de forma colaborativa en el desarrollo de proyectos, frecuentemente a través de Internet. Son sistemas que ponen marcas en las diferentes versiones para identificarlas posteriormente, facilitan el trabajo en paralelo de grupos de usuarios, permiten analizar la evolución de los diferentes módulos del proyecto, y mantienen un control detallado sobre los cambios que se han realizado; funciones que son indispensables durante la vida del proyecto.

Estos sistemas no sólo tienen aplicación en el desarrollo del software, sino que además son ampliamente utilizados en la creación de documentación, sitios web y en general cualquier proyecto colaborativo que requiera trabajar con equipos de personas de forma concurrente.

Los sistemas de control de versiones se basan en mantener todos los archivos del proyecto en un lugar centralizado, normalmente un único servidor, aunque también hay sistemas distribuidos, donde los desarrolladores se conectan y descargan una copia en local del proyecto. Con ella, envían periódicamente los cambios que realizan al servidor y van actualizando su directorio de trabajo que otros usuarios a su vez han ido modificando.

Los sistemas de control de versiones de código están integrados en el proceso de desarrollo de software de muchas empresas. Cualquier empresa que tenga

más de un programador trabajando en un mismo proyecto acostumbra a tener un sistema de este tipo, y a medida que crece el número de personas que se involucran en un proyecto, más indispensable se hace un sistema de control de versiones.

Los sistemas de control de versiones modernos suelen aportar una amplia gama de funcionalidades, de las cuales se extraen continuación las más relevantes:

- Control de usuarios. Establecer qué usuarios tienen acceso a los archivos del proyecto y qué tipo de acceso tienen asignado.
- Mantener un control detallado de los cambios realizados en cada archivo. Disponer de un control de los cambios efectuados en el archivo, fecha, causa que los motivó, quién los realizó y mantener las diferentes versiones.
- Resolver conflictos de actualización de archivos. En caso de que dos usuarios modifiquen un archivo proporcionar herramientas para gestionar este tipo de situaciones.
- Bifurcación de proyectos. A partir de un punto concreto, crear ramas del proyecto para que puedan evolucionar por separado. Es habitual utilizar esta técnica cuando se ha liberado una versión de un producto y se requiere que siga evolucionando.

2.3. Gestión de artefactos

Durante el proceso de compilación suele ser necesario realizar unas tareas fijas: generar una documentación del código, generar un ejecutable .jar, etc. Estas acciones se pueden automatizar mediante el uso de herramientas específicas que se detallarán más adelante.

Durante este proceso, además, pueden necesitarse artefactos externos para cumplir las tareas especificadas, y se plantean diferentes opciones sobre qué hacer con estos artefactos. Existe la opción de almacenarlos junto al proyecto, de modo que la compilación sea rápida y sin dependencias externas (ver figura 2.1). Sin embargo esto exige la replicación descontrolada de los mismos artefactos, con el consiguiente perjuicio en cuestiones de espacio y sobre todo de organización, por ejemplo al publicarse una nueva versión de un artefacto.

La otra opción, que estén siempre disponibles en Internet en sus páginas de origen (o incluso en un servidor externo), es una solución que tampoco es del todo adecuada, pues introduce un factor de riesgo en el proceso: la compilación será una tarea de tiempo variable y muy dependiente del ancho de banda disponible (ver figura 2.2).

Para evitar las desventajas de uno u otro método, se introduce una solución intermedia, que es la de construir un repositorio de artefactos, que contendrá una única vez los artefactos y que podrá estar en un entorno de red local, lo

Figura 2.1: Fuentes y artefactos de 3as partes en mismo Repositorio

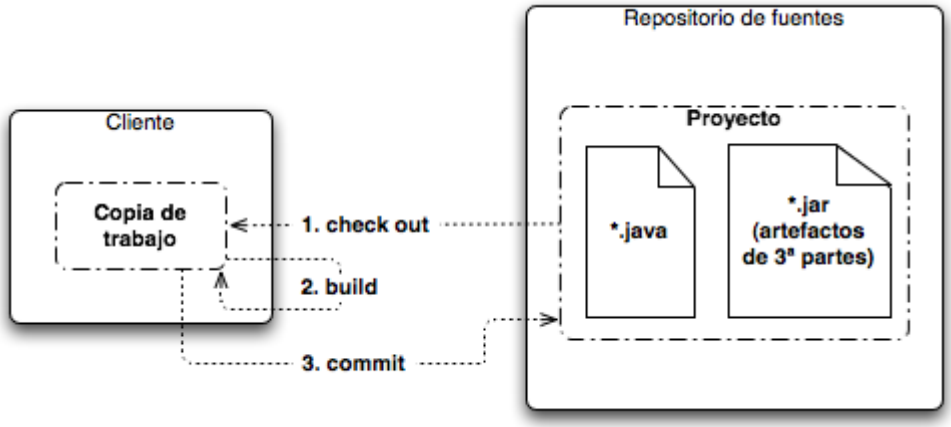
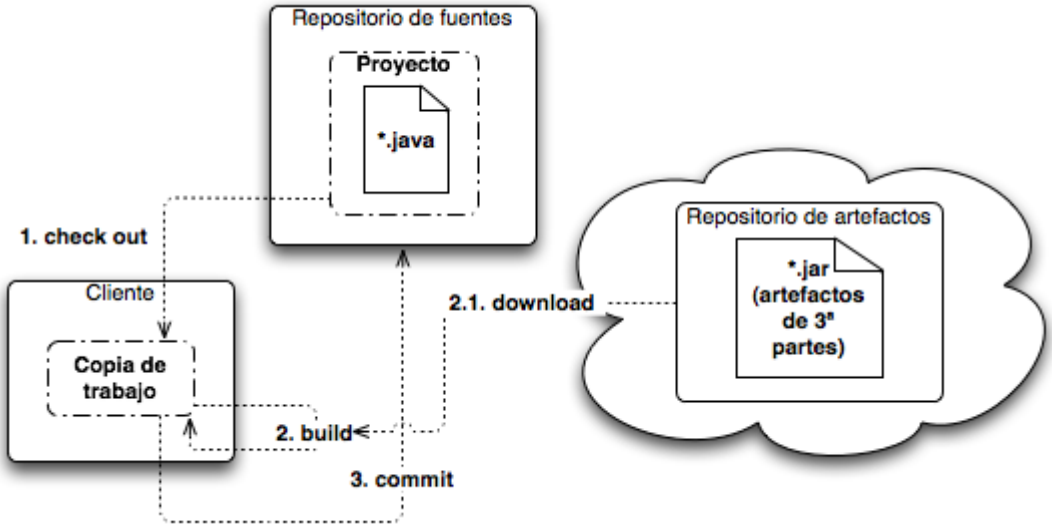


Figura 2.2: Fuentes en servidor local y artefactos de 3as partes en Internet

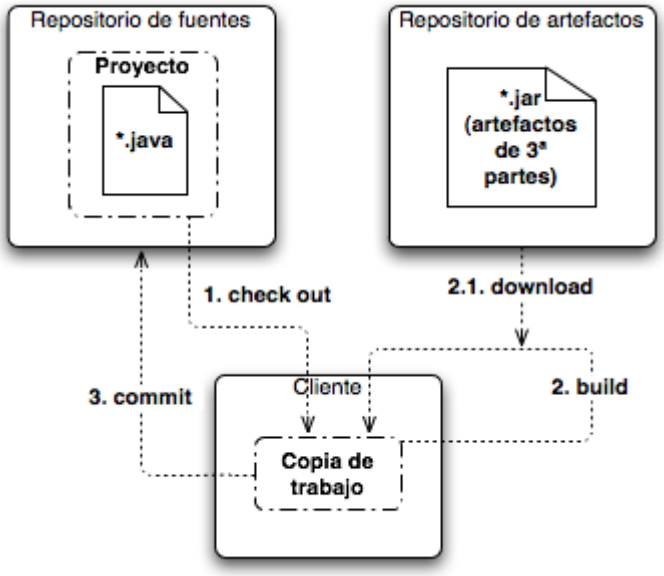


cual mejorará el tiempo de compilación, aparte de otras ventajas que se verán posteriormente (ver figura 2.3).

2.4. Gestión de dependencias

Se puede pensar que en proyectos pequeños o de poca complejidad, las dependencias entre distintos componentes es fácilmente manejable mediante una buena disciplina de documentación. El modelo de desarrollo de software libre, por propia necesidad, ha implantado una manera de hacer y de trabajar en la que el

Figura 2.3: Fuentes en servidores locales separados



control de las dependencias es clave.

Los ficheros Makefile ya no resultan extraños, se sabe que cuando se descarga un paquete que contiene el mencionado fichero habitualmente hay que ejecutar:

```
configure
make
make install
```

y se obtiene como resultado que cada componente pasa por fases de verificación de requisitos, compilación, empaquetado y distribución en el sitio adecuado. De estas etapas, la de verificación de requisitos no es más que un control sobre las dependencias que una pieza software tiene sobre otra.

La comunidad Java rápidamente vio esta necesidad y de ahí surgieron primero el proyecto Ant [CSL01c] y más adelante el proyecto Maven [CSL02]. Otros lenguajes tienen sus propios generadores como los Gems de Ruby [CSL07f], los Eggs de Python [CSL07e], los Módulos de Perl [CSL01a] [Wil97], así como repositorios donde se mantienen los paquetes y sus dependencias como el Python Package Index de Python [CSL99], el CPAN de Perl [CSL98b], el CTAN de LaTeX [CSL98c], por citar solamente algunos ejemplos representativos. También los sistemas operativos gestionan las dependencias de los paquetes como parte inseparable de las herramientas de instalación. Ejemplos de esto son los `rpm` de la ditribución RedHat Linux, los `deb` del mundo debian, los `depot` de HP-UX, etc.

La gestión de dependencias es elemento clave y aunque el proyecto se considere mínimo es una buena práctica incorporar técnicas que contemplen esta problemática en las fases tempranas de desarrollo ya que en ocasiones es dificil prever el crecimiento de un proyecto.

2.5. Gestión de despliegues

En este trabajo se considera despliegue *deployment* al proceso de instalar y actualizar componentes software en el contenedor de componentes. Estos componentes es lo que se viene llamando a lo largo del estudio los artefactos. En un sentido más amplio y dependiendo del contexto se puede considerar como parte del proceso de despliegue también la generación de los artefactos y su mantenimiento, pero se ha preferido establecer distinción entre estos dos conceptos como procesos separados porque el modelo resultante aporta más claridad al conjunto y más flexibilidad a la hora de seleccionar herramientas o patrones.

Los artefactos java pueden tener distintos acabados o formatos: `jar`, `war`, `ear`, etc. Cuando un artefacto o conjunto de artefactos se considera que debe ser desplegado en el entorno de pruebas o el de producción, lo deseable es que dependiendo del propio artefacto y del destino elegido se conozca con detalle qué acciones deben realizarse como pasos previos y/o pasos posteriores a su puesta en funcionamiento o activación. No será lo mismo si se despliega en un servidor o en varios. Si se trata de producción, posiblemente habría que activar unas tareas de comprobación más exhaustivas, emitir alarmas, prever mecanismos de marcha atrás. La posibilidad de aplicar algún tipo de automatismo o planificación también son funcionalidades que pueden resultar útiles.

2.6. Integración continua

Los conceptos de Integración Continua –*Continuous Integration*– tomaron relevancia a partir de un conocido artículo de Martin Fowler [Fow06], que ha sido posteriormente revisado. El éxito de este sistema puede comprobarse, como veremos más adelante, en el gran número de herramientas existentes, tanto libres como comerciales, que llevan a cabo esta tarea. Hay que destacar que el proceso de Integración Continua supone algo más que utilizar una herramienta ya que conlleva adoptar nuevas buenas prácticas. Esta práctica se asocia a las metodologías ágiles como Extreme Programming.

Un sistema de Integración Continua consiste en un ciclo que básicamente se encarga de revisar los cambios existentes en el repositorio, compilar automáticamente los proyectos y ejecutar pruebas automáticas de forma *frecuente*. Esto permite detectar antes los fallos, y evitar que tenga que ser el propio usuario el que compile el proyecto cada cierto tiempo. Con esto se elimina el riesgo de que en algunas fases el usuario no compile o no ejecute las pruebas y se haya escrito mucho código cuando un error es detectado. Hay que tener en cuenta que en este proceso no participa activamente el desarrollador, aunque puede forzar a que se produzcan compilaciones aparte de las automáticas. Cuando un sistema de Integración Continua detecta un fallo lo comunicará a los interesados con rapidez y esta supervisión será la que mejore el conocimiento de los fallos y en general el proceso de desarrollo.

En ocasiones implantar un sistema de integración continua puede no tener demasiado sentido: si no se han elaborado pruebas automáticas y la fase de compilación es rápida puede prescindirse de esta metodología. Cuanta mayor es la envergadura del proyecto y mayor número de desarrolladores trabajan en él, más conveniente será para el proyecto.

2.7. Virtualización

La virtualización se usa a menudo cuando se necesita hacer, bien algo que no es factible implementar en un entorno real, bien algo para lo que no se dispone de equipamiento.

IBM fue la primera compañía en hacer uso de la virtualización en entornos de servidor. El objetivo era sacar el máximo provecho de hardware muy potente en entornos *mainframe*, además de añadir flexibilidad. El VMM (*Virtual Machine Monitor*) se introdujo a finales de los años 60 cuando IBM quería dividir los recursos de los mainframes para entornos multitarea. Hoy día la virtualización es la base de su gama alta de mainframes de la serie zSeries.

Con el tiempo, el hardware de consumo ha evolucionado de manera que ahora es considerablemente más barato y potente, lo que permite el uso de la virtualización en este segmento. Los distintos ordenadores virtuales que ejecutan sobre ordenadores físicos son completamente independientes entre sí y para comunicarse entre ellos hacen uso, por lo general, del protocolo IP . La ventaja estriba en que si una de las máquinas virtuales se ve comprometida debido a un agente externo, las demás no se ven afectadas. Al ejecutar un servicio en cada máquina física, se está minimizando el riesgo.

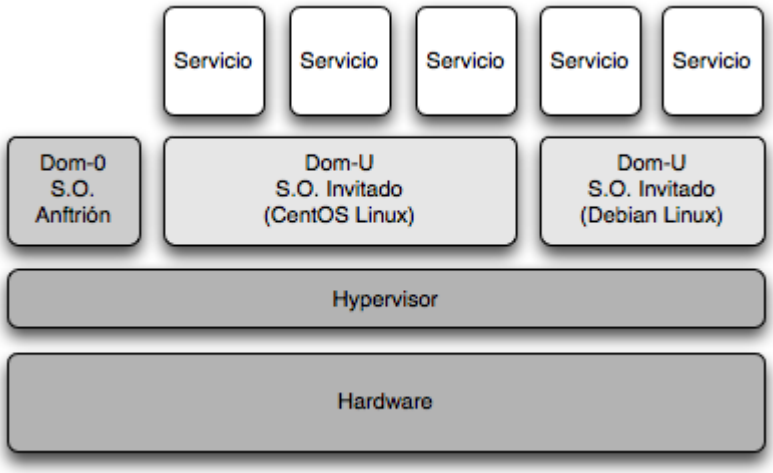
La tarea de llevar a cabo esta práctica, sin la virtualización, conlleva un aumento significativo de los costes debido al número de piezas hardware que habría que introducir además del coste derivado de su mantenimiento. Por medio de la virtualización, los servicios se pueden alojar en máquinas virtuales separadas que se ejecutan sobre un servidor físico. Este servidor constituye un punto único de fallo, pero al tener una instalación mínima y estable de sistema operativo capaz de ejecutar máquinas virtuales, el conjunto probablemente sea más fiable que complejos sistemas operativos dando servicio a multitud de tareas dispares.

La virtualización se puede acometer de varias maneras pero la más común es la de separar el contexto de las aplicaciones, del contexto del hardware, colocando entre éste y el sistema operativo, una capa que se encarga de la virtualización.

La Figura 2.4 (traducida del original de [Bra06]) muestra un ejemplo de una máquina física que está ejecutando tres sistemas operativos simultáneamente. Dos de ellos son máquinas virtuales (Dom-U = dominio de usuario) y una aloja el sistema operativo anfitrión (Dom-0). El nivel de abstracción denominado hipervisor proporciona encapsulación hacia arriba en las capas. El resultado es que todo el software que ejecuta dentro de una máquina virtual está fuertemente encapsulado incluyendo estados de conexión, estados de CPU y estados de me-

moria. Si una máquina virtual se ve comprometida, el sistema operativo anfitrión no resulta afectado.

