

Programador de tareas en código abierto

PLANIFICADOR DE TAREAS A LA CARTA

La planificación de tareas puede ser algo verdaderamente tedioso, sobre todo cuando se ha de realizar sobre un conjunto de máquinas. Presentamos una herramienta que nos lo pone mucho más fácil.

POR JAMES MOHR

Algo muy necesario para cualquier administrador de sistemas es poder realizar una tarea determinada en un momento dado o a intervalos regulares. El demonio *cron* proporciona un método sencillo para la programación de tareas en sistemas basados en Unix. Y a pesar de que *cron* ha sido mejorado a lo largo de los años, incluso las versiones más nuevas están diseñadas para la realización de planificaciones muy simples. Un administrador con unos propósitos inusuales tendrá que crear un nuevo script o incluir la funcionalidad adicional en sus scripts de *cron*.

Nos ahorraríamos mucho tiempo si no tuviésemos que crear estos scripts, ni hacer nada más para que nuestros programas reaccionasen ante condiciones de error y se ejecutasen en el momento exacto que los necesitamos – y en el orden exacto que necesitamos. Existen varios productos comerciales que ofrecen estas funcionalidades, pero pueden llevarse buena parte de nuestro presupuesto para TI. Por suerte, el mundo del código abierto ofrece también soluciones que van más allá de la planificación de tareas con *cron*.

En este artículo explicaremos cómo usar una alternativa muy potente: *Open Source Job Scheduler*. Con cualquier software de

planificación, la unidad mínima de administración es la *tarea*, que no es más que un script o un programa iniciado por el software de planificación. En muchos casos, *cron* es suficiente para gestionar las necesidades de planificación más simples, como la ejecución de un determinado programa una vez al día (para realizar, por ejemplo, backups). Podemos usar *cron* incluso para tareas que necesitemos ejecutar a intervalos menores de tiempo (cada 15 minutos), mayores (una vez al mes), o en momentos específicas (el día 1 de cada mes).

En el momento en que se presentan dependencias de cualquier tipo, comenzamos a ver las limitaciones de *cron* – imaginemos que queremos iniciar un programa específico cada vez que ocurre un evento determinado. En nuestro trabajo, tenemos una serie de cadenas de tareas que constan de hasta una docena de tareas individuales que han de ejecutarse en un orden preciso. Cada tarea sólo ha de ejecutarse si la tarea anterior de la cadena terminó con éxito. Si uno de los pasos fallase, la ejecución de las siguientes tareas de la cadena sólo haría crearnos problemas. Ni siquiera las últimas versiones de *cron* son suficiente para manejar esta situación, especialmente cuando necesitamos saltar a la mitad de

una cadena de tareas para continuar desde allí.

Los eventos iniciadores de tareas no tienen por qué ser sólo momentos específicos o la correcta terminación de tareas anteriores. Hay tareas que esperan hasta que se envía un archivo específico o llega cualquier cosa a un directorio determinado. Naturalmente, en muchas ocasiones, las tareas se podrán reiniciar bajo demanda, sin que se tenga que dar ninguna otra circunstancia.

Hay muchas tareas que se deben gestionar en varias máquinas, por lo que un buen software de planificación nos permitirá gestionarlas todas desde un único punto, iniciarlas de forma remota o hacer otro tipo de gestiones. Para distribuir las tareas de *cron* entre las distintas máquinas tendríamos que usar *rsync* o algún otro mecanismo. Dicha distribución se convertiría rápidamente en una pesadilla en materia de administración bajo condiciones específicas: si la configuración variase entre ellas, si hay que reiniciar tareas manualmente con frecuencia, o si hay que realizar cualquier otra tarea no incluida en la funcionalidad de *cron*.

Al Rescate

Podemos instalar versiones de *cron* de código abierto en máquinas con Windows,

pero lidiar con otros sistemas puede causarnos algún problema adicional. Por ejemplo, hay dialectos de Unix que no soportan un archivo central */etc/crontab*, por lo que habremos de instalar un archivo individual para cada usuario. En su día, cron fue una herramienta muy útil, y en ocasiones lo sigue siendo, pero no nos queda más remedio que descartarlo debido a las necesidades específicas de muchas empresas.

Las limitaciones de cron no han pasado desapercibidas, provocando la llegada al mercado de nuevos productos. Los productos comerciales pueden llegar a costar una pequeña fortuna, estando sus licencias, muchas veces, condicionadas por el número de servidores y, en algunos casos, incluso por el número de scripts que inician.

