Universidad Politécnica de Madrid

Facultad de Informática

Trabajo Fin de Carrera

Gestión y Distribución de Aplicaciones en Grandes Organizaciones

Tutor: Prof. Dr. Jesús Cardeñosa Lera

Autor: Germán Cancio Meliá

Agradecimientos

Deseo empezar dando las gracias al Profesor Dr. *Jesús Cardeñosa* por haber supervisado y guiado la realización del presente Trabajo Fin de Carrera, y por haberme abierto las puertas para participar en proyectos internacionales. Expreso mi gratitud al Dr. *Philippe Defert* por haberme dado la oportunidad de trabajar en el CERN, y por todo el apoyo, personal y profesional, que me ha brindado durante mi estancia en Ginebra. Mi agradecimiento a todos mis amigos y compañeros de la Facultad, especialmente del grupo VAI, y del CERN, especialmente del grupo DIS, por toda la ayuda y todo el soporte recibido. Muchas gracias a *Maria José*, por todo su afecto y paciencia. Finalmente y sobre todo, agradezco a mis padres y mis hermanos todo su cariño y comprensión durante la realización de la licenciatura.

Germán Cancio Meliá, Mayo de 1998.

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

A pesar de que el despliegue generalizado de la informática en grandes empresas y organizaciones ha acarreado beneficios en términos de mayor productividad, coordinación y creatividad, también ha generado serios problemas de soporte y un aumento considerable de costes. Las organizaciones dan cada vez en mayor medida acceso a sus empleados acceso a información y aplicaciones, lo cual deriva directamente en una mayor dedicación de recursos a servicios de soporte. Por otro lado, el dinamismo inherente a las Tecnologías de la Información obliga a una gestión de actualización y reconfiguración continua del parque tecnológico, sea a nivel de Software o de Hardware, si la organización no quiere correr el riesgo de quedarse con medios obsoletos.

La irrupción de los ordenadores de sobremesa, que jubilaron a los ordenadores centrales, trajo consigo la libertad del uso de la informática dentro de cualquier rincón de la empresa. Debido a su reducido coste y su flexibilidad se han hecho enormemente populares e imprescindibles en toda clase de tareas. Sin embargo, mediante su implantación se renunció a algunas de las ventajas del modelo de ordenador central, fundamentalmente la de la administración centralizada.

Si bien la inversión inicial en ordenadores de sobremesa y servidores, junto al software necesario, era mucho menor que el coste de adquisición de un nuevo mainframe, se desconocían los problemas y los costes derivados de la complejidad de gestionar información en un entorno descentralizado sobre múltiples tipos y variedades de ordenadores.

En una organización se ha de dar respuesta a las distintas necesidades corporativas en materias de tecnologías de información; no tienen las mismas necesidades un portero, para el cual es suficiente acceder a un terminal para el control de acceso al edificio, que para una secretaria, que necesita un entorno de ofimática potente y fiable, que para un arquitecto, quien desea una estación de trabajo con capacidades gráficas potentes.

Por otro lado, muchos usuarios tienen necesidades concretas que les llevan a modificar la configuración de su estación de trabajo, instalando hardware y software por su cuenta.

Tampoco es posible mantener las estaciones de trabajo de la misma clase al mismo nivel de actualidad, ya que la tecnología evoluciona rápidamente, por lo que normalmente convivirán distintas generaciones de sistemas.

Resumiendo, se puede dar el caso en una organización de que cada máquina tenga características únicas. Esto complica la labor por parte del personal de soporte, que se ve obligado a analizar las características de cada ordenador por separado, lo cual deriva en un aumento de costes.

Según un estudio del grupo Gartner, el coste de mantenimiento y soporte de un ordenador PC en un entorno empresarial ronda los 8'000 dólares por usuario y año, incluyéndose la formación del usuario. Especialmente en grandes organizaciones, con miles de usuarios y miles de ordenadores, se deben de identificar qué tareas de administración pueden ser controladas y centralizadas, y cuáles de ellas pueden ser automatizadas, para minimizar este gasto.

Así, una de las tareas de soporte que más recursos consume, en forma de personal y tiempo, es la gestión de las actualizaciones de las aplicaciones software, y su mantenimiento. Normalmente, las aplicaciones hay que instalarlas por separado en cada ordenador. La instalación y actualización debe ser llevada a cabo por un experto técnico debido a su complejidad, para mantener la consistencia entre los ordenadores, y para mantener un control sobre las licencias. Sin embargo, y especialmente en entornos grandes, donde los ordenadores se cuentan por centenares o miles, es prácticamente imposible ejecutar manualmente esta tarea de una manera eficiente.

La expansión actual de las redes corporativas, que comunican todos los ordenadores en una empresa, ha sentado la base para que la distribución de aplicaciones pueda ser automati-

zada. Pero hasta el momento, la gestión de distribución de aplicaciones no ha sido muy tenida en cuenta ni por el mundo empresarial ni el mundo académico, y actualmente no existe ni una terminología reconocida universalmente ni una estandarización de los procesos a llevar a cabo. Las soluciones comerciales son escasas y de muy reciente aparición; la mayoría de los sistemas existentes para la distribución automatizada de aplicaciones han sido desarrollados por grandes organizaciones para dar solución a sus propias necesidades.

Uno de estos sistemas es el desarrollado en el Laboratorio Europeo de Física de Partículas (CERN), llamado ASIS (Application Software Installation System). Este sistema almacena y sirve desde un repositorio central unas 600 aplicaciones y es accedido por unas 2000 estaciones de trabajo dentro del CERN. El sistema ASIS es asimismo utilizado en otros centros científicos, que replican el contenido del repositorio.

En el siguiente capítulo de este Trabajo se analizará el estado general de la gestión de distribución de aplicaciones, examinándose especialmente los aspectos relacionados con su centralización y su automatización. Se identificarán características comunes a los sistemas existentes, sus diferencias, y sus deficiencias.

A continuación, se describirá el sistema ASIS así como su entorno de uso en el CERN. El objetivo principal de este Trabajo Fin de Carrera es el análisis y mejora de algunos de los puntos débiles del sistema ASIS, en especial la falta de un mecanismo fiable y eficiente para modificar y replicar el repositorio, así como la falta de un modelo para describir las interrelaciones entre aplicaciones que permita mejorar la coherencia en el acceso y modificación del contenido del repositorio.

Se propondrá una solución, basada en el análisis efectuado, en la cual se diseñará e implementará para su uso operacional un sistema que mediante el uso de transacciones permitirá que las modificaciones sobre el repositorio sean llevadas a cabo de manera fiable. Asimismo, este sistema servirá para replicar de manera económica y fiable el contenido del repositorio. Adicionalmente, se diseñará un modelo de interrelaciones y herramientas a través de las que se pueda definir las dependencias entre aplicaciones, y entre aplicaciones y su entorno de ejecución; esta información permitirá mayor coherencia en el acceso y la modificación del repositorio.

Finalmente se explicarán qué otras futuras aportaciones se pueden hacer al presente Trabajo y se esbozarán conclusiones sobre la labor realizada.

Capítulo 2

Estado de la Cuestión

2.1 El problema de la distribución de aplicaciones

Uno de los problemas que se deben afrontar para la gestión centralizada de recursos informáticos es la distribución de aplicaciones, es decir, hacer accesible a los usuarios los programas, utilidades y herramientas que éstos requieren para aumentar su productividad en el trabajo. Así, un entorno mínimo debe incluir herramientas de ofimática como un editor de textos o una hoja de cálculo, pero también sistemas de comunicación electrónica, además de aplicaciones específicas por cada tarea y área dentro de la empresa.

Normalmente, el software instalado de fábrica en un ordenador no incluye más que el mínimo imprescindible; generalmente se limita al sistema operativo y algunas herramientas para gestionar su operación.

El coste de gestión y distribución de software se dispara por un lado debido a el número creciente de ordenadores dentro de la empresa, y su uso en los áreas más diversos lo cual implica mayor variedad de problemas a ser resueltos con la ayuda de sistemas software. Por otro lado, la rápida evolución de los sistemas software hace que éstos se queden obsoletos y tengan que ser sustituidos o actualizados, en ciclos que tienden a acortarse cada vez más. Se debe controlar y racionalizar la explosión en la cantidad y la variedad de productos software dentro de una organización. La ejecución de esta tarea puede ser facilitada mediante procesos que de manera centralizada y automatizada permitan controlar y ejecutar la

distribución de aplicaciones a los ordenadores de los usuarios.

2.1.1 Evolución de la distribución de aplicaciones

Hace 20 años, se disponía de sistemas "Mainframe" en los que se centralizaban las aplicaciones, los datos y los ciclos de computación. Los usuarios se conectaban mediante terminales "tontos" al ordenador central. Las aplicaciones eran desarrolladas para cumplir un fin específico en un entorno específico. Eran diseñadas para ajustarse su ejecución sobre el ordenador central; cuando éste se quedaba obsoleto, había que portar manualmente la aplicación al ordenador nuevo. El mercado de software era muy limitado, y las pocas aplicaciones disponibles tenían que ser adaptadas de manera artesanal a cada ordenador.

Los usuarios realizaban su trabajo adaptándose a los programas que se ejecutaban en el ordenador central. Si un usuario necesitaba una nueva aplicación o una modificación en una de las aplicaciones existentes, debía solicitarla al departamento de informática y entrar en una lista de espera. La operación de estas instalaciones se llevaba a cabo por un personal altamente cualificado y especializado. La administración afectaba a todos los usuarios por igual. Un cuello de botella de estos sistemas es que si estaba caído el computador, todos los terminales eran inoperativos.

Con la conquista del mercado por parte de sistemas estandarizados, como los mainframes IBM y Digital VAX, y los primeros sistemas basados en UNIX, se establece una base para el comercio de software a mayor escala.

Los terminales "tontos" acabaron dando paso a los ordenadores de sobremesa, donde se daba pleno control y responsabilidad al individuo sobre las aplicaciones y los datos. Aparecieron paquetes de software que podían ser utilizados por cada usuario por separado, sin depender de los servicios centrales. Esto conlleva una explosión del mercado de software. Las aplicaciones, distribuidas en discos o cintas, vienen junto a una guía o incluso un programa de ayuda para su instalación.

El estado actual está marcado por la aparición de las redes de área local (LAN, *Local Area Network*), mediante las cuales se introduce el concepto cliente-servidor. Este hace referencia a un modelo de computación distribuida, donde los clientes (las estaciones de trabajo de los usuarios) solicitan servicios (por ejemplo acceso a datos o recursos compartidos) a

servidores que procesan y satisfacen la solicitud.

Las aplicaciones ofrecen cada vez más funcionalidades que deben de ser adaptadas a los usuarios, lo cual deriva en procesos de configuración más complejos. Los recursos requeridos por las aplicaciones se disparan, en forma de potencia de cálculo y memoria de almacenamiento.

Algunos paquetes software se pueden emplazar en servidores de ficheros, para su acceso simultáneo por varios clientes. Se suele utilizar un servidor de ficheros por cada LAN, ésta a su vez aloja un número reducido de ordenadores cliente. Las aplicaciones tienen que ser instaladas cuidadosamente sobre el servidor, de tal manera que pueda ser utilizada por todos los clientes conectados a éste. Se debe comprobar el buen funcionamiento, y ocasionalmente adaptar la instalación por cada cliente de manera manual, dado que la configuración de éstos varía sensiblemente. Esto exige frecuentemente el desplazamiento de un técnico al emplazamiento físico de cada ordenador.

Cada LAN dentro de la organización tendrá su propia configuración en servidores y clientes. En la organización aparecen multitud de plataformas distintas e incompatibles entre sí, tanto en el mundo UNIX como en menor medida en el mundo PC.

Una nueva dimensión se abre con la aparición de redes corporativas que abarcan toda la organización y que intercomunican todos los ordenadores en ella. Las nuevas tecnologías en materia de comunicaciones permiten avanzar la implantación global del esquema cliente-servidor. La tendencia actual es la integración transparente de todos los servicios en la red, de manera que se ofrezca al usuario un entorno homogéneo, donde tenga acceso a todos sus datos y aplicaciones independientemente de la localización física. De hecho, la red se está convirtiendo "en el sistema nervioso central de la empresa, mediante el cual usuarios pueden comunicar con aplicaciones, con contenidos y entre ellos, y sirve como plataforma a la extensión de los sistemas de información" [7].

2.1.2 La gestión centralizada de aplicaciones software

Los usuarios normalmente no tienen los recursos necesarios, en forma de experiencia o tiempo, para encargarse ellos mismos de la gestión de las aplicaciones software necesarias para su trabajo. Sin embargo, cuando no existe ningún servicio central que se encargue

de ello de manera satisfactoria, se da el caso de que los usuarios cambian la configuración, instalan aplicaciones por su cuenta, añaden componentes hardware etc. lo cual hace que cada máquina tenga unas características particulares.

Muchas veces comprometen el buen funcionamiento del sistema informático, ya que por falta de competencia borran o sobreescriben aplicaciones vitales, o incluso introducen "viruses" en las redes de la empresa, lo que puede comprometer la seguridad de toda la organización. El mayor problema de seguridad en una empresa no son los ataques externos, sino los usuarios descuidados o malintencionados.

El intercambio de información entre usuarios se ve dificultado debido a la existencia de distintos paquetes software que cumplen la misma función, o distintas versiones de estos paquetes, eventualmente incompatibles entre sí. Esto deriva en pérdidas de productividad, debido al tiempo invertido en solucionar estos problemas. Las tareas del personal de soporte se complican, ya que deben analizar por separado cada ordenador antes de poder determinar las causas de un problema.

Cuando existe un servicio informático central, debe ser el personal de informática el responsable de ofrecer este servicio a los usuarios, de una manera consistente. El mantener un control central sobre quién, dónde y porqué se utiliza un determinado producto software es importante para minimizar los gastos derivados de su adquisición, por ejemplo calculando el número y tipos de licencias necesarias, y los recursos invertidos en la instalación y el mantenimiento.

Se pueden establecer centralmente políticas de configuración, para ofrecer al usuario un entorno de trabajo estándar y homogéneo.

Adicionalmente, la división de informática podrá operar de manera más eficiente el mantenimiento, debido al conocimiento sobre qué aplicaciones son utilizadas por quién y dónde; y a la existencia de una configuración global que implique a todo el parque informático, tanto para hardware como software, establecida normalmente por la división misma.

Además del control sobre las aplicaciones, es conveniente llevar un *inventario* centralizado, identificando y localizando cada ordenador, sus periféricos, sus características y configuración hardware y software. Este inventario sirve como ayuda para la distribución de aplicaciones, ya que a través de los datos proporcionados se puede estudiar la adecuación y

viabilidad de una distribución. También sirve a los técnicos de soporte para determinar la causa de un problema de mantenimiento y facilitar su solución, evitándoles desplazamientos físicos a cada ordenador.

2.1.3 Pasos en la selección, distribución y el mantenimiento de aplicaciones

La distribución de aplicaciones solamente tiene sentido si ésta se basa en ciertos criterios, que se pueden discretizar definiendo las fases en las que se basa.

- En una primera fase, hay que *definir* el servicio que se ofrecerá al cliente, y el software que da soporte a este servicio. Esto implica para el departamento de informática comprender las líneas de negocio de la organización, para poder preparar las bases para el servicio requerido. Para ello es necesario la existencia de una buena comunicación entre los usuarios y el personal de informática. Dependiendo de la naturaleza de la organización, serán los usuarios, la alta dirección, o ambos quienes establezcan los requisitos sobre qué tipo de actividades deben ser soportadas a través de un sistema software. Asimismo, se deberá llegar a un acuerdo sobre el tipo y nivel de servicio que será prestado por cada paquete de software.

  Para el personal de informática, esto supone ponerse de acuerdo con los usuarios y eventualmente con la dirección sobre qué soluciones software concretas deben de ser ofrecidas, y evaluar si los recursos hardware existentes son suficientes o deben de ser actualizados. Se llevará a cabo una selección de productos, determinando si un es necesario un desarrollo propio o si existe una solución aceptable en el mercado, que se ajuste tanto a las necesidades de los usuarios cómo a los recursos materiales, financieros y humanos disponibles.

  Asimismo, se deben de asignar las tareas de mantenimiento del producto software, evaluándose hasta qué nivel éste mantenimiento ha de ser llevado a cabo por personal propio o mediante subcontratas. El mantenimiento del software será el que causará más consumo de recursos humanos, por lo que se debe acordar de manera precisa qué tipo de soporte se ofrecerá, por quién, cuándo y bajo qué condiciones.

- Una vez que el producto software ha sido elegido y está disponible, se procede a *distribuir* éste en los lugares de trabajo del usuario. La existencia de una red corporativa

que permite interconectar los ordenadores de la empresa mediante un modelo cliente/servidor, da la base para que la distribución pueda ser llevado a cabo de manera centralizada. El software deberá ser configurado para ajustarse a los requisitos de los usuarios y las directivas generales, y a continuación deberá ser instalado físicamente sobre los servidores y/o ordenadores de los usuarios.

- La tercera fase es la *operación y el mantenimiento* del servicio en los términos acordados. Esto requerirá que la dirección de informática gestione la tecnología y la infraestructura sobre el cual está basado el funcionamiento del software. Por ejemplo se deben gestionar la red, los servidores, las bases de datos centrales, etc. Por otro lado se establece un mecanismo de soporte al usuario, por ejemplo a través de una ventanilla de ayuda, una línea telefónica o correo electrónico. También será necesario mantener el software en sí; es decir, detectar fallos y corregirlos (ya sea a través del personal propio o informando al vendedor del software), revisar y reajustar su configuración, y la instalación de nuevas versiones del software si así se requiere.

  Para llevar a cabo un mantenimiento consistente, es necesaria una estrecha coordinación entre los grupos de la división de informática encargados de cada función particular.

2.1.4 Grado de libertad de los usuarios

El control central sobre qué aplicaciones deben de ser instaladas sobre cada ordenador, limita la libertad del usuario sobre la que considera muchas veces *su* personal herramienta de trabajo. Esto deriva directamente del hecho de que el usuario ha sido quién se ha encargado clásicamente del control y mantenimiento de su ordenador.

Se puede establecer una clasificación de los usuarios en función del grado de libertad a la hora de establecer la configuración para cada máquina. Así, habrá usuarios que requieran acceder solamente a unas cuantas aplicaciones. Las máquinas que son accedidas por estos usuarios podrán tener una configuración más rígida, por lo que pueden ser gestionados de manera más uniforme y más económica por el personal de informática. El control sobre qué productos software se instalan y cuando se actualizan se lleva a cabo de manera central. La dirección de informática puede condicionar a un usuario el servicio de soporte a la prohibición de cambiar la configuración o instalar aplicaciones sin autorización.

En cambio, otros usuarios requerirán mayor libertad y una configuración personalizada sobre el conjunto de aplicaciones ofrecidas, para mantener un alto nivel de productividad. Especialmente en el mundo PC, el usuario está muy acostumbrado al alto grado de libertad que le ofrece; uno de los avances de los PC's sobre los ordenadores centrales es precisamente su flexibilidad. Limitar ésta sin fundamento o exclusivamente por razones de simplicidad y economía, puede significar mermar la productividad del usuario, dado que puede interpretarlo como una falta de confianza por parte de la dirección.

En los casos que el usuario deba mantener una cierta autonomía, la dirección informática deberá ceder parte del control, recomendando en vez de obligando una cierta configuración. Consiguientemente, parte de la responsabilidad del mantenimiento estará en manos del usuario.

En algunos de los entornos informáticos más populares hoy en día, como por ejemplo UNIX y Windows NT, existe un personaje intermedio, que se encarga de la administración individualizada de cada máquina, el *administrador*. El administrador es un usuario privilegiado que tiene acceso a todos los componentes de la máquina. En estos entornos, los usuarios finales que no tengan derechos de administración no están autorizados a modificar el conjunto de aplicaciones de una estación de trabajo. Este enfoque tiene la ventaja que la selección de aplicaciones puede ser llevada a cabo por personal experimentado. Sin embargo, en el caso de que el administrador y el usuario final no sean la misma persona, el primero tendrá que estar accesible para atender las peticiones de los usuarios, lo cual causa problemas por ejemplo cuando está de baja laboral.

2.1.5 Licencias de productos software

Antes de tomar la decisión sobre qué producto software hay que utilizar, se deberá evaluar qué tipo de licencia es la adecuada para este producto. Básicamente, existen dos tipos de licencias: licencias basadas en instalaciones efectuadas y licencias concurrentes. Dentro de las licencias concurrentes existen las licencias flotantes, y las licencias basadas en la media de usuarios. Las licencias flotantes enfuerzan mediante un mecanismo especial que no haya más usuarios utilizando simultáneamente el producto que el número máximo autorizado.

Si existe la posibilidad de elegir entre diversos tipos de licencia, la decisión dependerá, apar-

te de las condiciones económicas de cada una, de *dónde* y *cuándo* se utilizará el producto. Será importante tener estadísticas de uso de productos similares o relacionados. Así, un producto que no es utilizado frecuentemente por un gran número de usuarios simultáneo, pero cuyo uso se extiende por toda la organización, será más adecuado adquirirlo mediante una licencia de usuarios simultáneos. Si se tienen datos concretos sobre el pico esperado de usuarios simultáneos, se tiene la base para establecer una licencia flotante.

Software de libre distribución

Muchos usuarios, particularmente los que trabajan en los campos de educación y ciencia, pueden acceder un gran número de aplicaciones de manera gratuita o a bajo coste. Estas aplicaciones, llamadas de "dominio público" o de "libre distribución", son distribuidas sin ánimo de lucro, pero también sin ninguna clase de garantía ni ningún tipo de soporte implícito. Sin embargo, muchos de estos productos software se han convertido en un estándar *de facto*. Como ejemplo cabe citar las utilidades GNU (especialmente los compiladores y editores de texto), el entorno gráfico X-Windows del MIT, o el sistema operativo Linux.

Aunque se utilice un producto de libre distribución, habrá que fijarse exactamente en la licencia que viene con el producto. Así, la *licencia general pública GNU* [14], bajo la cual se distribuyen una gran parte de los productos de dominio público, exige que todos los productos que deriven de otro distribuido bajo esta licencia asimismo tengan que ser distribuidos bajo la misma licencia, por lo que serán también de libre distribución. Otras licencias limitan el uso del software en ciertos entornos, por ejemplo en organizaciones con ánimo de lucro. Es importante distinguir la libre distribución de software de la renuncia sobre la propiedad intelectual sobre él, o de la inexistencia de condiciones de uso.

2.1.6 Gestión de aplicaciones en entornos heterogéneos

El gran número de usuarios y campos de aplicación de la informática dentro de la empresa es la razón de la existencia de una cantidad y variedad importante de ordenadores y servidores, de varios fabricantes y con unas características ajustadas a cada necesidad concreta. Así por ejemplo, un sistema dedicado al diseño asistido por ordenador difiere sensiblemente de uno utilizado para la gestión de nóminas, tanto por su estructura hardware, como el

sistema operativo utilizado, o su entorno gráfico. Los ordenadores utilizados se pueden agrupar en clases llamadas *plataformas*. Bajo plataforma se entiende la combinación de hardware y el software de base, como el sistema operativo.[1]

Es importante establecer un *factor común* a las plataformas que permita su gestión centralizada, y también para facilitar a los usuarios la toma de contacto con cada plataforma. De hecho, muchos usuarios trabajan simultánea o sucesivamente sobre múltiples entornos hardware y software, y adaptarse a cada uno de ellos conlleva un tiempo de aprendizaje, el cual representa un gasto considerable y que hay que minimizar para optimizar la productividad.

Desde el punto de vista de la distribución de aplicaciones, el factor común estriba por un lado, en ofrecer aplicaciones compatibles entre sí, con un manejo parecido, y por otro lado, en ofrecer un mismo conjunto base de aplicaciones sobre cada plataforma utilizada. Cada aplicación deberá tener básicamente la misma *función*, la misma *apariencia* y la misma *configuración* sobre todas las plataformas. El usuario no debe verse obligado a adaptar la aplicación una y otra vez sobre cada plataforma utilizada; la configuración individualizada de cada usuario debe de almacenarse centralmente para que se tenga acceso a ella independientemente de la máquina usada.

De esta manera, le es posible al usuario tener las mismas herramientas de trabajo sea cual sea el tipo de plataforma y localización del ordenador utilizado, y por lo tanto se facilita establecer un *entorno homogéneo* en todas las plataformas y ordenadores.

2.1.7 Necesidad de compilación de aplicaciones

Ocasionalmente, las aplicaciones son entregadas por sus desarrolladores en formato *fuente*, independiente de la plataforma. Para poder utilizar estas aplicaciones, tienen que ser *compiladas* sobre cada plataforma por separado para generar el formato binario, específico a cada plataforma. El proceso de compilación se lleva a cabo durante la fase de configuración.

Normalmente las aplicaciones comerciales ya vienen compiladas para cada plataforma, por lo que la tarea de configuración se limita a su configuración adaptada al entorno operativo.

[1] algunos ejemplos de plataforma son: Microsoft Windows NT 4.0 bajo procesador Intel, Sun Solaris 2.5 bajo procesador SPARC.

El ofrecer la aplicación ya compilada acorta por un lado el proceso a recorrer para su utilización, y por otro lado resulta ser un mecanismo eficiente para ocultar su diseño interno ante ojos indiscretos. También sucede que muchas aplicaciones de dominio público son publicadas simultáneamente en formato fuente, independiente de la plataforma, como en formato compilado para algunas plataformas, porque la alta complejidad de su compilación y adaptación a cada plataforma suele sobrepasar la capacidad de un administrador medio. Este es el caso por ejemplo del entorno gráfico X-Windows o los compiladores GNU, cuya compilación y configuración sobre múltiples plataformas pueden llevar días incluso a un experto.

Sin embargo, siempre y cuando se disponga de los recursos en forma de tiempo, personal y herramientas adecuadas, es preferible la compilación propia de las aplicaciones cuando estén disponibles en formato fuente.