Figura 2.4: Capas de modelo para-virtualización. Una máquina física ejecuta tres sistemas operativos



2.7.1. Tecnologías de virtualización

Hoy día existen diferentes técnicas de virtualización y la elección de cualquiera de ellas depende de las necesidades del sistema. Las diferencias principales entre ellas atienden a razones de flexibilidad y de eficiencia. La flexibilidad en virtualización se refiere al grado de dependencia con respecto del sistema operativo y del hardware. La capa de abstracción añade flexibilidad pero también penaliza el rendimiento. Las distintas técnicas de virtualización son:

Virtualización total (*Full virtualization*) También llamada "virtualización hardware". Proporciona un alto grado de flexibilidad ya que presenta un hardware virtual a las máquinas virtuales. El sistema operativo invitado, es decir, la máquina virtual, no sabe que no está ejecutándose directamente en el servidor físico. Se emula el hardware por medio de un Virtual Machine Monitor (VMM) que puede ser implementado de dos maneras: "basado en host", y 'no basado en host". En el primer caso un sistema operativo anfitrión contiene los drivers específicos necesarios para las máquinas virtuales. Entre las herramientas que usan este método figuran VirtualPC, QEMU, VMware Workstation y VMware Server. En el segundo caso el VMM es un monitor puro, es decir no es un sistema operativo completo, sino que solamente ejerce las funciones de monitor de máquinas virtuales con lo que se elimina una parte de la sobrecarga. A cambio este modelo requiere mucho más soporte hardware y lo convierte por tanto en menos flexible. A este método pertenecen herramientas como VMware ESX Server.

Para-virtualización (*Para-Virtualization*) Para eliminar sobrecarga en la capa de abstracción esta solución propone que esta capa no se muestre como idéntica al hardware que pretende ser sino ligeramente modificada. Esto obliga a que las máquinas virtuales deben ser "conscientes" de que ejecutan sobre una plataforma virtual y de hecho comparten algunas tareas con el VMM que en este contexto se denomina hipervisor. Este modelo tiene como ventaja principal que la capa de abstracción es mucho más fina con lo que el rendimiento mejora ostensiblemente. En estudios comparativos de las dos herramientas principales de virtualización total y para-virtualización, VMWare ESX y Xen respectivamente, mientras el primero ofrecía una sobrecarga de 10-15%, el segundo mostraba cifras del 1-4%. La herramienta más conocida de este modelo es Xen.

Virtualización de Sistema Operativo (*OS-Virtualization*) Este modelo no virtualiza hardware, sino que virtualiza sistema operativo, de modo que un solo kernel está ejecutándose en la máquina física y es usado tanto por el sistema operativo anfitrión como por los invitados. Es muy eficiente pero carece de flexibilidad, ya que las máquinas virtuales, al compartir kernel, deben ser todas del mismo tipo en cuanto a sistema operativo, por ejemplo, todas linux, y si son de distintas distribuciones, deben poder ejecutar sobre el mismo kernel, lo que no siempre es factible. Las herramientas más conocidas de este tipo son OpenVZ y Virtuozzo. Ésta última es una herramienta comercial basada en el producto software libre OpenVZ.

2.7.2. Ventajas de la virtualización

La virtualización ofrece muchas ventajas que pueden ser utilizadas en clusters de alta disponibilidad:

Flexibilidad Es posible ejecutar más de una instancia de un sistema operativo en una máquina física. Es posible también migrar una instancia virtualizada a otra máquina física y también se pueden gestionar las máquinas virtuales desde el sistema anfitrión de manera sencilla. También en algunos casos se pueden cambiar algunos parámetros de configuración de las máquinas virtuales mientras están activas como el tamaño de la memoria ram, el del disco, la dirección IP, etc.

Disponibilidad Se pueden mantener activas instancias de máquinas virtuales, mientras se ha parado un nodo físico para labores de mantenimiento, por medio de las facilidades de migración.

Escalabilidad Es muy sencillo añadir y eliminar nodos, ya que la instalación del sistema operativo anfitrión es trivial y, una vez añadido al cluster, añadir máquinas virtuales mediante facilidades de clonación o copia, carece de complejidad además de ser un proceso rápido.

Aprovechamiento de hardware Es una consecuencia que se desprende de poner en una misma máquina física, varias máquinas virtuales y de observar el comportamiento de las mismas.

Seguridad La separación de servicios facilita el aislamiento de los problemas y su posterior resolución. También evita que los servicios "inocentes" resulten afectados.

2.7.3. Inconvenientes

Sobrecarga El gran inconveniente de la virtualización, aunque este aspecto está cada vez más controlado. Las cifras que publica Xen, que es un sistema paravirtualizado, son del 4% de sobrecarga, lo que se puede considerar muy cercano al performance de la máquina.

Punto único de fallo Si la máquina física sufre una caída, las máquinas virtuales se verán afectadas inevitablemente.

Interfaz de gestión Es altamente dependiente de la plataforma de virtualización, y dificulta ciertos modelos que tienen como objetivo la consolidación de distintas plataformas bajo un mismo entorno.

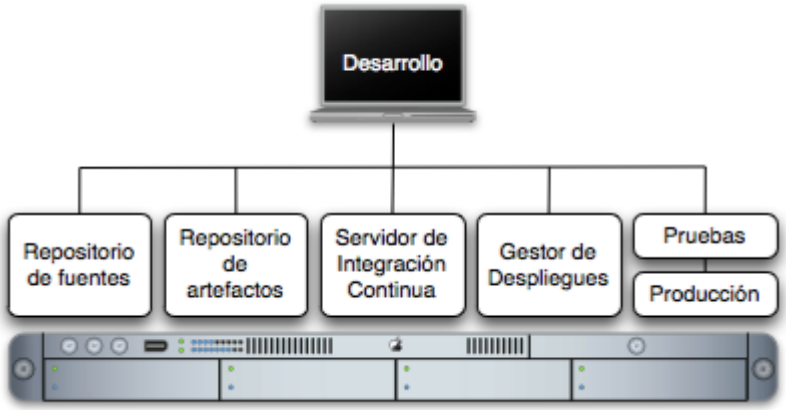
Capítulo 3

METODOLOGÍA

3.1. Arquitectura

Una parte importante del presente trabajo consiste en aprovisionar, en una sola máquina física, mediante técnicas de virtualización, varias máquinas virtuales que implementarán los distintos servicios clave del modelo de arquitectura que se muestra en la figura 3.1.

Figura 3.1: Aprovisionamiento de servicios

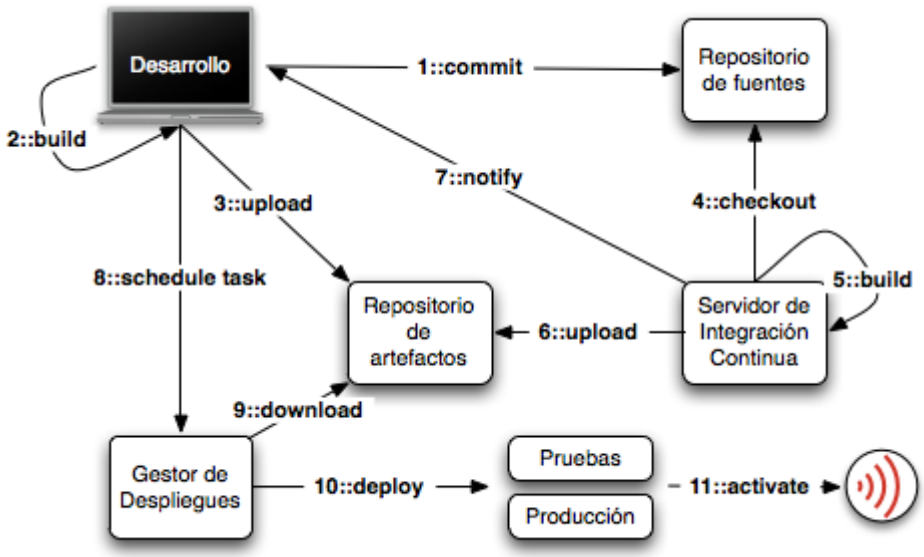


La figura 3.2 muestra la comunicación entre las distintas piezas:

1. Desarrollo consolida los cambios efectuados en el código al repositorio de fuentes
2. Desarrollo genera un artefacto en su estación de trabajo y cuando está conforme...
3. ... lo etiqueta y lo carga en el repositorio de artefactos.
4. El servicio de Integración Continua observa los cambios en el repositorio de fuentes y ...

5. ... comienza una tarea de generación de artefacto planificada generalmente nocturna.
6. El servicio de Integración Continua actualiza el repositorio de artefactos en caso de éxito y...
7. ... ante cualquier posible fallo en la generación emite notificación a Desarrollo
8. Cuando Desarrollo considera que el artefacto está listo para despliegue lo notifica al Gestor de Despliegues indicándole el destino del mismo.
9. El gestor de Despliegues dispone de información para cada artefacto y conoce por ejemplo las acciones que debe realizar en caso de fallo en el despliegue, el mejor momento del día para llevarlo a cabo, etc. Descarga el artefacto del repositorio de artefactos y...
10. ... lo distribuye.
11. Los servidores de Pruebas y de Producción reciben los artefactos del Gestor de Despliegues y proporcionan los servicios de negocio a los usuarios finales.

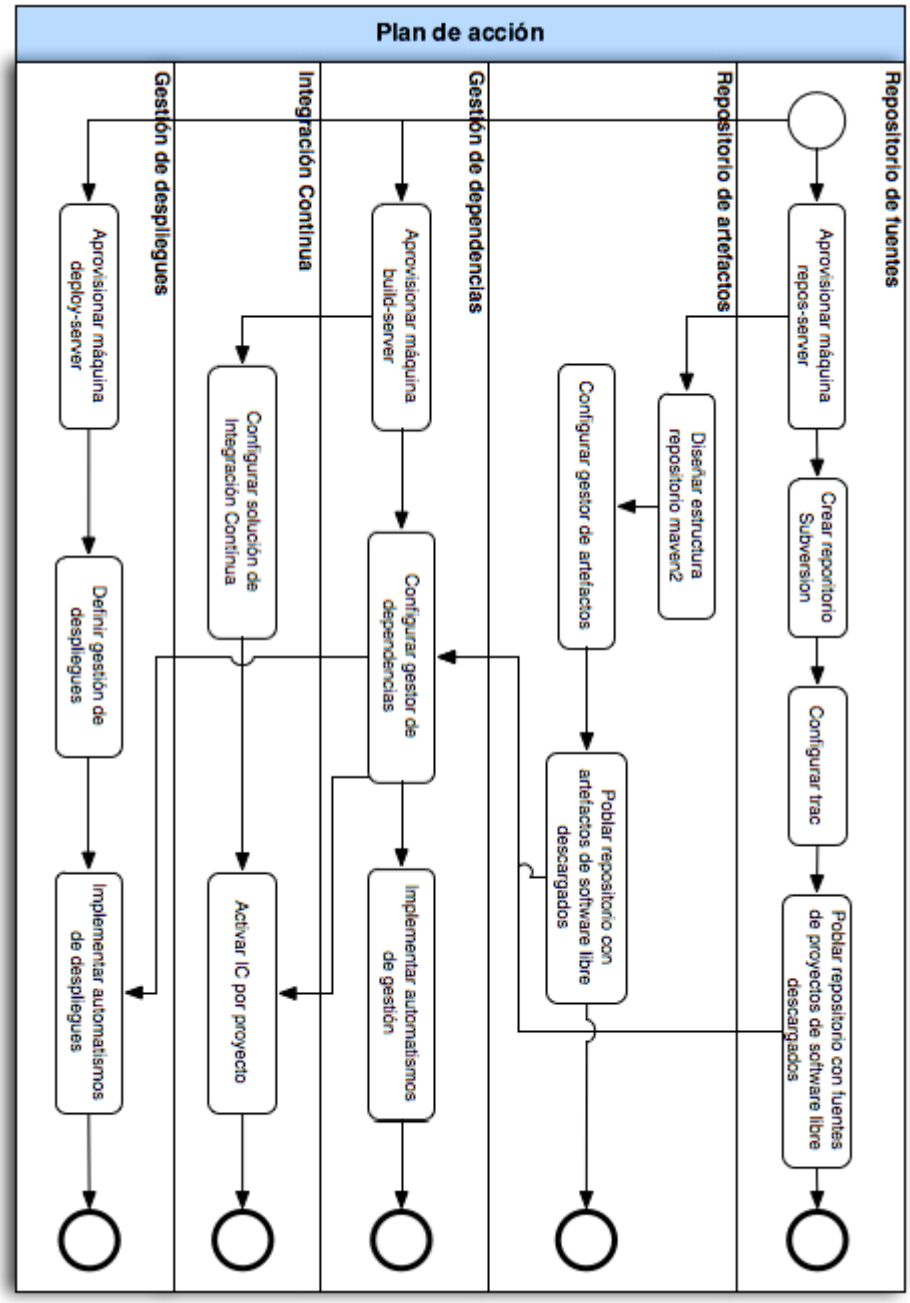
Figura 3.2: Comunicación entre los distintos componentes



3.2. Plan de acción

La figura 3.3 muestra las distintas secciones del proyecto, los procesos más relevantes, las dependencias entre ellos y la secuencia en que deben ser realizados.

Figura 3.3: Diagrama de procesos



3.3. Herramientas y equipamiento

Se han utilizado para el desarrollo del proyecto ordenadores de arquitectura x86, concretamente un equipo de sobremesa marca Dell con Intel Core 2 Duo, con 2GB de RAM y 200GB de disco duro y un portátil Apple MacBook Pro también con Intel Core 2 Duo 2GB de Ram y 120GB de disco duro. El portátil se ha usado básicamente para labores de prueba de conexión de red y de documentación. El equipo de sobremesa es el que se ha usado para instalar la virtualización y los servidores incluidos en la arquitectura.

Son equipos de bajo coste pero cuya versatilidad y parámetros elevados en cuanto a memoria, disco y capacidad de proceso, permiten el montaje de un pequeño laboratorio de pruebas, donde antes, no hace tanto tiempo, habría sido necesario aprovisionar más equipos y de coste muy superior.

Capítulo 4

DESARROLLO

4.1. Servidor físico

4.1.1. CentOS 5.1

El hardware con el que se realizará el proyecto soporta Linux con la distribución CentOS-5, versión libre compilada de Red Hat Entreprise 5. Puede encontrarse información en la página oficial de CentOS [CP04].

Se hará un esbozo de lo que se hizo durante la instalación del sistema operativo en el ordenador y sobre todo de los problemas encontrados durante su instalación, de modo que pueda servir de ayuda , aunque ciertas particularidades de la instalación que posteriormente se detallarán pueden ser específicas del equipo en el que se instaló y no aplicables a todos. Para un manual detallado se remite al lector a esta guía.

En primer lugar el equipo proporcionado venía con Windows XP instalado, pero fue formateado, por lo que no se hizo ninguna gestión adicional de particiones para soportar los 2 sistemas operativos. Como ante cualquier sistema operativo, fue necesario indicar al ordenador la secuencia correcta de arranque, para que arrancara antes con el cd que con el disco duro (modificando la BIOS). [1]

A partir de aquí el instalador proporciona un interfaz gráfico para facilitar los pasos de instalación. En este caso, dado que el ordenador iba a tratar asuntos de repositorios, se seleccionaron los grupos de paquetes relacionados con Virtualización, Desarrollo y Servidor web. No es necesario ahondar en detalles de paquetes durante la instalación ya que si resulta necesario instalar algún paquete

[1] Hubo algún pequeño problema, debido a que no arrancaba el instalador por alguna incompatibilidad con el disco duro, se incluyó una directiva que solucionó este error: `acpi=off`.Debido a la directiva introducida para que el instalador del sistema operativo se pudiera cargar, el disco duro sufrió una degradación en su rendimiento. Esto fue corregido con nuevas directivas añadidas a la anterior, y referentes a no hacer comprobaciones de disco en todo momento. Las directivas añadidas fueron `hda=noprobe hdb=noprobe hdc=noprobe`. Posteriormente se comprobó que las directivas anteriores *impiden que el equipo se apague automáticamente*. No obstante, cuando se decide apagarlo y aparece la secuencia *System Halted*, es seguro apagar manualmente el ordenador.

más, es muy sencillo extenderlo una vez instalado el sistema operativo por medio del gestor de paquetes *yum*. La opción de Virtualización instala una versión de kernel basada en la versión libre de Xen mejorada para este propósito y las herramientas de gestión de máquinas virtuales asociadas.

4.1.2. FreeNX

NX [Wik08a]es un software que proporciona acceso en modo gráfico a escritorios remotos. En este sentido no es distinto de otras herramientas como VNC [Wik08b]. Lo que convierte a NX en una herramienta de siguiente generación es que utiliza un protocolo que comprime la señal de tal modo que la sensación de velocidad de transferencia es mucho mayor.

Las librerías NX fueron creadas y liberadas bajo licencia GPL [GNU08] por la empresa NoMachine [NoM08]. El proyecto de software libre FreeNX [CSL08c] mantiene el software de servidor de nx lo que permite que algunas distribuciones como CentOS lo incluyan en su repositorio de paquetes extras. NoMachine facilita la descarga gratuita de una aplicación cliente de NX para las plataformas Desktop más comunes.

Software de este tipo es necesario si se quiere acceder remotamente a servidores en modo gráfico y para este trabajo, aunque se puede mediante sesión de terminal ssh administrar el entorno de virtualización y las distintas piezas instaladas, en algunos casos conviene tener disponible un acceso gráfico solvente.