Los problemas relacionados con la planificación de tareas y las licencias de software comercial tampoco han pasado desapercibidos para la comunidad de software libre. Una magnífica solución, desarrollada por *Software- und Organisations-Service GmbH* en Berlín, Alemania, es *Open Source Job Scheduler*. Disponemos de versiones para Linux, Solaris, HP-UX (PARISC, IA64), AIX y Windows. También da soporte para varias bases de datos, entre las que se encuentran DB2, Oracle, MS SQL Server, PostgreSQL y MySQL.

Dependiendo de nuestras propias necesidades, elegiremos entre las dos licencias que hay disponibles: *GPL* y *Guaranteed License*. Ambos tipos proporcionan las mismas bases, incluidas todas las funcionalidades, el código fuente y las actualizaciones. El producto es exactamente el mismo con las dos licencias.

Aunque en el mercado hay productos que ofrecen más funcionalidades en su versión comercial, el director de SOS, Andreas Püschel, opina que el negocio de la empresa para la que trabaja debe centrarse en los servicios, y no en el producto. Además, SOS no comercializa su producto, ni tampoco los servicios que ofrece, de un modo tradicional. En sus exposiciones, el objetivo final no es convencer al cliente potencial, sino demostrar lo que el producto es capaz de hacer y dejar que sean los clientes quienes decidan por sí mismos si cubre sus necesidades. Sólo viendo lo que este producto es capaz de hacer, muchos quedarán convencidos.

En la versión comercial se asigna un "responsable" a cada llamada de servicio,

garantizándose unos tiempos de respuesta preestablecidos. Las peticiones de nuevas funcionalidades tienen mayor prioridad ante una posible implementación. Además, esta versión no presenta las restricciones de la GPL, por lo que la podemos incluir dentro de nuestra propia aplicación (sin tener que adherirnos a la GPL).

A diferencia de la mayoría de software, tanto de código abierto como comercial, SOS nos brinda además una garantía limitada de dos años, así como una cláusula específica de indemnizaciones. Según nos cuenta Püschel, la empresa se siente obligada a dar al usuario aquello por lo que paga, incluida la coherencia entre el producto y su documentación.

El componente central del paquete es el motor *Job Scheduler*, que se ejecuta en cada una de las máquinas en las que se van a planificar tareas. La documentación ofrece un par de métodos para permitir la ejecución remota. Uno de ellos consiste en tener un planificador instalado en la máquina remota actuando como esclavo. Si fuese necesario, dicho planificador podría realizar sin ningún tipo de problema cualquier tarea de forma autónoma. Este mecanismo se puede ampliar para posibilitar el balanceo de carga entre varios servidores. En este método sólo hemos de gestionar una única cadena de tareas, que desde nuestra máquina distribuye las peticiones hacia el resto.

Un aspecto clave a tener en cuenta es lo que se conoce por *petición*, que no es más que una señal o baliza que pasa de una tarea a otra. Dependiendo de la configuración, puede que las tareas no se inicien hasta recibir una petición.

En su forma más simple, las peticiones son como el testigo de una carrera de relevos, pasando de una tarea a otra. Las peticiones pueden contener parámetros que las tareas pueden compartir (como por ejemplo el nombre del archivo que está siendo procesado). Además, podemos configurar en la petición un tiempo de inicio determinado, de modo que el sistema la genere en dicho momento. Tras generarla se iniciará la cadena de tareas asociada a dicha petición.

Cabe destacar que para que una tarea sea capaz de reaccionar ante una petición, hay que configurarla antes. Nótese que una petición sólo se puede asociar a una sola cadena de tareas, a diferencia de las tareas individuales. De este modo, la petición va pasando de una tarea a otra dentro

de la cadena de peticiones, pero realmente está asociada a la cadena y provoca su inicio. Si, por ejemplo, se quiere iniciar una tarea individual en un momento determinado, se puede hacer desde de la propia tarea.

Otro de los componentes clave son los *Hot Folders*, directorios que el planificador monitoriza constantemente en busca de cambios, como pueden ser nuevas tareas o tareas modificadas.