De esta manera, se ofrece mayor *disponibilidad* al poderse realizar la compilación sobre todas las plataformas deseadas, dado que muchas veces las aplicaciones precompiladas no están disponibles para el tipo exacto de plataforma utilizado, o simplemente no existen para una determinada plataforma. El proceso de instalación en el entorno de ejecución resulta más *flexible* al poderse acceder y modificar todos los componentes de la aplicación; y se aumenta la *seguridad* al poderse descartar la existencia de componentes extraños y hostiles, tipo "virus".

2.1.8 Necesidad de distribución automatizada de aplicaciones

El camino desde la adquisición de las aplicaciones a su utilización por los usuarios requiere usualmente un un proceso de instalación sobre el sistema donde debe de ser ejecutado, y un proceso de configuración a las características del entorno, es decir, de las necesidades del usuario y de las necesidades corporativas.

En una organización en la que proliferan los sistemas informáticos, esta tarea se complica y requiere una notable inversión en tiempo y personal experto debido a:

- la notable cantidad de aplicaciones a instalar,
- los procesos de configuración, ocasionalmente muy complejos, que además difieren

por cada aplicación y plataforma,

- la necesidad de repetir la instalación de las aplicaciones en todos los ordenadores a los que accedan los usuarios,
- la necesidad de actualizar las aplicaciones, para evitar que queden obsoletas, cuando son relevadas por nuevas versiones, lo cual implica repetir el proceso cíclicamente.

Aún definiendo y aplicando directivas de configuración estándar, la instalación o actualización de un producto software en todos los ordenadores puede implicar una inversión de recursos humanos en magnitudes de meses/hombre en organizaciones con miles de ordenadores.

Sin embargo, la ejecución repetitiva de esta tarea sobre cada máquina suele ser mayoritariamente monótona por lo que se presta para su *automatización*, lo cual reduciría sensiblemente el coste asociado a la distribución de software.

Un sistema centralizado para la configuración y distribución de aplicaciones puede automatizar gran parte de estas tareas. Una definición que esboza el campo de actividad de un tal sistema puede ser la siguiente: *un sistema que permite distribuir a clientes de manera automatizada y centralizada software de aplicación operativo.* El personal informático configura y prepara las aplicaciones para ser instaladas, y programan su distribución automática a los ordenadores cliente (los ordenadores de los usuarios).

Estos sistemas son de relativamente nueva aparición, y aún no existe ninguna clase de estándar sobre lo que se supone que deben de hacer o cómo deben de hacerlo. Dado que tampoco existe acuerdo común en su nombre, utilizaremos la denominación *sistemas de distribución de aplicaciones* (S.D.A.) para designarlos. Idóneamente, un S.D.A. debe mostrar las siguientes características:

- **acceso múltiple:** debe permitir la distribución concurrente de aplicaciones a múltiples clientes.
- **adaptable:** debe ser capaz de acomodar múltiples plataformas, y múltiples aplicaciones. Deberá ser flexible y escalable para aceptar múltiples métodos y estructuras de instalación que permitan dar acogida al máximo de aplicaciones distintas.

- **consistente:** debe ofrecer un servicio robusto y fiable, y evitar estados inconsistentes en las aplicaciones.
- **seguro:** debe de ser altamente seguro. Un problema en la seguridad del servidor o en alguna de las aplicaciones, puede comprometer la seguridad de todos sus clientes.
- **replicable:** debe ser replicable, de tal manera que exista redundancia en el acceso y que sitios distantes puedan acceder a las aplicaciones.
- **multiversión:** debe permitir una gestión de versiones de una misma aplicación, y una transición "sin traumas" de una versión a otra.

Actualmente, los sistemas de distribución de aplicaciones aún no están muy extendidos. Existen unos diez sistemas en el mercado. Muchos de ellos fueron creados en laboratorios y universidades para cubrir sus propias necesidades.

Existe cierta confusión a la hora de diferenciar estos sistemas de los *sistemas de gestión de configuración* utilizados en proyectos [11], y los *repositorios de software* comunes.

Esencialmente, la diferencia entre un S.D.A. y un sistema de gestión de configuración de proyecto es que el primero no está pensado para el proceso de desarrollo de software, sino para distribuir el resultado de un desarrollo, mientras que el segundo se limita al desarrollo de software. La tabla 1 establece una comparación más detallada.

La diferencia con los repositorios de software comunes (como pueden ser SimTel ó Netlib [6]) es que éstos almacenan software que ha de ser copiado, configurado e instalado sobre los ordenadores para poder ser utilizado. El proceso de distribución, configuración e instalación corre a costa del cliente. Estos repositorios son accesibles remotamente a través de Internet por un gran público, mientras que un S.D.A. es normalmente local y privado a una organización. Sin embargo, el contenido de los repositorios software comunes puede ser utilizado como fuente para suministrar productos software a un sistema de distribución de aplicaciones, como es habitual por ejemplo en el sistema de distribución Asis [13].

Existen también otros sistemas que sirven para la instalación y desinstalación de paquetes software, como Stow [16], Debian Dpkg [22] y SD-UX en HP Unix; sin embargo no distribuyen aplicaciones propiamente dicho, ya que están pensados para ser utilizados sobre un solo ordenador y plataforma y no en un entorno distribuido con múltiples ordenadores.

concepto	S.G.C.	S.D.A.
enfoque	desarrollo	distribución
uso en ciclo de vida	constante	fase de operaciones
contenido	aplicación en formato fuente, documentación de desarrollo	aplicación ejecutable y configurada, documentación de usuario
acceso	personal de desarrollo	usuarios finales
control de versiones	sí	posible
multiplataforma	posible	posible
granularidad de acceso mínima	ficheros individuales	aplicaciones

S.D.A.: sistema de distribución de aplicaciones
S.G.C.: sistema de gestión de configuración

Tabla 1: comparación entre sistemas de distribución de aplicaciones y sistemas de gestión de configuración

2.1.9 Gestión de distribución automatizada de aplicaciones

Un sistema de distribución de aplicaciones es esencialmente un sistema basado en en concepto cliente-servidor, donde el rol de cliente recae en los ordenadores de los usuarios. Estos ordenadores acceden a servidores encargados de entregar las aplicaciones. Los servidores almacenarán el software a distribuir en un *depósito* o *repositorio* de aplicaciones.

La gestión del repositorio se lleva a cabo de manera centralizada desde los servidores. Los clientes podrán tener más o menos libertad para seleccionar un conjunto de aplicaciones, pero no pueden alterar el contenido del repositorio.

Desde el punto de vista de su gestión, un S.D.A. debe ofrecer herramientas que permitan la manipulación del repositorio, implantar las políticas de distribución a los clientes, y el análisis de su contenido y uso.

El proceso de actualización del repositorio es similar en la mayoría de los sistemas y consiste en los siguientes pasos:

1. Una vez que se ha seleccionado y adquirido la aplicación a distribuir, el primer paso es

su *configuración*. En un entorno multiplataforma, será preciso repetir el proceso por cada una de ellas, si bien existen mecanismos, como AUTOCONF [15], que facilitan la adaptación a varias plataformas[2]. Esta tarea también es automatizable; el sistema ASIS por ejemplo ofrece una herramienta mediante la cual se puede configurar y compilar una aplicación de manera concurrente sobre varias plataformas.

2. El siguiente paso es definir por cada plataforma un *paquete* conteniendo la aplicación para su registro en el repositorio. En algunas plataformas, este paquete también debe de contener comandos o reglas de instalación a ejecutar localmente sobre la máquina cliente, para completar la instalación y configuración.[3] También es posible que el paquete contenga información sobre condiciones que debe de reunir el cliente para poder instalar la aplicación.

3. A continuación se determina qué clientes, o qué tipo de clientes, tendrán acceso a la aplicación. Así, algunos sistemas ofrecen la posibilidad de ofrecer una aplicación en modo de prueba a un grupo restringido de usuarios. De esta manera, y antes de efectuar un despliegue generalizado, se puede comprobar el buen funcionamiento de la aplicación y la consistencia y coherencia de su configuración.

4. La *distribución física* instalará la aplicación sobre los clientes seleccionados. Los ficheros que componen la aplicación podrán ser instalados sobre un servidor de ficheros al que acceda un grupo de clientes, o directamente sobre cada cliente. El instalar una aplicación sobre el servidor reduce los recursos (en forma de memoria permanente) necesarios en cada cliente, pero le hace dependiente del buen funcionamiento de la red.

   En algunos S.D.A., como DEPOT [24] o ASIS, los paquetes pueden ser accedidos directamente por los clientes dentro del repositorio, por lo que el servidor que almacena el repositorio puede ser utilizado como servidor de ficheros.

   En los S.D.A. SMS [10], NETWIZARD [8], y ASIS, el paquete puede ser replicado a

[2] en concreto, AUTOCONF genera *scripts* que automáticamente configuran código fuente para adaptarlo a múltiples plataformas UNIX.

[3] Este es el caso de las plataformas basadas en MS Windows. Se debe a que en el diseño de MS-Windows no fue prevista la separación entre los componentes del sistema operativo y las aplicaciones, por lo que tienen una fuerte interdependencia. Así, se deben actualizar diversos registros y ficheros del sistema operativo para notificar la existencia de la aplicación; en la instalación es frecuentemente necesario sustituir librerías pertenecientes al sistema operativo por versiones propias.

repositorios secundarios.

En el caso de existir procesos de instalación, éstos se ejecutarán. En caso de fallar la distribución (por ejemplo por detectarse un conflicto no resoluble, o si no se satisfacen condiciones para la instalación del paquete, o si hay fallos en la red), se abortará. En este caso, los sistemas SMS y NetWizard deshacen los pasos ya ejecutados y dejan el cliente en el estado previo a la instalación.

Una actualización de un cliente puede ser *compulsoria* o *libre*. Una distribución compulsoria significa que todos los clientes destino deberán instalar el paquete, mientras que las distribuciones libres es decisión del cliente aceptar o no un paquete. En las distribuciones compulsorias, se podrá verificar el estado de todos los clientes después de la distribución; en caso de que haya fallado en un cliente se puede informar al personal informático o deshacer automáticamente la distribución en todos los clientes. SMS y NetWizard permiten distribuciones compulsorias, mientras que ASIS y Depot no ofrecen esta posibilidad.

Existen dos técnicas básicas que pueden ser utilizadas para actualizar un ordenador cliente, llamadas *pull* y *push*:

- ***push* (empujar)**: un conjunto de objetos (en este caso, aplicaciones) es "empujado" por un servidor hacia un cliente. El control de la actualización se sitúa en el servidor.
- ***pull* (tirar)**: el cliente solicita ("tira") del servidor un conjunto de objetos (aplicaciones). El control sobre la actualización se sitúa en el cliente.

Mediante la técnica *push* es posible un control central de la actualización de los clientes. La técnica *pull* permite que el ordenador cliente sea actualizado asíncronamente del servidor, también permite configuraciones más independientes por parte del usuario.

Cuando existe *control de versiones* de productos software, se puede llevar a cabo de dos maneras. Por un lado, se puede considerar cada versión de una aplicación como un ente independiente, como se hace en Depot, UPS/UPD [19] y ASIS. Otra técnica es guardar por cada versión solamente sus diferencias con la anterior, como se hace en NetWizard. De esta manera, se ahorran recursos de almacenamiento, pero se complica o se imposibilita el acceso a una determinada versión de una aplicación. En ASIS, SMS y NetWizard existe

la posibilidad de corregir ficheros de la aplicación ("parche") distribuyendo los ficheros corregidos a los clientes.

Para la gestión de licencias es muy interesante disponer de almacenar centralmente la información sobre quién, dónde, cuándo y con qué frecuencia se utiliza una cierta aplicación, y la configuración de cada cliente. De esta manera, tienen datos precisos para determinar el número y tipo de licencias necesitadas en cada caso. Los S.D.A. SMS y NetWizard ofrecen representaciones visuales sobre estadísticas de uso de productos.

2.1.10 Replicación de repositorios

En entornos grandes, es interesante que se pueda *replicar* las aplicaciones de un servidor o repositorio a otro, para aumentar su disponibilidad, potencia, y redundancia. En una misma empresa, se pueden distribuir los clientes entre varios servidores, para minimizar la carga y evitar que una caída del servidor afecte a toda la organización.

Muchas empresas tienen sedes o filiales distantes geográficamente, cuyo acceso a los recursos informáticos centrales son limitados debido a que el ancho de banda de la línea de comunicación que las une con la central es estrecho. Estas sedes pueden establecer un servidor secundario, que replica periódicamente el contenido del servidor central. Cuanto más frecuente la replicación, mayor sincronización existirá entre los servidores, pero también se producirá un mayor consumo de recursos, especialmente en ancho de banda de comunicación. SMS, NetWizard y ASIS ofrecen mecanismos para replicación.

2.1.11 Distribución inatendida

La instalación sobre el cliente debe de ser inatendida, es decir, que se ejecuta sin intervención humana. Idóneamente, el usuario no debería ser interrumpido en su trabajo durante una actualización de las aplicaciones. Sin embargo, durante la fase de instalación se pueden producir inconsistencias, por ejemplo por reemplazarse una aplicación con la cual el usuario está trabajando en ese justo momento.

Por esta razón, y por la larga duración de algunas instalaciones, y para minimizar la carga de la red, las actualizaciones deberían ser llevadas a cabo fuera de horas de trabajo.

SMS permite la distribución asíncrona de aplicaciones; el responsable informático puede programar una cierta distribución para que sea llevada a cabo de noche. Por contra, la mayoría de los demás sistemas no disponen de esta facilidad, lo cual deja dos posibilidades: que los usuarios no usen sus ordenadores durante su actualización, con la consiguiente pérdida de productividad, o que el personal informático ejecute la distribución fuera de horas de trabajo, debiéndose remunerar apropiadamente.

2.1.12 Coherencia de configuración

Una de las tareas de configuración más complejas de la distribución de aplicaciones es construir un conjunto *coherente* de aplicaciones para ser ofrecidas a los usuarios. Una tendencia en alza en el desarrollo del software es de ofrecer aplicaciones que interactúan con otras aplicaciones, en vez de ofrecer productos autocontenidos e independientes. Cada aplicación puede ser desarrollada y distribuida por un fabricante software distinto. Así, un navegador de *web* no suele incluir todas las herramientas necesarias para visualizar documentos que no sean de hipertexto, sino que interactúa con otras aplicaciones en esta tarea. Una herramienta de correo electrónico no es operativa sin la ayuda de un servidor de correo, una aplicación de inventario requiere una base de datos determinada, etc. El uso de tecnologías distribuidas que permiten reutilización y comunicación interproducto, como CORBA, JAVABEANS y OLE, hace que la complejidad de las relaciones y dependencias entre sistemas software sea cada vez mayor.

Otro tipo de dependencias son las establecidas sobre objetos del entorno de ejecución, como por ejemplo, el espacio libre en el disco duro local, la memoria disponible, librerías compartidas u otros componentes del sistema operativo, etc.

Los conflictos entre aplicaciones o entre una aplicación y objetos del entorno también se pueden clasificar como dependencias, en términos negativos. Así, dos aplicaciones pueden ser incompatibles por competir por un mismo recurso, por ejemplo un determinado fichero o servicio.

El problema es que muchos fabricantes de software, y debido a la falta de un estándar al respecto, no definen de manera precisa las dependencias, o lo hacen de manera incompleta, lo que es especialmente cierto cuando los productos requeridos no son del mismo fabricante.

Así, una aplicación A puede depender de una determinada versión de la aplicación B, pero no existe certeza ni garantía de que funcione con una versión anterior o posterior de B. La información sobre las dependencias se encuentra en los más variados sitios, como manuales, notas de instalación, código fuente, etc.

La falta de un estándar obliga al personal informático a detectar estas dependencias, y establecer heurísticamente una configuración coherente. Este proceso es complicado y basado en la técnica de prueba y error, y hay que repetirlo cada vez que se actualiza el repositorio, ya sea para añadir una aplicación, o al borrarla (hay que verificar que las dependencias establecidas por otras aplicaciones no serán violadas al desaparecer una aplicación). El problema de las dependencias también se presenta a los usuarios que establecen una configuración individualizada sobre su ordenador.

La mayoría de los S.D.A. actuales carecen de un modelo para definir dependencias. Excepciones son sistemas como UPS/UPD [19], en el cual se permiten declarar dependencias de un producto a otro. El sistema LUDE [23] define dos niveles de dependencias, obligatorias y opcionales, y permite definir dependencias hacia el entorno de ejecución[4].

Sin embargo, el modelo más avanzado para la definición de dependencias no lo ofrece un S.D.A., sino un sistema de empaquetamiento de aplicaciones perteneciente al sistema operativo DEBIAN LINUX, llamado DPKG [22]. En él se definen cuatro niveles de dependencias: dependencias absolutas, recomendaciones, sugerencias, y conflictos. Adicionalmente, permite clasificar las aplicaciones por su funcionalidad y establecer dependencias hacia aplicaciones que presenten una determinada funcionalidad. También se pueden definir alternativas en las dependencias, de tal manera que varias aplicaciones puedan satisfacer una misma dependencia.

Agrupación de distribuciones

Es importante manipular conjuntamente los productos software que tengan una fuerte interrelación. Así por ejemplo, se usan para un compilador de JAVA unas librerías externas; si se decide actualizar el compilador posiblemente sea necesario adquirir asimismo la nueva

[4]las dependencias hacia el entorno de ejecución no son registradas explícitamente sino que se detectan mediante un programa tipo *script* que se ejecuta en la instalación del producto, por lo que es difícil localizar y gestionar esta información.

versión de las librerías adaptadas al nuevo compilador. El proceso de sustituir las versiones antiguas por las nuevas, tanto de las librerías como del compilador, deberá ser realizado en un paso único e indivisible, dado que de lo contrario pueden aparecer inconsistencias.

La mayoría de los S.D.A. no ofrece ninguna clase de mecanismos para agrupar distribuciones. El problema es que en distribuciones independientes, el fallo de una distribución no implica a las demás. Así, y siguiendo el anterior ejemplo, si la distribución del compilador de Java tiene éxito, pero no la distribución de las librerías, el estado resultante es inconsistente dado que ahora se encuentra instalado el nuevo compilador de Java, pero siguen las librerías antiguas. Por lo tanto, el personal informático deberá revertir la distribución del compilador mientras que se analiza el problema de la distribución de las librerías.

2.1.13 Desventajas de los Sistemas de Distribución de Aplicaciones

Debida a la poca madurez de los S.D.A. todavía existen una serie de problemas a resolver para que lleguen a ser completamente fiables y eficientes.

Frecuentemente, no todo el proceso de instalación de un producto software puede ser llevado a cabo de manera automática. Así, algunas aplicaciones han de ser imperativamente actualizadas manualmente. Aunque los los procesos de instalación existan desde los comienzos de la informática, todavía no existe ningún procedimiento estándar y unificado, por lo cual la instalación sigue basándose en heurísticas.

En las plataformas basadas en Windows, donde en la mayoría de los casos se deben de ejecutar procesos locales de instalación, la probabilidad de que ocurra un fallo es relativamente alta. Además, el informático encargado de una aplicación tiene que definir él mismo estos procesos de instalación locales; el sistema se limita a ejecutarlos. Cualquier variación en la configuración de un determinado cliente, no tenida en cuenta al establecer los procesos, puede hacer fallar su ejecución sobre este cliente. La actual inexistencia de estándares hace que los procesos de instalación locales sean a veces complejos de modelizar. Esta es una de las razones por lo cual se piensa que los sistemas para Windows no hacen lo que prometen.

Los S.D.A. están más extendidos sobre UNIX, por un lado dado que el proceso de distribución es más sencillo, debido a la madurez de este sistema operativo, y por otro lado,

porque los S.D.A. existen desde hace más tiempo sobre UNIX.[5]

Otro problema de los S.D.A. es su elevado precio inicial, debido a la poca competencia y su novedad, y el coste asociado a adaptar un proceso de gestión centralizada en la empresa. Así, una licencia de NetWizard para 100 clientes, lo cual representa una organización de tamaño pequeño-mediano, cuesta 7'500 libras esterlinas (alrededor de dos millones de pesetas al cambio actual). Si a esta inversión inicial hay que sumarle el tiempo invertido en su instalación (que requiere reconfigurar y adaptar todos los clientes al S.D.A.), la necesidad de reajustar el proceso de trabajo para centralizar el soporte y mantenimiento, la formación de los usuarios y el personal informático, y el tiempo invertido y la experiencia necesaria para definir correctamente los paquetes de instalación, el precio total puede multiplicarse fácilmente. Es por lo cual, especialmente en organizaciones pequeñas o medianas, se debe evaluar cuidadosamente si compensa la adquisición de un S.D.A., teniendo en cuenta el número de actualizaciones software previstas dentro de la organización; o si por lo contrario resulta más económico ejecutar el proceso a mano.

Aunque existen S.D.A.'s de libre distribución, éstos vienen sin ninguna clase de garantía, y en muchos casos su documentación es insuficiente. Por lo tanto es arriesgado intentar instalar uno de libre distribución sin contar con la asistencia de alguien que tenga experiencia con ese sistema.

2.1.14 El futuro de los Sistemas de Distribución de Aplicaciones

En la actualidad los sistemas de distribución de aplicaciones aún están poco extendidos fuera de organizaciones grandes, y faltos de madurez y estandarización. Sin embargo, la tendencia de crecimiento de la utilización de ordenadores en toda clase de situaciones, ya sea dentro de la empresa como en el hogar, deriva en una creciente diversidad de aplicaciones cuya complejidad crece cada vez más, asimismo como el esfuerzo necesario en adaptarlo a las necesidades del usuario.

[5]Sin embargo, la fase de configuración sobre UNIX es más compleja que sobre Windows y mucho menos amigable. Especialmente, cuando se trata de compilar, configurar e instalar una aplicación sobre múltiples plataformas, el proceso requiere la asistencia de un técnico especialista en este sistema operativo y sus variantes. Pero una vez que una aplicación está disponible sobre todas las plataformas, su integración en el cliente es mucho menos compleja que sobre sistemas basados en Windows.

Los sistemas cada vez están más interconectados, lo que abre cada vez más las puertas para la distribución automática de software. Dentro de las empresas se empiezan a establecer redes corporativas que alcanzan todos los ordenadores. Cada vez son menos las empresas y hogares que no tienen acceso a Internet, lo cual abre una nueva dimensión: la actualización de versiones directamente por parte del vendedor. De hecho, en la nueva versión de Microsoft Windows será posible solicitar al sistema operativo una su actualización a través de la red. Se aprecia un intento por parte de Microsoft en alcanzar un estándar para la instalación remota de software, a través de su reciente "iniciativa para la administración cero" (ZAW, *Zero Administration for Windows initiative* [27]).

Sin embargo, la configuración a las necesidades de la empresa no puede ser llevada a cabo por parte del vendedor, ni se debe ceder el control sobre el estado y versiones de las aplicaciones ofrecidas. Es por lo cual la gestión de estos sistemas debe de ser llevada a cabo desde la empresa y no por el vendedor.

2.1.15 *Network Computers*: ordenadores de red

Una tendencia relativamente nueva para limitar los costes de mantenimiento, tanto de hardware como de software, son los llamados NC, *Network Computers* u ordenadores de red. La idea básica en la que se basa este concepto es la de ofrecer un sistema cliente de sobremesa, basado en un entorno gráfico y conectado en red a un servidor, al cual solicita datos y aplicaciones. Es por lo tanto una evolución del concepto ordenador central - terminal de usuario.

La diferencia con los terminales gráficos habituales (como los terminales basados en X-Windows), es que en éstos la ejecución de las aplicaciones se lleva a cabo sobre un ordenador central, y la representación gráfica se prepara en el terminal. En un NC, la totalidad de la ejecución de la aplicación es local, lo que descarga al servidor.

Un NC no tiene dispositivo de almacenamiento permanente propio (como discos duros y flexibles o CD-ROM), en cambio todos los datos son gestionados centralmente, así como todas las aplicaciones, incluido el sistema operativo. Los factores de configuración se minimizan por lo que se alcanza una gran homogeneidad entre los ordenadores. La totalidad de la configuración se lleva a cabo de manera central y está fuera del alcance de los usuarios,

por lo que el esfuerzo de mantenimiento de estos sistemas es mínimo. Debido a su sencillez estos sistemas también son llamados "clientes delgados" (*thin clients*), en comparación con las estaciones de trabajo UNIX y los PC's (*fat clients*).

En principio, se simplifica la gestión de aplicaciones en estos sistemas, ya que se limita a instalar y configurar las aplicaciones sobre el servidor, y decidir qué clientes tienen acceso a qué aplicaciones. Dado que los clientes son homogéneos, no es necesario llevar a cabo instalaciones sobre múltiples plataformas, ni evaluar la factibilidad de la configuración sobre cada cliente. Sin embargo, actualmente no existen muchas aplicaciones que sean operativas, ni tampoco un estándar para la gestión de aplicaciones.

Actualmente existen dos especificaciones de ordenadores de red. La primera es de Sun, que propone una plataforma basada en tecnología Java (*JavaStation* [28]), y la segunda de Microsoft (*NetPC*) [9], donde la plataforma está basada en una versión especial de Windows.

Una limitación de este modelo es la dependencia absoluta del buen funcionamiento y potencia suficiente de la red y del servidor. Las redes cada vez son más fiables y disponen de un ancho de banda en aumento; el auténtico cuello de botella es la capacidad de los servidores. Un servidor de NC's soporta típicamente 20 a 25 clientes. Así, en una organización con cinco mil usuarios, la cantidad de servidores necesarios rondaría los 200, que a su vez tendrían que ser administrados centralmente, lo cual puede llevar a nuevos problemas de gestión.

Asimismo, el usuario carece de flexibilidad alguna dado que la configuración de los NC, incluyendo el acceso a las aplicaciones, está fuera de su alcance. Es debido a estas limitaciones que los clientes ligeros serán una solución restringida a áreas donde el usuario realice tareas mecánicas, como por ejemplo control y vigilancia, facturación en aeropuertos, etc. El grupo Gartner calcula que su cuota de mercado no excederá el 10%.