La instalación de freenx en el servidor físico es muy sencilla:

```
yum install freenx
```

En otras versiones de CentOS se puede instalar siguiendo las instrucciones de esta página [CP08]

El servicio se arranca en el servidor con la siguiente instrucción:

```
service freenx-server start
```

La instalación del cliente es guiada por instalador gráfico y el paquete se puede descargar, para la plataforma deseada, de la página oficial de NoMachine.

4.1.3. Xen

De la página de preguntas más frecuentes del sitio oficial de Xen [CSL03c] se han traducido las dos primeras:

> **¿Qué es Xen?** El hypervisor Xen® es una tecnología única de código abierto, desarrollada de manera colaborativa por la comunidad de Xen y por ingenieros de más de 20 de los fabricantes más innovadores en soluciones para centros de datos, incluyendo AMD, Cisco, Dell, HP, IBM, Intel, Mellanox, Network Appliance, Novell, Red Hat, SGI, Sun, Unisys, Veritas, Voltaire, y obviamente,

Citrix. Xen está licenciado bajo la GNU General Public License (GPL2) y está disponible sin coste tanto el código fuente como el código binario. Xen es y será siempre software de código abierto, en el que la industria y el ecosistema Xen se unen para acelerar la adopción de la virtualización en el mundo empresarial.

¿Quién creó Xen? El hypervisor Xen se creó en 2003 en el Laboratorio de Informática de Universidad de Cambridge en lo que se conoce como el proyecto Xen Hypervisor liderado por Ian Patt con equipo formado por Keir Fraser, Steven Hand, y Christian Limpach. Este equipo junto con los tecnólogos emprendedores de Silicon Valley, Nick Gault y Simon Crosby, fundaron la empresa XenSOurce que fue adquirida por Citrix Systems en octubre de 2007.

El software Xen incluido en la distribución CentOS-5 es el que va a permitir la creación de las máquinas virtuales de la figura 3.1. Dado que solamente se dispone de un total de 2GB de RAM se crearán tres máquinas virtuales cada una configurada con 384MB de RAM para dejar memoria libre suficiente para que pueda trabajar el hypervisor. Los dos repositorios, el de fuentes y el de artefactos serán servidos por una de ellas que se llamará repos-server. El servidor de Integración Continua tendrá su propia máquina que se llamará build-server. El gestor de despliegues y lo que sería el servidor destinado a un entorno producción también comparten máquina y se llamará deloy-server. Esto deja libre para el servidor anfitrión unos 896MB lo que le permitirá trabajar con cierto margen.

Las máquinas virtuales contendrán una instalación mínima del sistema operativo CentOS-5, de tipo servidor y sin escritorio gráfico. El equipo de sobremesa usado para el presente trabajo viene provisto con un chip Intel Core 2 Duo con funcionalidad *Intel Virtualization Technology* lo que permite la posibilidad de instalar maquinas virtuales en modo *para-virtualized* pero añade una pequeña complejidad en la instalación ya que el instalador debe obtener la distribución vía nfs, ftp o http, en lugar de una imagen de CD como sería el caso si se tratara del modo *fully-virtualized*. Esto se resuelve añadiendo el punto de montaje del CD/DVD con la distribución, una vez insertado y montado, al fichero de exportación de nfs `/etc/exports`

```
/media/cdrom *(ro,no_root_squash)
```

y después arrancar el servicio nfs

```
service nfs start
```

Tras esta preparación se lanza el instalador de máquinas virtuales con el comando

```
virt-install -n repos-server -r 384 -s 20 -p \
  --nographics --nonsparse -f /var/lib/xen/images/repos-server.img \
  -l nfs://localhost:/media/cdrom
```

Con esto se le está indicando que genere una máquina virtual llamada repos-server de 384MB de memoria tomando 20GB de disco, que no use instalador gráfico, que sea paravirtualizado y que la distribución está accesible en la dirección nfs señalada.

Esto arrancará un proceso de instalación de CentOS-5 en modo texto y como se ha dicho más arriba, se procurará seleccionar solamente lo imprescindible en materia de paquetes a instalar. Lo mismo se puede conseguir utilizando el instalador gráfico `virt-manager` y proporcionando los datos de manera interactiva. Se lanzaría simplemente así:

```
virt-manager
```

Una vez creada y activada la primera máquina virtual se puede comprobar su actividad con virt-manager. Para hacer una parada de la máquina virtual se ejecuta el comando

```
xm shutdown repos-server
```

Es el momento de clonar la máquina repos-server para crear las otras dos máquinas virtuales necesarias:

```
virt-clone \
   --original repos-server \
   --name build-server \
   --file /var/lib/xen/images/build-server.img

virt-clone \
   --original repos-server \
   --name deploy-server \
   --file /var/lib/xen/images/deploy-server.img
```

El comando xm create se usa para arrancar las máquinas virtuales:

```
xm create repos-server
xm create build-server
xm create deploy-server
```

Con esto ya se dispone, por lo tanto, de las tres máquinas operativas.

4.2. Repositorio de fuentes

El siguiente paso es la configuración del servidor de repositorio de fuentes para lo cual se explorarán las distintas alternativas disponibles, siempre dentro del mundo del software libre.

4.2.1. CVS

Se puede decir que CVS [CSL89] es todavía el sistema más utilizado en el mundo del software libre para el control de versiones. Basado en el modelo cliente-servidor, el propio sistema es software libre y existen versiones del software para multitud de plataformas. Su solidez y su probada eficacia, tanto en grupos pequeños de usuarios como en grandes, le han convertido en la herramienta que utilizan proyectos de software libre como Mozilla, OpenOffice.org, KDE o GNOME, por mencionar solamente algunos.

El manual oficial conocido como *el Cederqvist* está considerado como la biblia del CVS [ea99]. Un libro muy completo en el que se aborda el proceso de desarrollo de manera muy práctica es el Fogel [FB03] también disponible en versión libre en formato pdf.

4.2.2. Subversion

Subversion [CSL01b] es un software libre de control de versiones que nació en el año 2001 con el objetivo de solventar las principales carencias de CVS. Muchos desarrolladores ven en Subversion el sistema de control de versiones que sustituirá a CVS.

Entre las principales mejoras de Subversion respecto a CVS, destacan:

- Posibilidad de mover archivos. Una de las grandes limitaciones de CVS ha sido la carencia de un comando para poder mover archivos entre diferentes partes del repositorio. Subversion finalmente corrige esta carencia.
- Commits atómicos. La posibilidad de enviar un conjunto de cambios al repositorio en bloque evitando los problemas que suceden en CVS cuando e envía una gran cantidad de cambios al repositorio, que son todos parte de un mismo trabajo, pero sólo parte de ellos son aceptados por CVS.
- Metadatos. Subversion permite guardar por cada archivo o directorio un conjunto de llaves emparejadas con su valor que permiten almacenar metadatos y luego recuperarlos.
- Versionado por directorios. CVS sólo ofrece control de versiones por archivo, con Subversion se puede conseguir también control por directorio.
- Soporte para diferentes transportes de red. Subversion ha sido diseñado para que sea muy sencillo añadir nuevos transportes de red, como conexiones seguras mediante SSH o mediante WebDAV.
- La comparación de archivos se realiza con un algoritmo binario que es mucho más efectivo.

Se recomienda la lectura de los libros [CSFP04] –también disponible en versión libre online y en formato pdf– y [Roo05] pues son excelentes referencias. Y

con un enfoque más cercano a la metodología ágil, el libro de la serie Pragmatic [Mas05]

4.2.3. Otras alternativas

Hay muchas más herramientas de gestión de código fuente –*SCM source code management*– y algunas de ellas están plantando cara a Subversion con funcionalidades no contempladas en ésta, como la descentralización de los fuentes y la capacidad de trabajar con grandes proyectos distribuidos. Bazaar [CSL08a], Mercurial [CSL07b], Darcs [CSL04] son algunas de ellas. Se pueden estudiar en detalle las funcionalidades que aporta cada una de ellas en páginas como esta [CSL08b] donde aportan comparativas y opiniones de usuarios.

4.2.4. Trac

Trac [ES08] es una herramienta capaz de administrar la información que le aporta Subversion y mostrarla en un interfaz web. La aplicación se apoya en una base de datos (SQLite, PostgreSQL o MySql pueden valer) y es capaz de mostrar la información gracias al sistema de templates que proporciona ClearSilver (HTML) o Genshi (XML). Además del propio visor de archivos incluye opciones que lo hacen interesante de cara al desarrollo de proyectos como:

- Posibilitar el acceso con diferentes permisos a los archivos o a edición de contenidos en función del usuario.
- Un formato wiki integrado con la herramienta, de modo que se pueden relacionar versiones de código fuente con explicaciones detalladas en páginas wiki.
- Posibilidad de informar mediante RSS de cambios.
- Un subsistema de *tickets* integrado en Trac, de uso más sencillo que Bugzilla.

4.2.5. Instalación

Estos son los componentes que se van a instalar en esta sección:

- Servidor web de Apache.
- Subversion.
- Módulo de Apache para acceder vía WebDAV al repositorio
- Trac

A continuación los comandos de instalación:

```
yum install httpd
yum install subversion
yum install mod_dav_svn
yum install trac
```

4.2.6. Configuración

Configurar Subversion con Apache

Se procede a configurar Subversion para que pueda ser accedido vía http con WebDAV. El detalle de la configuración de Subversion con Apache se puede consultar en el apéndice A.1.

Una vez configurado el repositorio se podría navegar por http desde la dirección:

```
http://respos-server/repos
```

Configurar Trac con Subversion

Una vez instalado Trac, lo siguiente es configurarlo para poder ver en el navegador el repositorio Subversion. Las primeras pruebas irán orientadas a la visualización de los datos, para pasar posteriormente a modificaciones y configuración de acceso.

Es necesario, además de que exista un repositorio Subversion, crear un entorno para que Trac trabaje con él. Esto se consigue utilizando *trac-admin* mediante la instrucción:

```
trac-admin /var/lib/trac initenv
```

donde `/var/lib/trac` es el path donde se ubicará el entorno de Trac. Posteriormente a esta instrucción, la aplicación pedirá nuevos datos: el nombre del proyecto, el tipo de base de datos utilizada, el tipo de repositorio, el path al repositorio y el directorio donde se encuentren los templates de Trac. En algunos casos ya tienen valores por defecto adecuados, por lo que no será imprescindible introducir todos los datos, y además si no se conocen los datos pueden modificarse posteriormente en el fichero de configuración `trac.ini` sin tener que crear de nuevo el entorno.

Trac es esencialmente una aplicación web escrita en Python y el servicio puede arrancarse de varias maneras. La más inmediata es arrancar el demonio `tracd` por un puerto:

```
tracd --port 8000 /var/lib/trac
```

con lo que el acceso se haría a través de la dirección `http://repos-server:8000/`. Pero aunque sea un poco más costoso de configurar es preferible ejecutarlo dentro

del entorno de Apache sirviéndose de `mod_python` ya que se gana en rendimiento y flexibilidad. Un ejemplo de configuración con Apache se puede encontrar en el apéndice A.2. y en este caso la dirección válida de acceso sería:

```
http://repos-server/trac
```

A partir de aquí puede visualizarse el código, utilizar la wiki, el sistema de tickets, etc. Quedarían por configurar las opciones de autentificación ya que en este punto, si se hace click en el apartado *login*, la aplicación emitirá un mensaje diciendo que el módulo no está instalado. Éstas y otras opciones de Trac pueden consultarse en el apartado de la guía oficial [ES08]

Las modificaciones pueden hacerse por consola, aunque precisamente la mayor utilidad de Trac es presentar la información de forma ordenada y agradable por interfaz web. Las modificaciones, creación de tickets, etc. son procesos guiados por la aplicación y de uso sencillo.

Aunque esta opción no fue explorada, se debe mencionar que Trac en principio no permite el acceso con un repositorio Subversion remoto. No obstante, se puede utilizar mediante SVK [Kao03] que facilita establecer un *mirror* con el repositorio.

Durante la fase de pruebas se mantuvo un servidor con el Trac configurado y con algunos proyectos de software libre descargados para comprobar la bondad de la instalación. No obstante, para ver un ejemplo de su aplicación en un proyecto real, se remite al lector al Trac del repositorio de proyectos de software libre MacPorts de la plataforma Mac [CSL06d]

Otras utilidades de Trac

Utilizar Trac simplemente como visor de código es una opción válida, pero supondría obviar las otras funcionalidades que incluye, con especial atención al gestor de tickets y al sistema wiki integrados. Se da a continuación una breve explicación de sus características.

Trac como gestor de tickets Un ticket en Trac es el medio de comunicar una incidencia, sugerencia, o posible fallo del programa. Utilizar un fichero de texto para incidencias, aunque esté centralizado (por ejemplo en una página wiki o en el propio servidor cvs), se antoja insuficiente para gestionar eficientemente la lista de tareas a realizar en un proyecto: Es posible en estos casos que no se consulte y actualice con asiduidad, y será inevitable que haya falta de información sobre quien tiene asignada una tarea, su estado, y las observaciones realizadas. Además, cuando un problema sea solucionado normalmente será borrado y sin embargo es información que puede ser adecuado consultar en el futuro, sin perderse entre cientos de incidencias descritas.

El sistema de tickets de Trac permite introducir cualquier nueva tarea a realizar a través de un sencillo interfaz web, que pedirá información sobre el

título de la tarea, quién lo reporta y a quién se asigna, la versión del ticket, palabras claves para ser fácilmente identificado, etc. Estos campos podrán ser posteriormente modificados, e incluso existe la posibilidad de configurar qué campos se desea que aparezcan por defecto. Su modo de uso no se detalla pues trabajar con los tickets desde el interfaz web es una tarea sencilla y completamente intuitiva.

Trac no es el único sistema que permite gestionar de forma eficiente los bugs: está también muy extendido el uso de Bugzilla [CSL98a], aunque la popularidad del sistema de tickets Trac va en aumento y cada vez son más usuarios los que utilizan esta herramienta. Existen opiniones diversas sobre cual es la mejor herramienta sin un resultado claro y por supuesto dependiente del uso que le dé cada usuario, aunque se destaca que Trac tiene un interfaz más claro además de la ventaja de centralizar toda la información y no tener solo la de los bugs. Existe un script de python, `bugzilla2trac.py`, para que si se ha comenzado a trabajar con Bugzilla en algún proyecto y se desea cambiar de gestor de tickets, se pueda pasar a Trac sin perder todo lo anotado hasta la fecha. Puede encontrarse en la sección de contribuciones de la página de Trac, aunque no ha sido analizado en profundidad y habría que comprobar su funcionamiento. Existe más información sobre conversiones entre sistemas gestores de tickets (también de Mantis a Trac) en una wiki de phpdoctrine.

Wiki integrada en Trac La expansión del formato wiki, con sus ventajas de fácil edición por cualquier usuario y la facilidad para organizar los contenidos, ha provocado que muchas herramientas que tienen que gestionar información incluyan un sistema wiki. En este caso la función principal de la wiki es servir de soporte a explicaciones detalladas de bugs y cambios en el código, pudiendo hacer referencias desde los mensajes de commits o los tickets a estas páginas wiki. Esta forma de trabajar permite en primer lugar poder dar un formato adecuado a las explicaciones, que facilite su lectura, y por otra aprovechar la inclusión de enlaces desde múltiples puntos a las páginas wikis, de modo que la información está mucho mejor organizada y es más fácilmente accesible.

A partir de aquí, una cuestión que puede surgir es si se puede o debe integrar esta wiki con otras wikis quizá ya existentes en un entorno determinado como puede ser una intranet corporativa. Probablemente lo más aconsejable sea separar el tipo de información que hay en cada wiki. En este caso la wiki de Trac se utilizaría simplemente para aclaraciones sobre temas relativos a código fuente, gestión de cambios en desarrollo, y explicación de fallos de funcionamiento o nuevas funcionalidades por implementar.

Plugins La aplicación viene con unas opciones por defecto, pero pueden añadirse nuevos *plugins* que añadan características, para centralizar la información del proceso de compilación en menos herramientas. Puede consultarse un

listado completo de *plugins* en el sitio oficial de Trac [ES08]. Se hará referencia en esta sección a los *plugins* de Trac utilizados en el proyecto:

- *Plugin* de integración de Trac con CruiseControl [vL05].
- *Plugin* de integración de Trac con Continuum [CSL03b].
- *Plugin* de integración de Trac con Hudson [CSL08d].

Se hace referencia a su instalación y configuración en la sección posterior de Integración de Trac con Herramientas de Integración Continua.

4.3. Repositorio de artefactos

El repositorio central en internet de artefactos java, Maven2 [CSL05b] surge del proyecto Maven [CSL02] de Apache como aplicación práctica del modelo de gestión de dependencias que incluye la herramienta. Se llama Maven2 porque la primera versión de Maven como herramienta software, proponía una organización de los artefactos, que se vio superada en la versión siguiente por otra sensiblemente distinta, lo que hizo que el primer repositorio central quedara desfasado.