Se pueden configurar las tareas desde la GUI del *Job Scheduler Editor* (Figura 1; al que de aquí en adelante nos referiremos como *Editor de Tareas*) o editando los archivos XML directamente. La interfaz del Editor de Tareas es una aplicación basada en Java con la que se pueden configurar tareas, cadenas y otros aspectos del sistema. Toda la configuración del servidor está almacenada en archivos XML, así que lo único que hay que hacer para realizar los cambios es abrir el correspondiente archivo en el Editor de Tareas. Desde nuestro punto de vista, y en comparación con otros planificadores de tareas, el hecho de que podamos usar vi para editar los archivos de configuración es más que una bendición, sobre todo cuando tenemos que realizar cambios de forma masiva.

Los archivos XML se pueden copiar a otras máquinas o guardarlos directamente a través de FTP desde el Editor de Tareas. Al aterrizar en un *Hot Folder*, los archivos pasan a estar disponibles directamente para el motor de planificaciones de la máquina remota.

La interfaz del Editor de Tareas no es todo lo intuitiva que debería. Tampoco está del todo claro a primera vista el propósito de cada campo. En algunos casos, podemos hacernos una idea de su propósito a partir de las descripciones de los archivos XML en la documentación.

Aunque el formato predeterminado para el almacenamiento de la configuración es XML, es posible configurar el planificador para usar una base de datos. A este tipo de tareas se las denomina "tareas gestionadas". Cada uno de los planificadores que instalamos puede configurarse para que acceda a las tareas de la base de datos, por lo que no tendremos que copiar los archivos manualmente.

Las operaciones del día a día se gestionan con *Job Scheduler operations GUI* (de aquí en adelante la *GUI de Operaciones*), al que podemos acceder a través del navegador, permitiéndonos la gestión de nuestras

tareas desde cualquier máquina. Con esta interfaz, no sólo podemos monitorizar tareas, sino que también podemos iniciarlas, pararlas, gestionar errores o realizar otras funciones.

El planificador ofrece además una API, a través de la cual es posible también la gestión externa de tareas. La API soporta varios lenguajes, entre los que se cuentan Perl, VBScript, JavaScript y Java. Nos extrañó que no soportase PHP, puesto que las tareas son gestionables a través del navegador y en la documentación aparecen ejemplos de scripts en PHP.

La Instalación

En los ejemplos de este artículo hemos usado la versión 1.3.4 para Linux, que podemos descargar desde SourceForge [1]. Si se planea usar una base de datos MySQL, como hicimos nosotros, hay que tener en cuenta que no se proporciona un driver JDBC para MySQL, aunque sí para Oracle y otras bases de datos. El driver JDBC para MySQL lo podemos descargar directamente desde el sitio web de MySQL [2]. Tan sólo hay que introducir la ruta al archivo .jar apropiado en el momento de la instalación.

Antes de comenzar recomendamos la lectura de la guía de instalación en PDF que viene con el paquete. Además, hay algunos PDF más en el sitio web de la compañía [3], que profundizan más en varios aspectos. Aviso: Para sacar un mínimo partido a la información de la documentación debemos estar familiariza-

dos con la programación orientada a objetos y XML, ya que en dicha documentación no se explica nada al respecto. Y tampoco está demasiado organizada, lo que nos obliga a buscar mucho en ella. Es bastante extensa y su usabilidad no es muy buena, pero la empresa piensa mejorarla.

En Linux, la instalación se realiza mediante un asistente en Java y, en caso de hacerla con un usuario sin privilegios de administración, la ubicación predeterminada de la instalación es `$HOME/scheduler`. Si lo hacemos como `root`, se realiza en `/usr/local/scheduler`. Después de instalar el producto, vemos que hay un archivo README que no nos recomienda instalar *Job Scheduler* como `root`, aunque la guía de instalación y configuración no lo mencionaba en ningún momento. Para evitar posibles complicaciones, reinstalamos desde un usuario sin privilegios.

Durante la instalación se nos pide información sobre la conexión y se nos pregunta el tipo de base de datos. No está claro si al decir “parámetros para la base de datos” se refiere a los parámetros de la conexión una vez iniciada la base de datos, o si por el contrario se refiere a los parámetros de la conexión que se usarán para crear la base de datos. Lamentablemente, aunque se recomienda crear un usuario en la base de datos para uso exclusivo de *Job Scheduler*, este extremo sólo se menciona al finalizar la instalación. La solución pasa por crear la base de datos manualmente y asignar privilegios antes de comenzar la instalación.