2.2 El enfoque del CERN

El Laboratorio Europeo para la Física de Partículas (CERN) tiene su sede en Ginebra. Se trata de un organismo fruto de la cooperación internacional, cuya misión es el estudio experimental de la física de altas energías (HEP, *High Energy Physics*). Fue fundado en 1954, y está emplazado en la frontera franco-suiza. Hay trece estados miembros, entre los que se encuentra España. Los idiomas oficiales del CERN son el inglés y el francés.

Los experimentos más importantes se llevan a cabo en aceleradores (o colisionadores) subterráneos circulares de gran diámetro, siendo el SPS (*Super Proton-Synchotron*) y el LEP (*Large Electron-Positron Collider*) los principales. En estos aceleradores se hacen chocar partículas de mismo tipo pero de carga opuesta entre sí (usualmente electrones e^- y positrones e^+), bajo condiciones variables. El tipo, forma y movimiento de las subpartículas resultantes es registrado por detectores subterráneos y posteriormente evaluado. La notable cantidad de datos producidos por estos experimentos obliga al CERN a disponer de avanzados medios de tratamiento de información.

Igualmente, se necesitan medios importantes para permitir el desarrollo de nuevos experimentos y sobre todo para la concepción y realización del nuevo acelerador protón-antiprotón LHC (*Large Hadron Collider*), que estará listo en el año 2005. Además, se calcula que se tendrá que hacer frente a una cantidad de información a registrar y evaluar que representará 200 veces la cantidad actual.

El CERN emplea directamente cerca de 3000 personas, un tercio de las cuales son científicos e ingenieros que trabajan con universidades y centros de investigación de alrededor del mundo. También se ofrecen alrededor de 100 becas anuales a estudiantes universitarios en los campos de física e ingeniería con el fin de promocionar el contacto con científicos de todo el mundo, y de la educación y formación en un entorno multinacional y multicultural.

El CERN está dirigido por un director general, elegido por los países miembros. La organización se estructura en doce divisiones, cada una con una determinada misión funcional. Las divisiones a su vez se componen de grupos y secciones.

Los programas de experimentos e investigación atraen a alrededor de 6'500 físicos, ingenieros y técnicos de otros centros y universidades, que a menudo alternan estancias en el

CERN con estancias en su lugar de origen, por lo que es importante la disponibilidad de medios que permitan la comunicación del laboratorio con otros centros.

Muchos desarrollos y descubrimientos del CERN han sido galardonados con importantes premios, como el Nobel por el descubrimiento de la fuerza nuclear débil, mientras que otros han llamado la atención de la sociedad en general, como la generación de átomos de antimateria (átomos de anti-hidrógeno compuestos por un antiprotón y un positrón) en 1996.

Otros desarrollos e inventos del laboratorio han sido adoptados por la industria en general. El ejemplo más notable es la creación de la *World Wide Web* en 1993, pensada inicialmente para el intercambio electrónico de documentos en la comunidad científica.

2.2.1 El CERN y la informática

En términos generales, el servicio informático en el CERN gira alrededor de tres ejes:

- La comunicación entre personal científico, dentro del CERN y con otros centros.
- El registro, tratamiento, evaluación y presentación de datos de experimentos actuales.
- la simulación y modelización de futuros experimentos.

La informática en el CERN es omnipresente, y todas las divisiones tienen su grupo informático. Asimismo, existe una división completamente dedicada a la informática: la división de Tecnologías de la Información, IT (*Information Technologies*).

La división IT presta servicios a todos los usuarios del laboratorio. Entre estos servicios se pueden resaltar los siguientes:

- instalación y mantenimiento de la red informática
- soporte general (instalación y configuración de hardware y software), y ventanilla de ayuda al usuario (*help desk*)
- gestión de ordenadores centrales

- soporte a usuarios de estaciones de trabajo y PC's y su integración en un entorno homogéneo
- implementación y desarrollo de arquitecturas para adquisición, registro y almacenamiento de datos, y sistemas de control para los experimentos
- desarrollo de software de análisis de datos y cálculo numérico

Dentro de la división IT, existe un grupo encargado de los sistemas informáticos de uso general, el grupo DIS (*Distributed Infrastructure Services*). Los sistemas soportados están basados mayoritariamente en Microsoft Windows 95 y NT para PC's de tamaño pequeño a mediano, y UNIX para estaciones de trabajo y servidores de tamaño mediano a grande. Existe una clara diferenciación entre las secciones dedicadas a Windows y las dedicadas a UNIX; a pesar de tener en muchos casos objetivos comunes no se desarrollan apenas proyectos en cooperación.

Administración centralizada

Existe una gran cantidad y variedad de PC's, estaciones de trabajo y servidores en el laboratorio debido a las múltiples áreas en los que son utilizados, desde la gestión de facturas al desarrollo de detectores de partículas.

El número de servidores y estaciones de trabajo UNIX en el laboratorio supera las 2400 unidades, y hay más de 8500 usuarios registrados, de los cuales una media de 5000 acceden diariamente.

No es política del CERN *imponer* máquinas, configuraciones o software estándar, ni de forzar su gestión centralizada. Pero dado que el personal de soporte es limitado, y que las diferencias entre los distintos sistemas UNIX son sensibles, se restringe el soporte por parte de la división IT a las plataformas UNIX más importantes y de mayor difusión.

Dentro del grupo DIS, existen cuatro áreas orientados a la integración de las estaciones de trabajo y servidores UNIX existentes en el laboratorio, en un entorno común, homogéneo y centralizado, tanto para su administración como para el acceso de usuarios:

- El mantenimiento de los *sistemas de ficheros* que permiten un acceso distribuido

e independiente de su localización a aplicaciones, datos compartidos y cuentas de usuario. El sistema de ficheros distribuido principal es AFS (*Andrew File System*, de Transarc Corporation) [2]. Una particularidad de este sistema de ficheros es su directorio unificado global, que permite que el camino de acceso a un determinado fichero dentro del directorio, independiente del emplazamiento físico de éste, sea el mismo desde cualquier máquina en cualquier parte del mundo.

Adicionalmente, se ofrecen los sistemas de ficheros NFS (*Networked File System*, de Sun Microsystems) [1], y de manera experimental DFS (*Distributed File System*) de la Open Systems Foundation. Dentro del grupo DIS existe una sección dedicada al mantenimiento de los sistemas de ficheros distribuidos, llamada DFS (*Distributed File Systems*).

- El sistema SUE (*Standard Unix Environment*, entorno estándar bajo Unix) [35] se define para ayuda a la centralización y automatización de la instalación, configuración y mantenimiento de máquinas UNIX del laboratorio. SUE es desarrollado y mantenido por el grupo DIS. A través de SUE se puede configurar el acceso a la red, la sincronización de relojes, el correo electrónico, entornos de seguridad utilizados etc., además de programar la ejecución automática y regular de diversas tareas de administración. Asimismo permite elaborar inventarios sobre las características de las estaciones de trabajo existentes.

- HEPiX (*UNIX in High Energy Physics*) es un proyecto cuyo fin es ofrecer un ambiente de trabajo común independiente del tipo y localización de la estación de trabajo usada en cada momento. Este proyecto se desarrolla en colaboración con otros centros HEP. Las actividades principales son la definición de procedimientos y estándares para la gestión de cuentas, y entornos de usuario gráficos y de texto.

- ASIS (*Application Software Installation System*) es el sistema de distribución de aplicaciones en entorno UNIX. Es desarrollado y gestionado por la sección OSE (*Open Systems Environment*) del grupo DIS. Este sistema ofrece un repositorio multiplataforma y multiversión en el cual se almacenan y distribuyen aplicaciones (programas, librerías y utilidades) listas para su uso a las estaciones de trabajo y servidores del laboratorio. El contenido del repositorio es replicado por algunos centros HEP.

Cada máquina en el laboratorio tiene su propio administrador dentro su división y grupo.

En las estaciones de trabajo pequeñas el usuario final hace de administrador. Los servicios ofrecidos no son compulsorios, por lo que la responsabilidad de mantenimiento y gestión se distribuye entre los administradores y el grupo DIS.

Otro campo de actividad del grupo DIS es la gestión y el mantenimiento de los ordenadores centrales de acceso público, genéricamente denominados PLUS (*Public Login User Servers*, servidores públicos para acceso de usuarios). Estos sistemas, basados en agrupamientos (*clusters*) de estaciones de trabajo UNIX, ofrecen un acceso interactivo a gran número de aplicaciones y herramientas de uso general a los usuarios registrados. Adicionalmente, se gestionan *clusters* de acceso privado por grupos de trabajo formados por ejemplo en experimentos, denominados WGS (*Work-Group Servers*, servidores para grupos de trabajo).

Arquitectura CUTE

Para la estandarización de la gestión de ordenadores de acceso público, estaciones de trabajo y PC's, distribución de aplicaciones y cuentas de usuario se definen las arquitecturas CUTE y NICE.

La arquitectura CUTE (*Common Unix and X-Terminal Environment*, entorno común para Unix y terminales X[6]) se establece para el mundo UNIX. La arquitectura NICE (*Network Integrated Computing Environment*, entorno integrado de computación en red) [18] es exclusiva a los sistemas PC basados en Windows, pero funcionalmente análoga a CUTE.

La arquitectura CUTE se muestra en la figura 1.

Esta arquitectura se asienta en la red interna del CERN, cuyo núcleo se constituye de subredes de alta velocidad (basadas en tecnología ATM y FDDI), y que presta servicio a todas las divisiones. Sobre esta red interna se implanta el sistema de ficheros distribuido AFS. En la arquitectura se integran los distintos servidores PLUS y WGS, las estaciones de trabajo y los terminales X del laboratorio. Los PC's basados en Windows tienen acceso a aplicaciones X-Windows mediante un emulador X. Las cuentas de usuario disponen de un servicio regular de respaldo (*backup*). Los datos provenientes de experimentos son accesibles a través de un sistema de almacenamiento masivo (CORE). Las aplicaciones en el repositorio ASIS son servidas por servidores de ficheros dedicados; estas aplicaciones son

[6] un terminal X es un terminal gráfico para conexión con máquinas UNIX bajo protocolo X-Windows

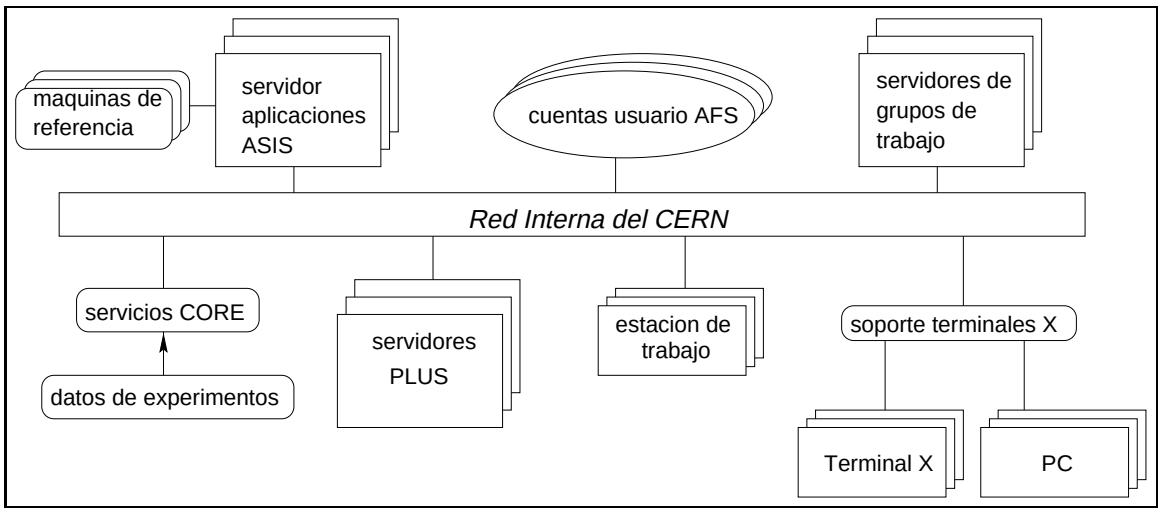

Figura 1: arquitectura CUTE

compiladas y configuradas por cada plataforma en máquinas llamadas "de referencia", que representan un entorno estándar por cada plataforma.

La ventaja de esta arquitectura es la transparencia de acceso a recursos, como cuentas de usuario y aplicaciones. Además mediante SUE y HEPiX, tanto los usuarios como los administradores tienen un entorno homogéneo de trabajo, sea cual sea la localización física o la arquitectura de la estación de trabajo utilizada.

El punto débil de la arquitectura es la red. Si ésta, o el sistema de ficheros, fallan, entonces no se puede acceder ni a cuentas de usuario ni a aplicaciones en los servidores centrales. Es por lo cual algunos grupos, sobre todo pertenecientes a experimentos, se mantienen al margen de CUTE.

2.3 Distribución de Aplicaciones en el CERN: el sistema ASIS

ASIS es el acrónimo de *Application Software Installation System*, sistema instalador de aplicaciones "software". Es un servicio ofrecido desde 1994 por la sección DIS/OSE, cuyo responsable es el Dr. Philippe Defert.

ASIS es un servicio pilar de la arquitectura CUTE. Este servicio incluye:

- un repositorio multiplataforma y multiversión que contiene aplicaciones de uso extendido, junto a su documentación, en formato listo para su uso por un gran número de máquinas.
- un conjunto de herramientas para la generación y configuración de aplicaciones, y para el acceso y manipulación del repositorio.

Las aplicaciones gestionadas son de tipo comercial, o desarrolladas por el CERN, o bien de libre distribución. Actualmente existen alrededor de 600 aplicaciones en el repositorio del CERN.

Estas aplicaciones se ofrecen a los usuarios configuradas y listas para su uso. Junto a los binarios se incluye la documentación correspondiente (manuales, documentación técnica, ejemplos de uso, etc.).

Gran parte del contenido del repositorio es accesible gratuitamente por los centros de física de altas energías (*HEP, high energy physics*), y por centros educativos asociados.

Existen ocho réplicas actualizadas regularmente del repositorio fuera del laboratorio; por ejemplo en CASPUR-INFN (*Instituto Nazionale de Fisica Nucleare*, Roma, Italia), y en el Reino Unido por *Rutheforth Laboratories*.

La EPFL, *École Politécnique Féderale de Lausanne*, Escuela Politécnica Federal de Lausanne (Suiza), recientemente se ha unido al desarrollo de ASIS. Mantienen independientemente del CERN un repositorio, donde generan y distribuyen sus propias aplicaciones.

2.3.1 Usuarios de ASIS

Hay cuatro tipos de usuarios del sistema ASIS: Los *usuarios finales*, los *administradores de estaciones de trabajo*, los *gestores de producto*, y los *gestores del repositorio*.

- Al *usuario final* es a quién va orientado el servicio prestado por el sistema. Es quien hace uso de las aplicaciones del repositorio, y quien solicita la disponibilidad de nuevos productos o nuevas versiones de éstos. En el CERN, la mayoría de los usuarios finales son físicos, científicos o ingenieros que no tienen por qué tener conocimientos de

informática. Se calcula que hay alrededor de 5000 usuarios de aplicaciones contenidas en ASIS.

- El *administrador de estación de trabajo* es el responsable de una o varias estaciones de trabajo (o de un *cluster*) que tienen acceso al sistema ASIS. Existen alrededor de 2000 estaciones de trabajo con acceso a ASIS. El administrador se encarga de seleccionar un conjunto de aplicaciones del repositorio y de hacerlas accesibles a los usuarios. No puede modificar el contenido del repositorio.

- El *gestor de producto* (*product maintainer*) es el responsable de uno o varios productos distribuidos por ASIS. Se encargará de compilar, configurar y probar las aplicaciones requeridas por los usuarios y de introducirlas en el repositorio. Asimismo, contestará dudas y sugerencias de los usuarios sobre estas aplicaciones. Existen alrededor de treinta gestores de producto en el CERN.

- Los *gestores de repositorio* (*repository managers*), mantienen el sistema en sí. Son responsables del funcionamiento correcto de todas las aplicaciones para consulta y manipulación del sistema. También son responsables de todos los productos contenidos en el repositorio que no tienen gestor de producto explícito. En el CERN existen dos a tres gestores de repositorio, que son los mismos que llevan a cabo el desarrollo del sistema ASIS.

Al margen de las aplicaciones ofrecidas por ASIS, los administradores podrán seguir instalando por su cuenta otras aplicaciones en "sus" estaciones de trabajo.

2.3.2 Plataformas en ASIS

Normalmente, cada aplicación distribuida por ASIS está disponible para las plataformas soportadas oficialmente, y sobre cada una de ellas está configurada de manera similar, con lo que se favorece la homogeneidad entre plataformas.

Aparte de las plataformas soportadas, existen plataformas cuyo uso es aceptado pero no soportado oficialmente por el CERN (por ejemplo, las estaciones que operan bajo el sistema operativo de dominio público Linux). Se distribuirán aplicaciones, sin ninguna clase de garantía, para estas plataformas siempre y cuando exista amplia demanda por parte de los usuarios.

El tipo y cantidad de plataformas usadas varía en función de la demanda de los usuarios y de las directivas de alto nivel. El repositorio se "congela" para las plataformas que dejan de ser activas; es decir que no sufre cambio ni modificación alguna. De esta manera se permite que estaciones de trabajo cuyo hardware o sistema operativo no puede ser actualizado sigan teniendo acceso a ASIS. Cuando una plataforma se queda definitivamente obsoleta y en desuso, se borran las aplicaciones del repositorio para esta plataforma para liberar recursos.

2.3.3 Mecanismo de distribución

El sistema ASIS está construido basándose en la existencia de un sistema de ficheros que permita el acceso directo (*on-line*) de los clientes al repositorio a través de la red.

Las aplicaciones se distribuyen por servidores de ficheros dedicados, que son los mismos que acogen el repositorio. El sistema ASIS no se encarga de forma directa del tipo, ni de la potencia y cantidad de servidores agrupados necesarios para hacer accesibles las aplicaciones. Estas tareas son propias del modelo de sistema de ficheros en red utilizado; en el CERN su mantenimiento corre a cargo de la sección DIS/DFS.[7]

El repositorio ASIS es accesible en el CERN a través de los sistemas de ficheros AFS y NFS, existiendo también un acceso experimental vía DFS. ASIS solamente hace uso de características comunes a los sistemas de ficheros bajo UNIX, por lo que es, generalmente, independiente del tipo concreto de sistema de ficheros utilizado.

Los repositorios sobre AFS y DFS se basan en el mismo repositorio físico, mientras que para NFS existe una réplica físicamente independiente que es actualizada regularmente. La replicación de repositorios físicamente independientes sí es gestionada por ASIS.

2.3.4 Organización lógica

La distribución de las aplicaciones se efectúa desde un repositorio, en el cual se almacenan centralmente las aplicaciones, hacia las estaciones de trabajo a las que se conectan los usuarios.

[7] sin embargo, los gestores de repositorio colaboran a la hora de equilibrar la distribución de la carga de los servidores.

familia	contenido
ASIS	herramientas para el acceso y manipulación del repositorio
CERN	utilidades y herramientas internas al CERN
CERNLIB	librerías de análisis y cálculo
GNU.EDIT	editores de texto y derivados
GNU.LANG	compiladores de C, C++, FORTRAN, Java, etc.
GNU.SYS	herramientas de administración de sistemas
GNU.MISC	diversas aplicaciones bajo licencia GNU
HEPiX	aplicaciones y configuraciones HEPiX
LIC	software comercial licenciado
MAIL	herramientas para correo electrónico
MISC	software de origen diverso
PERL	lenguaje PERL y herramientas
TCL	lenguaje TCL y herramientas
TEX	sistema de preparación de documentos LaTeX
X11	entorno gráfico X-Windows

Tabla 2: familias en el repositorio del CERN

En el repositorio ASIS, las aplicaciones están organizadas en *Familias*, *Productos* y *Versiones*.

- Una *familia* agrupa un conjunto de aplicaciones que tienen similar funcionalidad o un origen común. Por ejemplo, la familia CERNLIB contiene las librerías de cálculo numérico y análisis de datos del CERN, la familia X11 contiene aplicaciones y librerías para el sistema de ventanas X-Windows, y la familia GNU.LANG contiene los compiladores de GNU. La lista completa de familias en el CERN se representa en la tabla 2.

- Una familia está compuesta de *productos*. Representan todas las versiones de una aplicación. Así, por ejemplo, el producto `gcc`, perteneciente a la familia GNU.LANG, es el compilador GNU de C, y `pine` es el conocido lector de correos, clasificado dentro de la familia MAIL.

- Un producto es compuesto de una o varias *versiones*. Una versión representa una aplicación a nivel lógico, y se diferencia de versiones anteriores por haber sufrido

correcciones y/o modificaciones. Cada versión tiene asociado un identificador, usualmente un número de versión. Así, la versión en uso actual del editor emacs es la 19.34, mientras que una versión antigua es por ejemplo la 18.58.1. Las versiones son completamente autónomas entre sí.

Una aplicación quedará identificada unívocamente por el nombre de la familia, el nombre del producto y el identificador de versión, en la forma *Familia/Producto-Versión* (por ejemplo, `GNU.EDIT/emacs-20.1beta`).

Paquetes

Cada versión de un producto o aplicación consiste en un conjunto de ficheros, organizados jerárquicamente en directorios: ficheros binarios ejecutables, librerías software, documentación, etc. Este conjunto de ficheros se llama *paquete*.

Dado que ASIS es un sistema multiplataforma, existe un paquete por cada plataforma y aplicación, con los ficheros específicos a esta plataforma, ya que normalmente los ficheros binarios no son portables entre las plataformas.

Paquetes específicos y paquetes compartidos

Algunos tipos de ficheros, como la documentación o librerías de alto nivel sin embargo son idénticos a todas las plataformas. Por lo tanto éstos forman una parte común a todas las plataformas y se emplazan en un paquete especial, llamado el *paquete compartido o común* (*share*). Los ficheros de una aplicación específicos a una plataforma, por contener binarios compilados y adaptados a características especiales de esta plataforma, se emplazan en un *paquete específico* a esa plataforma.

Por ejemplo, el núcleo del sistema de preparación de documentos LaTeXconsiste de una docena de ficheros ejecutables (`tex`, `latex`, `mf`,...), y cientos de directorios con más de mil ficheros de macros y de tipografía, además de una multitud de ficheros de documentación. Los ficheros ejecutables irán en los paquetes específicos a cada plataforma, mientras que los ficheros de macros, tipografía y documentación irán al paquete compartido dado que son independientes de la plataforma. De esta manera, se reduce el espacio ocupado por cada aplicación.

Ciertas aplicaciones no tienen parte compartida, y otras en cambio consisten *exclusivamente* de una parte compartida; por ejemplo aplicaciones escritas en lenguajes de alto nivel, como Perl, Java o TCL, que no han de ser compiladas y que por lo tanto son independientes de la plataforma.

2.3.5 Emplazamiento de las aplicaciones

Las aplicaciones del repositorio se emplazan dentro del repositorio en dos áreas, llamados *área de paquetes* y *área de Producción*. El área de paquetes contiene todas las aplicaciones del repositorio. En cambio, el área de producción ofrece una preselección de aplicaciones que son las accedidas por defecto por las estaciones de trabajo.

Area de paquetes

En éste área se acomodan todas las aplicaciones del repositorio, para todas las plataformas disponibles.

Este área está organizado jerárquicamente según familias, productos y versiones. Cada aplicación estará emplazada en un dominio independiente de las demás aplicaciones. Los niveles de jerarquía se establecen mediante *directorios* del sistema de ficheros. Esta estructura de directorios se aprecia en la parte superior de la figura 2:

- *Familias*: directorio por cada familia, dentro del cual se sitúan todas las aplicaciones pertenecientes a la familia
- *Aplicaciones*: por cada aplicación, existe un directorio, en el cual se almacenan todos los paquetes pertenecientes a cada plataforma de la aplicación.
- *Plataformas*: los paquetes específicos se almacenan dentro de un directorio de mismo nombre que la plataforma, mientras que el paquete compartido se almacena en un directorio llamado *share*.
- *Directorio prefijo*. Para ser ejecutada correctamente, la aplicación ha de ser emplazada en ese directorio en la máquina cliente. Si el prefijo consiste en varios directorios, éstos se separan mediante un punto. Así, si una aplicación ha sido configurada para

residir en el directorio `/usr/local`, el directorio prefijo se almacena como `usr.local`. Otro directorio prefijo es `/cern`, raíz de los productos de la familia CERNLIB.

- *Directorios y ficheros* de los que se compone el paquete.

En la figura 2 se muestra a nivel de ejemplo, el emplazamiento de las aplicaciones `GNU.EDIT/lyx-0.12` y `GNU.LANG/gcc-2.5.2` para la plataforma Silicon Graphics IRIX 5.2, cuyo identificador es `sgi-52`. Se aprecia que las dos aplicaciones residen en jerarquías independientes. El directorio de acceso dentro del área de paquetes a `GNU.EDIT/lyx-0.12` para la plataforma `sgi-52` es `GNU.EDIT/lyx-0.12/sgi-52`.

Dado que la organización del área de paquetes se fundamenta en dar acogida a todas las aplicaciones, pero no refleja la estructura de ejecución, no se puede usar de manera inmediata para ejecutar las aplicaciones. A este fin existe una herramienta llamada `ASISupdate`, mediante la cual el administrador de estación de trabajo seleccionará el conjunto de aplicaciones que desea hacer accesible sobre su máquina.

Area de producción

En el área de producción se incluye una selección de aplicaciones, lista para su uso inmediato en estaciones de trabajo. Las estaciones de trabajo accederán por defecto a las aplicaciones de este área.