El antiguo repositorio sigue existiendo pero apenas tiene actividad. El repositorio central vigente, esto es, Maven2 contiene una cantidad considerable de los artefactos java existentes, pero no están todos los que son ni los que están se encuentran actualizados a la última versión. Esto se debe a que, a pesar del gran esfuerzo que la comunidad Maven y su mentor la Fundación Apache, ponen en motivar a las distintas comunidades java para que actualicen sus artefactos en el repositorio, no es tarea facil mover las conciencias de todos por igual. El resultado es que se tiene un repositorio incompleto pero en muchos casos, bastante funcional ya que los artefactos comunes más usados suelen estar en el repositorio actualizados.

El uso de un repositorio central facilita la obtención de los artefactos, permitiendo que la tarea de obtener los artefactos sea transparente al usuario. No obstante, sería también bastante ineficiente que para compilar cada proyecto siempre fuera necesario descargar los artefactos al ordenador local, incluso aunque se hubiera descargado ya antes. Por ello la idea que se aplica en la gestión del repositorio es similar a la jerarquía de memoria de un computador, donde en primer lugar se busca el dato en una memoria cache, posteriormente en una memoria cache de segundo nivel (y en maś niveles si hay), luego en memoria principal y finalmente en memoria virtual (disco), de modo que se ordenan las búsquedas por la rapidez en que se producirán.

En este caso, la aplicación buscará los artefactos en un directorio /lib, dentro del proyecto que se quiera compilar (método más rápido). Posteriormente buscará en el repositorio local creado en el ordenador, por defecto en el directorio `$home/.m2` si se trabaja con maven, como herramienta de construcción, o en

`$home/.ivy` si se trabaja con ivy [2], que reúnen todos los artefactos que se han descargado. Si no se encuentra ya se buscará en un repositorio compartido, y finalmente, y sólo en el caso de que no se haya descargado nunca en el repositorio local, se descargará el artefacto de Internet de los repositorios públicos. En éste último caso se mantendrá cierta incertidumbre acerca del tiempo de compilación, por la dependencia con respecto al ancho de banda.

Los repositorios de tipo Maven2 pueden ser especificados fácilmente en los ficheros de configuración de cada proyecto, de modo que cada proyecto puede utilizar también unos u otros repositorios y en un orden determinado. Los repositorios públicos de Maven tienen un gran número de artefactos almacenados que incluye los más habituales. El repositorio central de Maven2 puede consultarse de manera más amigable en el sitio MVNRepository [CSL06b].

Otra iniciativa de generar un repositorio central vino de la mano de los fabricantes originales de la herramienta de gestión de dependencias Ivy [CSL05a]. IvyRep era el repositorio por defecto que utilizaba Ivy pero no tuvo mucha repercusión y ahora usa Maven2. La idea de estos repositorios parte de un concepto diferente, ya que los repositorios públicos de Ivy almacenan los ficheros ivy.xml de los proyectos y no los artefactos, que sí están en los repositorios MavenX. Por tanto se suelen utilizar repositorios Maven desde Ivy –existe compatibilidad en este sentido, pero no en el contrario– para obtener los artefactos. Como ya se ha indicado anteriormente, cuando se instala maven, el repositorio local por defecto está en `$home/.m2`.

4.3.1. Gestión de artefactos

Existen herramientas que facilitan la gestión, la catalogación, la navegación por el árbol de dependencias, de los artefactos. Las más relevantes se describen en los siguientes apartados.

Artifactory

Artifactory [Jfr06] es un gestor del repositorio Maven2 . Proporciona una aplicación web para la gestión y navegación y capacidad de proxy, cache, de seguridad de acceso. Existen otras herramientas de este tipo, pero se reducen en la mayoría de casos a aportar un proxy para el repositorio y no incluyen este tipo de interfaces y posibilidades.

Para ejecutar Artifactory como servidor autónomo, hay que situarse en el directorio donde se ha instalado y ejecutar:

```
./artifactory.sh
```

[2] Maven e Ivy como software de construcción se explican en detalle más adelante en el apartado de Gestión de dependencias

También se puede ejecutar desde un servidor Tomcat, copiando el fichero .war de la carpeta WebApps de artifactory sobre el directorio del mismo nombre de Tomcat.

Sobre un manual más completo de Artifactory, aunque su uso es muy sencillo, se remite al lector a este artículo [Man07] donde además se explica en detalle cómo se puede crear un repositorio Maven centralizado.

Apache Archiva

Apache Archiva [CSL07a] es un gestor de un repositorio de artefactos que surgió del grupo de desarrollo de Maven pero que ha cobrado vida propia lo que indica que tiene un ritmo de desarrollo y un nivel de aceptación elevados ya que es un proyecto que en abril del 2007 lanzó su primera versión alpha. En estos momentos se considera todavía una herramienta más probada y desarrollada Artifactory, pero entre las características que se detallan en la documentación de Archiva parece encontrarse un mayor control y seguimiento de las actividades del repositorio, punto débil de Artifactory que no informa del número de veces que los artefactos son descargados, por ejemplo, por lo que sería adecuado en el futuro hacer un seguimiento a la evolución de estas herramientas.

Existe una distribución llamada Maestro [Glo08] que reúne en un mismo paquete Maven, Archiva y Continuum [3] como solución integral.

Proximity -Nexus

La otra alternativa hasta hace poco se llamaba Proximity [Cse05], aunque las opiniones encontradas en la red la descartaban, por no presentar herramientas que facilitaran el backup ni la importación/exportación de artefactos y por la lentitud de la aplicación. Seguía un esquema más parecido a Archiva pero no llegó a tener versiones estables y una gran actividad en su desarrollo, lo que quizás provocara que fuera retomada por otro patrocinador y relanzada como Nexus [CSL07d] con nuevas funcionalidades.

4.3.2. Buenas prácticas en la gestión de artefactos

Los medios que se están estudiando presentan ventajas evidentes pero obligan a mantener más orden en lo que se refiere a la gestión de los artefactos. Por ejemplo, si se necesita un artefacto para un proyecto concreto, y se quiere que se descargue desde el repositorio local, debe estar claramente identificado.

Los criterios que se presentan aquí están orientados a conseguir la máxima uniformidad y se han tomado a partir de experiencias observadas en distintos

[3] Continuum es una solución de Integración Continua que surgió como proyecto satélite de Apache Maven

proyectos de Software Libre. Aún así, en muchos casos estos criterios no son incompatibles con otros métodos de actuación que ya estén implantados en un entorno de trabajo determinado. Lo importante es que los criterios que se utilicen sean comunes y debidamente conocidos por todos los integrantes del equipo de desarrollo de un proyecto.

Dar nombre a los artefactos de acuerdo a un patrón

Es recomendable asignar un nombre al artefacto siguiendo siempre un patrón, por ejemplo `[nombre]-[revisión].[ext]`, ya que así resulta más fácil categorizar y gestionar los artefactos, de modo que no haya lugar a dudas sobre el artefacto que se está descargando en un proceso determinado.

Esto evita posibles errores por versiones. Se pueden guardar distintas versiones del mismo artefacto, ya que cada proyecto puede usar una versión diferente, y no tener que descargar siempre la última.

Es aconsejable adoptar nomenclaturas similares a las que usan los repositorios públicos, ya que facilita la descarga sin tener que hacer apenas modificaciones en los ficheros de configuración relativos a gestión de dependencias.

Indicar siempre versión del artefacto

Aunque no sea requerida una versión concreta de un artefacto en un proyecto, es obligatorio en su fichero ivy.xml indicar la versión. La versión se incluirá en el campo `rev` de cada dependencia, que es un campo obligatorio.

En el caso de que no haya una versión, por tratarse de un artefacto interno sin versión, puede incluirse por defecto siempre como versión "1", o una cadena `latest`. Es bastante habitual que todos los artefactos tengan un número de versión asociada.

Organization y name: GroupId y ArtifactId

Es obligatorio indicar el campo `name` del artefacto, que se corresponderá en general al `artifactId` –no incluye versión ni extensión–. El campo organization `org` no es obligatorio, aunque se recomienda su uso pues se corresponderá con el campo `groupId` de cada artefacto en el fichero `ivy.xml`, con la importancia añadida de que estos nombres en general formarán el path hasta el artefacto.

Es recomendable respetar los datos de los repositorios públicos e indicar correctamente estos campos, pues ante cualquier incidencia en el repositorio local siempre se podrían descargar con los datos correctos la mayoría de artefactos de los repositorios públicos automáticamente.

Si por ser un artefacto interno no tiene estos campos, es una buena práctica indicar el nombre del proyecto como `groupid` y el nombre del artefacto, sin extensión ni revisión, como `artifactid`. Así, si dos aplicaciones necesitan un artefacto de otro proyecto, ambas tendrán la misma forma de llegar al artefacto y no será necesario almacenarlo varias veces en carpetas internas del proyecto.

Indicar tipos de dependencias

Existen diferentes tipos de dependencias entre los proyectos y los artefactos, y si este aspecto se gestiona bien, un desarrollador puede hacer uso de esta información y así obtener sensibles mejoras de tiempo de compilación.

Si se desea sólo compilar para hacer una comprobación de sintaxis, se pueden comentar algunas dependencias en el fichero de configuración para evitar su descarga. Y aún más importante, se puede categorizar la información, de modo que se pueden prevenir riesgos y conocer el posible impacto en el proyecto como resultado de aplicar cambios en los artefactos. Si se indica que una dependencia sólo se produce en la fase de compilación, no es necesario hacer pruebas de ejecución sobre su correcto funcionamiento.

Se aportan como ejemplo las configuraciones de dependencias empleadas en algunas pruebas en Ivy, aunque se pueden especificar a partir de éstas otro tipo de dependencias más concretas que digan más sobre el proyecto.

```
<configurations>
<conf name="default" visibility="public" description="runtime
    dependencies and master artifact can be used with this conf"
    extends="runtime,master"/>
<conf name="master" visibility="public" description="contains only
    the artifact published by this module itself, with no transitive
    dependencies"/>
<conf name="compile" visibility="public" description="this is the
    default scope, used if none is specified. Compile dependencies
    are available in all classpaths."/>
<conf name="provided" visibility="public" description="this is much
    like compile, but indicates you expect the JDK or a container to
    provide it. It is only available on the compilation classpath,
    and is not transitive."/>
<conf name="runtime" visibility="public" description="this scope
    indicates that the dependency is not required for compilation,
    but is for execution. It is in the runtime and test classpaths,
    but not the compile classpath." extends="compile"/>
<conf name="test" visibility="private" description="this scope
    indicates that the dependency is not required for normal use of
    the application, and is only available for the test compilation
    and execution phases."/>
<conf name="system" visibility="public" description="this scope is
    similar to provided except that you have to provide the JAR which
     contains it explicitly. The artifact is always available and is
    not looked up in a repository."/>
</configurations>
```

Organización de repositorios

Se puede definir una sucesión de repositorios de los que descargar los artefactos. Por un lado, definir un único repositorio supone que pueda ser poco manejable por el rápido crecimiento del número de archivos. Por otro lado, definir numerosos repositorios puede desembocar en que no se llegue a saber claramente donde están los artefactos, haciendo demasiadas búsquedas sin éxito, y de nuevo terminando en una sobrecarga considerable de artefactos repetidos.

Si se desea mantener más de un repositorio, debe estar debidamente documentado cuál es el adecuado para importar/exportar y qué tipos de artefactos, con el fin de no perder el control sobre el proceso. También en el caso de múltiples repositorios es aconsejable tener en cuenta el orden de búsqueda más óptimo.

Qué guardar en el repositorio de artefactos

Todo fichero `.jar` o `.ear`, entre otros, puede ser considerado un artefacto. Por ello una solución puede ser volcar todo fichero con alguna extensión de este tipo en el repositorio. En artefactos externos, se recomienda esta solución, pues aunque puedan obtenerse de servidores públicos, se perdería la ventaja de tener todos los contenidos en una red interna y que el tiempo de compilación sea independiente del ancho de banda. Sólo si por motivos de espacio esto no resulta adecuado, se recomienda entonces almacenar en el repositorio los artefactos cuyo uso sea más frecuente.

En artefactos internos, esto es, los generados dentro del propio proyecto, debe evaluarse si se quiere volcar o no, el artefacto generado. En general, lo recomendable es almacenar unas versiones concretas de interés, ya que en cualquier momento se puede generar una versión con los archivos de una fecha concreta.

No repetir

Esta idea está subyacente en los anteriores puntos, no obstante se va a mencionar de manera expresa ya que no sólo se trata de una cuestión de espacio, sino también de organización. Mantener una organización adecuada en el repositorio de control de versiones puede suponer una reducción de un 70 % en el tiempo de importación de un proyecto. Mantener un repositorio de artefactos ordenado y sin redundancias facilita la labor de los scripts de construcción y también tiene impacto sobre el tiempo final de construcción.

4.3.3. Conclusiones

En el apartado de repositorio de artefactos no hay muchas decisiones que tomar al ser Maven2 la única opción viable, si se quiere hacer uso de un repositorio central en internet como auxiliar al repositorio local, ya que es conveniente que ambos sigan la misma estructura organizativa y protocolos.

En el apartado de gestión de artefactos, tras navegar por las listas de funcionalidades que proporciona cada herramienta de las disponibles, Artifactory es la que más madura se encuentra en estos momentos y por lo tanto, es la seleccionada para llevar a cabo la instalación.

Aunque un poco desfasada, la entrada de este blog [Chi07] contiene información interesante acerca de las herramientas mencionadas en el apartado de gestión de artefactos.

4.4. Gestión de dependencias

En lo que se refiere a herramientas de gestión de dependencias, las opciones se reducen a dos, en el ámbito del lenguaje Java. Por un lado está Ivy [CSL05a] como producto software con clara vocación de cumplir esta demanda. Por otro lado está Maven [CSL02], de cuya lista de funcionalidades, la gestión de las dependencias es una de ellas, como una parte de un método de trabajo que incluye además, la compilación, organización y gestión de la documentación.

Ivy se pensó para añadir funcionalidad de gestión de dependencias a Apache Ant [CSL01c] y es ideal para proyectos en los que ya se viene usando Ant, y en los que no se tiene intención de cambiar o se quiere un impacto mínimo en la organización de los objetos.

Maven es una solución completa que obliga a tomar la decisión de reorganizar el proyecto si se está trabajando con Ant por ejemplo. Si se trata de un nuevo proyecto, la elección de Maven puede ser la adecuada pero hay que tener en cuenta que tiene un coste de adaptación más elevado, ya que Maven impone ciertas restricciones sobre la organización del código y de la documentación. Pero gracias a estas restricciones el producto resultante tiene un acabado más "estandarizado", puesto que sus normas de uso están basadas en la adopción de una serie de buenas prácticas recogidas a partir de experiencias con metodologías ágiles de desarrollo.

Al usar Ivy como repositorio central, Maven2, y ser más flexible en sus planteamientos, en este trabajo se ha descartado Maven en favor de Ivy. Pero esto no quiere decir que se piense que una sea mejor que la otra, sino simplemente que la opción elegida responde mejor al modelo de arquitectura planteado en este trabajo.

4.4.1. Ant

Ant es la primera aplicación que surgió como complemento del lenguaje Java para facilitar la compilación multiplataforma. La instalación con yum

```
yum install ant
```

deja el entorno de modo que el ejecutable `ant` está accesible para cualquier usuario y la variable `ANT_HOME` apunta al directorio `/usr/share/ant`

En Ant es necesario definir todas las operaciones que se quieren realizar, incluso las habituales, Estas tareas se definen en un fichero con estructura XML llamado `build.xml` y su tamaño suele ser elevado en proyectos grandes. Además, si se requiere especificar propiedades de compilación, puede crearse un archivo *build.properties* que se utilizará cuando se compile un proyecto.

Un ejemplo de tarea que borra el contenido de un directorio podría ser:

```
<target name="clean">
    <delete dir="build"/>
</target>
```

Para compilar un proyecto, debe introducirse en el terminal de comandos y situado sobre el directorio del proyecto, que a su vez contendrá el fichero `build.xml`

```
ant
```

Y para ejecutar tan sólo una tarea concreta:

```
ant <nombretarea>
```

Para más detalles, se remite al lector al manual de Ant [CSL01c]

Su uso como herramienta única de compilación cada vez es menor. Se requieren otras herramientas que hagan una mejor gestión de la información del proyecto, de modo que especificar las tareas sea un proceso mucho más rápido. No obstante, al ser muy extensible la comunidad ha generado una cantidad muy grande de *plugins* de modo que prácticamente se ha convertido en una *shell* más, con la peculiaridad de que usa XML como lenguaje.

4.4.2. Maven

Maven surge después de Ant para aportar mayor facilidad y claridad de uso, permitiendo mayor automatización de ciertas tareas y una mejor gestión de dependencias. No obstante, su uso en este proyecto se reducirá a utilizar los comandos propios de gestión de un repositorio Maven2, no así a especificar los proyectos de acuerdo a este software como se ha explicado anteriormente.

Su instalación no se realiza por medio de yum sino que se descarga la aplicación de la zona de descargas del sito de Maven [CSL02], y se extrae sobre un directorio que especifique el usuario.