La instalación es bastante intuitiva, pero tarda varios minutos en crear las tablas de la base de datos. Si tenemos intención de instalar el planificador en varias máquinas con los mismos parámetros, nos vendrá bien la pregunta que se nos hace al finalizar la primera instalación, momento en el que podremos decidir si queremos crear un script para la automatización de la instalación.

Independientemente de que hayamos instalado *Job Scheduler* como `root`, o como usuario sin privilegios, hemos de iniciar el planificador manualmente. Normalmente, el motor se controla a través del típico script para `rc`:

```
$HOME/scheduler/bin/➤
jobscheduler.sh start
$HOME/scheduler/bin/➤
jobscheduler.sh stop
```

Si queremos que el planificador se inicie automáticamente con cada inicio del sistema, nuestra recomendación es crear un usuario específico para el motor y un script para `rc` que haga `su` a ese usuario y lo inicie. Si hay tareas que se tienen que ejecutar como `root`, o como cualquier otro usuario, hay que preparar un entorno de `sudo` adecuado.

Creción de Tareas

Para crear nuevas tareas, podemos editar los archivos XML directamente o bien a través del Editor de Tareas. En Linux, ejecutamos el script `jobeditor.sh`, que por

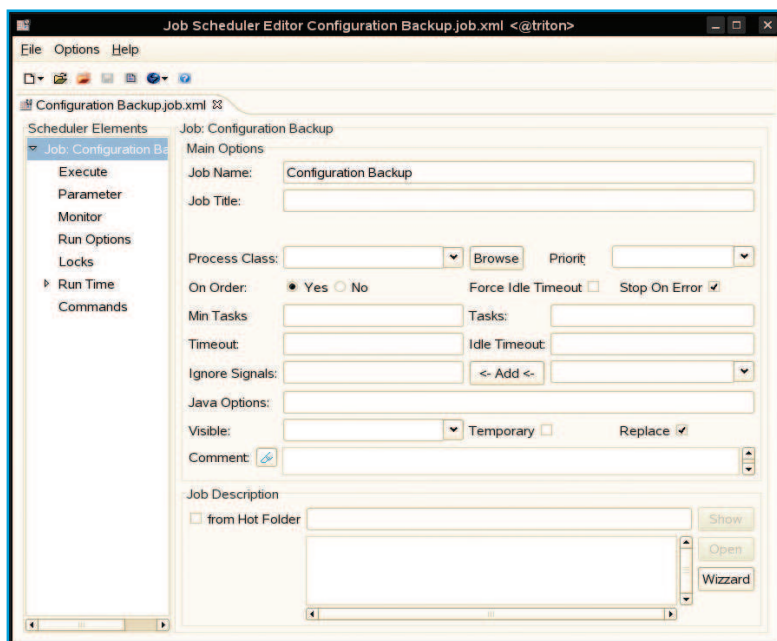


Figura 1: El Editor de Tareas.

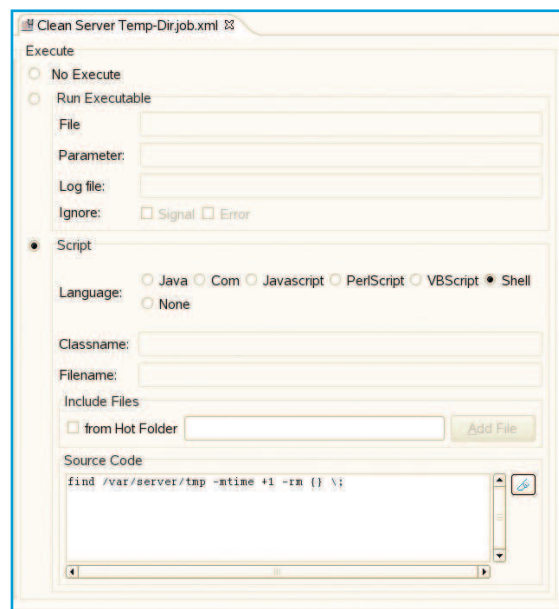


Figura 2: Introducimos el código directamente en el Editor de Tareas.

defecto se ubica en `/usr/local/scheduler/bin` o `$HOME/scheduler/bin`.

Imaginemos la típica tarea con la que se hace una copia de respaldo de la configuración de nuestro sistema. Supongamos que la queremos hacer diariamente y que usaremos el script `/usr/local/bin/config_backup.sh`.