Esta selección es efectuada por los gestores de producto, que deciden qué versiones de sus productos se incluyen en éste área. Normalmente, se incluye una sola versión de cada producto en el área. Esta versión es la actualmente soportada del producto. En algunos casos, se tienen varias versiones de un mismo producto "en producción". Por ejemplo, para facilitar la transición gradual a una nueva versión durante algún tiempo pueden convivir dos o más versiones "en producción".

El área de producción está diseñado para ser enlazado directamente por una estación de trabajo, para que forme parte de su jerarquía del sistema de ficheros. La estructura por lo tanto refleja los distintos directorios prefijo y los directorios propios a las aplicaciones conteniendo los ficheros tal y como han de ser instalados para su correcto funcionamiento. Esto significa que las aplicaciones se integran en un dominio común (por cada directorio

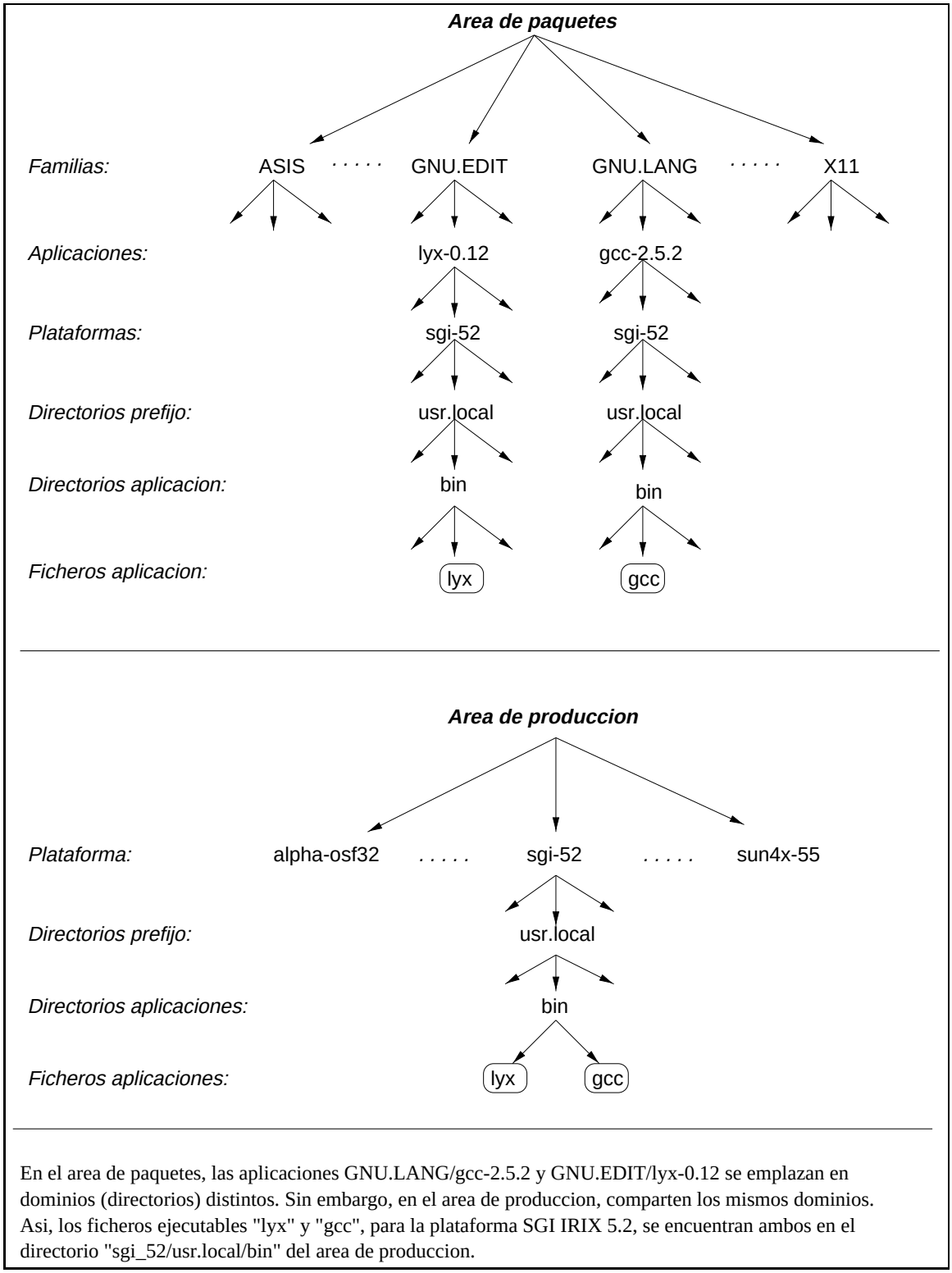

Figura 2: estructura de las áreas del repositorio

prefijo) dentro del sistema de ficheros, como se muestra en la parte inferior de la figura 2. Al enlazar el área de producción se accederá a *todas y cada una* de las aplicaciones del área.

En UNIX, los directorios utilizados por las aplicaciones suelen seguir ciertas pautas organizativas. Así, las aplicaciones dentro del directorio prefijo `/usr/local` suelen emplazar sus ejecutables en el subdirectorio `bin`, las librerías objeto en el subdirectorio `lib`, la documentación en los subdirectorios `doc` y `man`, etc. El formato exacto recomendado en el CERN se describe en [36].

Una ventaja de esta convención es la comodidad para el usuario: podrá encontrar por ejemplo los ejecutables bajo el directorio `/usr/local/bin` sin tener que rastrear extensas jerarquías de directorios. Asimismo, se favorece la comunicación entre aplicaciones; por ejemplo, al definirse un directorio común para la documentación, un lector de documentos[8] sabrá donde encontrar la documentación del software instalado, un compilador encontrará todas las librerías en el mismo lugar, etc.

Sin embargo, presenta un inconveniente. Dado que en un mismo directorio los nombres de fichero han de ser únicos, dos aplicaciones que contengan ficheros del mismo nombre en un mismo directorio no pueden estar simultáneamente "en producción", de lo contrario se produce un *conflicto* en el espacio de nombres. En repositorios grandes, con cientos de aplicaciones donde cada una de ellas puede contener cientos de ficheros, éste es un problema habitual.

Cada plataforma dispone de su propio área de producción. El área de producción de una plataforma es independiente del de las otras, si bien se procura que todas las plataformas dispongan de las mismas aplicaciones en producción para homogeneizar el acceso a las aplicaciones por parte de los usuarios, independientemente del tipo de estación de trabajo utilizada.

Debido a que el área de producción sufre un acceso directo y continuo por parte de las estaciones de trabajo, ha de ser emplazado en servidores de alto rendimiento.

La razón de existencia del área de producción viene dada por la necesidad de poder ofrecer un acceso preconfigurado a las aplicaciones más importantes del repositorio, de manera

[8] como el lector de páginas `man`

directa y libre de conflictos.

Además, en las estaciones de trabajo que enlazan este área son los gestores de producto, y no el administrador, quienes determinan de manera centralizada el conjunto de aplicaciones disponibles a los usuarios.

Relación entre ambas áreas

Las aplicaciones que pasan a estar "en producción" se mueven del área de paquetes al área de producción. Para ahorrar en espacio de disco, se borran los ficheros del área de paquetes; para mantener la integridad del área de paquetes, se sustituye cada fichero por una referencia al fichero en su nuevo emplazamiento[9]. Cuando una aplicación abandona el área de producción los ficheros se volverán a mover a su lugar de origen dentro del área de paquetes.

Al llevar una aplicación al área de producción hay que detectar posibles conflictos en el espacio de nombres; el no detectar esta situación puede llevar a la corrupción del área.

2.3.6 Base de datos ASIS

La función principal de la base de datos del sistema ASIS es la de establecer una correspondencia entre los ficheros en los áreas del repositorio y los paquetes. También se almacena la siguiente información:

- la lista de las familias, productos y versiones junto a su descripción respectiva
- el estado y la evolución de todas las versiones de un producto para cada plataforma
- información de soporte por cada producto; en concreto, la dirección de contacto del autor y del gestor del producto.
- la lista de plataformas contenidas en ASIS, junto a su estado (soportada o no, activa o no)

[9] esto se realiza a través de ficheros de tipo *enlace simbólico*

2.3.7 Acceso al repositorio

Los usuarios tienen acceso a las aplicaciones contenidas en ASIS, a su descripción y a su documentación.

Acceso a la base de datos y la documentación

A diario, se genera una copia de la base de datos en formato HTML, que permite al usuario consultar y buscar información según diversos criterios. Adicionalmente, se generan enlaces hacia los documentos y manuales distribuidos en los paquetes. Estas páginas son accesibles en la red interna del CERN mediante el uso de cualquier navegador de *web*.

Existe también una herramienta llamada `ASISinfo`, accesible desde cualquier estación de trabajo con acceso a ASIS, que sirve para consultas abreviadas desde la línea de comandos.

Acceso a las aplicaciones

El modelo de acceso de una máquina cliente a las aplicaciones se puede clasificar en tres categorías: *directo*, *semidirecto* e *híbrido*.

Acceso directo

El acceso directo se efectúa a través del área de producción. Se habilita la estación de trabajo para acceder a éste área, en un proceso inicial conocido como "montaje", en el cual se enlaza el área de producción, emplazado físicamente en un servidor central, dentro de la jerarquía del sistema de ficheros del cliente. El cliente verá el área de producción como si perteneciese a su propia máquina, y accederá a todas y cada una de las aplicaciones del área de producción. Todos los cambios en el área de producción se reflejarán de manera automática y transparente en la estación de trabajo.

Esta categoría de acceso se puede clasificar como técnica *push*, dado que el proceso de actualización es controlado desde los servidores.

La ventaja de este enfoque es que se decide centralmente (por los gestores de producto) qué aplicaciones son distribuidas a las estaciones de trabajo, siendo innecesario una intervención

local sobre la máquina cliente. El consumo en recursos locales, en forma de espacio de disco, es nulo.

La desventaja es que ofrece un grado de libertad nulo al cliente, impediéndole tener una configuración propia sobre qué aplicaciones y qué versiones desea acceder y cuáles no. Además, el acceso está condicionado al funcionamiento de los servidores y del estado de la red: si los servidores o la red están caídos las aplicaciones no estarán disponibles; si la red está saturada el acceso será muy lento. Otra desventaja es que no es posible instalar localmente aplicaciones en las jerarquías de directorios reclamadas por el área de producción.

Acceso semidirecto

Si se desea acceder a todas las aplicaciones del repositorio, es necesario tener acceso al área de paquetes del repositorio vía el sistema de ficheros distribuido. Además se requiere construir una correspondencia entre el entorno de ejecución y las aplicaciones del área de paquetes, dado que éste área está construido para acoger todas las aplicaciones y no refleja la estructura necesaria para la ejecución de éstas.

A través de la herramienta `ASISupdate` el administrador de la estación de trabajo selecciona el conjunto de aplicaciones que desea hacer accesible a los usuarios. La herramienta creará enlaces[10] por cada uno de los ficheros de las aplicaciones, desde el repositorio hacia su lugar correspondiente dentro de la jerarquía del sistema de ficheros local, realizando la integración entre las distintas aplicaciones en un dominio común por cada directorio prefijo, de manera similar a como se hace en el área de producción.

Se permite que la jerarquía de ficheros sea compartida indistintamente por aplicaciones del repositorio y aplicaciones locales, siempre y cuando no se produzcan conflictos de nombres.

Por cada aplicación se puede solicitar que sea copiada físicamente al sistema de ficheros local, en vez de ser enlazada, con lo que se logra independencia respecto a los servidores. Al poderse elegir por cada aplicación el tipo de acceso deseado (local o por red), se logra equilibrar la necesidad de independencia con el consumo en recursos locales, por ejemplo, copiando solamente aplicaciones importantes y accediendo al resto a través de la red.

Esta categoría de acceso es de tipo *pull*; el cliente solicita del servidor un conjunto de

[10] de tipo *enlace simbólico*

aplicaciones.

La ventaja de este tipo de acceso es que la flexibilidad es muy alta. Sin embargo, obliga que la selección y actualización de las aplicaciones disponibles sea llevada a cabo manualmente por parte del administrador.

Acceso híbrido

La mayoría de los usuarios están interesados en acceder las aplicaciones "en producción", pero con algunas variaciones. Por ejemplo, para una máquina usada en procesos de compilación será importante la estabilidad del compilador, por lo que se fijará una versión determinada de éste; las demás aplicaciones serán las disponibles "en producción".

La herramienta `ASISupdate` permite combinar ambos tipos de técnicas, *pull* y *push*. Las aplicaciones accedidas serán las correspondientes del área de producción, excepto en las que se solicita una versión específica, o genéricamente la última versión disponible de un determinado producto. También es posible anular el acceso a aplicaciones concretas.

En el modelo híbrido, la herramienta se debe ejecutar cada vez que se produce un cambio en el repositorio para recalcular los enlaces hacia las aplicaciones que actualmente satisfacen la configuración seleccionada.

Un problema que se presenta ocasionalmente es el siguiente. Dado que el conjunto de aplicaciones "en producción" varía, aunque éste conjunto de aplicaciones no presente conflicto alguno, al unirlo con las aplicaciones seleccionadas por el usuario pueden aparecer conflictos en algún momento. Esto puede dejar la estación de trabajo en un estado inconsistente, dado que los conflictos se detectan cuando se generan físicamente los enlaces y/o copias hacia el repositorio.

Acceso indirecto

También existe la posibilidad de acceder al repositorio a través de un servidor `ftp`[11]; de esta manera se permite acceder al contenido del repositorio por nodos externos que no disponen de otro medio de acceso. El acceso se efectúa al área de paquetes del repositorio basado en AFS. Sin embargo, no se ofrece ninguna herramienta para automatizar este

[11] `ftp`, *file transfer protocol* es un protocolo para la transmisión remota de ficheros

acceso.

Dependencias en aplicaciones

Un administrador de estación de trabajo puede acceder libremente a todas y cada una de las aplicaciones del repositorio. A la hora de establecer la configuración, deberá prestar especial atención a que ésta sea coherente; es decir, que las dependencias entre aplicaciones sean satisfechas, asimismo como las dependencias de aplicaciones hacia el entorno local de ejecución.

Actualmente, la configuración se establece basándose en la experiencia del administrador. El problema estriba en que un administrador no puede tener la experiencia necesaria para determinar todas las interactuaciones de las todas las aplicaciones, especialmente si éstas son recientes. Esto deriva en que los administradores tengan que probar manualmente las aplicaciones, revisando iterativamente la configuración hasta que sea completa y coherente.

Una limitación del sistema ASIS es que no existe ningún modelo o mecanismo que permita a un gestor de producto definir las relaciones entre su aplicación y otras, o describir el entorno de ejecución requerido. Esta información podría ser consultada por un administrador de estación de trabajo para establecer una configuración coherente.

2.3.8 Modelo de gestión de aplicaciones y del repositorio

Modelo de procesado de "software"

El ciclo de vida de distribución de una aplicación, desde que es publicada, pasando por su configuración y su entrega a los usuarios hasta que es retirada, ha sido cuantificado heurísticamente en una serie de pasos discretos. Estos pasos han sido modelizados mediante un diagrama de transición de estados, llamado el modelo de procesado de "software" (*Software Processing Model*) [12].[12] Este modelo se representa gráficamente en la figura 3.

El modelo define los estados siguientes:

[12] estrictamente hablando, este modelo debería llamarse *Software Distribution Processing Model* ya que analiza solamente la fase de distribución, pero no la de desarrollo.

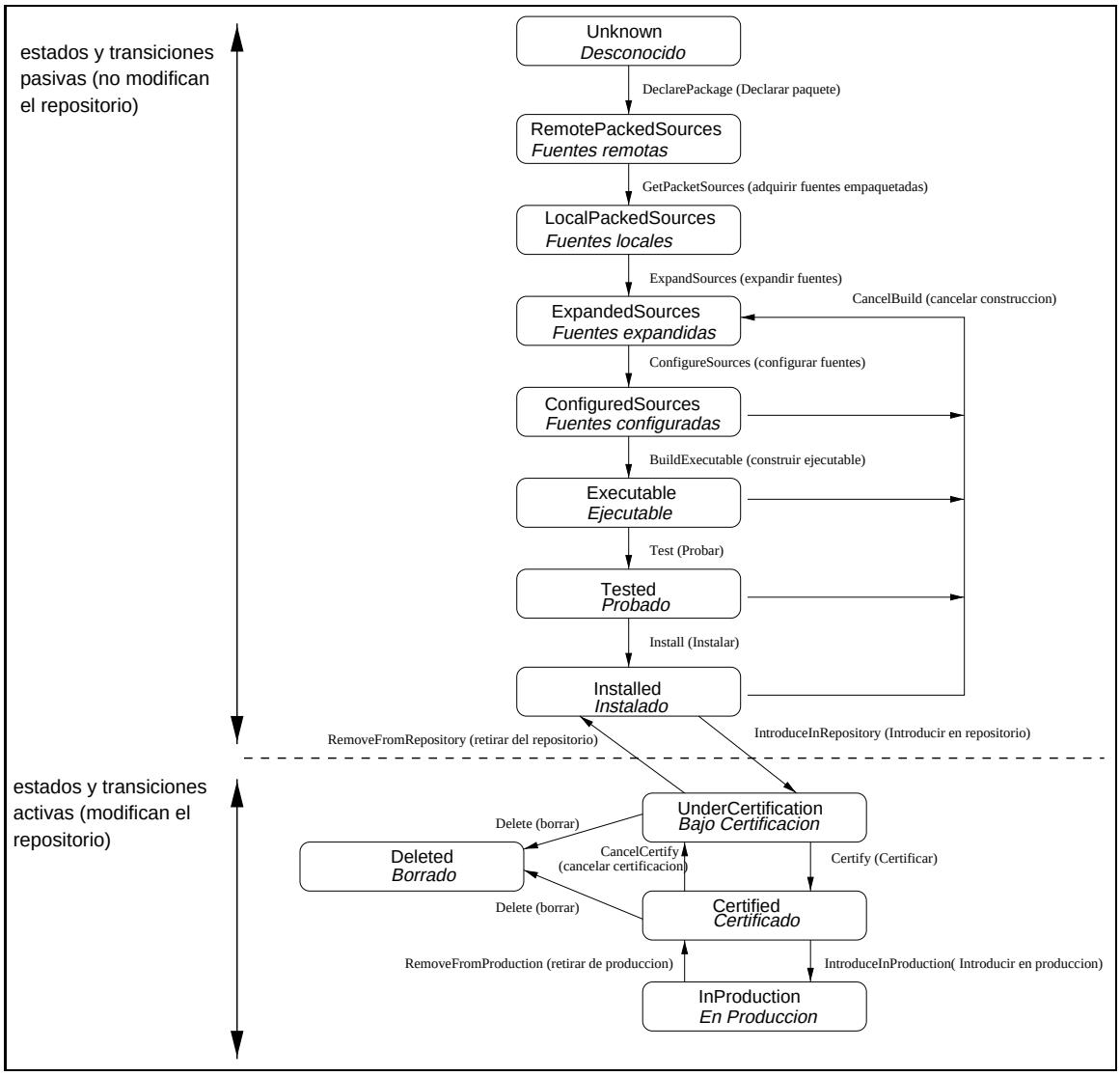

Figura 3: modelo de procesado de software

- **Desconocido (*Unknown*):** No se tiene conocimiento de la aplicación.
- **Fuentes empaquetadas remotas (*RemotePackedSources*):** Se conoce el lugar de donde obtener la aplicación en formato fuente empaquetado[13].
- **Fuentes empaquetadas locales (*LocalPackedSources*):** se dispone de la aplicación en formato fuente empaquetado.
- **Fuentes expandidas (*ExpandedSources*):** las fuentes han sido desempaquetadas.
- **Fuentes configuradas (*ConfiguredSources*):** las fuentes han sido configuradas específicamente para cada plataforma, y están listas para ser compiladas.
- **Ejecutable (*Executable*):** la aplicación ha sido compilada y se han generado los binarios.
- **Probado (*Tested*):** las pruebas distribuidas con la aplicación han sido ejecutadas satisfactoriamente.
- **Instalado (*Installed*):** los ficheros ejecutables, sus librerías y demás componentes han sido configurados para su uso operativo y conforman junto a la documentación un conjunto listo para ser utilizado.
- **Bajo Certificación (*Under Certification*):** la aplicación está accesible a un grupo de usuarios reducido para ser evaluada.
- **Certificado (*Certified*):** la aplicación está disponible a todos los usuarios después de haber superado la fase de certificación.
- **En Producción (*InProduction*):** la aplicación se incluye en el entorno de usuario de por defecto.
- **Borrada (*Deleted*):** la aplicación ha sido retirada.

En un entorno multiplataforma, las transiciones entre los estados *Fuentes Configuradas* y *En Producción* serán independientes por cada plataforma. Esto significa que una misma aplicación puede estar en un estado distinto por cada plataforma. En caso de que una aplicación contenga una parte compartida (*share*) a todas las plataformas, ésta será tratada como una plataforma más.

[13]Este estado es relevante cuando la aplicación se distribuye desde un nodo internet remoto.

Las transiciones entre los estados se pueden agrupar de acuerdo con su impacto sobre un repositorio:

- **pasivas**, no modifican el contenido del repositorio, pero sí la aplicación. Es el caso para todas las transiciones entre los estados *Desconocido* e *Instalado*. El resultado es un objeto que contiene la aplicación lista para su uso.
- **activas**, modifican el contenido del repositorio, sin modificar la aplicación en sí. Lo son todas las transiciones entre los estados *Instalado* y *En Producción*.

Adicionalmente a las transiciones activas, existen tres operaciones para llevar a cabo "parches" sobre aplicaciones en el repositorio. Estas son ejecutadas excepcionalmente en caso de detectarse una anomalía o un error en una aplicación, cuando el error se puede localizar en un fichero concreto. Las operaciones son:

- **añadir un fichero a un paquete (*AddFileToPackage*)**
- **borrar un fichero de un paquete (*RemoveFileFromPackage*)**
- **reemplazar el contenido de un fichero de un paquete (*ReplaceFileInPackage*)**

En el sistema ASIS, un gestor de producto puede efectuar todas las transiciones, asimismo como las operaciones "parche", usando una herramienta llamada `happi` (*Heterogeneous Automated Product Processor and Installer*, procesador e instalador automatizado heterogéneo de productos). Esta herramienta puede ser usada simultáneamente por los gestores de producto para generar aplicaciones y actualizar el repositorio.

Entorno automatizado de generación

La fase de compilación y configuración de una aplicación, correspondientes a la ejecución de las transiciones pasivas, se lleva a cabo fuera del repositorio, con lo cual las aplicaciones hasta ese punto quedan fuera del acceso de los usuarios.

La herramienta `happi` permite automatizar la generación de aplicaciones sobre múltiples plataformas, para lo que se debe definir los pasos a seguir para instalar una aplicación.

Normalmente, las aplicaciones vienen con un manual de instalación o incluso con utilidades que automatizan parte de la instalación. El gestor de producto describe en un "fichero de configuración" por cada transición *pasiva* la secuencia exacta de operaciones que deben de ser ejecutadas satisfactoriamente para pasar de un estado del diagrama a otro; se ofrecen librerías de utilidades y procedimientos de instalación predefinidos para facilitar el trabajo.

El fichero de configuración (implantado en PERL [37]) es específico a cada aplicación y se almacena en la base de datos del sistema.

Este fichero contiene el conocimiento necesario para generar la aplicación sobre cada plataforma, y se puede consultar y adaptar para la generación de nuevas versiones, o de aplicaciones cuyo proceso de generación sea similar. También se permite definir ficheros de configuración genéricos a todas las versiones de un producto o a una familia entera.

La herramienta permite interrumpir la generación de una aplicación en cualquier estado y reanudarla posteriormente.

La generación de una aplicación finaliza al llegar ésta al estado *Instalado*. Por cada plataforma se define un paquete listo para ser llevado al repositorio.

Las transiciones hasta el estado *Fuentes expandidas* son ejecutadas sobre un servidor central, mientras que las transiciones entre *Fuentes expandidas* e *Instalado*, que son específicas a cada plataforma, se ejecutan concurrentemente sobre máquinas dedicadas, las *máquinas de referencia*, de las cuales existe una por cada plataforma.

Las máquinas de referencia tienen una configuración lo más estándar y genérica posible para minimizar las dependencias que puedan surgir en el proceso de instalación y compilación hacia una configuración específica. Muchos procesos de configuración e instalación de aplicaciones asumen que la máquina sobre la cual se genera la aplicación será la misma sobre la cual se ejecutará, lo cual no es cierto en el caso de ASIS. No se pretende imponer uniformidad en la configuración de las estaciones de trabajo, pero sí en el acceso al software.

Sin embargo, dado que existe una notable variedad en las configuraciones de las estaciones de trabajo debido al gran número de éstas, ocasionalmente es inevitable hacer suposiciones sobre el entorno de ejecución en la estación de trabajo cliente. El gestor de producto no puede definir formalmente qué requisitos y dependencias ha detectado que condicionen el correcto y eficaz funcionamiento de la aplicación generada, debido a la inexistencia de un modelo de definición de dependencias.

Modificación del repositorio

La modificación del repositorio es asímisma efectuada mediante la herramienta `happi`.

La ejecución de la transición *Introducir En Repositorio* implicará que la aplicación sea transferida al área de paquetes del repositorio. La transición *Certificar* no alterará ni la aplicación ni su emplazamiento, sino que significará luz verde para su acceso por parte de todos los clientes. El paso hacia el área de producción se toma mediante *Introducir En Producción*. Por cada una de estas transiciones existe su transición inversa, lo cual permite retirar un producto de producción, cancelar su certificación y retirarlo del repositorio. Desde el punto de vista del contenido del repositorio, las transiciones *Borrar* y *Retirar del Repositorio* son idénticas, ya que la aplicación es eliminada del repositorio.