Posteriormente se actualizan las variables de entorno adecuadas, en este caso, `MAVEN_HOME` y `PATH`:

```
MAVEN_HOME=/usr/local/maven
PATH=$MAVEN_HOME/bin:$PATH
export PATH MAVEN_HOME
```

Maven es una herramientas más ambiciosa que Ant: tiene unas tareas definidas por defecto y permite mayor uniformidad en la definición del proceso de

construcción. Su finalidad es más cercana a controlar la información del proyecto, no sólo de la construcción. Las tareas de Ant aquí se definen como objetivos *goals*.

Los objetivos se definen en un fichero XML que recibe el nombre de `maven.xml`, y que es opcional pues ya hay muchos objetivos definidos por defecto. Un objetivo se puede definir de esta forma:

```
<goal name="foobar-dist">
   <attainGoal name="war:install" />
</goal>
```

El fichero principal que se ejecuta por cada proyecto tiene por nombre `pom.xml` –*Project Object Manager*–. En este fichero se pueden definir:

- Propiedades del proyecto

```
<pomVersion>3</pomVersion>
   <id>maven_dummy_project</id>
   <artifactId>maven_dummy_project_demo</artifactId>
   <name>maven dummy project</name>
   <groupId>metaware</groupId>
```

```
   <currentVersion>0.0.1</currentVersion>
   ...
```

- Dependencias:

```
<dependency>
      <groupId>commons-beanutils</groupId>
      <artifactId>commons-beanutils</artifactId>
      <version>1.6.1</version>
      <type>jar</type>
      <properties>
         <war.bundle>true</war.bundle>
      </properties>
</dependency>
```

- Repositorios de los que tomará los artefactos que supongan dependencias, o donde dejará los artefactos que construya:

```
<repository>
   <connection>scm:cvs:pserver:anoncvs@cvs.apache.org:/home/
      cvspublic:maven-plugins/examples</connection>
   <url>http://cvs.apache.org/viewcvs/maven-plugins/examples/</
      url>
</repository>
```

Para ejecutar Maven, bastará con posicionarse sobre el directorio donde esté el proyecto y teclear:

```
mvn
```

Si se quiere ejecutar un objetivo concreto:

```
mvn <nombreobjetivo>
```

Las opciones de Maven son muy diversas y profundizar más en detalle, se alejaría del alcance de este trabajo, por lo que se recomienda consultar el manual oficial [CSL02].

En principio Ant y Maven son herramientas separadas, aunque existe una opción de combinar ambas mediante Antlib + Maven, dada la popularidad que adquirió ant y para facilitar la transición de una a otra herramienta. Hay también una referencia al uso de Ant + Maven en una sección del manual de maven, pero no se ha profundizado en estas opciones.

Ejemplos prácticos de uso y del método de trabajo con Maven se pueden encontrar en este libro [MO05] escrito a modo de cuaderno de notas. Cubre la primera versión de Maven pero para obtener una visión general sirve perfectamente. De la versión 2 de Maven hay dos libros editados: [vZCR08] y [CMPS06]

4.4.3. Ivy

En este artículo [Duv08] se explica bastante bien la esencia y la manera de trabajar con Ivy. Ivy es el software que se utilizará en este proyecto para especificar las dependencias. Es dependiente de Ant y su instalación es aún más sencilla. Simplemente se requiere descargar el programa del sitio oficial [CSL05a], extraer su contenido en cualquier directorio, y seleccionar después uno de los dos archivos .jar –el archivo cuyo nombre sea `ivy-<version>-incubating.jar`– y copiarlo al directorio `lib` de la aplicación Ant. Siguiendo la lógica de directorios que se ha empleado en este documento, supondría copiar este fichero sobre el directorio `usr/share/ant/lib`.

A partir de aquí, si al ejecutar ant sobre un directorio concreto se encuentra en el fichero `build.xml` una referencia a un fichero `ivy.xml`, Ant se encargará de llamar a Ivy y resolver las dependencias.

Esta herramienta era anteriormente parte de la empresa *Jayasoft*, pero ahora forma parte de Apache. Con el cambio se ha producido un proceso de adaptación de Ivy que ha provocado que en este momento –julio de 2008– sólo haya disponibles dos versiones beta.

Su uso no es autónomo, y va asociado a Ant. De este modo, en cada proyecto se requerirá un archivo `build.xml` como el visto anteriormente, y un nuevo fichero `ivy.xml` que gestione las dependencias. Para hacer llamadas desde el fichero de Ant al de Ivy y resolver las dependencias se utiliza el código:

```
<target name="resolve" description="--> retreive dependencies with
    ivy">
    <ivy:retrieve/>
```

```
</target>
```

Y en general, para cualquier llamada a una tarea de Ivy, sería:

```
<ivy:nombretarea/>
```

Pueden especificarse en el fichero `ivy.xml` las dependencias especificando el nombre, organización y revisión (versión):

```
<dependencies>
    <dependency org="apache" name="commons-lang" rev="2.1"/>
</dependencies>
```

Hay otras herramientas surgidas después de Ant y de Maven con funcionalidades similares pero que introducen algunas diferencias como el caso de Gant [Win07] que es un gestor de tareas Ant que usa Groovy como lenguaje, en lugar de XML.

4.4.4. Aplicación práctica: Manejo de dependencias Ivy con repositorio Maven2

Se trata de crear un repositorio de artefactos remoto y utilizado por Ivy durante la compilación, conectando la aplicación con uno o más servidores y permitiendo además la gestión de artefactos tanto por línea de comandos como mediante un interfaz web (Artifactory).

En primer lugar, hay que reseñar que los repositorios locales y de ibiblio vienen dados por defecto. Sólo es necesario definir los repositorios que vayamos a crear específicos para nuestro entorno de trabajo, en este caso el repositorio de los artefactos de la red en un servidor virtual creado para el efecto. Por ello, dado que se va a utilizar un repositorio Maven2, crearlo es el primer paso, aunque no hay una instrucción específica asociada pues no se crea ninguna estructura de directorios específica, es más un contenedor "ordenado" de artefactos.

Estos pasos se harán con Artifactory, que ayuda a gestionar el almacenamiento de artefactos e información asociada, y que aporta un interfaz web para realizar las opciones más habituales –ver ficheros, búsqueda de artefactos, cargar y descargar artefactos, permitir distintos accesos en función de usuario y realizar backups entre otras–.

Una vez puesto en marcha el servidor artifactory, ya se podrá abrir en un navegador la siguiente dirección:

```
http://repos-server:8080/artifactory/
```

Tras el intercambio de credenciales se verán unos subdirectorios creados por defecto, para que los se utilicen si se quiere mantener los nombres habituales de los repositorios –libs-releases, libs-snapshots, etc–. Se puede crear uno nuevo, o configurar la aplicación con uno de estos. En este caso se va a instar a Ivy a que utilice el repositorio remoto indicado por:

```
http://repos-server:8080/artifactory/libs-releases
```

Si se accede a esta dirección desde un navegador, se obtiene un error porque no hay plantilla asociada –se supone que para visualizar los artefactos, está la opción *Browse repository*–. Pero el repositorio concreto está configurado y se podrá indicar en los ficheros que utilice Ivy. En primer lugar, se edita el fichero que aparecerá en el directorio de instalación *de Maven*, en `./maven/conf/settings.xml`, para incluir la información del servidor asociado y los datos de acceso, dentro de las etiquetas `<servers>`:

```
<server>
  <id>myArtifRepository</id>
  <username>admin</username>
  <password>password</password>
</server>
```

Esto servirá, por tanto, para identificar el servidor sin tener que incluir datos de acceso en cada archivo de proyecto, pero también para poder ejecutar comandos de maven por línea de comandos. Posteriormente tras situarse en el directorio concreto del proyecto que se quiera compilar con Ivy (para hacer las pruebas se utilizó el proyecto de ejemplo que incluye Ivy en su instalación llamado `chainedresolvers-project`). Se añade en el fichero `build.xml`, además de las tareas creadas, dónde se encuentra el fichero de características `ivysettings.xml` que se va a utilizar: en este caso será sobre el directorio del proyecto:

```
<ivy:settings file="${ivy.settings.dir}/ivysettings.xml"/>
```

Además de la habitual y ya presentada antes tarea resolve:

```
<target name="resolve" description="--> retrieve dependencies with
    ivy">
   <ivy:retrieve/>
</target>
```

Después en ese fichero `ivysettings.xml` se pueden indicar los métodos para obtener los artefactos, por medio de las etiquetas *resolvers*. Se indica un repositorio para los artefactos y otro para las dependencias `ivy.xml` –en este caso el de los ficheros `.ivy` es local, pero podría ser de la misma forma remoto–, ya que al trabajar en un repositorio Maven2, los ficheros que Artifactory generará por defecto como "acompañantes" de los artefactos serán de tipo maven (pom.xml). Se incluye aquí el fichero completo:

```
<ivysettings>
  <settings defaultResolver="chain-example"/>
  <resolvers>
    <chain name="chain-example">
      <url name="exampleArtifactory" m2compatible="true">
```

```
        <ivy pattern="file://home/.../example/chained-resolvers/
            chainedresolvers-project/ivy-[module][revision].xml"/>
        <artifact pattern="http://repos-server:8080/artifactory/
            plugins-releases/[module]/[module]/[revision]/[artifact
            ]-[revision].[ext]"/>
      </url>
      <ibiblio name="ibiblio"/>
    </chain>
  </resolvers>
</ivysettings>
```

donde cabe destacar que se debe indicar que es maven2 compatible con el atributo *m2compatible* con valor *true*, y que los ficheros se buscan de acuerdo a un patrón dependiendo del nombre de organización, módulo, revisión, etc. Este convenio puede cambiar dependiendo del artefacto, pero es bueno fijarlo para que el repositorio sea uniforme y los artefactos puedan encontrarse siempre, evitando repeticiones. Además se indica que si no se encuentran en nuestro repositorio, se busquen en el repositorio público de ibiblio.

Si ahora se va al directorio del proyecto, y se teclea *ant* o *ant resolve*, el programa tratará de buscar en orden por los repositorios disponibles, primero en su directorio, luego en su repositorio local, luego en nuestro nuevo repositorio Maven y luego en ibiblio, hasta descargar los artefactos. En uno de los ejemplos más sencillos en que se probó, se descargó un fichero de un repositorio local, y otro desde Maven2, obteniendo un resultado como éste:

```
$ ant
Buildfile: build.xml

resolve:
[ivy:retrieve] :: Ivy 2.0.0-beta2-local-20080213132935 -
    20080213132935 :: http://ant.apache.org/ivy/ ::
[ivy:retrieve] No ivy:settings found for the default reference 'ivy.
    instance'. A default instance will be used
[ivy:retrieve] :: loading settings :: file = /home/carlinux/
    PruebasIvy/example/chained-resolvers/chainedresolvers-project/
    ivysettings.xml
[ivy:retrieve] :: resolving dependencies :: [ apache | chained-
    resolvers | working@tfc ]
[ivy:retrieve] confs: [default]
[ivy:retrieve] found [ apache | commons-lang | 2.0 ] in ibiblio
[ivy:retrieve] found [ apache | test | 1.0 ] in exampleArtifactory
[ivy:retrieve] downloading http://repo1.maven.org/maven2/commons-
    lang/commons-lang/2.0/commons-lang-2.0.jar ...
[ivy:retrieve] ..............................................
    (165kB)
[ivy:retrieve] .. (0kB)
[ivy:retrieve] [SUCCESSFUL ] [ commons-lang | commons-lang | 2.0 ]/
```

```
    commons-lang.jar[jar] (2437ms)
 [ivy:retrieve] downloading http://repos-server:8080/artifactory/
    plugins-releases@repo/test/test/1.0/test-1.0.jar ...
 [ivy:retrieve] .. (1kB)
 [ivy:retrieve] .. (0kB)
 [ivy:retrieve] [SUCCESSFUL ] [ apache | test | 1.0 ]/test.jar[jar]
    (23ms)
 [ivy:retrieve] :: resolution report ::
      ---------------------------------------------------------------------
      |           |            modules            ||   artifacts   |
      |   conf    | number| search|dwnlded|evicted|| number|dwnlded|
      ---------------------------------------------------------------------
      |  default |   2   |   2   |   0   |   0   ||   2   |   2   |
      ---------------------------------------------------------------------
[ivy:retrieve] :: retrieving :: [ apache | chained-resolvers ]
[ivy:retrieve] confs: [default]
[ivy:retrieve] 0 artifacts copied, 2 already retrieved

run:
   [mkdir] Created dir: /home/carlinux/PruebasIvy/example/chained-
      resolvers/chainedresolvers-project/build
   [javac] Compiling 1 source file to /home/carlinux/PruebasIvy/
      example/chained-resolvers/chainedresolvers-project/build
    [java] standard message :example world !
    [java] capitalized by org.apache.commons.lang.WordUtils : Example
        World !
    [java] upperCased by test.StringUtils : EXAMPLE WORLD !
```

Si se hubiera ejecutado con algún fichero en el repositorio local del ordenador, el resultado podría terminar como:

```
  [ivy:retrieve] :: resolution report ::
       ---------------------------------------------------------------------


       ---------------------------------------------------------------------


       ---------------------------------------------------------------------
 [ivy:retrieve] :: retrieving :: [apache | chained-resolvers]
 [ivy:retrieve] confs: [default]
 [ivy:retrieve] 1 artifacts copied, 1 already retrieved
```

Con la consiguiente mejora en tiempo, y sin tener que preocuparse en adelante de donde están los artefactos, ya que siempre estarán disponibles una vez usados al menos en el repositorio local.

Sobre un ejemplo de cómo funcionarían los comandos por terminal, para comunicarse con el servidor local, usando Maven, se aporta el siguiente ejemplo de cómo se subiría un artefacto con los datos introducidos:

```
mvn deploy:deploy-file -DrepositoryId=myArtifRepository \
     -Durl=http://repos-server:8080/artifactory/libs-releases@repo \
     -DgroupId=test -DartifactId=test -Dversion=1.0 -Dpackaging=jar
        \
     -Dfile=test.jar
```

Sobre las instrucciones de línea de comandos, pueden encontrarse dentro del manual de maven [CSL02], buscando por la instrucción concreta que se quiera ejecutar.

4.4.5. Conclusiones

En conclusión, el uso de Ant, Maven e Ivy supone una ayuda considerable para el proceso de compilación, aunque el tener que utilizar parte de las tres herramientas puede llevar a cierto desconocimiento inicial de los procesos a seguir y a ciertas dudas por parte de las personas que tengan que crear los ficheros en ant + ivy, utilizando comandos de maven si se desean subir los archivos manualmente.

No obstante el proceso de configuración de las herramientas implicadas no ha conllevado demasiadas complicaciones, más allá de establecer las primeras conexiones entre el servidor y las herramientas, por lo que superada esa fase el resto de procesos son mecánicos y en su mayoría se hacen a través de un interfaz web, con lo que sólo quedaría como proceso manual escribir los scripts de dependencias de Ivy.

4.5. Integración continua

Para implantar un sistema de Integración Continua, se debe planificar la frecuencia con la que deben compilarse y ejecutarse las pruebas de los proyectos. Este tiempo debe ser como mínimo de una vez al día, y se recomienda que sea mucho más a menudo (incluso es habitual que se ejecute cada vez que se detecten cambios en el repositorio).

El servidor –*build-server*– estará conectado con el repositorio del código –*repos-server*– de modo que en cada momento programado revisa si hay nuevos datos en el repositorio, y si es así compila el proyecto. Este proceso en general será visible para todos los usuarios de alguna forma –la mas habitual, por un interfaz web, aunque también incluso en una pantalla externa que pueda ver el equipo de desarrollo–, y desembocará en un resultado –éxito o fracaso de la compilación–.

Posteriormente ejecutará las pruebas automáticas, y si se pasan con éxito, esta rama principal es estable y puede seguir trabajándose en ella. En caso contrario, ha ocurrido un error, y puede volverse al código anterior, responsabilizando al desarrollador que incorporó los últimos cambios para que arregle el código en una rama propia y mantener la rama principal disponible para compilación y en pleno funcionamiento. Se le puede avisar por email, chat, rss, etc., a esa persona o a todas las que estén asignadas a un determinado proyecto.

También puede tratar de solucionarse el fallo en la rama principal durante un tiempo, pero la idea que hay debajo es que la rama principal esté casi siempre libre de bugs e incidencias. Este funcionamiento puede verse con explicaciones adicionales en la siguiente entrada de un blog [Unk06]. No obstante, hay que añadir que pese a servirnos de ellos en la explicación, el funcionamiento por ramas no es exclusivo de la integración continua.

Existe un número bastante elevado de herramientas de integración continua con bastantes opciones, por lo que finalmente se han tenido que elegir solo algunas para probar su funcionamiento. Sin embargo, aunque algunas no serán explicadas aquí, se incluirán referencias a todas las herramientas y sus características propias en la sección de comparativas, para que pueda elegirse la que mejor se adapte a las necesidades específicas.