Primero, arrancamos el Editor de Tareas y vamos a *New | Hot Folder Element | Job* (Figura 1). En el primer formulario, introducimos la información básica de nuestra tarea. En nuestro ejemplo, sólo vamos a poner "Copia de Seguridad" en el campo *Job Name*; en el campo *Job Title* podríamos añadir una breve descripción, pero nosotros lo dejaremos en blanco.

En el panel izquierdo, pulsamos sobre *Execute* para introducir los detalles acerca del programa o script que pretendemos usar. Como queremos ejecutar un script externo, seleccionamos *Run Executable* e introducimos su ruta completa. Aquí podemos definir también los parámetros que le queremos pasar al script. Por ejemplo, para comprimir los archivos que vamos a respaldar podríamos añadir `-c`.

También podemos incluir etapas individuales de ejecución seleccionando el botón *Script* y el tipo de código del programa e introduciendo dicho código en el cuadro correspondiente. Como puede apreciarse, se pueden definir estructuras de programación en el lenguaje seleccionado (Figura 2).

Para guardar la tarea pulsamos el botón *Save* o escogemos *Save* desde el menú *File*. La primera vez que guardamos el archivo se nos pregunta su nombre. Vamos a `../config/live/` y lo guardamos como *Respaldo de la Configuración*. La extensión `.xml` se añade automáticamente.

Al guardar el archivo en el directorio *live* pasa a ser visible por el sistema inmediatamente. Éste es, de forma predeterminada, un *Hot Folder* que el sistema inspecciona constantemente. Llegados a este punto, la tarea ya está añadida al sistema, pero no está programada. Para iniciar la tarea, ejecutamos la *GUI de Operaciones* apuntando nuestro navegador a `http://localhost:4444`.

Al conectar desde el navegador, aparece una GUI similar a la mostrada en la Figura 3. Para ver los detalles de la tarea pulsamos dos veces en ella, en la columna de la parte izquierda. Si la queremos ejecutar inmediatamente, pulsamos sobre el botón de menú *Job* y seleccionamos *Start task now*.

No es que sea muy espectacular, ya que podemos iniciar el script desde la línea de comandos.

Volviendo al Editor de Tareas y pulsando sobre la entrada *Run Time* en el panel izquierdo, podemos seleccionar el momento en el que queremos que se ejecute la

tarea. Para hacerlo, pulsamos sobre *Every-day* y definimos un nuevo período con el botón *New Period*. En *Start Time* ponemos algo como `09:00`, en el campo *Single Start*. Luego guardamos con el botón *Save* y ya estará activada la configuración – la tarea comenzará cada día a las 9:00 a.m.

En términos de creación de tareas, lo único que cron no nos proporciona y el editor de *Job Scheduler* sí, es una GUI – pero no hemos hecho más que arañar la superficie. Cuando empecemos a trabajar con cadenas de tareas comenzaremos a ver el potencial del planificador.

Después de crear una segunda tarea que realiza una copia de seguridad de la base de datos, vamos a hacer que se ejecute inmediatamente después de la copia de seguridad de la configuración del sistema. Una posibilidad sería crear un script que primero hiciese la copia de la configuración, e inmediatamente después hiciese la copia de la base de datos. Pero las cadenas de tareas son útiles en situaciones mucho más complejas en las que no nos servirá un simple script, como veremos más adelante. En favor de la simplicidad, nos vamos a quedar sólo con dos tareas.

Igual que hicimos con la primera tarea, creamos un nuevo elemento de tipo *Hot Folder*, pero seleccionando esta vez *Job Chains*. Aquí introducimos un valor para *Chain Name* y, si lo deseamos, para *Title* (descripción breve). A cada elemento de la cadena se le denomina nodo (*Node*). Lo siguiente es añadir un nuevo nodo con