Las modificaciones sobre el repositorio se efectúan independientemente por cada plataforma. La parte compartida (*share*) a todas las plataformas, cuando ésta exista, deberá ser llevada manualmente con la plataforma cuyo estado sea más alto. Así, si una aplicación está en producción para cierta plataforma, la parte compartida también deberá estar en producción para que la aplicación esté disponible completamente y no aparezcan inconsistencias.

Si una aplicación consiste solamente de una parte compartida, cada vez que se manipula su estado se actualiza el estado de todas las plataformas.

Un problema deriva de la modificación concurrente del repositorio por parte de los gestores de producto. Esta lleva a inconsistencias si se solapan ficheros de dos aplicaciones durante una actualización simultánea del área de producción. Aún utilizando un mecanismo de cerrojos sobre el acceso al área de producción se pueden generar inconsistencias. Así, si el gestor A desea reemplazar una aplicación por una nueva versión, deberá ejecutar dos

transiciones, entre las cuales debe ceder el cerrojo, momento en el cual puede aparecer otra solicitud de modificación por parte de un gestor B, que podría hacer inviable la segunda transición del gestor A.

Dado que el acceso de las estaciones de trabajo al contenido del repositorio se realiza "en línea", la modificación del repositorio ha de ser efectuada con mucha precaución. Sobre todo cuando se trata de introducir y retirar un producto del área de producción, o borrar una aplicación, hay que tener en cuenta que durante la ejecución de la operación el contenido del repositorio es inconsistente para esa aplicación, lo cual quiere decir que su acceso no es posible. Modificar el emplazamiento físico de una aplicación puede llevar bastante tiempo, que en el caso de aplicaciones de gran tamaño se contabiliza en orden de decenas de minutos.

En el CERN se solventa este problema basándose en una facilidad específica al sistema de ficheros AFS, que permite generar una copia del repositorio de sólo lectura, que es a la que tienen acceso los usuarios. Esta copia es actualizada por las noches, de tal manera que los usuarios ven los cambios "de golpe". Sin embargo, el uso de esta técnica genera muchos problemas y fallos cuya recuperación resulta compleja. Además se establece una dependencia hacia el tipo de sistema de ficheros concreto. En la EPFL no se aplica dado que el sistema de ficheros usado es NFS, que no ofrece esta facilidad. Una solución es llevar a cabo la modificación fuera de horas de trabajo, pero está condicionada a que los gestores de producto estén dispuestos a ello, lo cual no es razonable asumir.

Tampoco es posible prever el resultado de una transición. Si por ejemplo, durante una transición de tipo *Introducir En Producción* aparece un conflicto de nombres o de falta de espacio, hay que deshacer toda la transición, volviendo a llevar los ficheros copiados al área de paquetes. Si además fallos como éste ocurren al efectuar una serie de operaciones dependientes entre sí, como por ejemplo sustituir un grupo de aplicaciones interrelacionadas por sus versiones nuevas, esto puede significar un tiempo de baja considerable.

Debido a la inexistencia de un modelo o mecanismo para establecer dependencias entre aplicaciones se producen problemas desde el punto de vista de la coherencia del contenido del repositorio. Ocurre frecuentemente que una aplicación mantenida por un determinado gestor depende de otras aplicaciones no gestionadas por él. Si se producen cambios en

éstas, por ejemplo por ser retiradas del área de producción o borradas del repositorio, las aplicaciones que dependen de ellas pueden sufrir limitaciones o dejar de funcionar completamente. Así, por ejemplo, un cambio en el área de producción de la versión del intérprete del lenguaje gráfico TCL/TK puede conllevar que numerosas aplicaciones dejen de funcionar debido a incompatibilidades entre las versiones del intérprete.

Dada la multitud de aplicaciones existentes en el repositorio, es imposible memorizar qué aplicaciones dependen de cuales, y en qué medida. Debido a que para un gestor de producto le es difícil evaluar qué impacto tendrá un cambio en una aplicación sobre otras, no tiene más remedio que anunciar con antelación las modificaciones sobre la aplicación a todos los usuarios y los demás gestores de producto, y confiar en que evalúen la compatibilidad de sus aplicaciones con la nueva versión del producto con anterioridad a su puesta en producción. Sin embargo, este procedimiento no deja de ser informal y poco fiable, y ocasionalmente se ha dado el caso que aplicaciones se quedan temporalmente fuera de servicio, debido a dependencias no satisfechas, afectando a cientos de usuarios. También se puede dar el caso de que un gestor de producto deje la organización, llevándose todo el conocimiento sobre las interrelaciones de sus aplicaciones.

2.3.9 Replicación del repositorio

El repositorio ASIS es replicado dentro y fuera del CERN. En el CERN, la replicación se efectúa para aumentar la disponibilidad del repositorio, ofreciendo el acceso sobre otro tipo de sistema de ficheros (NFS) que el principal, que es AFS, al cual algunas máquinas no tienen acceso.

De esta manera, también se logra redundancia; en caso de fallar AFS las máquinas se pueden reconfigurar automáticamente (mediante SUE) para acceder al repositorio basado en NFS.

Los centros que replican el repositorio fuera del CERN acceden a la multitud de aplicaciones ofrecidas para muchas plataformas. De esta manera, se benefician del esfuerzo invertido y de la experiencia necesaria para la generación y configuración de las aplicaciones, sin tener que invertir recursos propios. Algunos centros HEP, aparte de replicar ASIS, también hacen lo mismo con SUE y HEPiX con lo que su entorno "software" es funcionalmente muy

similar al del CERN.

La replicación del repositorio se realiza duplicando la jerarquía de directorios y ficheros del repositorio[14]. Si bien solamente se copian los ficheros del origen que difieren o no existen en el destino, es necesario recorrer toda la jerarquía de directorios del repositorio para replicar su contenido, consultando, comparando y en su caso copiando fichero a fichero. Por lo tanto, el coste de cada replicación es *proporcional al tamaño del repositorio*. En el caso concreto del CERN, el tamaño se mide por miles de ficheros sobre cada plataforma, por lo que la replicación es un proceso que lleva horas. La replicación del repositorio AFS al de NFS tarda aproximadamente 3 horas. Este tiempo es sensiblemente superado por nodos remotos, externos al CERN, cuyo ancho de banda de acceso es reducido.

Esto causa un problema importante de carga en los servidores. El repositorio ha de ser replicado regularmente sobre NFS, por un lado, y por otro lado los nodos externos también acceden a los mismos ficheros. Se calcula que los procesos de replicación consumen hasta el 20% de la potencia de los servidores. Además, debida a la larga duración, crece la posibilidad de que ocurra un fallo en la red durante la replicación que provoque una actualización incorrecta o incompleta en el repositorio replicado.

[14] se realiza mediante una herramienta llamada MIRROR, vía el sistema de ficheros distribuido o por `ftp`

2.4 Otros sistemas de distribución de aplicaciones

A continuación se realizará una breve descripción de otros sistemas existentes para la distribución automatizada de aplicaciones, en concreto, SUP, DEPOT, SMS y UPS/UPD. El primer intento para organizar la distribución de aplicaciones fue el presentado por P. Anderson en [3]; sin embargo, no tuvo aplicación práctica debido a diversos problemas técnicos[15]. SUP y DEPOT pertenecen a la primera generación de S.D.A.'s operativos, mientras que UPS/UPD y SMS son sistemas más recientes y completos.

Otros sistemas no descritos aquí, pero igualmente importantes, son LUDE [23], XHIER [31], y NETWIZARD [8]. Asimismo, algunos sistemas tradicionales para la gestión de red, como OPENVIEW de Hewlett-Packard, o SOLSTICE de Sun Microsystems, ofrecen capacidades de distribución de software. Sin embargo, su enfoque primordial es la gestión de hardware y redes, misión para la cual fueron desarrollados, y sus capacidades para la gestión de software son rudimentarias.

2.4.1 SUP, *Software Upgrade Protocol*

SUP (*Software Upgrade Protocol*) [32] fue desarrollado en 1990 por la Universidad Carnegie-Mellon. El sistema, construido en torno al sistema operativo MACH, se basa en la cooperación entre ordenadores cliente conectados a un servidor central. Las aplicaciones son almacenadas en un repositorio central, el cual no es accesible directamente por los clientes.

Regularmente, un agente en el cliente solicita actualizar una "colección" de ficheros, similar a un paquete en ASIS, al servidor. En el servidor central se crea un agente por cada cliente, encargado de atenderle, que verificará si el cliente requiere ser actualizado, comparando el estado de los ficheros de la colección del cliente con la colección actual. En ese caso le entregará los ficheros de la colección vía una conexión TCP/IP. Cada cliente recibirá una copia de la colección que instalará sobre su propio sistema de ficheros. La actualización del cliente es de tipo "push".

SUP no ofrece un control de versiones, ni permite el uso de múltiples plataformas, por lo

[15] Esencialmente, Anderson proponía distinguir los ficheros pertenecientes a cada aplicación asignándoles a cada una un identificador de usuario distinto, éste enfoque resulta inviable cuando se trata de gestionar un número elevado de aplicaciones.

que su dominio de aplicación se limita a replicar información idéntica sobre un conjunto de clientes similares.

2.4.2 NIST Depot

Depot [24] surgió en 1990, desarrollado por el NIST (*National Institute of Standards and Technology*) estadounidense. Este sistema está preparado para acomodar múltiples plataformas y aplicaciones UNIX. Las aplicaciones, centralizadas en un repositorio, están organizadas en jerarquías divididas por aplicaciones y plataformas (de manera similar al área de paquetes de ASIS). Al igual que en ASIS, la parte de la aplicación común a todas las plataformas se guarda separadamente de la parte específica a cada plataforma. Los clientes acceden directamente al repositorio, emplazado en un servidor de ficheros, a través del sistema de ficheros NFS.

Los clientes acceden directamente al subconjunto del repositorio donde están almacenadas las aplicaciones, enlazando la jerarquía del repositorio sobre un directorio raíz (`/depot`). El cliente solamente verá la parte común de las aplicaciones, y la parte específica a su plataforma. La actualización de las aplicaciones disponibles sobre el cliente es tarea del servidor; el acceso del cliente es tipo "push".

No existe ningún modelo de acceso escalonado, por lo que un cliente verá todas y cada una de las aplicaciones en el repositorio. Tampoco se pueden generar copias locales de las aplicaciones, por lo que el acceso al repositorio está condicionado al buen funcionamiento de la red y del servidor.

El usuario deberá añadir la localización de los ficheros ejecutables de cada aplicación al camino de búsqueda[16] (*path*). No existe integración de los ficheros en una jerarquía común a todas las aplicaciones dentro de un directorio prefijo[17], como es estándar en Unix, es decir: las aplicaciones residen en espacios separados y no comparten directorios entre ellas. Debido a esto, el camino de búsqueda deberá contener una entrada por cada aplicación, por lo que puede convertirse excesivamente largo. Además, esta estructura es inviable para muchas aplicaciones, que requieren instalar ficheros en directorios compartidos con otras aplicaciones para intercambiar información. Otras aplicaciones no pueden ser configuradas

[16] un camino de búsqueda es una secuencia de punteros a directorios donde residen ficheros ejecutables.

[17] un ejemplo de directorio prefijo estándar es `/usr/local/`

para ser instaladas fuera de directorios prefijo estándar.

Depot no prevée el caso de que se instalen múltiples aplicaciones que contienen ficheros ejecutables con los mismos nombres. Si se da este caso, la aplicación ejecutada será la primera encontrada en el camino de búsqueda. Esto trae consigo la aparición de conflictos por el orden del camino de búsqueda. Por esta razón, Depot no es adecuado para gestionar múltiples versiones de una misma aplicación, donde los nombres de los ficheros ejecutables suelen coincidir.

Depot no ofrece mecanismos para replicación, actualización fuera de horas de trabajo, ni para evitar inconsistencias en los clientes al modificar el repositorio. Se centra en la distribución física, y no existen herramientas para la configuración de aplicaciones.

2.4.3 SMS: *Systems Management Server*

Sms [10, 29] de Microsoft consiste en un conjunto de herramientas para la gestión centralizada de ordenadores personales en un entorno distribuido. Entre sus funcionalidades se encuentran la gestión de red, la teleasistencia (control remoto de clientes), y la distribución de aplicaciones en entornos basados en DOS y Windows. La distribución de aplicaciones se controla desde un servidor central, que está comunicado con "servidores de distribución", los cuales despachan las aplicaciones a los nodos finales, que pueden ser estaciones de trabajo o servidores de ficheros. Cada servidor de distribución puede estar geográficamente separado del servidor central.

Para distribuir una aplicación, se debe crear un paquete que contiene la aplicación, junto a un programa tipo *script*, que contiene órdenes para el proceso de configuración e instalación. A continuación se seleccionan los ordenadores destino del paquete. El paquete es distribuido desde el servidor central a los servidores de distribución. Sobre los ordenadores destino existe un agente local que periódicamente interroga, de manera análoga a SUP, al servidor de distribución si tiene algún paquete para él. En caso de ser así, este agente copia el paquete al ordenador y lanza el *script* de instalación. La instalación de un paquete puede ser optativa, pudiendo ser pospuesta o descartada por el usuario, o compulsoria, obligándose la actualización del cliente. SMS ofrece la posibilidad de ejecutar instalaciones fuera de horas de trabajo, aunque solamente se puede programar por adelantado una

distribución a la vez.

En vez de distribuirse aplicaciones ya instaladas y configuradas, se entrega al ordenador destino el paquete junto a los comandos que se deben de ejecutar para su instalación. SMS no ofrece herramientas para la construcción del *script* de instalación, ni analiza su contenido, por lo que su elaboración está basada en la técnica de ensayo y error. Tampoco existe un modelo de interrelaciones que permita definir las dependencias de una determinada aplicación, o entre paquetes distribuidos. Debido a esto, es difícil prever el éxito de una distribución, dado que no se conoce el posible impacto sobre cada cliente. En caso de fallo de la instalación, el agente local intentará reconstruir el estado previo al inicio de la instalación. El agente local informará del resultado de la distribución al servidor central, que registrará todos los resultados, los cuales se pueden utilizar para refinar el procedimiento de instalación. La configuración de cada cliente es accesible centralmente.

SMS no mantiene un repositorio de aplicaciones, sino que se limita a distribuir paquetes a un conjunto de ordenadores destino, sea cual sea su contenido. Una vez que un paquete ha sido distribuido, puede ser borrado del servidor central. La oferta de aplicaciones se restringe a las que son distribuidas centralmente. De esta manera, se tiene un mayor control sobre qué software es accesible por qué clientes. Sin embargo, el grado de libertad de éstos es mínimo.

2.4.4 UPS/UPD: *Unix Product Support / Unix Product Distribution*

El sistema Ups/Upd [19] es desarrollado por el *Fermi National Accelerator Laboratory* (EE.UU.) para la distribución de software a los ordenadores bajo Unix del laboratorio. Las aplicaciones son almacenadas empaquetadas sobre repositorios multiplataforma y multiversión. Estos repositorios se emplazan en servidores llamados "nodos de distribución", de los cuales existe uno principal, que sirve aplicaciones a todo el laboratorio, y varios secundarios, de acceso restringido a grupos o departamentos. Estos nodos de distribución son accedidos por "nodos locales" que replican las aplicaciones deseadas del repositorio. Los nodos locales pueden ser estaciones de trabajo, o servidores de ficheros para un grupo de estaciones de trabajo.

Las aplicaciones son instaladas sobre el sistema de ficheros del nodo local, por lo que la

ejecución de las aplicaciones sobre el nodo local se realiza con independencia respecto del nodo de distribución. De hecho, el emplazamiento del nodo local puede ser geográficamente distinto del nodo de distribución, la comunicación entre ambos se realiza mediante el protocolo TCP/IP. En cada nodo local, existe una base de datos con toda la información sobre las aplicaciones instaladas. A diferencia de ASIS, donde todas las estaciones de trabajo acceden a un solo repositorio común, un nodo local puede solicitar aplicaciones a varios nodos de distribución, que pueden ser gestionados independientemente.

La actualización del conjunto de aplicaciones existente sobre el nodo local es efectuado directamente por su administrador. Desde los nodos de distribución no se puede dirigir este proceso, por lo cual no es posible llevar a cabo distribuciones controladas centralmente, al contrario de SUP, DEPOT y ASIS. El método de acceso de los clientes por lo tanto es de tipo "pull". Es responsabilidad de cada administrador de acceder regularmente al repositorio para actualizar las aplicaciones ofrecidas, y de hacerlo cuando no interfiera con el trabajo de los usuarios. Al igual que en DEPOT, las aplicaciones son instaladas en directorios separados, lo cual implica los problemas expuestos anteriormente. Un avance respecto a DEPOT es que existe una herramienta, que accediendo a la base de datos permite configurar el camino de acceso para controlar qué aplicaciones se desean acceder y cuáles no. De esta manera se pueden gestionar múltiples versiones de una misma aplicación, dando acceso solamente a una versión a la vez.

Existe un modelo simple de relaciones, que permite establecer una relación de dependencia entre una aplicación y otra. Al configurar el acceso a una determinada aplicación, se dará acceso a todas las aplicaciones de las que depende.

Parecido al existente en ASIS, existe un modelo de acceso escalonado a las aplicaciones. Una aplicación puede estar "en desarrollo", "bajo prueba" (ofrecida a los usuarios finales para su evaluación, análogo a "bajo certificación" en ASIS), "nueva" (aplicación ya probada, análogo a "certificado"), "actual" (la recomendada, análogo a "en producción"), o "vieja" (en desuso).

Dado que el camino de acceso es un atributo local a cada usuario final, éste puede configurar de manera independiente a los demás usuarios qué aplicaciones y qué versiones desea utilizar sobre las disponibles en la estación de trabajo. Debido a su mayor flexibilidad, esto representa una ventaja sobre ASIS, donde todos los usuarios finales acceden al mismo

conjunto de aplicaciones seleccionado por el administrador.

Existen herramientas para actualizar el repositorio sobre los nodos de distribución. Sin embargo, no existe ningún entorno automatizado para la generación de aplicaciones. Al no acceder los clientes de manera directa al repositorio, y al no existir ningún área común a todas las aplicaciones, la modificación del repositorio es bastante más simple que en ASIS y no plantea los mismos problemas de consistencia.

Capítulo 3

Planteamiento del Problema e Hipótesis de Trabajo

La entrada por parte de la EPFL en el proyecto ASIS, conlleva una evaluación general del sistema existente. Una de las razones de la colaboración entre el CERN y la EPFL se fundamenta en el intercambio de conocimiento técnico, habitual entre ambas instituciones. En este caso concreto es el CERN que actúa de fuente de conocimientos, al tener mayor experiencia en el área de sistemas distribuidos.

El concepto y la estructura general del repositorio ASIS se considera correcta y válida para que éste sea portado a la EPFL.

Sin embargo, las herramientas de acceso y modificación fueron diseñadas para cubrir necesidades propias del CERN. Es por lo cual que existen en su diseño profundas dependencias hacia el entorno del laboratorio, que exigen una reimplementación antes de poder ser utilizadas por la EPFL. Se aprovecha la ocasión y se procede a un *rediseño completo* del juego de herramientas para manipular y acceder el repositorio, teniendo en cuenta especialmente:

- la fiabilidad, consistencia y coherencia tanto en el acceso como en la manipulación del repositorio
- la portabilidad de las herramientas, identificando y eliminando dependencias de lugar
- flexibilidad en la configuración de las herramientas

Adicionalmente al rediseño del juego de herramientas existen algunos aspectos que requieren un análisis más profundo. Así por ejemplo, la falta de un entorno de seguridad propio y portable, que proteja el repositorio y consecuentemente las máquinas cliente de ataques hostiles, y también los problemas derivados de la gestión del contenido del repositorio.

- Uno de los puntos flojos del actual sistema son los mecanismos de modificación y replicación de repositorios. Estos giran alrededor de tres vertientes:

  1. la imposibilidad de poder agrupar operaciones individuales de manera inseparable
  2. la imposibilidad de prever el resultado de una operación de modificación
  3. la falta de un mecanismo de replicación eficiente.

  Aunque el modelo de procesado de "software" describe correctamente las operaciones sobre una aplicación, no es adecuado cuando aparecen interacciones entre aplicaciones del repositorio. Un ejemplo típico es el de sustituir una versión de una aplicación en producción por una nueva. En este caso deben de ejecutarse dos operaciones: primero, retirar la versión antigua mediante la transición "retirar de producción", y segundo, "introducir en producción" la nueva. Ambas operaciones deben de terminar satisfactoriamente para asegurar que el entorno de usuario no se queda en un estado inconsistente. Este ejemplo se generaliza a todas las operaciones (transiciones activas y operaciones parche) que guardan alguna relación entre sí.

  Estas operaciones deben de ser ejecutadas en orden y de manera atómica, por lo que representan una *transacción*. Una transacción es una secuencia ordenada de operaciones cuya ejecución atómica ha de ser garantizada. Es decir, o se ejecutan todas las operaciones que conforman la transacción, o ninguna de ellas.

  Al agrupar operaciones de modificación del repositorio, un gestor de producto puede estar seguro que en caso de producirse un problema durante una operación, el estado del repositorio quedará idéntico a como estaba antes de ejecutarse la transacción. De esta manera, la manipulación de aplicaciones relacionadas entre sí se puede incluir en la misma transacción para garantizar su atomicidad.

  La ejecución de una transacción se puede separar en dos fases: en una primera fase, se *valida* la viabilidad de ejecutar todas y cada una de las operaciones de la transacción,

y en la segunda, se efectúa la *ejecución física* de la transacción. Una transacción validada tiene garantizada su ejecución. Separando temporalmente ambas fases, un gestor de producto puede validar en horas de trabajo una transacción, llevándose a cabo su ejecución en un momento que no haya usuarios accediendo al repositorio.

Consultando las transacciones ejecutadas sobre un repositorio se obtiene la información sobre los cambios efectuados sobre éste. Basando la replicación de un repositorio en estos cambios, se modificarán solamente los objetos afectados por los cambios. De esta manera, el esfuerzo de replicación será proporcional al tamaño de los cambios habidos, en vez de ser proporcional al tamaño del repositorio.

Basándose en estos conceptos se construirá un *sistema transaccional para la modificación y replicación del repositorio.*

Si bien en 1995 se inició el desarrollo de un prototipo transaccional, basado en la herramienta `happi`, se decide descartar este enfoque, reanalizando y diseñando un sistema independiente, que pueda ser utilizado de manera autónoma tanto para modificar como para replicar un repositorio.

- El uso de transacciones no modeliza por completo las relaciones que puedan existir entre aplicaciones. Mediante el uso de transacciones se puede establecer un *nexo temporal* entre aplicaciones, en el marco de la ejecución de una transacción. Sin embargo, no existe un modelo para describir las distintas dependencias existentes entre aplicaciones o entre una aplicación y su entorno de ejecución. Su definición se hace necesaria debido al grado de complejidad de las dependencias de aplicaciones.

  Mediante un modelo similar, un gestor de producto puede documentar de manera exacta y explícita los objetos que son necesarios para la ejecución de una aplicación, y cuales son incompatibles con ella. Un administrador se valdrá de este conocimiento para determinar qué aplicaciones conforman un conjunto que sirve para establecer un entorno de trabajo coherente sobre una determinada estación de trabajo.

  Los gestores de producto también se beneficiarían de este conocimiento, al poder conocer las relaciones que otras aplicaciones, no gestionadas por ellos, tienen con las suyas. Dado que el área de producción es común a todas las aplicaciones, se pueden establecer reglas para que el área de producción sea coherente en su conjunto. De la misma manera, se pueden establecer reglas para que el repositorio en su conjunto sea lo más coherente posible.

Será por lo tanto necesario definir un *modelo de dependencias*, y herramientas y procesos para su gestión.

El autor del presente Trabajo Fin de Carrera fue becado por el CERN durante 14 meses, en los cuales llevó a cabo las siguientes tareas de desarrollo:

- diseño e implementación del sistema de modificación y replicación del repositorio
- diseño e implementación del sistema para describir y comprobar dependencias

En el próximo capítulo se presentará el diseño de estos sistemas. Adicionalmente, para facilitar la interactuación de los usuarios con los sistemas, está previsto el uso de una interfaz gráfica; se presentará el diseño del modelo cliente/servidor que sirva para la comunicación de las herramientas ASIS con su respectiva interfaz gráfica.

Capítulo 4

Solución Propuesta

Se pretende dar en este capítulo una solución a los problemas planteados anteriormente. En concreto, se presenta el diseño de un sistema basado en transacciones para la modificación y replicación del repositorio, y el de un sistema para describir y comprobar las relaciones y dependencias que plantean los objetos del repositorio.

En primer lugar, se describirá el diseño conceptual y funcional de los sistemas propuestos, pasándose a continuación a la descripción de la arquitectura física y del diseño detallado.

4.1 Diseño conceptual y funcional

A continuación se definirá el modelo lógico y funcional de los sistemas como los requisitos que deberá cumplir el sistema software que le dé soporte.

El diseño estará dividido en tres partes; la primera de ella tratará el problema del sistema de modificación y replicación del repositorio, mientras que la segunda estará orientada a la gestión de interrelaciones. En una tercera parte se determinará el esquema cliente/servidor que será utilizado para comunicar estos sistemas con sus interfaces gráficas. Los requisitos específicos se listan de forma tabular en el apéndice C, a partir de la página 129.

4.2 Modificación y replicación del repositorio

4.2.1 Propósito

Tanto la modificación y replicación de repositorios ASIS actualmente presentan graves inconvenientes que pueden ser resumidos como:

- la falta de serialización del acceso al repositorio
- la imposibilidad de preveer el resultado de las modificaciones
- la falta de un sistema de replicación eficiente, económico y tolerante a fallos.

El modelo que se presenta pretende dar también solución a los problemas derivados de la modificación del repositorio en horas de trabajo, cuando hay usuarios utilizándolo en línea.

Se establecerán dos sistemas interdependientes: uno que **valide** las operaciones de modificación enviadas por los usuarios en forma de transacciones y otro que las **ejecute** asíncronamente fuera de horas de trabajo.