4.5.1. CruiseControl

CruiseControl [TCSL04] es la herramienta con más tradición, y ya estaba presente en el primer artículo de Martin Fowler [Fow06] ya mencionado. Fue inicialmente construida para funcionamiento interno de la compañía ThoughtWorks, pero desde hace tiempo es una aplicación liberada a la comunidad. El proyecto sigue vivo, aunque la comunidad de usuarios está usando también otras herramientas que se están desarrollando de manera muy activa.

Para las pruebas se utilizó la versión 2.7.0. No fue requerido un proceso como tal de instalación, tan solo:

1. Descargar la aplicación en .tar.gz
2. Descomprimir sobre el directorio que pretendemos utilizar, por ejemplo en /usr/local/cruisecontrol
3. Ejecutar el archivo cruisecontrol.sh, dentro de la subcarpeta bin.

La aplicación web que maneja y configura el proyecto está basada en páginas jsp. Su apariencia no es muy vistosa, aunque genera informes estadísticos sobre el estado de las compilaciones que otras aplicaciones no incluyen. No es difícil de usar, e incluye otra aplicación más adaptada al usuario para visualizar el estado de las compilaciones (dashboard). Sin embargo, ninguna de las dos incluye gestión de usuarios, lo cual podría supone una desventaja en determinados entornos corporativos.

Para añadir un proyecto, debe modificarse su fichero `config.xml` de configuración, incluyendo la parte relativa a un nuevo proyecto:

```
<cruisecontrol>
 <project name="tfc">
  <listeners>
    <currentbuildstatuslistener file="logs/${project.name}/status.txt
        "/>
```

```
  </listeners>
  <bootstrappers>
    <svnbootstrapper localWorkingCopy="projects/${project.name}"/>
  </bootstrappers>
  <modificationset quietperiod="30">
    <svn RepositoryLocation="http://repos-server/repos/${project.name
        }/trunk"/>
  </modificationset>
  <schedule interval="600">
    <ant anthome="/usr/share/ant/" buildfile="projects/${project.name
        }/trunk/build.xml"/>
  </schedule>
  <log>
    <merge dir="projects/${project.name}/target/test-results"/>
  </log>
  <publishers>
    <onsuccess>
     <artifactspublisher dest="artifacts/${project.name}" file="
         projects/${project.name}/target/${project.name}.jar"/>
    </onsuccess>
  </publishers>
 </project>
</cruisecontrol>
```

Si se observa el fichero de configuración anterior se puede ver que se ha indicado la dirección del repositorio remoto –*repos-server*–.

A partir de ese momento, puede compilarse el proyecto eligiendo la opción build, o esperar a la compilación automática. Es una aplicación bastante robusta y sencillamente funciona. Se puedo obtener más información en la documentación que la comunidad vuelca en la página del proyecto [TCSL04] y también en este libro [Cla06] en el que se repasa todo el ciclo de construcción de software tomando CruiseControl como herramienta de apoyo.

La empresa originaria del proyecto ThoughtWorks ha puesto a disposición de la comunidad el paquete Buildix [Tho08a] que es una pequeña distribución de herramientas preconfiguradas que incluye:

- Subversion
- Trac
- CruiseControl
- Mingle –herramienta de gestión de proyectos–

El paquete se distribuye en formato LiveCD pero no se han podido incluir pruebas de la misma en este trabajo de modo que no se dispone de mayor información.

4.5.2. Continuum

Continuum [CSL03b] es una aplicación desarrollada anteriormente bajo el proyecto Maven pero que recientemente ha adquirido presencia propia dentro del espectro de proyectos de la Fundación Apache. Por ello cabe esperar que las expectativas sobre el potencial de este producto se vean materializadas y de hecho ya hay una versión estable tras muchos intentos con versiones alpha y beta. La versión que se ha probado es la versión 1.1. Puede verse un ejemplo de uso en vivo en [CSL03a].

No es necesario proceso de instalación para ejecutarlo como servidor autónomo. Por ello los pasos son los triviales:

1. Descargar la aplicación .zip
2. Descomprimir sobre el directorio que pretendemos utilizar, por ejemplo en /usr/local/continuum
3. Ejecutar el archivo run.sh, dentro de la subcarpeta bin, y de la carpeta con la arquitectura adecuada (linux-x86-32 o linux-x86-64).

Presenta una web bastante clara e intuitiva, con gestión de usuarios y la información bien jerarquizada. Aunque no incluye demasiadas opciones, lo que siempre se destaca sobre la aplicación es la comodidad y facilidad de uso.

Para añadir los proyectos, Continuum diferencia el tipo de proyecto en función del tipo de script de compilación utilizado (Ant, Maven 1.x, Maven 2.x). En el caso general, con scripts en Ant, se deberá añadir un nuevo proyecto con datos como estos:

```
Project Name*: tfc
Version*: 2.0
SCM Url*: scm:svn:http//repos-server/repos/tfc/trunk
```

El campo SCM Url es una convención que se utiliza para muchos sistemas de control de versiones, pero no es más que incluir la palabra scm, luego el tipo de sistema de control de versiones –en este caso Subversion– y seguidamente la dirección donde se encuentra el repositorio, que en este caso apunta a la máquina *repos-server*.

Una vez rellenados estos campos, la primera vez que se quiera compilar un proyecto la aplicación bajará los datos a una copia local, a partir de la que compilará en las siguientes ocasiones (obteniendo los nuevos datos del repositorio cuando estos se produzcan). Por ello la primera compilación puede retrasarse e incluso dar algún fallo inicial en el checkout, sin que implique fallos en la compilación.

Tras esto, se podrá consultar los datos del proyecto a partir de *Show Projects Groups*. pudiendo ver toda la estructura de archivos y sus contenidos –de forma similar a Trac, aunque mucho menos vistosa–. Entre las opciones del proyecto se podrá forzar la compilación, pulsando sobre *Build Now*, o comprobar el estado

de las compilaciones anteriores. Además destaca que todas las opciones están bastante claras e integradas, y se pueden añadir opciones como notificadores ante eventos.

El problema principal que se encontró durante la compilación es que al descargarse los contenidos, Continuum crea un directorio adicional que no está en la ruta, y que proporcionaba problemas con el script de compilación, pese a haber hecho uso de paths relativos. Por ello fue necesario modificar ligeramente el fichero `build.xml` creando uno nuevo para ser el utilizado por Continuum y manteniendo el anterior para las compilaciones manuales. Sin duda es una aplicación por mejorar, aunque es recomendada para iniciarse en las prácticas de integración continua y el respaldo de Apache es un factor importante a su favor.

4.5.3. Otras

Del resto de actores que han entrado en escena últimamente Hudson [CSL08d] es el que está obteniendo una mayor atención, por su extensibilidad, por su arquitectura, por su facilidad de uso ha conseguido en muy poco tiempo que la comunidad haya generado una gran cantidad de *plugins*. No se han realizado pruebas de esta herramienta pero todo apunta a que debe ser tenida en cuenta.

A continuación se incluyen referencias a páginas donde se comparan las fortalezas de las distintas herramientas disponibles de esta categoría, incluidas las seleccionadas en el trabajo:

- Comparación detallada entre Cruise Control, Continuum y Luntbuild [Duv06]: Concluyen que Continuum es el más fácil de usar, CruiseControl el más extendido y con gran comunidad.
- Comparación esquemática entre la mayoría de opciones disponibles [Tho08b].
- Otra comparación entre los cuatro sistemas principales de integración continua open source, al menos por uso y popularidad: Cruise control, Continuum, Luntbuild y Hudson [Sma06].

4.5.4. Integración

A medida que se van probando y analizando herramientas de todos los campos involucrados en el proyecto, se observa que se obtienen grandes cantidades de información en numerosas aplicaciones e informes. Tener disponible mucha información radicará en una mejora de los procesos que se ejecuten, pero siempre y cuando la información obtenida sea utilizada.

Para ello resultará útil que las herramientas tengan interfaces web claros e intuitivos, y no sólo salidas de terminal, pero puede que esto no sea suficiente, ya que si se consigue que la información que cada pieza de la arquitectura emite, pueda llegar a relacionarse, mediante algún mecanismo de integración, llámese

plugin o extensión, entonces se habrá conseguido un estadio mas cercano a la gestión del conocimiento.

Una posible solución de integración es tratar de reunir toda la información en una herramienta, aunque se permita acceder a otras herramientas para ver otro tipo de informes. Este apartado recoge esa idea e integra las herramientas de Integración Continua más comunes con Trac, el entorno visor de Subversion, aprovechando algunos *plugins* que se han encontrado.

Integrar Trac con CruiseControl

Se ha encontrado un *plugin* que integra CruiseControl con Trac, de modo que, una vez instalado, aparece una nueva pestaña en el aspecto habitual de Trac con la opción de CruiseControl. Se encuentra en el listado de *plugins* de Trac en su web oficial, y el propio *plugin* está en esta dirección [vL05].

El *plugin* que se descargó en estas pruebas era el perteneciente a la versión `0.1.3-py2.4.egg`. Se siguieron las instrucciones indicadas en la página oficial. No obstante se incluyen aquí con mayor nivel de detalle, para que su instalación no presente problemas:

1. Descargar TracCC-0.1.3-py2.X.egg y copiarlo en el subdirectorio *plugins* del directorio del entorno Trac concreto sobre el que se quiere aplicar el *plugin*.

2. Añadir una sección nueva llamada *cruisecontrol* al fichero de configuración trac.ini, que se encuentra en el subdirectorio conf. Esta sección debe contener algo similar a esto:

```
[cruisecontrol]
ccpath = /usr/{\dots}/cruisecontrol/logs/tfc
buildstatusfile = status.txt
xslfile = /usr/{\dots}/cruisecontrol/reporting/jsp/webcontent/
    xsl/buildresults.xsl
```

3. Añadir el permiso de acceso a la vista de CruiseControl a los usuarios correspondientes. Si deseamos hacer esto para el usuario genérico que no necesita login, bastaría con insertar la siguiente instrucción:

```
trac-admin /var/lib/trac permission add anonymous
   CRUISECONTROL_VIEW
```

4. Comprobar que la instalación de Trac está configurada para aceptar los `egg` de Python. Lo más habitual es que esta opción esté activada por defecto, por lo que se pudo omitir este paso.

Tras ejecutar estos pasos ya se puede contemplar, en la dirección del Trac local,

```
http://repos-server/trac
```

la nueva funcionalidad. El primer cambio que se observará será la pestaña *CruiseControl*. Este *plugin* añade la posibilidad de:

- Mostrar la información de las compilaciones y sus resultados, en la pestaña de Cruise Control.
- Mostrar los ficheros logs específicos de cada compilación.
- Las construcciones aparecen en la sección *Timeline* de Trac, junto a los otros cambios de código, tickets o modificaciones de wiki.

Se encontraron algunos problemas en la opción de ver los logs individuales de cada compilación, en lo que al parecer es un error del *plugin*. Además, es claramente insuficiente esta información respecto a la que proporciona CruiseControl, por lo que esta aplicación puede adoptarse para añadir información a Trac, pero no sustituye a toda la información que proporciona CruiseControl. Hay que hacer constar que este *plugin* no tiene demasiada actividad de desarrollo.

Integrar Trac con Continuum

El *plugin* que se utiliza para visualizar los resultados de Continuum en Trac se llama ContinuTrac. Sigue más o menos el funcionamiento y proceso de instalación del *plugin* de integración con Trac para CruiseControl, aunque no incluye la asignación de un permiso específico como CRUISECONTROL_VIEW.

Las instrucciones de instalación para la versión 0.0.1 utilizada (fichero *ContinuTrac-0.0.1-py2.4.egg*) son:

1. Traer los contenidos del servidor Subversion, con la instrucción:

```
svn co https://svn.rectang.com/continutrac
```

2. Ejecutar luego la instalación sobre python:

```
python setup.py bdist_egg
```

3. Copiar el fichero .egg creado en el subdirectorio dist dentro de la carpeta *plugins* del entorno creado por Trac para un proyecto concreto.

4. Añadir la siguiente sección al fichero trac.ini, dentro del subdirectorio conf del entorno de Trac:

```
[components]
continutrac.* = enabled
[continuum]
url = http://localhost:8000/
projects = 5
```

Cabe destacar que el puerto que se incluye en el fichero de configuración es el 8000, que es el puerto por defecto que utiliza Continuum para el interfaz de XMLRPC (no el puerto para el interfaz web, que por defecto es el 8080).

5. Comprobar que la instalación de Trac está configurada para aceptar los `egg` de python.

6. Reiniciar Trac y/o el servidor web.

Al arrancar de nuevo Trac se pudo comprobar que habían tomado correctamente los datos del *plugin*, y aparecía la nueva sección de Continuum, aunque aparece un error de conexión con el servidor de Continuum al tratar de acceder a los datos. Esto parece deberse a ejecutarlo en el modo de servidor dedicado (sin servidor web como Tomcat) o a problemas de autorización por necesitar los datos del usuario. En cualquier caso el proyecto lleva tiempo sin actividad y apenas hay documentación en la web sobre posibles errores con el programa, por lo que no hay guías ni listas de mensajes detalladas sobre como solucionar estos problemas.

Integrar Trac con Hudson

Aunque finalmente esta aplicación de integración continua no fue probada, se indica la referencia a la web oficial de Hudson [CSL08d] pues incluye un *plugin* para trabajar con Trac.

4.5.5. Conclusiones

Se han comentado a lo largo del apartado, las ventajas que tiene un sistema de integración continua. Como cualquier nueva implantación, exigirá de un pequeño esfuerzo inicial, ya que pese al progreso de las herramientas, y a que no debería ser un proceso diferente al de la compilación manual, los proyectos que compilaban manualmente no siempre compilan con éxito en el primer intento con estas herramientas. Si bien es cierto que como se ha destacado en actividades similares este esfuerzo sería sobre todo inicial, pues una vez implantado para cada proyecto no deberían surgir nuevos problemas, y la administración y consulta de los datos, al ser un proceso en interfaz web, no necesitaría prácticamente aprendizaje.

Al expandirse el uso de Trac han comenzado a surgir *plugins* que integran Trac con prácticamente cualquier herramienta. No obstante, su desarrollo está poco regulado oficialmente (desde los desarrolladores de Trac) y los *plugins* no tienen demasiado soporte, por lo que ante un fallo se encuentran pocas soluciones. No se tratan de proyectos muy activos, y si el desarrollador del *plugin*, generalmente único, lo abandona, puede generar incompatibilidades con otras versiones.

Por todo ello, se considera que los *plugins* pueden ser interesantes como complemento de información en Trac, pero si se busca la información completa y segura de las herramientas se recomienda usar sus informes de salida oficiales. Con éstas, por su mayor uso y desarrollo más organizado, se evitarán problemas

adicionales. Al menos por los *plugins* observados, ya que puede que éstos sigan evolucionando hasta obtener un mejor funcionamiento, por lo que se recomienda revisar la lista de *plugins* de Trac regularmente para actualizar esta información.

4.6. Servidor de despliegues

Se pretende conseguir un servidor –denominado deploy-server en este trabajo– que reciba las órdenes de despliegue de los artefactos en posibles servidores destino, que permita almacenar información acerca de acciones o eventos que deben realizarse para cada artefacto o tipo de artefacto, que notifique del resultado de dichas acciones y que sea fácil de usar y administrar.

No hay demasiadas herramientas en esta categoría para el lenguaje Java y las que hay están muy ligadas a los servidores de aplicaciones, como funcionalidad básica y sin grandes posibilidades de expansión ni separación. En algunos casos proporcionan algún API para facilitar el desarrollo mediante lenguaje Java de funcionalidades más complejas. Pero no se ha encontrado ningún caso de herramienta independiente específica para java con el objetivo de resolver este problema.

Quizá se deba a que en algunos servidores como el Tomcat o JBoss baste con colocar el artefacto en un directorio determinado para que aquéllos se den cuenta y activen un proceso de despliegue. Pero esto no resuelve, por ejemplo, un plan de contingencia para el caso de que lo desplegado contenga errores o una planificación de despliegue fuera del horario habitual, etc.

Algunos servidores comerciales proporcionan entornos basados en la combinación Java, Python, Jython y tareas Ant que permiten parar y arrancar el servidor, comprobar errores, planificar despliegues, acceder a Beans de monitorización mediante JMX, etc. En este sentido se ha encontrado alguna documentación en la que se explica cómo se pueden implementar tareas de Ant que contemplan algunas acciones de este tipo para servidores de aplicaciones libres como Tomcat.

Probablemente sea por la facilidad con que se puede extender Ant adaptándolo a las necesidades de cada proyecto, que no haya herramientas específicas en Java para esta categoría.

En donde sí se han encontrado piezas de software que puedan ayudar a conseguir estos objetivos es en otros lenguajes como Ruby o Python. Dado que estos lenguajes son de tipo *script* se puede pensar en estas herramientas como piezas de lenguaje de *shell* con cierto nivel de sofisticación.