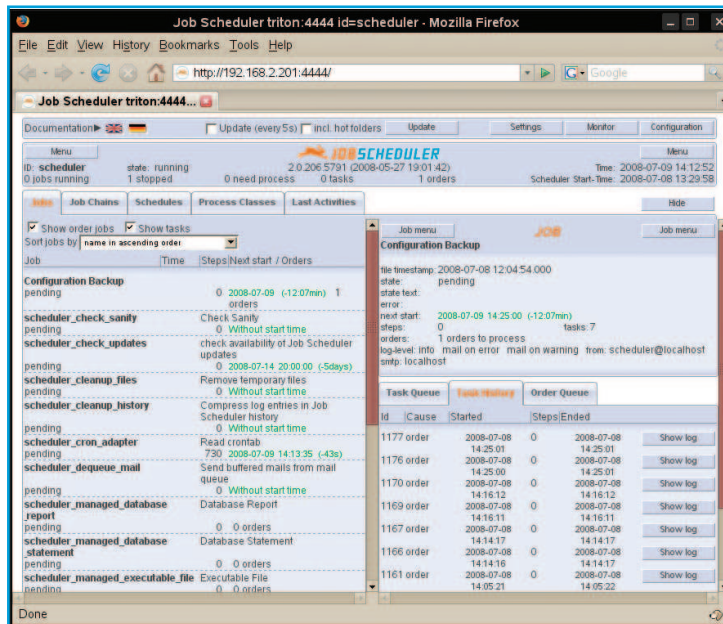


Figura 3: La GUI de Operaciones.

New Chain Node. Si sabemos el nombre de la tarea, podemos introducirla manualmente. En caso contrario podemos elegirla desde el botón *Browse*.

En el campo *State*, podemos definir un estado para el nodo. Mediante la definición de estados, podemos crear un flujo de trabajo más complejo. Por ejemplo, podríamos definir un estado llamado *Error* para que, cuando uno de los nodos encuentre un error, salte inmediatamente a esta tarea, ignorando el resto de los nodos.

Crear una cadena de un solo nodo tiene sus ventajas, pero no vamos a detenernos aquí. Como mencionamos anteriormente, nuestra cadena de tareas constará de dos tareas, así que vamos a crear un segundo nodo con la tarea encargada de la creación de copias de seguridad de la base de datos, y la configuraremos de forma similar. En este ejemplo, definimos el primer nodo con un estado *Start*, y el segundo con un estado *End*, aunque verdaderamente no es necesario.

Al pulsar sobre el botón *Save*, podemos ver la nueva cadena en la pestaña *Job Chains* de la *GUI de Operaciones*. La casilla *Show Jobs* muestra las tareas de nuestra cadena. Haciendo doble click sobre la cadena, se nos abre a la derecha un panel con los detalles.

En este punto, la cadena de tareas no se ejecutaría, debido a que no tiene *peticiones* de avance, que podemos crear manualmente pulsando en *Add Order* desde el botón *Job Menu* de la derecha. Al hacerlo se abre una nueva ventana que nos per-

mite definir ciertas características de la petición, como el ID, el momento de inicio o el estado al que debe saltar. En este ejemplo, lo dejamos todo en blanco, encargándose el sistema de crear por nosotros un ID de petición.

Al no definir condición alguna, la cadena se inicia inmediatamente; aún así, podríamos haber definido un *Time Slot* para que, o bien las tareas o el mismo planificador, tengan que esperar a que se cumpla el tiempo especificado antes de iniciar la tarea.

Las tareas deben ser capaces de aceptar peticiones ante las que reaccionar. Esto se consigue desde la ventana de configuración de cada tarea. En la ventana *Main Options* hay un botón de selección llamado *On Order*, que hay que poner a *Yes*; de lo contrario, la petición no iniciará la tarea.

Hasta ahora hemos iniciado todo más o menos manualmente. Ahora hay que definir una petición con la que arrancar la cadena de tareas. Por eso necesitamos alguna forma de crear peticiones dinámicamente. Una opción es crear una petición para un momento dado, para que ésta inicie la cadena. Se puede hacer desde el menú *New | Hot Folder Element | Order*. Después de asignar un nombre a la petición en la ventana *Job Chains*, seleccionamos la cadena de tareas específica que queremos asociar a esta petición. Recordemos que una petición sólo se puede asociar a una cadena. Luego definimos un nuevo período de tiempo (*Time Period*) y *Single Start* como *09:00*. Para que los cambios tengan efecto inmediato, hemos de guardarlos en el directorio *config/live/*.

Al volver a la *GUI de Operaciones*, vemos que hay una petición asociada a la cadena que habíamos creado. Debajo de la petición aparece una entrada *next start*, que muestra la fecha y hora en que comenzará la petición.

Controlar los Tiempos de Inicio

Disponemos de un par de opciones a la hora de definir períodos para nuestras tareas, cadenas y peticiones. En primer lugar, podemos definir un espacio de tiempo – por ejemplo, para cuadrar cuentas diariamente después de que todas las transacciones de entrada y de salida se hayan extraído de la base de datos. Debido a la carga que este tipo de tareas produce

en el sistema, no deberían ejecutarse hasta pasadas las 11:00 p.m., ni siquiera aunque las otras tareas hayan finalizado.