También se separarán funcionalmente las fases de *generación* de aplicaciones de las de *distribución*. El sistema descrito recibirá como entrada paquetes listos para ser introducidos en el repositorio. La generación de estos paquetes queda fuera de su ámbito.

4.2.2 Consideraciones de entorno

Los *usuarios* que se verán afectados directamente por el diseño del sistema de modificación y replicación del repositorio serán los gestores de producto y los responsables de los repositorios. Los administradores de estaciones de trabajo y los usuarios finales se beneficiarán de la mayor estabilidad que deberá proporcionar el sistema.

El *entorno de funcionamiento* en el que se instale el sistema deberá seguir las pautas siguientes:

- el equipo informático deberá ser ampliamente extendido y accesible por el tipo de

organización destino (CERN y EPFL), debiendo ser este sistema de fiabilidad probada.[1]

- el entorno software en el que se instale el sistema deberá ser altamente difundido y proveer una fiabilidad probada. A ser posible, el sistema deberá ser instalable en un número alto de plataformas distintas.

4.2.3 Relación con otros sistemas

Dentro del proyecto ASIS, del cual forma parte el sistema de modificación y replicación del repositorio, se encuentran otros sistemas con los que se interactuará:

- el gestor de estación de trabajo (`ASISupdate` o su nueva versión `ASISwsm`) que es la herramienta para el acceso de las estaciones de trabajo cliente al repositorio;
- el generador de paquetes (`happi` o su nueva versión `ASISgen`) que prepara los paquetes para ser instalados en el repositorio;
- el sistema de dependencias, que permitirá definir las interrelaciones de aplicaciones entre sí y con su entorno;
- el entorno de seguridad ASIS [17], actualmente en desarrollo.

La versión actual del generador de aplicaciones (`happi`) incluye entre sus funcionalidades operaciones de modificación del repositorio principal. Sin embargo, la nueva versión desarrollada (`ASISgen`) preparará el paquete para ser entregado al sistema de modificación y replicación del repositorio, pero en ningún momento modificará el repositorio.

El *entorno de seguridad ASIS* proveerá los mecanismos necesarios para garantizar la autentificación de los usuarios que pretendan modificar el repositorio, de la misma manera que protegerá el contenido del repositorio de ataques hostiles externos.

[1] en el CERN, los sistemas hardware y software deben someterse a un proceso de certificación antes de recibir su visto bueno para su uso como servidores.

4.2.4 Restricciones generales

Las restricciones aplicadas en el desarrollo del sistema se verán esencialmente guiadas por:

- la importancia de la integridad del contenido del repositorio. El sistema debe ser altamente fiable y tener capacidad de recuperarse de fallos, evitando dejar el repositorio en un estado inconsistente.
- la replicación debe ser económica en el concepto de accesos al repositorio a replicar y las transferencias de datos a través de la red. De esta manera se minimiza el tiempo de replicación y se posibilita la replicación a lugares geográficamente distantes que disponen de un ancho de banda reducido con el repositorio a replicar.
- deberá existir un sistema de ficheros común entre el repositorio y su réplica (por ejemplo, un sistema de ficheros tipo AFS).

4.2.5 Modelo de transacciones

Se utilizará un modelo basado en transacciones para la modificación del repositorio principal y su replicación a otros repositorios. Las transacciones sobre el repositorio principal se llevarán a cabo en dos fases asíncronas: primero, se comprueba la validez de una transacción, y si esta es aceptada, se ejecutará físicamente fuera de horas de trabajo. La replicación de un repositorio se llevará a cabo propagando los cambios del repositorio principal al ejecutarse sobre el repositorio replicado las transacciones ya ejecutadas sobre el principal.

Las transacciones consistirán en una secuencia ordenada, válida y consistente de operaciones de modificación sobre el repositorio. La secuencia de operaciones deberá conformar una transformación correcta y coherente del estado del repositorio. Las operaciones podrán ser de dos tipos:

- *transiciones activas* del modelo de procesado de "software", que cambian el estado de una aplicación en el repositorio
- *operaciones "parche"* que modifican una aplicación, sin cambiar su estado.

Una transacción podrá contener tantas operaciones como sea necesario. Se podrán usar todos los tipos de operaciones en cada transacción.

Como parte de la misma definición de transacción, toda transacción debe cumplir las propiedades ACID [34] (*Atomic, Consistent, Isolated, Durable*; atómicas, consistentes, aisladas, permanentes):

- **Atómicas**: o bien se ejecutan todas las operaciones, o no se ejecuta ninguna.
- **Consistentes**: El conjunto de las operaciones de la transacción debe ser un cambio correcto de estados.
- **Aisladas**: Cada transacción se ejecutará independientemente del efecto de otras transacciones concurrentes.
- **Permanentes**: El efecto de transacciones validadas es permanente.

La *atomicidad* y la *consistencia* se asegurará validando primero la transacción, antes de empezar su ejecución, operación por operación en secuencia estricta. Si se detecta que una operación de la transacción es inválida, por ejemplo por detectarse un conflicto en el espacio de nombres del área de producción, o que no puede efectuarse siguiendo los pasos del diagrama de estados del modelo de procesado de "software", entonces la transacción se descarta en su totalidad.

Se garantizará que las transacciones se ejecutan de manera *aislada*, validándolas y ejecutándolas en secuencia estricta y una a la vez. Es decir, que durante la validación o ejecución de una transacción no se podrá validar/ejecutar otra. Las transacciones serán ejecutadas en el mismo orden que han sido validadas.

Una vez que una transacción haya sido validada, su posibilidad de ejecución sobre el repositorio habrá sido comprobada y su ejecución ha sido programada de manera definitiva. De esta manera, el efecto de la transacción será *permanente*.

Tolerancia a fallos

En un entorno distribuido, la ejecución con éxito de una transacción validada está condicionada a que no se produzcan fallos en la red en el momento que se lleve a cabo esta

ejecución. Dado que ésto es un factor impredecible, la *ejecución* de la transacción también será atómica: Si al ejecutarse una operación de una transacción se produce un fallo externo, se deshace la operación en curso y las operaciones anteriores a ésta que pertenezcan a la transacción. De esta manera, el punto de fallo será *entre dos transacciones*. Para implantar este mecanismo de "vuelta atrás" o *rollback*, las *operaciones* deberán ser atómicas (al igual que la transacción), e invertibles (por cada operación se puede ejecutar un procedimiento que la anule).

De esta manera, la atomicidad de una transacción se garantiza totalmente y aún en circunstancias adversas. Al representar la transacción una transformación consistente del repositorio, el estado será consistente antes y después de la ejecución de una transacción, es decir, entre dos transacciones.

Esta propiedad de tolerancia a fallos se aplicará tanto a la modificación del repositorio como a su replicación. Si aparece un fallo en la replicación, se limitará a una interrupción de la ejecución *entre* transacciones pero no en mitad de una transacción.

4.2.6 Estructura general

En la figura 4 se aprecian los sistemas principales; El gestor de transacciones, que se llamará `ASIStm` (*ASIS transaction manager*), y el gestor de copia local, que se llamará `ASISlcm` (*ASIS local copy manager*):

- El gestor de transacciones se encargará de recibir en horas de trabajo las transacciones de los usuarios, verificar la autorización de éstos, chequear la factibilidad y buen sentido de las transacciones sobre el repositorio principal. Si la transacción es válida se introduce en una lista, donde se guardarán en orden todas las transacciones validadas. El usuario es informado del resultado de la comprobación.
- El gestor de copia local se activará en horas de baja actividad, leerá las transacciones de la lista y las ejecutará sobre el repositorio en el mismo orden en el que han sido validadas. El gestor de copia local escribirá las transacciones ejecutadas en una lista.

El gestor de copia local también será utilizado para llevar a cabo la replicación sobre otros repositorios. Dado que se tienen las modificaciones efectuadas sobre el repositorio

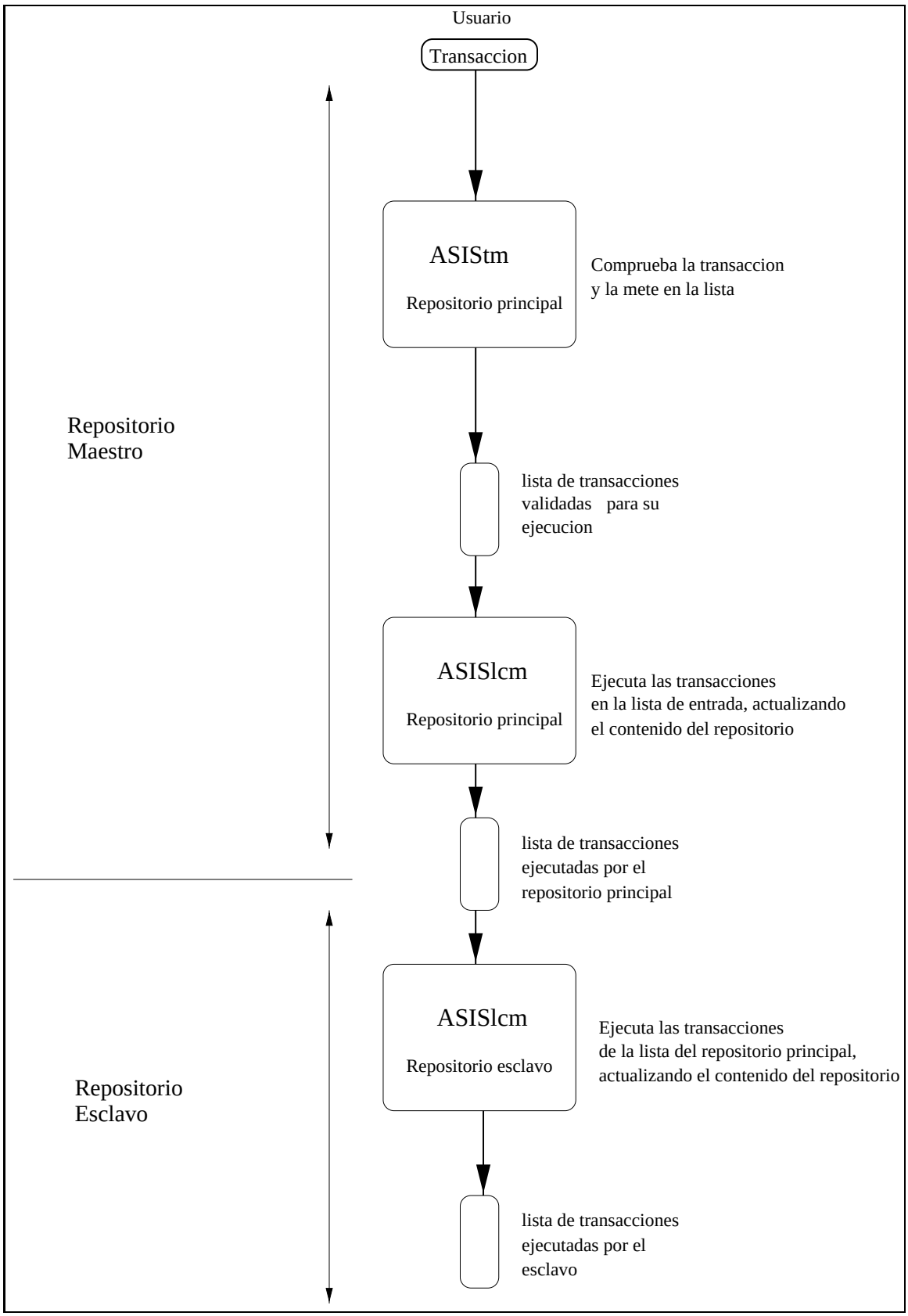

Figura 4: modificación y replicación de repositorios

principal, en forma de la lista de transacciones ejecutadas, se pueden ejecutar estas mismas transacciones sobre el repositorio replicado.

El gestor de transacciones no garantizará el éxito de la ejecución sobre los repositorios que no sean el repositorio principal (sobre el que se lleva a cabo la validación de la transacción).

Se adopta un *modelo maestro-esclavo*, en el cual las modificaciones en el repositorio maestro serán propagadas hacia los repositorios esclavos. Los repositorios esclavos, en cambio, no modificarán al repositorio maestro.

Esto posibilita la existencia de múltiples repositorios encadenados, asumiendo tanto el rol de maestro como de esclavo, que vayan propagando los cambios desde un repositorio central (el repositorio principal).

Se permitirá que un repositorio maestro sea replicado simultáneamente por varios esclavos. Junto a la encadenación se consigue un esquema de replicación *en estrella*, como se muestra en la figura 5.

El repositorio principal *no será responsable* de ninguno de los repositorios esclavos, es decir: no controlará ni vigilará su replicación. Los esclavos serán capaces de limitar la replicación al subconjunto del contenido del repositorio que les interese.

4.2.7 Descripción del modelo

Gestor de transacciones

Observando la figura 6 se describe con mayor detalle el proceso del gestor de transacciones. Los pasos seguidos serán:

1. el usuario (normalmente, un responsable de producto) generará el paquete mediante la utilidad `happi` (o la nueva versión `ASISgen`) y lo depositará en un repositorio auxiliar. Este repositorio auxiliar solamente contendrá paquetes durante el tiempo que éstos no han sido transferidos al repositorio principal.

2. A continuación diseña una transacción en la que expresa las operaciones básicas e indivisibles que quiere efectuar sobre el repositorio principal. Luego contactará al

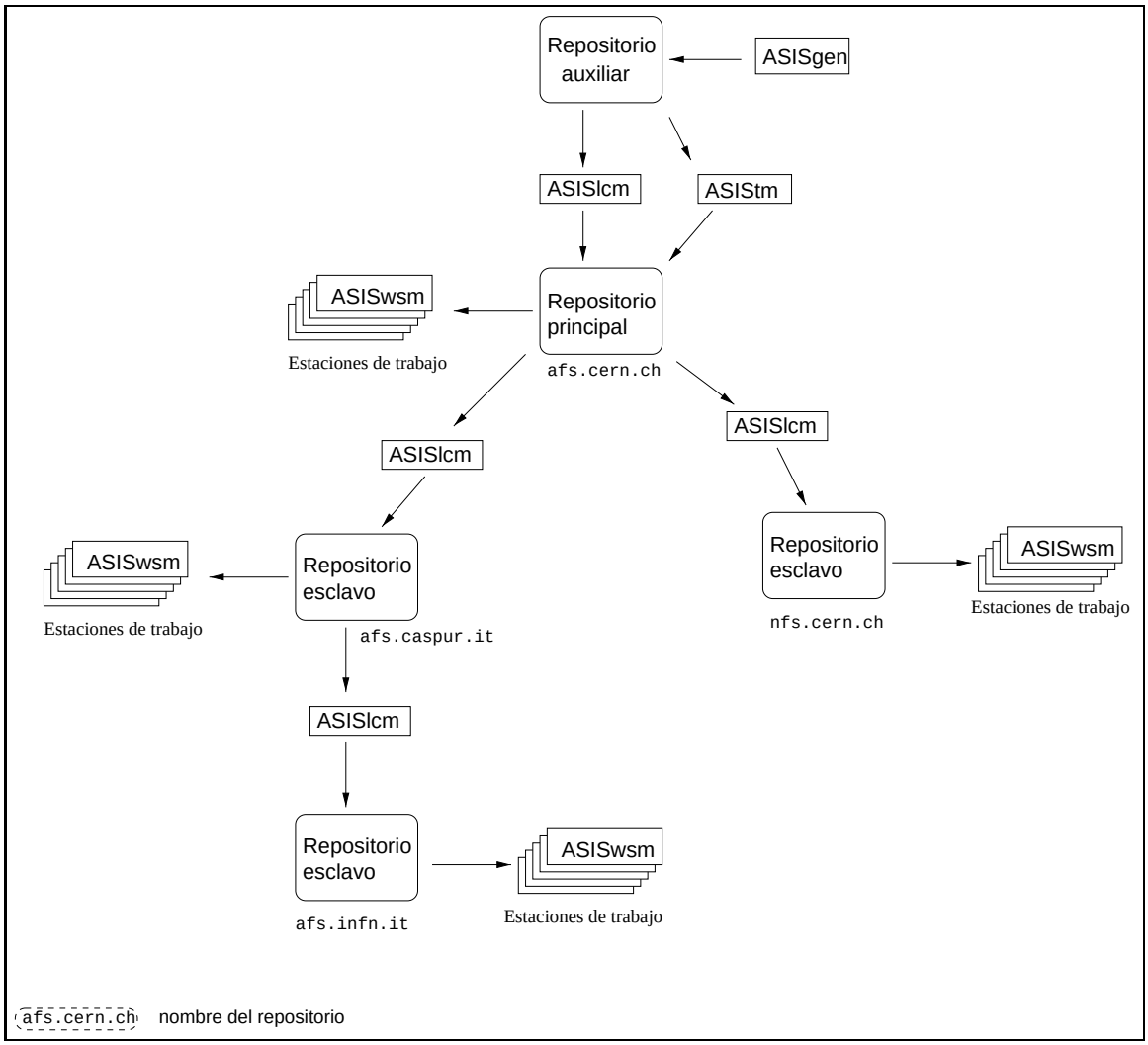

Figura 5: ejemplo de topología de replicación

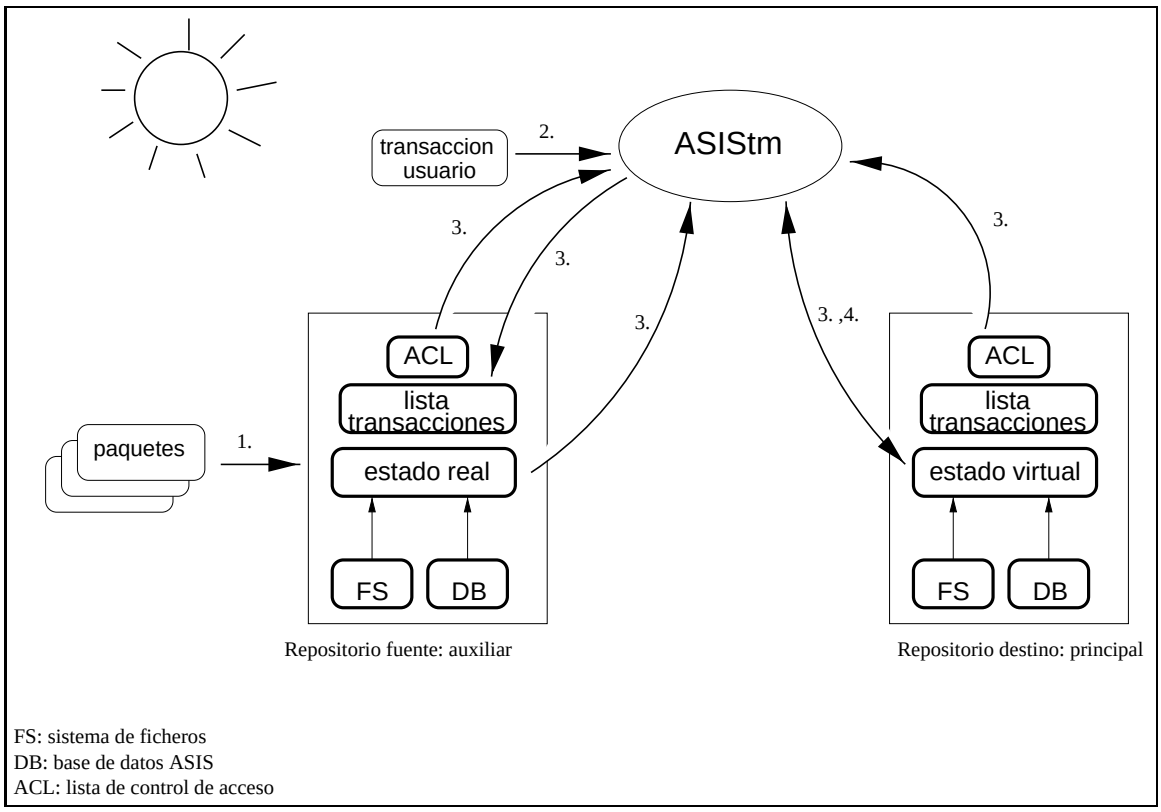

Figura 6: gestor de transacciones

gestor de transacciones (directamente o a través de un interfaz gráfico), facilitándole su identidad y la transacción.

3. El gestor de transacciones comprueba la validez sintáctica de la transacción, y por cada operación en la transacción

   - comprobará que el usuario está autorizado para modificar el producto afectado por la operación
   - ejecutará "virtualmente" la operación (sin modificar el sistema de ficheros ni la base de datos ASIS), comprobando la factibilidad de la operación sobre el repositorio principal. Se prestará atención de que cada operación sobre un producto, caso de tratarse de una transición, sea válida dentro del diagrama de transición de estados del modelo de procesado de "software". También se verifica que no se generan conflictos de nombres entre ficheros de paquetes, y que haya suficiente espacio para acomodar los paquetes en el repositorio principal.

4. Se actualiza la lista de transacciones en el repositorio auxiliar y se actualiza el estado

"virtual" del repositorio principal, en caso de ser válida la transacción.

5. se notifica al usuario el resultado de la operación.

En caso de ser rechazada una operación, se descartará la transacción entera, quedándose el estado virtual tal y como estaba al inicio del proceso. Si la transacción es aceptada, el usuario tiene la garantía de que la ejecución de la transacción será llevada a cabo con éxito sobre el repositorio principal.

Dado que la ejecución será "virtual", sin tener que por ejemplo copiar físicamente paquetes, la velocidad de respuesta del sistema será alta.

Estado virtual y estado real del repositorio

El *estado virtual* del repositorio es la representación del estado del sistema de ficheros y de la base de datos si se hubiesen ejecutado todas las transacciones validadas pero aún no ejecutadas físicamente por el gestor de copia local.

Las aplicaciones del repositorio no se deben alterar por la ejecución del gestor de transacciones, ni tampoco la base de datos (consultada por aplicaciones cliente y gestores de copia local de repositorios esclavos). La aplicación que modifica las aplicaciones contenidas en el repositorio es única y exclusivamente, el gestor de copia local.

Por ejemplo, supóngase que se valida una transacción en la que se borra una aplicación A, por lo cual sus ficheros desaparecer'an. Si se desea inmediatamente a continuación validar una transacción que instale una aplicación B que comparta nombres de ficheros en el mismo directorio con A, se producirá un *conflicto* si no se "recuerda" que los ficheros del primer paquete han sido borrados, ya que en el repositorio real siguen presentes (no se ha ejecutado ASISlcm entre la validación de una transacción y la otra).

Otro ejemplo es el de introducir una nueva aplicación en el repositorio, y algo más tarde intentar "certificarla" mediante otra transacción. Al introducir el paquete se crearán nuevos objetos que deberán ser visibles por operaciones posteriores, aunque éstos objetos no estén aún físicamente en el repositorio.

En resumidas palabras, para aceptar la transacción t_n hay que conocer necesariamente el

estado en t_{n-1}, por lo que la imagen de este estado deberá estar disponible, lo cual hace necesario mantener un "estado virtual", tanto del sistema de ficheros como de la base de datos.

Comprobaciones de coherencia

El gestor de transacciones no solamente comprobará que la transacción es válida, sino que también realizará ciertas comprobaciones respecto al su buen sentido. La función de estas comprobaciones es la de verificar que el autor de la transacción no ha cometido involuntariamente una omisión o una incorrección semántica al formular la transacción.

Las comprobaciones efectuadas son:

- *¿Se alteran aplicaciones pertenecientes a alguna plataforma no activa?* Generalmente, cuando una plataforma deja de ser activa, no se deberá alterar el contenido del repositorio para ésta.
- *¿Se queda para alguna plataforma un producto sin ninguna versión en estado "En Producción" que antes tenía al menos una en ese estado?* Quizás se desea retirar ese producto intencionadamente, o quizás se olvida reemplazar la versión antigua por otra. Los cambios de versiones en producción, es decir, retirar la versión antigua e introducir la nueva de un producto, deberán ser realizados siempre en la misma transacción, salvo circunstancias excepcionales.[2]
- (sólo para operación Introducir En Repositorio): *¿Se han omitido plataformas, es decir, está disponible una aplicación sobre más plataformas en el repositorio auxiliar que las indicadas en la transacción?* Quizás se trata de una omisión voluntaria, al disponer la aplicación generada para todas las plataformas pero no querer actualizar algunas.
- *¿Se borra la parte compartida ("share") de una aplicación sin borrar todas las partes*

[2]de esta manera, y al garantizarse la atomicidad de las operaciones de una transacción, nunca sucederá que un producto desaparezca inintencionadamente del área de producción. Dado que no se garantiza la atomicidad de ejecución de *toda* la lista de transacciones, si se emplazan estas operaciones en transacciones distintas podría darse el caso de que se ejecute la transacción que ejecuta la retirada de la versión antigua, pero no la transacción que introduce la versión nueva.

específicas a las plataformas? Al formar, para una plataforma, la parte específica junto con la parte compartida la unidad mínima la aplicación podría quedarse incompleta para esas plataformas. Se puede dar este caso al querer borrar la parte compartida de una aplicación que debe ser mantenida para plataformas que han dejado de ser activas.

- *¿Queda la parte compartida de una aplicación en un estado distinto al de todas las demás partes específicas, si éstas existen?* No tiene sentido que la parte compartida esté sola en un estado distinto al de a todas las demás plataformas.

Estas comprobaciones se deben a que los estados de una misma aplicación *podrán ser distintos por cada plataforma*, permitiendo el gestor de transacciones manipular los paquetes por separado. Por ejemplo, la aplicación A podrá estar "En Producción" para ciertas plataformas, en otras podrá estar "Bajo Certificación" y para otras quizás no existir.

Gestor de copia local

El gestor de copia local leerá las transacciones del repositorio maestro y las ejecuta sobre el repositorio esclavo. Esta actualización se llevará a cabo normalmente de manera automática y fuera de horas de trabajo, sin que el usuario intervenga.