4.6.1. Despliegues con tareas Ant

El artículo [Cha06] muestra cómo utilizar Ant para incorporar tareas de despliegue al `build.xml` de un proyecto de desarrollo. Un ejemplo, extraído de dicho artículo, de despliegue en varios hosts que se apoya sobre `rsync`:

```
<target name="deploy-live" depends="webapp">
```

```
 <!-- first, copy the live version of web.xml -->
 <copy file="${web.xml.live}" tofile="${webapp}/WEB-INF/web.xml"
    overwrite="yes"/>

 <!-- now, copy all the files -->
 <exec executable="${rsync}">
  <arg line=" --rsh=${ssh} -Cavz ${webapp.rsync}/* ${live.user}@${
      live.server}:${live.tomcat}/webapps/irv" />
 </exec>

 <!-- now recopy the old web.xml so the local version still works
    -->
 <copy file="${web.xml}" tofile="${webapp}/WEB-INF/web.xml"
    overwrite="yes" />
</target>
```

También se incluye un ejemplo de tarea de distribución:

```
 <target name="init">
  <property name="project" value="irv" />
  <tstamp/>
  <property name="now" value="${DSTAMP}-${TSTAMP}" />
  <property name="tarball.tar" value="${project}-${now}.tar" />
  <property name="tarball.tar.gz" value="${tarball.tar}.gz" />
...
 <target name="dist" depends="init">
  <tar tarfile="${build}/${tarball.tar}"
      basedir=".."
      includes="${project}/**"
      excludes="${project}/build/**, ${project}/test/**" />
  <gzip zipfile="${build}/${tarball.tar.gz}" src="${build}/${tarball.
     tar}" />
  <delete file="${build}/${tarball.tar}"/>
 </target>
```

Existen *plugins* de Ant para protocolos o herramientas como scp, http, Subversion, rsync, etc. lo que la convierte en una pieza de software muy potente.

4.6.2. Capistrano

Capistrano [Buc04] es un software desarrollado en lenguaje Ruby para el framework de desarrollo Ruby on Rails [Han06]. Ruby on Rails es un completo framework de desarrollo de aplicaciones web que utilizan bases de datos, sirviéndose del patrón arquitectural del Modelo-Vista-Controlador para organizar la información. Mediante este patrón se separa la representación –mediante plantillas– de los datos de modo que se facilita la extensibilidad y la limpieza de código. Está programado en Ruby y presenta peculiaridades respecto a otros lenguajes, que

son resumidas en 2 expresiones:

- *Don't repeat yourself* (No te repitas), hace referencia a que la integración de los módulos facilita que sólo haya que definir una vez lo que debe hacerse, lo que supone un ahorro considerable de código que deriva en un programa de código más legible y de fácil mantenimiento.
- *Convention over configuration* (Convención sobre configuración), se refiere a que sólo se define explícitamente lo que no es convencional. Cuando el comportamiento que se quiere asociar es el usual de nuevo se obtiene la ventaja de tener que escribir menos líneas de código. Para más información acerca del lenguaje y su filosofía se puede visitar la documentación de la wikipedia.

El software Capistrano fue diseñado para facilitar los despliegues de aplicaciones Ruby on Rails pero dado que es bastante configurable es posible hacer uso del mismo para distribuir artefactos java.

A la hora de instalar Capistrano y por tanto Ruby más Ruby on Rails, existen bastantes dependencias entre los programas, por lo que no todos se instalan por defecto con la facilidad que aporta el instalador yum. Se hace necesario seguir una secuencia similar a ésta en la instalación: primero Ruby, después RubyGems que es un gestor de paquetes o si se quiere artefactos Ruby, y finalmente Rails haciendo uso de RubyGems.

Si se tienen los repositorios adecuados configurados, la instalación de Ruby puede hacerse desde yum. La secuencia de paquetes necesaria es ésta:

```
yum install ruby ruby-devel ruby-libs ruby-irb ruby-rdoc ruby-mysql
```

Si se instala directamente Ruby (yum install ruby), y muestra las dependencias, comprobar que estén todas las indicadas. Puede que algo no sea requerido en primera instancia, pero seguramente sea de utilidad cuando se quiera instalar Ruby on Rails.

Es necesario descargar RubyGems [CSL07f] de RubyForge[CSL06c], y tras descomprimirla, sobre el directorio se ejecutaría la siguiente instrucción:

```
ruby setup.rb
```

Si se quieren cambiar las opciones de instalación por defecto se remite al lector a la guía de instalación oficial.

Si se ha instalado correctamente RubyGems, la instalación de Rails requiere solamente de una instrucción:

```
gem install rails --include-dependencies -y
```

Finalmente la instalación de Capistrano es similar a la de Rails

```
gem install -y capistrano
```

Veamos un ejemplo de script de Capistrano

```
task :scenario_a do
  set :deploy_to, "/path/to/scenario_a"
  role :war, "scenario_a.war"
  role :jar, "scenario_a.jar"
end

task :scenario_b do
  set :deploy_to, "/path/to/scenario_b"
  role :war, "scenario_b.war"
  role :jar, "scenario_b.jar"
end
```

Con Capistrano basta una sentencia del tipo:

```
cap scenario_a deploy
```

para desplegar un escenario de tipo predefinido cuya descripción reside en algún otro script.

4.6.3. Vlad the Deployer

Vlad the Deployer [hs07] surgió en el mundo Ruby como alternativa a Capistrano buscando sencillez de uso al pensar sus desarrolladores que Capistrano se estaba complicando en exceso. La instalación de este software vía RubyGems es fácil:

```
gem install vlad
```

Un ejemplo de script de despliegue:

```
set :application, "tfc"
set :domain, "dominio.com"
set :deploy_to, "/webapps/"
set :repository, 'http://repos-server/repos/tfc/branches/stable/'
```

4.6.4. Fabric

Fabric [Han07] es una propuesta similar a las anteriores de Ruby solo que para el lenguaje Python. Un ejemplo de uso:

```
set(
  project = 'tfc',
  fab_hosts = ['deploy-server.cluster.com', 'deploy-server2.cluster.
      com'],
)

def deploy():
  "Compilar y desplegar."
```

```
local('mvn tfc')
put('webapps/$(project).war', '$(project).war')
put('instalar.sh', 'instalar.sh')
sudo('instalar.sh')
```

Para lanzar el script anterior bastaría con teclear

```
fab deploy
```

4.6.5. Conclusiones

De las herramientas mostradas, Capistrano sería la que más se podría adaptar para poder conseguir los objetivos propuestos, ya que es la más madura y flexible. Sin embargo, la potencia de Ant, el hecho de que su entorno natural es Java y de que la mayoría de los proyectos ya la utilizan para los procesos de construcción, la convierten en la candidata más aconsejable.

4.7. Información analítica

La utilidad principal de un sistema de control de versiones es facilitar la gestión de un proyecto, permitiendo almacenar de forma centralizada o distribuida el código fuente, y posibilitando en última instancia el trabajo concurrente de grandes equipos. Cada actualización o *commit* que se realiza sobre el servidor queda registrado y asociado a diferentes parámetros: nombre de usuario, fecha, archivo modificado, versión, líneas introducidas, etc. para que los desarrolladores puedan conocer los últimos cambios y la razón de estos. Toda esta información que se utiliza para esta finalidad puede ser interesante también desde el punto de vista estadístico.

Existe otra categoría de herramientas que aprovechan esos datos, y permiten categorizar la información en función de, entre otros:

- Líneas de código añadidas o modificadas por autor, fecha, archivo.
- Archivos del proyecto con más modificaciones.
- Evolución del número y tamaño de los archivos.
- Comparación entre líneas de código existentes y modificadas.

En el conjunto del software libre también existen herramientas de este tipo que permiten obtener todas las estadísticas mencionadas con unos pasos sencillos.

4.7.1. StatSVN

La herramienta más utilizada tanto por realizar informes muy completos, como por tener un mayor soporte y documentación, es StatSVN [App06]. Esta herramienta hereda de StatCvs [App02], que también aporta información similar para proyectos en CVS.

Es una herramienta de código libre, multiplataforma y está desarrollada en Java. Se ejecuta desde un archivo .jar y no necesita instalación. A fecha de Julio de 2008, la versión operativa 0.4.1, con la que se trabaja aun es beta. Los pasos de instalación y funcionamiento también pueden ser consultados en la wiki oficial.

Es una herramienta que funciona de forma muy sencilla con su comportamiento por defecto. Estos serían los pasos para generar todos los informes de un repositorio SVN:

- Sobre el directorio que actuará de working copy, descargar los datos del repositorio Subversion.

```
svn co file:///path/to/repos
```

- Sobre el directorio (entrando en /repos, en este caso), crear un archivo log con la información (el fichero que se utilizará para generar las gráficas):

```
svn log --xml -v > svn.log
```

- Sobre el directorio en el que se encuentre StatsSvn (o sobre el directorio el que deseemos almacenar la información web, pero debemos hacer una copia del fichero statsvn.jar sobre este nuevo directorio), ejecutar:

```
java -jar statsvn.jar /path/to/repos/svn.log /path/to/repos/
```

Hay ejemplos en la página oficial de las gráficas que se pueden obtener como resultado de inspeccionar los repositorios de fuentes de los proyectos.

4.7.2. Otras

Se ha encontrado información de otras herramientas como MPY SVN STATS [CSL07c], aunque sus resultados son menos completos. Incluye información relativa al número de commits, cambios de paths y longitud de mensajes de log ordenados por usuario y mostrando información por fechas. No incluye prácticamente ninguna información relativa al propio proyecto y por tanto tampoco a las ramas de desarrollo, por lo que sólo es válido para obtener información rápida de los usuarios. No obstante, si se quiere una solución que no tenga un coste tan elevado en cuanto a espacio almacenado, ésta puede ser una solución aceptable.

Carnarvon [CSL06a] ofrece una gran variedad de estadísticas y gráficas, pero parece que el proyecto no tiene actividad desde hace tiempo.

4.7.3. Conclusiones

No se han encontrado demasiados proyectos dedicados específicamente a obtener estadísticas. En algunas herramientas, como *plugins* o interfaces para repositorios, se puede obtener información analítica como funcionalidad complementaria.

También hay servicios online que proporcionan estadísticas de proyectos. Basta con registrar el proyecto e incluir la ruta al repositorio de fuentes del proyecto, siempre y cuando éste se encuentre accesible por internet. Presentan gráficas de número de líneas de código, de lenguajes, de nivel de actividad, etc. Y también permiten comparar proyectos entre si, y hacer búsquedas en los códigos fuente, y en este aspecto se puede decir que adquieren la categoría de herramientas de gestión del conocimiento. Son sitios como Ohloh [Ohl06], Krugle [Kru06], o Koders [Kod06]. Puede resultar interesante seguir su evolución.

Capítulo 5

RESULTADOS Y CONCLUSIONES

El camino hasta llegar aquí ha sido largo y apasionante: se partía de la idea de buscar una solución integral para el trabajo diario, cíclico, hasta rutinario si se quiere, de la fase que comienza en el momento en que el programador decide que los cambios realizados en una copia de trabajo son válidos y activa la tarea de *commit*.

Esta fase, que se ha denominado a lo largo del proyecto *ciclo de vida del proceso de construcción de software*, adquiere especial relevancia cuando se trata de proyectos en los que intervienen equipos de desarrollo distribuidos, hecho que cada vez es más frecuente. Este proceso es determinante por cuanto afecta a aspectos clave como la productividad, la planificación de puesta en producción, la calidad del software generado, la detección temprana de errores, la coordinación de fuerzas de desarrollo, etc.

En este trabajo se han presentado propuestas para cada bloque del ciclo, se han instalado las mejor valoradas por los usuarios, y en algunos casos, las que seguían de cerca a aquéllas en potencial. Se han comparado entre sí y se ha verificado su funcionamiento con proyectos descargados de forjas de Software Libre en Internet.

También se ha llegado en algún bloque a caminos cortados, o cuando menos susceptibles de revisión para el futuro, como el caso de la gestión centralizada de los despliegues.

En resumen, ha habido un esfuerzo considerable de trabajo de campo, combinado con toma de decisiones continuada, como si del paradigma de Integración Continua introducido por Fowler se tratara [Fow06]: descargar, instalar, configurar, probar, descartar, integrar, vuelta a empezar.

La calidad de las herramientas de Software Libre es contrastada y reconocida, tanto por los fabricantes de software del modelo de negocio clásico, como de corporaciones de tradición conservadora, tanto que en el caso de aquéllas están patrocinando los propios proyectos de Software Libre, y en el caso de éstas últimas, están acometiendo renovaciones o migraciones haciendo uso a su vez de Software Libre. En el caso de las herramientas probadas durante el estudio, se ha observado también este patrón de calidad.

En lo que se refiere al modelo de arquitectura propuesto, se ha comprobado que es viable disponer de varias máquinas dedicadas a etapas distintas del proceso, ya que no ha supuesto en ningún momento impedimento alguno el hecho de que no residan las distintas piezas en la misma máquina. También se ha constatado que es una buena práctica tener físicamente segregadas las competencias.

La utilización de las técnicas de virtualización para poder aprovisionar estos entornos de pruebas ha sido providencial: hace tan solo dos años no habría sido tan sencillo hacer un montaje similar ni por equipamiento, ni por tecnología. Obviamente en un entorno de producción real habría que disponer de más equipamiento hardware, pero en el caso del software bastaría con redimensionar los parámetros de las máquinas virtuales creadas y activarlas en hardware más apropiado y estarían aptas para funcionar con algunos pequeños ajustes.

La idea que subyace es que sería factible y no demasiado complicado, generar con todo lo analizado a lo largo de este trabajo, una distribución de herramientas pre-configuradas, en formato imagen virtualizada de sistema operativo o *virtual appliance*, del tipo *instalar y listo para funcionar*.

Finalmente habría que mencionar a las metodologías ágiles de desarrollo, que combinadas con herramientas de última generación como las que se han evaluado, han aportado frescura y empuje a la Ingeniería del Software, y han propiciado la aparición de iniciativas como la indexación de los códigos fuente de los proyectos de Software Libre para facilitar su búsqueda y poder obtener información valiosa, información que se convierte en conocimiento cuando cae en manos de grupos de investigación como LibreSoft [GSy03].

El conocimiento fluye, y esto siempre es una buena noticia, porque genera más conocimiento.

Bibliografía

Libros

[Cla06] Mike Clark. *Pragmatic Project Automation*. Pragmatic Bookshelf, 2006.

[CMPS06] John Casey, Vincent Massol, Brett Porter, y Carlos Sanchez. *Better Builds with Maven*. DevZuz, 2006. http://www.exist.com/?q=node/151.

[CSFP04] Ben Collins-Sussman, Brian W. Fitzpatrick, y C. Michael Pilato. *Version Control with Subversion*. O'Reilly, 2004. http://svnbook.red-bean.com/.

[ea99] Per Cederqvist et al. *Version Management with CVS*. Ximbiot, 1999. http://ximbiot.com/cvs/manual/.

[FB03] Karl Fogel y Moshe Bar. *Open Source Development with CVS*. Paraglyph Press, 2003. http://cvsbook.red-bean.com/OSDevWithCVS_3E.pdf.

[GP05] Marc Gibert y Álvaro Peña. *Ingeniería del software en entornos de SL*. Universitat Oberta Catalunya, 2005.

[Mas05] Mike Mason. *Pragmatic Version Control using Subversion*. Pragmatic Bookshelf, 2005.

[MO05] Vincent Massol y Tim O'Brien. *Maven: A Developer's Notebook*. O'Reilly, 2005.

[Roo05] Garret Rooney. *Practical Subversion*. Apress, 2005.

[vZCR08] Jason van Zyl, John Casey, y Eric Redmond. *Maven: The Definitive Guide*. Sonatype, 2008. http://www.sonatype.com/book/pdf/maven-definitive-guide.pdf.

Recursos online

[App02] Appendium. Statcvs. 2002. http://statcvs.sourceforge.net/.

[App06] Appendium. Statsvn. 2006. http://www.statsvn.org/.

[Bra06] Espen Braastad. Management of high availability services using virtualization. Master's thesis, University of Oslo, 2006. http://research.iu.hio.no/theses/pdf/master2006/espenbraastad_thesis.pdf.

[Buc04] Jamis Buck. Capistrano. 2004. http://capify.org/.

[Cha06] Alex Chaffee. Building with ant: Deployment and distribution. 2006. http://www.developer.com/java/other/article.php/998241.

[Chi07] Deng Ching. The hype about repository managers. 2007. http://blogs.exist.com/oching/2007/11/05/the-hype-about-repository-managers/.

[CP04] CentOS-Project. CentOS - The Community ENterprise Operating System. 2004. http://www.centos.org.

[CP08] CentOS-Project. Howtos/freenx. 2008. http://wiki.centos.org/HowTos/FreeNX.

[Cse05] Tamas Cservenak. Proximity. 2005. http://proximity.abstracthorizon.org/px1/.

[CSL89] Comunidad-Software-Libre. Cvs. 1989. http://www.ximbiot.com.

[CSL98a] Comunidad-Software-Libre. Bugzilla. 1998. http://www.bugzilla.org/.

[CSL98b] Comunidad-Software-Libre. Comprehensive perl archive network. 1998. www.cpan.org.

[CSL98c] Comunidad-Software-Libre. CTAN - The Comprehensive TeX Archive Network. 1998. www.ctan.org.