La segunda opción es definir un momento específico para el inicio. Por ejemplo, la actividad debería empezar a las 10:00 p.m., o cada 12 horas. Naturalmente, para una cadena de tareas en la que los trabajos se van a realizar secuencialmente, no necesitamos tiempos de inicio fijos. De cualquier modo, es posible definir una petición que se inicie diariamente a las 7:00 a.m., momento en el que comenzaría la cadena de tareas.

No estamos limitados a ejecutar tareas en momentos específicos, también podemos configurar tareas y peticiones para que se inicien en días específicos de la semana o del mes conforme a una definición más compleja, como pueden ser el segundo Lunes del mes, el penúltimo día del mes, e incluso días determinados como el 8 de Enero o el 16 de Febrero.

A pesar de lo útil que nos puede resultar la ejecución de tareas y cadenas de tareas en momentos específicos, no siempre es posible saber con antelación cuándo vamos a necesitar ejecutarlas (como ocurre con la consulta de información de una base de datos tras la recepción de un archivo dado). Lo más directo es escribir el script de la tarea para que termine en cuanto detecte que el archivo aún no ha llegado. De todas formas, si el sistema se satura con mensajes innecesarios, se nos pueden pasar por alto eventos importantes. Por eso, para resolver este problema, *Job Scheduler* pone a nuestra disposición una serie de directorios vigilados. Se trata de directorios que son inspeccionados en busca de archivos específicos, incluso mediante la utilización de expresiones regulares (Figura 4).

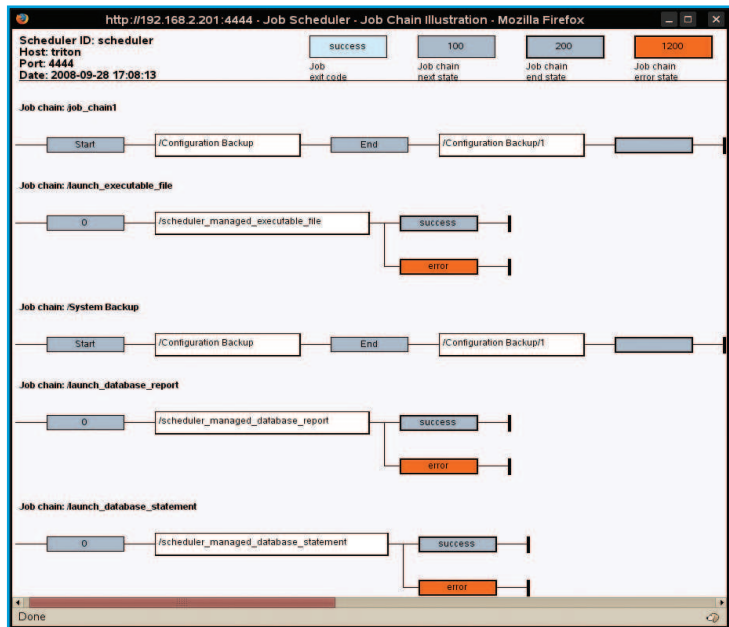


Figura 4: Las dependencias entre las distintas tareas se pueden ver mediante Job Chain Illustration.

Definimos Nuestra Propia Petición

Tras superar confusiones y conceptos erróneos, producto de las experiencias con otras soluciones de planificación de tareas, le fuimos tomando aprecio a *Open Source Job Scheduler*. Una vez nos hacemos a él, el producto acaba resultando fácil de configurar y administrar.

Nuestra experiencia con SOS GmbH fue realmente buena. Desde el recepcionista hasta el director, pasando por el soporte técnico, todas y cada una de las personas con las que hablamos parecía convencida acerca de la calidad de su producto y de la propia compañía. Quedamos impresionados cuando, tras descubrir un bug, el archivo .jar parcheado estaba en el servidor en menos de un día.

Incluso con una red pequeña, *Open Source Job Scheduler* ofrece funcionalidades muy útiles. En instalaciones de mayor tamaño se hace una herramienta casi indispensable.

RECURSOS

- [1] Open Source Job Scheduler: <http://jobscheduler.sourceforge.net/>
- [2] Driver para MySQL JDBC: <http://www.mysql.com/products/connector/>
- [3] Software und Organisations Service GmbH: <http://www.sos-berlin.com/scheduler>