El repositorio maestro puede ser bien el repositorio auxiliar, en el cual se depositan los paquetes, bien un repositorio que se desea replicar. Al ser la estructura en ambos casos la misma (la de un repositorio ASIS), el gestor de copia local no tiene porqué conocer el tipo del repositorio maestro. De esta manera, se consigue que el gestor de copia local sirva para la actualización de cualquier repositorio, ya sea el repositorio principal, sobre el cual se han validado las transacciones, o uno replicado del principal.

En la figura 7 se representa el proceso seguido por el gestor de copia local:

1. obtiene del repositorio maestro las transacciones que no han sido ejecutadas aún sobre el repositorio esclavo
2. por cada transacción y operación:

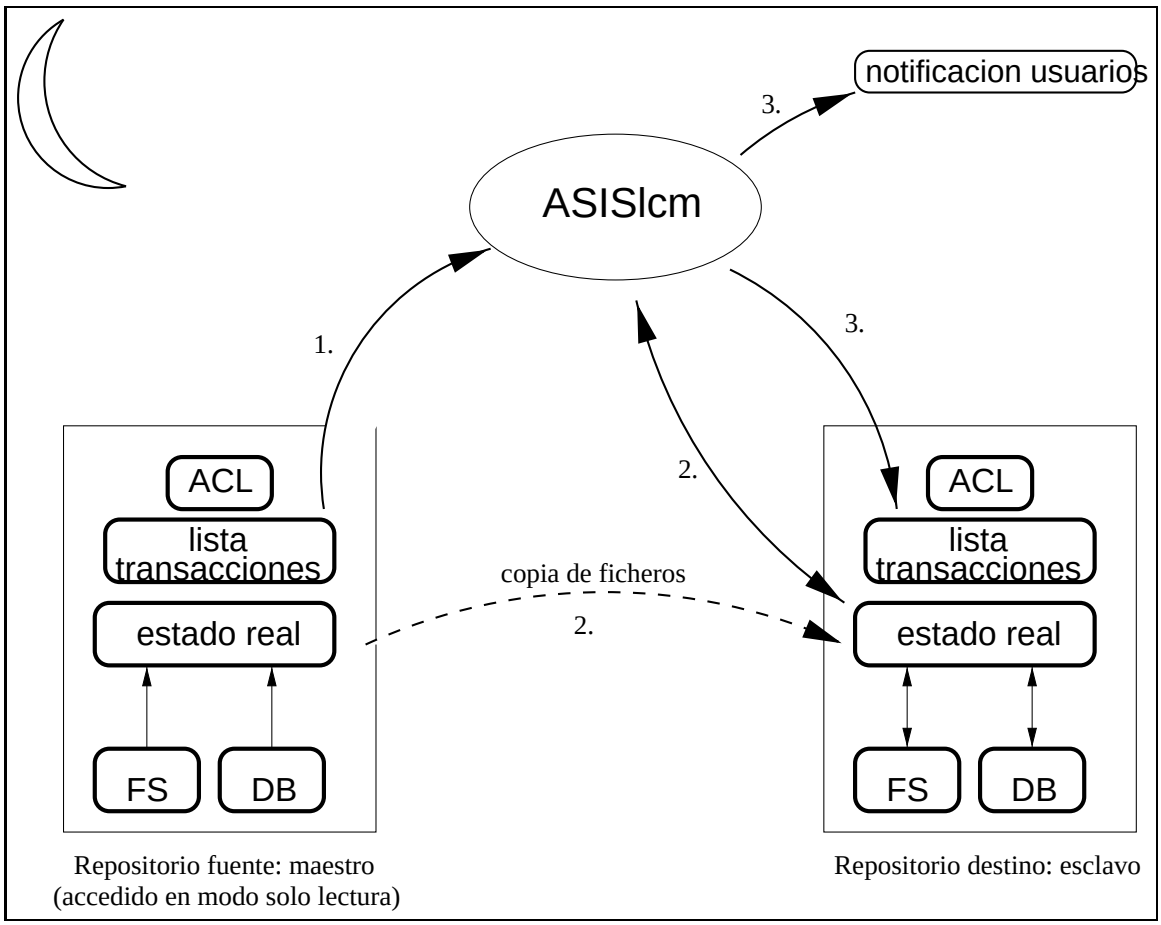

Figura 7: gestor de copia local

- ejecución "real" de la operación sobre el repositorio destino. Se accederá al maestro solamente para copiar aplicaciones nuevas o modificadas, el resto de la ejecución es local.

3. se actualiza la lista de transacciones ejecutadas en el repositorio destino, y se notificará a los usuarios del repositorio de los cambios habidos en él, por correo electrónico.

El proceso es muy similar al seguido por el gestor de transacciones, con la diferencia de que aquí se modificará el estado "real", es decir, la base de datos y el sistema de ficheros del repositorio esclavo. En cambio, el repositorio maestro o fuente será accedido sólo en modo lectura y no modificará nada en él.

Dado que la ejecución implicará acceso físico a objetos del repositorio, ésta ejecución no será tan rápida como la ejecución virtual llevada a cabo por el gestor de transacciones.

El contenido del repositorio maestro será accedido solamente cuando hay que transferir paquetes nuevos o ficheros modificados; el resto de las operaciones se *ejecutan localmente*. Así por ejemplo, el paso de una aplicación de un área a otro puede ser ejecutada sin tener que acceder al repositorio maestro. De esta manera, la carga de red es mínima.

El gestor de copia local no efectuará operación alguna si no ha habido cambios en el repositorio maestro. Solamente copiará aplicaciones del maestro cuando sea necesario, es decir, cuando haya un objeto nuevo sobre éste. De esta manera, la replicación se limita a lo imprescindible.

Debido a que el gestor de transacciones ha validado la posibilidad de la ejecución de las transacciones sobre el repositorio inicial, se garantiza la ejecución sobre éste repositorio principal. No se garantiza la ejecución sobre los repositorios replicados del principal.

En este caso, igual que en los casos que haya un problema de red, se deshace la transacción fallida, se para el proceso de ejecución y se envía un correo electrónico al responsable del repositorio esclavo, para que resuelva el problema y pueda procederse con la ejecución de las transacciones. Sin embargo, aunque no todas las transacciones hayan sido ejecutadas, el estado del repositorio es consistente al interrumpirse el proceso de actualización entre dos transacciones.

Simplificación y reducción de transacciones

La frecuencia de actualización del repositorio esclavo deberá ser configurable por el responsable de éste. El repositorio principal debería ser actualizado cada noche; así los cambios validados a lo largo del el día por el gestor de transacciones estarán visibles a la mañana siguiente. Asimismo habrá repositorios que junto al principal formen una agrupación o *cluster* y que por tanto deberán mantener idéntico contenido, por lo cual se lanzará la replicación justo cuando el repositorio principal haya finalizado su actualización.[3]

En el caso de repositorios remotos, sus responsables pueden decidir realizar las actualizaciones en fines de semana, o una vez al mes etc. Pero entre una actualización y otra, el repositorio maestro puede haber sufrido una secuencia de cambios que no sean linealmente

[3] un ejemplo de cluster es el existente en el CERN, donde los repositorios basados en AFS y en NFS deben estar sincronizados.

propagables al repositorio esclavo. Por ejemplo, entre dos actualizaciones del esclavo puede haberse introducido un paquete en el repositorio maestro, y al poco tiempo se borra este paquete. Si el esclavo intenta reproducir linealmente la secuencia de operaciones habidas en el repositorio maestro, al ver la operación de introducir el paquete en el repositorio, solicitará del repositorio maestro ese paquete, el cual ya no lo tiene al haber sido borrado. Por lo tanto, el esclavo deberá primero simplificar la lista de transacciones del maestro. Para ello, se apoyará en una *tabla de reducciones*, que se incluye en el apéndice B a partir de la página 125, junto a una descripción más detallada del proceso seguido.

Esta simplificación se guiará por el uso de las siguientes reglas:

1. **si** dos operaciones afectan a una misma aplicación **y** sobre la misma arquitectura **entonces** serán *candidatas* a ser simplificadas.
2. **si** existen dos candidatas en una misma transacción **entonces** se consultará la tabla de reducción y se simplificará si procede.
3. **si** existen dos candidatas en transacciones distintas **entonces** se procederá a fusionar las dos transacciones en una sola.

Se aplicarán estas reglas sobre el total de la lista de transacción para obtener la lista mínima. Al fusionar dos transacciones no se pierde la atomicidad de éstas, simplemente se aumenta su granularidad. La lista mínima es la que finalmente ejecutará el gestor de copia local del repositorio esclavo.

Filtrado de transacciones

El gestor de copia local podrá replicar un subconjunto del contenido del repositorio maestro, configurando el responsable del repositorio esclavo qué familias y plataformas desea replicar de las disponibles en el maestro.

4.2.8 Diagrama de contexto

En la figura 8 se muestra el diagrama de contexto del gestor de transacciones y del gestor de copia local. Las entidades externas serán:

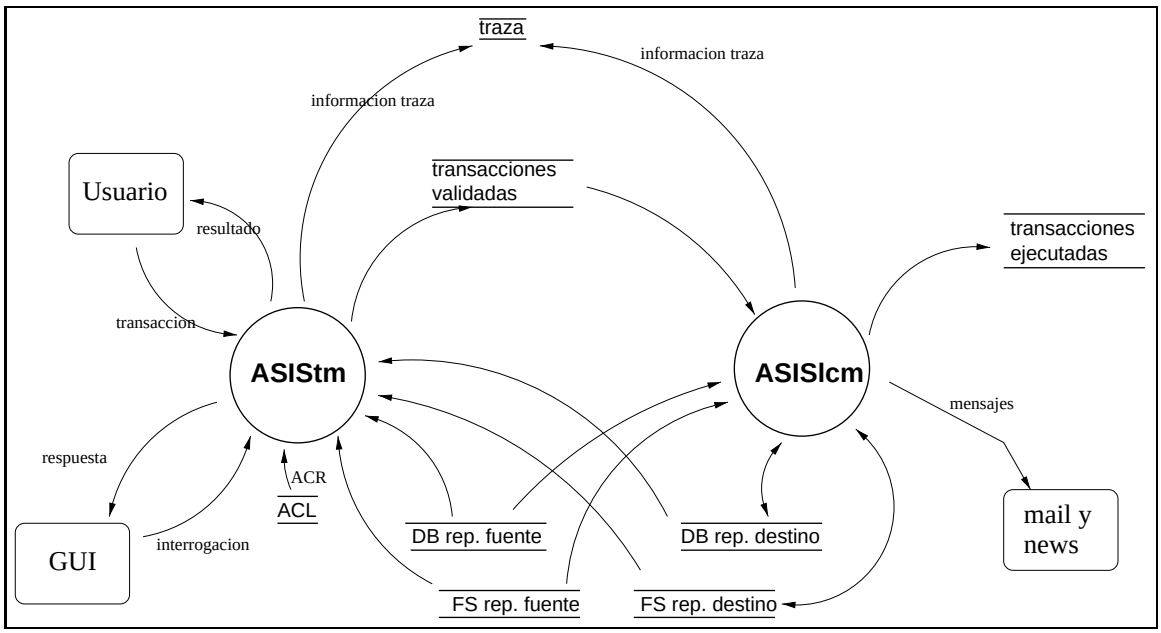

Figura 8: diagrama de contexto

- El usuario, que enviará la transacción al sistema y recibirá el resultado de ésta
- El interfaz gráfico (GUI), que interactuará con el gestor de transacciones a través de un modelo cliente/servidor
- el sistema de mensajería electrónica (mail y news), que distribuirá a los usuarios los mensajes generados por el gestor de copia local.

El usuario podrá interactuar con el gestor de transacciones a través de un interfaz gráfico. Se dispondrá de una traza de todas las operaciones efectuadas durante la ejecución de ambos sistemas.

La descripción funcional del gestor de transacciones y del gestor de copia local se encuentran en el apéndice A.1, a partir de la página 112.

4.2.9 Formato de transacciones

Se guardará la siguiente información por cada transacción validada y/o ejecutada:

- identificador de transacción

- descripción de la transacción
- fecha y hora de la transacción
- autor de la transacción
- máquina de ejecución
- lista de operaciones:
  - nombre de la operación
  - Familia/Producto-Versión afectada
  - en las operaciones tipo "patch", el fichero afectado
  - lista de (identificadores de) plataformas afectadas

Sintaxis de entrada

La sintaxis de entrada de las transacciones para el gestor de transacciones será:

```
Description: aquí viene la descripción

Operacion1 Fam/Prod-Ver (plataf1,plataf2..,platafN)
Operacion2 Fam/Prod-Ver (plataf1,plataf2..,platafN)
.
.
OperacionN ...
```

Una transacción semánticamente simplificada, pero sintácticamente correcta para ser enviada al gestor de transacciones es la mostrada a continuación:

Supongamos que deseamos sustituir la aplicación GNU.EDIT/emacs-19.34 por la nueva versión GNU.EDIT/emacs-20.2 en las plataformas Solaris 2.5 (`sun4x_55`), Digital Unix 3.2 (`alpha_osf32`), HP Unix 10.2 (`hp_ux102`) y la parte compartida (`share`). Además, se deberá actualizar simultáneamente la aplicación `vm` dado que depende de emacs. La plataforma Digital Unix 4.0 (`alpha_dux40`) es nueva y le son añadidas estas aplicaciones.

La formalización de la transacción es la siguiente:

```
Description: actualización coordinada de emacs y vm.

RemoveFromProduction GNU.EDIT/emacs-19.34
(share,sun4x_55,alpha_osf32,hp_ux102)

RemoveFromProduction GNU.EDIT/vm-5.96beta
(share,sun4x_55,alpha_osf32,hp_ux102)

IntroduceInProduction GNU.EDIT/emacs-20.2
(share,sun4x_55,alpha_osf32,hp_ux102,alpha_dux40)

IntroduceInProduction GNU.EDIT/vm-6.0
(share,sun4x_55,alpha_osf32,hp_ux102,alpha_dux40)
```

4.3 Gestión de dependencias

Aún con el uso de transacciones, el modelo presentado hasta el momento para la manipulación del repositorio es incompleto. Si bien a nivel de una transacción el gestor de transacciones dispone de mecanismos para evitar que aparezcan incoherencias (a través de las comprobaciones de coherencia descritas y del concepto mismo de transacción), no se considera ninguna clase de comprobación que abarque la interrelación entre aplicaciones y la relación de una aplicación con su entorno.

Es por lo cual existe la necesidad de establecer un modelo para describir explícitamente las interacciones y dependencias entre los objetos software contenidos en un repositorio multiplataforma, y también las interacciones de estos objetos y el entorno de ejecución local. Este modelo deberá servir para facilitar tanto a los gestores de producto como a los administradores determinar cuando un conjunto de aplicaciones forman un conjunto coherente y válido para ser ofrecido a los usuarios finales.

Seguidamente, se describirá el *modelo* desarrollado para describir las dependencias en ASIS. A continuación se mostrará el *proceso* definido para la gestión de dependencias dentro del ciclo de vida de distribución de una aplicación contenida en un repositorio ASIS. Finalmente, se presentará el diseño de las *herramientas* para la edición y comprobación de las dependencias.

4.3.1 Análisis de relaciones y dependencias en ASIS

Las relaciones que se pueden establecer para las aplicaciones en ASIS se pueden clasificar dentro de dos clases: las relaciones entre aplicaciones y las relaciones de una aplicación con su entorno de ejecución.

Relaciones entre aplicaciones del repositorio

Analizando las relaciones entre aplicaciones almacenadas en el repositorio ASIS del CERN se pueden resaltar las siguientes observaciones:

- Muchos paquetes son componentes de un sistema mayor, que ha sido subdividido

en partes para tener mayor flexibilidad en su gestión, o bien para permitir el uso y mantenimiento por separado de estas partes. Como ejemplo en ASIS cabe citar las librerías de cálculo matemático CERNLIB, que son divididas por su enorme tamaño, por ser gestionadas por varias personas y por haber partes que permiten su uso por separado.

- También se da el caso de que aplicaciones independientes permiten cooperar entre sí, siendo su uso conjunto altamente útil. Un ejemplo es el conjunto formado por el editor de texto EMACS, el compilador de LaTeX y el visualizador de PostScript GHOSTVIEW.

Se puede definir una *asociación* de aplicaciones independientes que forman una unidad mayor.

Las interacciones entre aplicaciones son las siguientes:

- Una aplicación requiere de otras, por ejemplo: la aplicación A exige o recomienda que la aplicación B esté instalada ya que requiere funcionalidades suyas. Puede ser incluso que la aplicación A funcione solamente con ciertas versiones de B, pero no con otras.
- También se puede dar el caso inverso, es decir, que una aplicación combinada con otra de lugar a incompatibilidades o disfuncionalidades y que por lo tanto haya que evitar su uso simultáneo. Este es el caso si la aplicación A y la aplicación B no deben ser instaladas simultáneamente por competir por un mismo recurso que no son capaces de compartir.[4]

Se establecen *dependencias* entre aplicaciones, que pueden ser de tipo "positivo" (cuando una aplicación hace uso de otra) o "negativo" (cuando una aplicación es incompatible con otra).

- Una aplicación puede requerir que de un conjunto de aplicaciones al menos una esté disponible. Por ejemplo, para ejecutar una cierta funcionalidad la aplicación A puede cooperar con la aplicación B o con la aplicación C.

[4]Es el caso de muchos lectores de correo que exigen acceso exclusivo al servidor de correos SMTP.

Por lo tanto, las dependencias positivas pueden dar lugar a alternativas.

- Una aplicación puede requerir la presencia de otra aplicación que cubra ciertas características funcionales, pero sin necesidad de concretar sobre qué aplicación debe ser. Por ejemplo, un programa de gráficos puede requerir o recomendar la presencia de una aplicación genérica para procesar trabajos de impresión.

Se pueden *clasificar* las aplicaciones en grupos según sus *funcionalidades* para luego poder establecer dependencias hacia un grupo sin tener que referirse a una aplicación en concreto.

Relaciones con el entorno de ejecución

Gran parte de las interactuaciones de una aplicación se hacen hacia el entorno en el cual se está ejecutando; es decir, hacia los *elementos locales* de la estación de trabajo en la que se lanza la aplicación. Bajo estos elementos se puede incluir por ejemplo la memoria libre disponible, las librerías y paquetes software instalados localmente, la versión y revisión del sistema operativo, ficheros de configuración, etc. La aplicación podrá necesitar que se cumplan determinadas condiciones sobre algunos de estos elementos. Algunos ejemplos pueden ser: que la memoria RAM sea superior a 16MB, o que exista un fichero de configuración del sistema de correos en un determinado directorio[5], o que esté instalada una determinada versión de cierta librería necesaria para la ejecución de la aplicación. Para una misma aplicación, estas dependencias varían mucho en función de la plataforma, por lo que conviene poder distinguir cuando una relación de este tipo es establecida para todas las plataformas o solamente para una específica.

Clasificación de objetos

Los dos tipos de agrupación descritos se pueden formalizar como:

- **paquetes virtuales (*virtual packages*):** En los *paquetes virtuales* se expresa la *asociación* de aplicaciones que forman una unidad mayor.

[5] este es el caso del programa `sendmail` bajo UNIX, que según el sistema operativo utilizado busca el fichero de configuración en el directorio `/etc/sendmail.cf` o en `/etc/mail/sendmail.cf`

- **grupos funcionales (*functional groups*):** Los miembros de un grupo funcional serán aplicaciones que cumplan una determinada funcionalidad.

De esta manera, se definen meta-aplicaciones; en los paquetes virtuales por asociación y en los grupos funcionales por clasificación.

Dependiendo de sus características, una aplicación podrá pertenecer a ninguno, uno o varios paquetes virtuales o grupos funcionales. A su vez, los paquetes virtuales y grupos funcionales serán divididos en versiones para una mayor flexibilidad a la hora de definirlos.

Los *elementos locales* también pueden ser clasificados:

- **configuración local (*local settings*):** Aquí se incluyen las características del hardware, la versión exacta del sistema operativo, el espacio libre en el disco duro local, etc.
- **paquetes locales (*local packages*):** Muchas de las plataformas incluidas en ASIS tienen su propio y particular sistema para actualizar el software de una estación de trabajo, como DEBIAN DPKG [22], SD-UX en HP Unix, etc. Estos sistemas se utilizan para instalar localmente paquetes no contenidos en el repositorio ASIS, por ejemplo utilidades y actualizaciones del sistema operativo, "drivers" para periféricos, etc. La mayoría de estos sistemas ofrecen herramientas para el control y seguimiento de los paquetes instalados localmente. Así, en las plataformas que dispongan de éstos sistemas se podrá definir una interfaz para consultar su estado.
- **ficheros:** En algunos casos una aplicación establece una dependencia hacia un determinado fichero; por ejemplo hacia un fichero de configuración, una librería dinámica, o un fichero ejecutable. Por ejemplo, el sistema de control de versiones CVS requiere la presencia de los siguientes ejecutables: `ci, co, rcs`. Otro ejemplo son las aplicaciones tipo *script*, que buscan el intérprete que requieren para ejecutarse en un determinado directorio y bajo un determinado nombre.[6]

[6] así, un *script* en lenguaje TCL que empieza por `#!/usr/local/bin/tclsh7.6` requiere que se encuentre en el directorio `/usr/local/bin` un ejecutable llamado `tclsh7.6`, de lo contrario no será capaz de

A continuación se muestra el formato exacto de los objetos identificados:

formato de los *objetos* componentes de dependencias:

- **Paquetes Virtuales**: `VP:nombre-versión`, ejemplo: `VP:TeX_base-1`
- **Grupos Funcionales**: `FG:nombre-versión`, ejemplo: `FG:PS_visualizer-1`
- **Ficheros y Ejecutables**: para los ficheros: `X[F]:/dir1/dir2/nombre` (nombre con el camino de acceso completo), ejemplo: `/etc/sendmail.cf`; y para los ejecutables: `X[E]:nombre` (con o sin camino de acceso)
- **Paquete local**: `LP:nombre-paquete`, ejemplo: `LP:SUNWsms`
- **Configuración local**: `S[tipo,operador]:valor`. El *tipo* podrá ser por ejemplo la memoria RAM (RAM), el nivel de revisión del sistema operativo (patchlevel), etc. Los *operadores* serán igual ('='), mayor('>'), menor('<'); también mayor igual ('>=') y menor igual ('<='). Por ejemplo: `S[RAM,>=]:32` (memoria RAM mayor igual a 32MB).
- **aplicaciones ASIS**: `Familia/Producto-versión`. En caso de que varias versiones de un mismo producto sean objeto de una misma dependencia, en la forma version1 ó version2 ó ..., se podrá utilizar la abreviación `Familia/Producto-(version1,version2,...,versionN)`

Tipos y niveles de dependencias

Una aplicación podrá expresar dependencias hacia cualquiera de los objetos identificados. Estas dependencias pueden ser positivas o negativas. Las dependencias positivas indican que una aplicación hace uso de ciertos objetos.

Habrá dos *tipos* de dependencias positivas:

- **requiere (*requires*):** expresa una dependencia absoluta. Es decir, la aplicación necesita imperativamente de los objetos en cuestión.[7]
- **recomienda (*recommends*):** la aplicación es más operativa con los objetos en cuestión, pero puede utilizarse independientemente.

ejecutarse.

[7] un ejemplo en el CERN puede ser la dependencia del paquete que contiene el lector de correos `vm` para emacs (GNU.EDIT/vm) hacia el paquete que contiene a `emacs` (GNU.EDIT/emacs)

El incumplimiento de una dependencia de tipo "requiere" provocará un error mientras que una dependencia incumplida de tipo "recomienda" dará lugar a una advertencia.

El expresar una dependencia hacia un grupo funcional significa que *se deberá encontrar al menos un componente de este grupo funcional para que la dependencia se verifique*, dado que un grupo funcional agrupa aplicaciones con funcionalidades similares. En el caso de los paquetes virtuales, *se deberán encontrar todos los componentes del paquete para que la dependencia se verifique*, dado que un paquete virtual define un conjunto de aplicaciones que interoperan.

Las dependencias *negativas* expresan rechazo hacia los objetos indicados. Habrá dos tipos de dependencias negativas:

- **choque (*clashes*):** rechazo absoluto. La aplicación no puede ser utilizada simultáneamente con el objeto sobre el cual se expresa la dependencia.[8]
- **conflicto (*conflicts*):** no es recomendable el uso simultáneo de la aplicación con el objeto.

Análogamente a las dependencias positivas, el incumplimiento de una dependencia de tipo "choque" provocará un error; una dependencia incumplida de tipo "conflicto" dará lugar a una advertencia.

Dado que las dependencias en una misma aplicación pueden variar de una plataforma a otra, se definen dos *niveles* de dependencias, para mayor flexibilidad a la hora de definirlas:

- las dependencias *generales* son independientes de la plataforma (afectan a todas ellas).

[8] un ejemplo de un *choque* es cuando dos aplicaciones tienen ficheros del mismo nombre en el mismo directorio, lo cual impide que las dos aplicaciones puedan ser accedidas al mismo tiempo. Esta circunstancia es detectada por el gestor de transacciones comprobando fichero a fichero, pero estableciendo esta dependencia entre dos aplicaciones se "marcarán" directamente como incompatibles entre sí.

- las dependencias *de sistema* son las que son específicas a una plataforma, no afectando a las demás plataformas.

Por cada aplicación, se podrán definir tantas dependencias como sea necesario, del tipo y al nivel necesario. Existen algunas restricciones respecto al uso de los objetos en las dependencias:

- los objetos de tipo configuración local y paquetes locales solamente se permitirán en las dependencias de sistema, ser específicas a una plataforma.
- los grupos funcionales y paquetes virtuales no pueden ser utilizados como destino de una dependencia "negativa".
- las dependencias hacia paquetes virtuales y grupos funcionales deberán establecerse sobre una versión concreta de éstos. No se aceptan dependencias hacia paquetes virtuales o grupos virtuales de manera genérica, ya que son ambiguas.
- análogamente, las dependencias hacia productos ASIS han de indicar la versión del producto del que se depende.