[CSL99] Comunidad-Software-Libre. Python package index. 1999. pypi.python.org/pypi.

[CSL01a] Comunidad-Software-Libre. Perl documentation. 2001. http://perldoc.perl.org/perlnewmod.html.

[CSL01b] Comunidad-Software-Libre. Subversion. 2001. http://subversion.tigris.org.

[CSL01c] Comunidad-Software-Libre. The Apache Ant Project. 2001. http://ant.apache.org.

[CSL02] Comunidad-Software-Libre. Apache Maven Project. 2002. http://maven.apache.org.

[CSL03a] Comunidad-Software-Libre. Apache Continuum. 2003. http://vmbuild.apache.org/continuum/.

[CSL03b] Comunidad-Software-Libre. Apache Continuum. 2003. http://continuum.apache.org.

[CSL03c] Comunidad-Software-Libre. Xen hypervisor. 2003. http://www.xen.org.

[CSL04] Comunidad-Software-Libre. Darcs. 2004. http://darcs.net/.

[CSL05a] Comunidad-Software-Libre. Ivy - the agile dependency manager. 2005. http://ant.apache.org/ivy/.

[CSL05b] Comunidad-Software-Libre. Maven 2 repository. 2005. http://mirrors.ibiblio.org/pub/mirrors/maven2/.

[CSL06a] Comunidad-Software-Libre. Carnarvon. 2006. http://carnarvon.tigris.org.

[CSL06b] Comunidad-Software-Libre. Maven repository search/browse/explore. 2006. http://www.mvnrepository.com/.

[CSL06c] Comunidad-Software-Libre. Rubygems. 2006. http://rubygems.rubyforge.org/.

[CSL06d] Comunidad-Software-Libre. The MacPorts Project. 2006. http://trac.macports.org/browser/.

[CSL07a] Comunidad-Software-Libre. Apache archiva. 2007. http://archiva.apache.org.

[CSL07b] Comunidad-Software-Libre. Mercurial. 2007. http://www.selenic.com/mercurial.

[CSL07c] Comunidad-Software-Libre. MPY SVN STATS. 2007. http://mpy-svn-stats.berlios.de/.

[CSL07d] Comunidad-Software-Libre. Nexus. 2007. http://nexus.sonatype.org.

[CSL07e] Comunidad-Software-Libre. Pythoneggs. 2007. http://peak.telecommunity.com/DevCenter/PythonEggs.

[CSL07f] Comunidad-Software-Libre. Rubygems. 2007. http://www.rubygems.org/.

[CSL08a] Comunidad-Software-Libre. Bazaar version control. 2008. http://bazaar-vcs.org/.

[CSL08b] Comunidad-Software-Libre. Better scm initiative. 2008. http://better-scm.berlios.de/.

[CSL08c] Comunidad-Software-Libre. Freenx - the free nx. 2008. http://freenx.berlios.de/.

[CSL08d] Comunidad-Software-Libre. Hudson. 2008. https://hudson.dev.java.net/Hudson.

[Duv06] Paul Duvall. Choosing a Continuous Integration server. 2006. http://www.ibm.com/developerworks/java/library/j-ap09056/index.html.

[Duv08] Paul Duvall. Automation for the people: Manage dependencies with Ivy. 2008. http://www.ibm.com/developerworks/library/j-ap05068/index.html.

[ES08] Edgewall-Software. Trac. 2008. http://trac.edgewall.org.

[Fow06] Martin Fowler. Continuous integration. Mayo 2006. http://www.martinfowler.com/articles/continuousIntegration.html.

[Glo08] Exist Global. Maestro. 2008. http://www.mergere.com/maestro.html.

[GNU08] GNU.org. Licencias gnu. 2008. http://www.gnu.org/licenses/licenses.es.html#GPL.

[GSy03] GSyC. Libresoft - We study Libre Software. 2003. http://www.libresoft.es.

[Han06] David Heinemeier Hansson. Ruby on Rails. 2006. http://www.rubyonrails.org/.

[Han07] Christian Vest Hansen. Fabric: simple pythonic deployment. 2007. http://www.nongnu.org/fab/.

[hs07] Ruby hit squad. Vlad the deployer. 2007. http://rubyhitsquad.com/Vlad_the_Deployer.html.

[Jfr06] Jfrog.org. Artifactory. 2006. http://www.jfrog.org/sites/artifactory/latest/.

[Kao03] Chia-Liang Kao. Svk. 2003. http://svk.elixus.org.

[Kod06] Koders. Koders. 2006. http://www.koders.com/.

[Kru06] Krugle. Krugle. 2006. http://www.krugle.com.

[Man07] Avneet Mangat. Setting Up a Maven Repository. 2007. http://www.theserverside.com/tt/articles/article.tss?l=SettingUpMavenRepository.

[NoM08] NoMachine. NoMachine. 2008. http://www.nomachine.com/.

[Ohl06] Ohloh. Ohloh. 2006. http://www.ohloh.net.

[Sma06] John Ferguson Smart. Which open source CI tool is best suited for your application's environment? 2006. http://www.javaworld.com/javaworld/jw-11-2006/jw-1101-ci.html.

[TCSL04] ThoughtWorks y Comunidad-Software-Libre. CruiseControl. 2004. http://cruisecontrol.sourceforge.net/.

[Tho08a] ThoughtWorks. Buildix. 2008. http://buildix.thoughtworks.com/.

[Tho08b] ThoughtWorks. Ci feature matrix. 2008. http://confluence.public.thoughtworks.org/display/CC/CI+Feature+Matrix.

[Unk06] Unkasoft. Integración continua y gestión de la configuración. 2006. http://eskasiunblog.blogspot.com/2006/04/integracin-continua-y-gestin-de-la.html.

[vL05] Tammo van Lessen. Cruisecontrol plugin for trac. 2005. https://oss.werkbold.de/trac-cc/.

[Wik08a] Wikipedia. Tecnología nx. 2008. http://es.wikipedia.org/wiki/Tecnología_NX.

[Wik08b] Wikipedia. Virtual network computing. 2008. http://es.wikipedia.org/wiki/Virtual_Network_Computing.

[Wil97] Ken Williams. Creating (and maintaining) perl modules. 1997. http://mathforum.org/~ken/perl_modules.html.

[Win07] Russel Winder. Gant. 2007. http://gant.codehaus.org.

Apéndice A

CONFIGURACIONES EN DETALLE

A.1. Configuración Subversion con Apache

1. Añadir usuario y configurar directorio del repositorio

```
mkdir /svnroot
groupadd svn
useradd -g svn -d /svnroot -s /sbin/nologin svn
chgrp -R svn /svnroot
chmod o-rwx /svnroot
chmod ug+rwx /svnroot
```

2. Crear el repositorio

```
mkdir -p /svnroot/repos
svnadmin create /svnroot/repos
```

3. Permisos en el repositorio

```
chown -R svn:svn /svnroot/repos
chmod -R g+w /svnroot/repos
chmod -R o-rwx /svnroot
```

4. Para que apache tenga acceso a los repositorios

```
usermod -G apache,svn apache
```

5. Configurar el servicio en bajo el dominio de `xinetd.d` Añadir la línea `groups = yes` a la sección default de `xinetd.conf`

```
#
# Simple configuration file for xinetd
#
# Some defaults, and include /etc/xinetd.d/
```

```
defaults
{
       instances      = 60
       log_type       = SYSLOG authpriv
       log_on_success = HOST PID
       log_on_failure = HOST
       cps            = 25 30
       groups         = yes
}

includedir /etc/xinetd.d
```

6. Añadir el servicio al directorio `/etc/xinetd.d`. El contenido del fichero `/etc/xinetd.d/svn` debe ser el siguiente:

```
service svn
{
       socket_type    = stream
       protocol       = tcp
       user           = svn
       group          = svn
       umask          = 002
       wait           = no
       disable        = no
       server         = /usr/bin/svnserve
       server_args    = -i -r /svnroot
       port           = 3690
}
```

7. Rearrancar el servicio xinetd

```
service xinetd restart
```

8. Contenido de `/etc/httpd/conf.d/subversion.conf`

```
# Needed to do Subversion Apache server.
LoadModule dav_svn_module  modules/mod_dav_svn.so

# Only needed if you decide to do "per-directory" access control
    .
LoadModule authz_svn_module modules/mod_authz_svn.so

<Location /repos>
   DAV svn
   SVNParentPath /svnroot
```

```
   # Limit write permission to list of valid users.
   <LimitExcept GET PROPFIND OPTIONS REPORT>
      AuthType Basic
      AuthName "Repositorio Subversion"
      AuthUserFile /path/to/passwdfile
      Require valid-user
   </LimitExcept>
</Location>
```

A.2. Configuración Trac con Apache

1. Añadir a `/etc/httpd/conf.d/subversion.conf`

```
Alias /trac "/var/lib/trac"
<Directory "/var/lib/trac">
    Options Indexes FollowSymLinks
    AllowOverride None
    Order allow,deny
    Allow from all

    # mod_python speeds things up considerably
    SetHandler mod_python
    PythonInterpreter main_interpreter
    PythonHandler trac.web.modpython_frontend
    PythonOption TracEnv "/var/lib/trac"
    PythonOption TracUriRoot "/trac"

    # authentication
    AuthType basic
    AuthName "Trac auth"
    Require valid-user
    # authorization is handled internally by trac
</Directory>
```

Apéndice B

DOCUMENTACIÓN

B.1. Contenido del CD

El CD contiene el fichero empaquetado `CGS-TFC.tgz` que a su vez incluye:

```
CGS-TFC-books.bib
CGS-TFC-web.bib
CGS-TFC.tex
CGS-TFC.pdf
Makefile
README
images/
       fig2_1.png
       fig2_2.png
       fig2_3.png
       fig2_4.png
       fig3_1.png
       fig3_2.png
       fig3_3.png
       fig2_1.svg
       fig2_2.svg
       fig2_3.svg
       fig2_4.svg
       fig3_1.svg
       fig3_2.svg
       fig3_3.svg
       FacInformatica_new.eps
       FacInformatica_new.pdf
       cc-by-nc-nd.png
tex/
       alphaurl-es.bst
```

B.2. Cómo se hizo...

Se incluyen a continuación las primeras líneas del focumento TEX que da origen a este documento final. Aunque el fichero completo se puede obtener del CD, se ha considerado que puede ser útil, para futuros redactores de Trabajos de Fin de Carrera, tener visibles los trucos de la mágia del LATEX

```
\documentclass[a4paper,12pt,twoside,openright]{report}

%%%
% Idioma, ajustes de pagina, enlaces...
%%%

\usepackage{graphicx}
\usepackage[spanish,es-noindentfirst]{babel}
\usepackage[utf8x]{inputenc}
\usepackage[T1]{fontenc}
\usepackage{pslatex}
\usepackage{palatino}
\usepackage[left=4cm,top=3.5cm,right=2cm,bottom=3.5cm]{geometry}
\setlength{\textwidth}{15cm}
\setlength{\textheight}{22.5cm}
\graphicspath{{.}{images/}}
\usepackage{bibtopic} % bibliografias agrupadas por concepto
\usepackage{hyperxmp} % para incluir metadatos en el pdf
\usepackage[pdftex,unicode,citecolor=blue,colorlinks=true,linkcolor=
    blue,urlcolor=blue]{hyperref}

%%%
% Metadatos
%%%

\newcommand{\licencia}{
Esta obra est\'a bajo la licencia de Creative Commons Reconocimiento-
    No comercial-Sin obras derivadas 2.5 Espa\~{n}a.
}
\newcommand{\urllicencia}{http://creativecommons.org/licenses/by-nc-
    nd/2.5/es}

\hypersetup{
pdftitle={Generaci\'on, gesti\'on y distribuci\'on de artefactos Java
     con t\'ecnicas de Integraci\'on Continua y Software Libre},
pdfauthor={Carlos Gonz\'alez S\'anchez <karlhangas@gmail.com>},
pdfsubject={Proceso de construcci\'on de software},
pdfkeywords={Integraci\'on Continua, Software Libre, Creative Commons
    },
```

```
pdfcopyright={\licencia},
pdflicenseurl={\urllicencia}
}

%%%
% Listados de codigo
%%%

\usepackage{color}
\definecolor{light-gray}{gray}{0.95}
\definecolor{lighter-gray}{gray}{0.85}

\usepackage{listings}
\lstset{
  basicstyle=\small\ttfamily,
  commentstyle=\itshape,
  extendedchars,
  showstringspaces=false,
  breaklines,
%  postbreak=\usefont{U}{pzd}{m}{n}\char"E5,
  columns=flexible,
  frame=tlrb,
  frameround=tttt,
  backgroundcolor=\color{light-gray},
  xleftmargin=20pt,
  xrightmargin=20pt,
  fontadjust=true}

%%%
% Cabeceras
%%%
\usepackage{fancyhdr}
\pagestyle{fancy}
% para que las cabeceras muestren el capitulo y seccion seguida del
    numeral solamente
\renewcommand{\chaptermark}[1]{\markboth{\thechapter.\ #1}{}}
\renewcommand{\sectionmark}[1]{\markright{\thesection.\ #1}}
% para que las paginas impares vacias no tengan cabecera ni pie
\makeatletter
\def\cleardoublepage{\clearpage\if@twoside \ifodd\c@page\else
\hbox{}
\vspace*{\fill}
\vspace{\fill}
\thispagestyle{empty}
\newpage
\if@twocolumn\hbox{}\newpage\fi\fi\fi}
\makeatother
```

```
\begin{document}

%%%
% Portada
%%%

\begin{titlepage}
\vbox to 0.5cm{
          \hskip0.75cm\llap{\includegraphics[scale=0.1]{
              FacInformatica_new.pdf}}
          \vskip 0.2cm
          \hskip-0.75cm \textsf{\textbf{FACULTAD DE ÁINFORMTICA}}
          \vskip -0.1cm
          \hskip-0.75cm \textsf{\footnotesize UNIVERSIDAD ÉPOLITCNICA
               DE MADRID}
          }
\begin{center}
\vspace*{1.8in}
{\LARGE UNIVERSIDAD POLITECNICA DE MADRID}
\par
{\Large FACULTAD DE INFORMATICA}
\par
\vspace{1.0in}
{\Large TRABAJO FIN DE CARRERA}
\par
\vspace{0.25in}
{\large GENERACION, GESTION Y DISTRIBUCION DE ARTEFACTOS JAVA CON
    TECNICAS DE INTEGRACION CONTINUA Y SOFTWARE LIBRE}
\end{center}

\vspace{3.2in}
\hskip-0.75cm AUTOR: Carlos Gonzalez Sanchez
\vskip 0.1cm
\hskip-0.75cm TUTOR: Francisco Gisbert Canto

\end{titlepage}

...
```

B.3. Herramientas utilizadas

Todas las herramientas que se describen a continuación son Software Libre.

B.3.1. LaTeX

Se ha usado la distribución TeXLive 2007 para la confección de este documento. El paquete `babel` y subpaquete `spanish` que combinados proporcionan funcionalidades propias de idioma español, se ha actualizado a la última versión disponible en CTAN [CSL98c] con respecto a la que traía la distribución del 2007, de modo que ha habido que regenerar las macros y reconstruir los formatos. Esto se ha hecho con los comandos:

```
export TeX-DIST=/usr/local/texlive/2007/texmf-dist
cd $TeX-DIST/source/generic/babel
tex babel.ins
mv *.def $TeX-DIST/tex/generic/babel
mv *.ldf $TeX-DIST/tex/generic/babel
mv *.sty $TeX-DIST/tex/generic/babel
mv *.fd $TeX-DIST/tex/generic/babel
fmtutil --all
```

Esta actualización permite, entre otras cosas, que el primer párrafo justo al inicio de una sección o capítulo no aparezca *indentado* gracias a una directiva introducida en la declaración del paquete.

Para crear la bibliografía se ha usado una versión ligeramente modificada del estilo `alphaurl.bst` que viene con la distribución mencionada, y que traduce al castellano algunas palabras clave utilizadas. El fichero se llama `alphaurl-es.bst` y se incluye en el CD. Seguramente hay maneras mejores de hacer esto con el paquete `babel` pero se ha optado por la *vía rápida* dada la escasa relevancia que tiene. Se incluye el fichero únicamente para facilitar la generación de la documentación en formato pdf sin errores.

B.3.2. TeXShop

Un excelente editor de documentos LaTeX para la plataforma Mac que viene con la distribución TeXLive de Mac. Si se opta por descargar de manera independiente esta es la página oficial:

http://www.uoregon.edu/~koch/texshop

B.3.3. BibDesk

Una herramienta que gestiona las entradas bibliográficas para su inclusión y referenciado en documentos LaTeX utilizando formato BibTeX. Disponible también en la distribución TeXLive de Mac y su página base es:

http://bibdesk.sourceforge.net

B.3.4. Inkscape

De la página oficial: http://www.inkscape.org

Inkscape es un editor de gráficos vectoriales de código abierto, con capacidades similares a Illustrator, Freehand, CorelDraw o Xara X, usando el estándar de la W3C: el formato de archivo Scalable Vector Graphics (SVG).

B.4. Licencia