En las dependencias de tipo "positivo" se pueden establecer *alternativas*: la dependencia se satisface si se satisface *al menos uno de los objetos dados en la lista.* En el caso de las dependencias de tipo "negativo" las alternativas carecen de sentido.

A continuación se muestra un ejemplo de modelización de dependencias:

Supongamos que tenemos la versión 0.10.7 de un producto llamado LyX, perteneciente a la familia GNU.EDIT. Este producto *requiere* para su funcionamiento otro producto incluido en ASIS, en concreto xforms, perteneciente a la familia X11; tiene que ser la versión 0.81 ó la versión 0.92. Además, requiere que se encuentre instalado el *paquete virtual* TeX_base (versión 1). También se *recomienda* que esté instalado alguno de los tres objetos: el paquete ASIS GNU.EDIT/ispell-3.1.20, o bien un ejecutable llamado ispell ó un ejecutable llamado spell. Asimismo se recomienda que esté instalado algún producto perteneciente al grupo funcional PS_visualizer (versión 2). En cambio, se produce un conflicto con la aplicación MISC/xemacs-19.14 al tener ambos un fichero del mismo nombre en el mismo directorio. En el caso de la versión para Sun Solaris 2.5, se requiere para su funcionamiento que esté instalado el paquete

X11/Xaw3d-0.6, o bien que se encuentre la librería /usr/lib/X11/libXaw3d.so.5.0. Además, tiene que estar instalado el paquete local SUNWsml y la memoria RAM deberá ser al menos 32MB.

La formalización de las dependencias generales queda:

```
Requires:
X11/xforms-(0.81,0.92)
VP:TeX_base-1

Recommends:
GNU.EDIT/ispell-3.1.20 OR X[E]:ispell OR X[E]:spell
FG:PS_visualizer-2

Conflicts:
MISC/xemacs-19.34
```

las dependencias adicionales para Solaris 2.5:

```
Requires:
X11/Xaw3d-(0.6,0.8) OR X[F]:/usr/lib/X11/libXaw3d.so.5.0
LP:SUNWsml
S[RAM,>=]:32
```

4.3.2 Consideraciones de entorno y relación con otros sistemas

Basándonos en la clasificación de los usuarios de ASIS, podemos definir cuales son los intereses de cada tipo de usuario por el sistema de gestión de interrelaciones.

- los *gestores de producto* definirán las interacciones entre "sus" productos y otros objetos. Ellos editarán y modificarán las dependencias, los paquetes virtuales y grupos funcionales. Al manipular el repositorio utilizando el gestor de transacciones (`ASIStm`), deberán ser informados de las consecuencias si quieren añadir, borrar o cambiar el estado de alguna aplicación considerando sus dependencias.
- Los *administradores* podrán verificar la coherencia de la configuración establecida con `ASISwsm`, verificando las dependencias entre los productos seleccionados para ser instalados en la estación de trabajo y su relación con la configuración local.
- los *gestores del repositorio* podrán obtener una visión general de la coherencia de las relaciones definidas entre objetos del repositorio. Asimismo, podrán consultar y

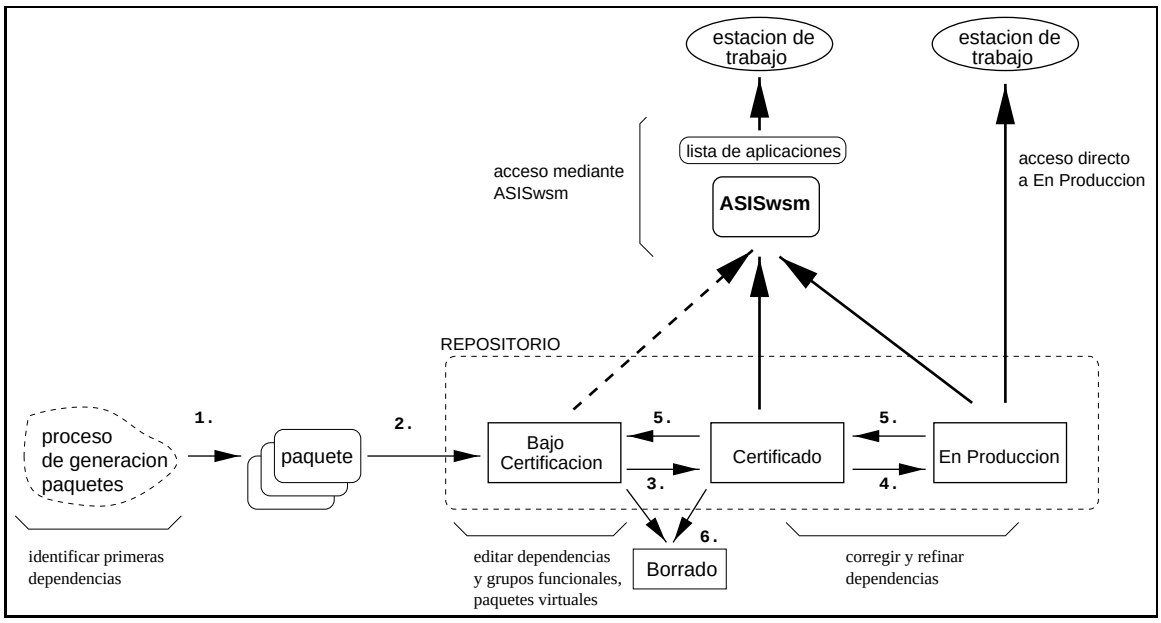

Figura 9: proceso de detección y edición de dependencias

modificar toda la información respecto de las relaciones.

Las consideraciones sobre el *entorno de funcionamiento* serán las mismas que en el sistema de modificación y gestión del repositorio.

4.3.3 Definición del proceso para detección y edición de dependencias

En la figura 9 se ilustra el proceso definido para crear, modificar y comprobar las dependencias dentro del ciclo de vida de una aplicación en el repositorio ASIS. Las dependencias serán establecidas inicialmente con el paso de la aplicación al repositorio y serán refinadas iterativamente.

- Cuando se genera un nueva aplicación (1), el gestor de producto detectará las primeras dependencias, a través de la documentación que viene con el producto, examinado partes del código fuente, la experiencia con versiones anteriores o productos similares, etc. Además, muchos procesos de configuración, compilación e instalación de las aplicaciones realizarán búsquedas y solicitarán información sobre qué otros productos y objetos están disponibles para basarse en ellos.

El gestor de producto intentará minimizar las dependencias hacia el entorno de ejecución, desplazando si es posible las dependencias hacia el repositorio ASIS, ya que el contenido de éste estará disponible a todas las máquinas. Por ejemplo, si una librería requerida por una aplicación está disponible tanto localmente como en ASIS, se configurará la aplicación para que utilice la contenida en ASIS, y se definirá una dependencia hacia ésta.

De esta manera, las dependencias hacia una configuración específica se minimizan, e inversamente, se evita imponer restricciones sobre el entorno operativo del cliente.

En general, las aplicaciones deben de ser lo más *autocontenidas* posible, tanto hacia el entorno de ejecución como entre ellas.

Al generarse la aplicación en paralelo para todas las plataformas usando la herramienta `ASISgen` el gestor de producto verá cuando una dependencia es general o afecta solamente a ciertos sistemas.

Durante la generación de paquetes el gestor de producto definirá las dependencias que va detectando. El gestor de producto podrá crear y editar las dependencias a través de la línea de comandos o de una interfaz gráfica. El proceso de detectar dependencias es heurístico y manual dado la falta absoluta de estándares al respecto.

En caso de existir versiones anteriores del producto en el repositorio, se podrá utilizar las dependencias definidas de una de estas versiones como *plantilla*, e ir adaptando progresivamente las dependencias plantilla a las existentes sobre esta nueva versión. De esta manera, se minimiza el esfuerzo necesario por parte del gestor para definir las dependencias, cambiándose solamente lo necesario.

- Una vez terminado el proceso de generación, el gestor de producto accederá al gestor de transacciones para introducir la aplicación en el repositorio (2). El gestor de transacciones no hará ninguna comprobación respecto al cumplimiento de las dependencias cuando el estado destino es "Bajo Certificación". Esto es así porque las aplicaciones en ese estado son probadas por un grupo de usuarios "beta" interesados en examinarlas, éstos serán los únicos en utilizarlas. Derivado de los informes y sugerencias de los usuarios "beta" se pueden modificar y detectar dependencias, que el gestor de producto modelizará, actualizando la base de datos. También se pueden editar o crear grupos funcionales y paquetes virtuales para incluir la aplicación.

- Si la aplicación ha sido probada suficiente y satisfactoriamente, se puede cambiar su

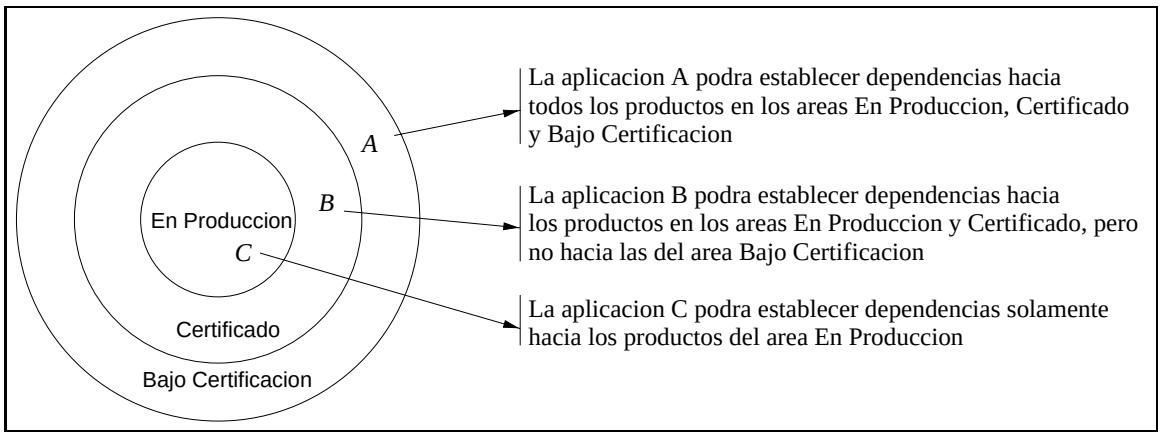

Figura 10: zonas del repositorio

estado a "Certificado" (3). El gestor de transacciones comprobará si las dependencias de la aplicación hacia otras aplicaciones del repositorio se satisfacen[9]. Las comprobaciones se harán por cada plataforma por separado, al ser la manipulación de versiones de producto independiente por plataforma. Por cada una se comprobarán tanto las dependencias generales como las de sistema.

Las comprobaciones se harán considerando el repositorio dividido en zonas correspondientes a los estados y el conjunto de aplicaciones que se encuentran en ese estado. La figura 10 ilustra esta división. Las aplicaciones de un estado, junto a las aplicaciones que se encuentran en estados superiores, deberán ser *autocontenidas*: no dependerán de aplicaciones que estén en un estado inferior.

Esto se debe a que las estaciones de trabajo tendrán normalmente acceso solamente a las aplicaciones que se encuentran en los estados "Certificado" y "En Producción". El acceso a las aplicaciones "Bajo Certificación" solamente se hará para la evaluación de éstas, y es restringido. Además, el acceso a un estado implica que también se puede acceder a los superiores. Por ejemplo, un usuario "beta" tendrá acceso a los estados "Bajo Certificación", "Certificado" y "En Producción", mientras que los usuarios normales tendrán acceso a los estados "Certificado" y "En Producción".

Adicionalmente, el conjunto de aplicaciones en el estado más alto ("En Producción") *deberá estar libre de conflictos* entre aplicaciones. Esto se debe al hecho de que todas

[9]Las dependencias hacia el entorno de ejecución se ignoran en las comprobaciones efectuadas por el gestor de transacciones, ya que solamente tienen sentido a la hora de comprobar la factibilidad del uso de la aplicación por una estación de trabajo.

las aplicaciones en este estado son accesibles directa y simultáneamente. Muchas estaciones de trabajo acceden directamente al área de producción. Estas estaciones tienen acceso a todas y cada una de las aplicaciones de este estado, pero a ninguna más, por lo que éste estado deberá ser autocontenido y coherente.

Por lo tanto, al querer subir una aplicación de estado habrá que evaluar si las dependencias de éste producto se ven cumplidas dentro de los estados *iguales o mayores al estado destino*. Por ejemplo, si se quiere pasar la aplicación A del estado "Bajo Certificación" al estado "Certificado", y resulta que A depende de las aplicaciones B y C, se exigirá que éstas se encuentren al menos en estado "Certificado".

Los grupos funcionales y los paquetes virtuales carecen de estado propio. Al definir una dependencia hacia ellos, se define hacia al menos uno de sus componentes (en el caso de los grupos funcionales) o hacia todos (paquetes virtuales). Por lo tanto si la aplicación A expresa una dependencia hacia el grupo funcional F, formado por las aplicaciones B y C, y se quiere pasar A al estado "Certificado", entonces bien B ó C deberán encontrarse al menos en "Certificado" para que la dependencia se verifique.

- Al pasar una aplicación al estado "En Producción" (4), además habrá que verificar, como se acaba de decir, que no se generen conflictos. Esto habrá que comprobarlo recíprocamente: por un lado, verificar que la aplicación no contiene una dependencia negativa hacia ninguna de las aplicaciones en este estado, y por otro lado habrá que verificar que ninguna de las aplicaciones contiene una dependencia hacia la aplicación en cuestión. De esta manera, el conjunto de aplicaciones en el estado "En Producción" *por construcción siempre será coherente.*

  La transición hacia el estado "En Producción" además permite detectar dependencias de tipo "choque", cuando se detectan conflictos en el espacio de nombres de ficheros (ficheros con mismo nombre en el mismo directorio) del área de "En Producción".

- Al *bajar una aplicación de estado(5) o al borrarla*(6), hay que verificar que no queden aplicaciones en el estado original o superiores a éste que dependan de ella. Por ejemplo, si tenemos que A y B están "En Producción" y que B depende de A, y decidimos bajar de estado a A, deberemos bajar también B al mismo estado que A, ya que de lo contrario B tendría una dependencia hacia un producto cuyo estado es más bajo que el suyo propio, y por lo tanto el estado "En Producción" dejaría de ser autocontenido.

También se examinará si la aplicación pertenece a algún grupo funcional o paquete virtual. En el caso de pertenecer a algún grupo funcional se comprobará que siguen quedando aplicaciones que forman parte del grupo en un estadio igual o superior al estado original de la aplicación. En el caso de pertenecer la aplicación a un paquete virtual se comprobará si existen aplicaciones que dependen de este paquete virtual siendo su estado mayor al estado destino de la aplicación, caso en el que esta dependencia sería incumplida.

El conjunto de comprobaciones realizadas por el gestor de transacciones se pueden modelizar en formato de regla como mostrado a continuación:

Al pasar una aplicación A del estado inicial e_i al estado final e_f se aplicarán las siguientes reglas para comprobar que las dependencias se cumplen (siendo C="Certificado", IP="En Producción"):

1. $e_i \notin (IP, C) \wedge e_f \in (IP, C) \Rightarrow$ comprobar que se han definido dependencias para A
2. $e_i = IP \wedge e_f \neq IP \Rightarrow$ comprobar si hay aplicaciones que dependen de A en IP
3. $e_i = C \wedge e_f \notin (IP, C) \Rightarrow$ comprobar si hay aplicaciones que dependen de A en C
4. $e_i \in (IP, C) \wedge e_f \notin (IP, C) \Rightarrow$ comprobar integridad dependencias a grupos funcionales y paquetes virtuales de los que forma parte A
5. $e_i \neq IP \wedge e_f = IP \Rightarrow$ comprobar si las dependencias se satisfacen en IP
6. $e_i \neq IP \wedge e_f = C \Rightarrow$ comprobar si las dependencias se satisfacen en (C, IP)

En cualquier momento se permitirá la edición de las dependencias de una aplicación a personas autorizadas (gestores de producto con permiso para modificar esa aplicación). De esta manera se podrán ir corrigiendo y refinando las dependencias establecidas.

Verificación de dependencias en el entorno de ejecución

Las estaciones de trabajo que accedan exclusivamente al área de "En Producción" tendrán la garantía que no habrá conflictos entre aplicaciones en ese área, como se acaba de ver.

En cambio, los administradores de estaciones de trabajo que deseen dar acceso a una configuración distinta a la dada en el área de "En Producción", correrán el riesgo que aparezcan conflictos y dependencias violadas entre las aplicaciones seleccionadas.

Por un lado, solamente en el área de producción se garantiza que no haya conflictos. Al ser el área de paquetes del repositorio el lugar donde se almacenan todas las aplicaciones "Certificadas" y "Bajo Certificación", no se pretende que la totalidad de las aplicaciones sean coherentes entre sí, ya que nunca se accederá simultáneamente al conjunto entero de aplicaciones.

Si bien se garantiza que un estado es autocontenido junto a sus superiores, esto simplemente significa que todas las dependencias *se pueden resolver* dentro de estos estados, no que no vaya a haber conflictos.

Un ejemplo: supongamos que un administrador de estación de trabajo decide seleccionar las aplicaciones A, B y C del área "Certificado". Puede ser que A dependa de X e Y, y además que A tenga una dependencia de conflicto con B. Una posible solución es eliminar B de la configuración y añadir a la configuración las aplicaciones X e Y, que por lo anteriormente dicho deberán estar en los estados "Certificado" o "En Producción", y por lo tanto accesibles al administrador.

Al comprobar las dependencias sobre una estación de trabajo se verificarán *todas* las dependencias: las dependencias entre aplicaciones, y también se comprobarán las dependencias de las aplicaciones sobre los elementos locales. Se verificarán las dependencias de cada una de las aplicaciones seleccionadas, y se informará al administrador si se han encontrado conflictos o dependencias incumplidas.

4.3.4 Diagrama de contexto

En la figura 11 se muestra el diagrama de contexto del sistema de dependencias. Las entidades externas serán

- el gestor de producto, que a través de un interfaz gráfico creará y editará las dependencias, los grupos funcionales y los paquetes virtuales
- el gestor de transacciones `ASIStm`, que solicitará la verificación de las dependencias

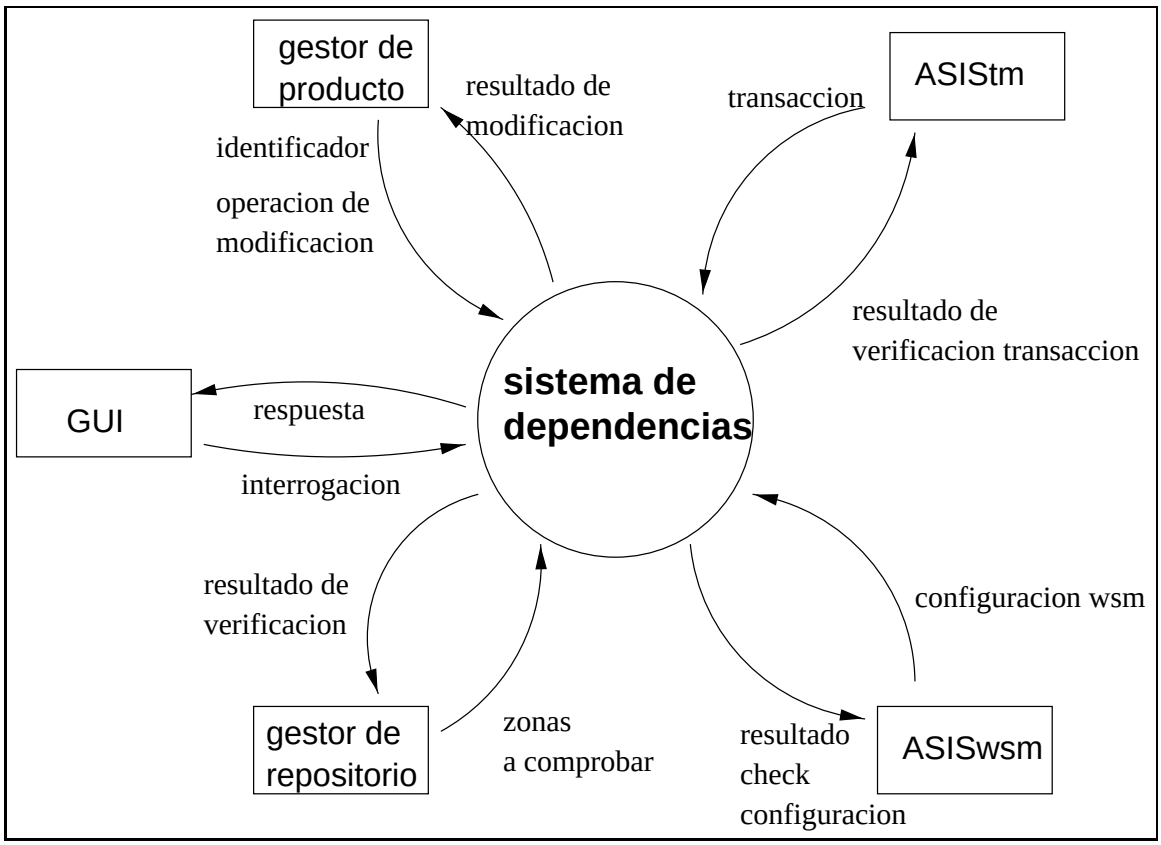

Figura 11: diagrama de contexto del sistema de dependencias

entre aplicaciones ASIS sobre los productos afectados por la transacción a validar

- el gestor de estación de trabajo `ASISwsm`, que solicitará la verificación de las dependencias de las aplicaciones que conforman la configuración seleccionada por el administrador. Se verificarán tanto las dependencias entre aplicaciones ASIS como las dependencias hacia el entorno de ejecución
- el gestor del repositorio, que solicitará la verificación de las dependencias de todas o un subconjunto de las aplicaciones del repositorio.

La descripción funcional del sistema se encuentra en el apéndice A.3, a partir de la página 118.

4.3.5 Modelo cliente/servidor

A continuación se describe el modelo cliente/servidor propuesto para la comunicación de las herramientas ASIS con sus interfaces gráficas.

Este modelo ofrecerá un método unificado, modular y eficiente para la comunicación entre las aplicaciones y sus interfaces gráficas.

El modelo existente hasta el momento tiene la limitación que exige a la interfaz gráfica lanzar, cada vez que ésta necesita algún dato sobre el sistema[10], un subproceso cuya salida son los datos requeridos. Este subproceso es normalmente la misma aplicación a quien se hace la interfaz, pero en un modo especial de consulta.

Este modelo deriva del hecho que las interfaces y las aplicaciones están implementados en lenguajes de programación distintos[11], que no permiten el intercambio directo de librerías. Al subproceso lanzado se le dota de opciones específicas por cada tipo de consulta, una vez computados los datos requeridos se entregan a la interfaz, y el subproceso finaliza. [12] Este modelo es altamente ineficaz dado que requiere lanzar una aplicación cada vez que se produce una consulta, y produce una ralentización inaceptable en estaciones de trabajo modestas.

Las primeras versiones de las interfaces gráficas ASIS existentes eran bastante primitivas y requerían pocos o poco complejos datos del sistema, por lo que el modelo adoptado era suficiente. Pero las interfaces han ido evolucionando y aumentando en complejidad, requiriendo más datos, y con ello hubo que implementar cada vez más funciones en los procesos servidores, y aumentaban las llamadas a éstos. Para contrarrestar este efecto, se intenta desplazar el peso de la transformación de datos a la interfaz. Pero esto da lugar a que muchos algoritmos y estructuras de datos tengan que ser duplicadas sobre la interfaz, lo cual dificulta el mantenimiento del sistema.

El resultado es una ralentización inaceptable en estaciones de trabajo modestas. Además, se complica para los desarrolladores la tarea de mantener todas las funciones que hacen

[10] ejemplos de datos solicitados al servidor pueden ser el nombre del repositorio, el número y nombre de familias, productos y versiones contenidas en éste, nombre y tamaño de los ficheros de un paquete etc.

[11] Se trata de Tcl/Tk para la interfaz y Perl para la aplicación

[12] La interfaz gráfica de la aplicación A lanza mediante un `fork` la misma aplicación A; A ejecuta la función e imprime el resultado por la salida estándar, la cual es recogida por la interfaz.

de interfaz entre una aplicación y otra, ya que éstas a veces se repiten en las distintas aplicaciones, ocasionalmente con nombre y formato distinto pero con funcionalidad similar, al no existir ninguna clase de formalismo al respecto.

Criterios de diseño

A la hora de rediseñar el método para comunicar una interfaz gráfica con su aplicación se decide el uso de un esquema cliente/servidor que:

- sea independiente de la localización de los procesos cliente y servidor.
- sea independiente de los lenguajes utilizados para la implementación de cliente y servidor.
- permita la agrupación de funciones de interfaz en componentes API [13] con el fin de favorecer su reutilización.
- pueda ser usado por todas las interfaces y aplicaciones ASIS.
- cuyo protocolo sea independiente del canal de comunicación utilizado.

El modelo cliente/servidor será utilizado por el *entorno de seguridad ASIS* [17]. Este llevará a cabo el proceso de autentificación de cliente y servidor, y una autorización de alto nivel (basada en nombre y dirección IP de las máquinas). Estos procesos se realizan independientemente de la autorización basada en los derechos de acceso a los productos, llevada a cabo por el gestor de transacciones y el editor de dependencias.

Este entorno de seguridad impone que las aplicaciones que modifican el repositorio, como el editor de dependencias `ASISwdep` y los gestores de transacciones y de copia local se ejecuten sobre un servidor dedicado y de acceso restringido. Solamente los gestores de repositorio tendrán cuenta en esta máquina y acceso directo al sistema de ficheros del repositorio. Las interfaces se conectarán mediante el modelo cliente/servidor a sus aplicaciones correspondientes, una vez que la conexión ha sido validada y autorizada por el entorno de seguridad.

[13] bajo módulo o componente API, *Application Programming Interface*, se entienden elementos software que permiten la intercomunicación entre aplicaciones.

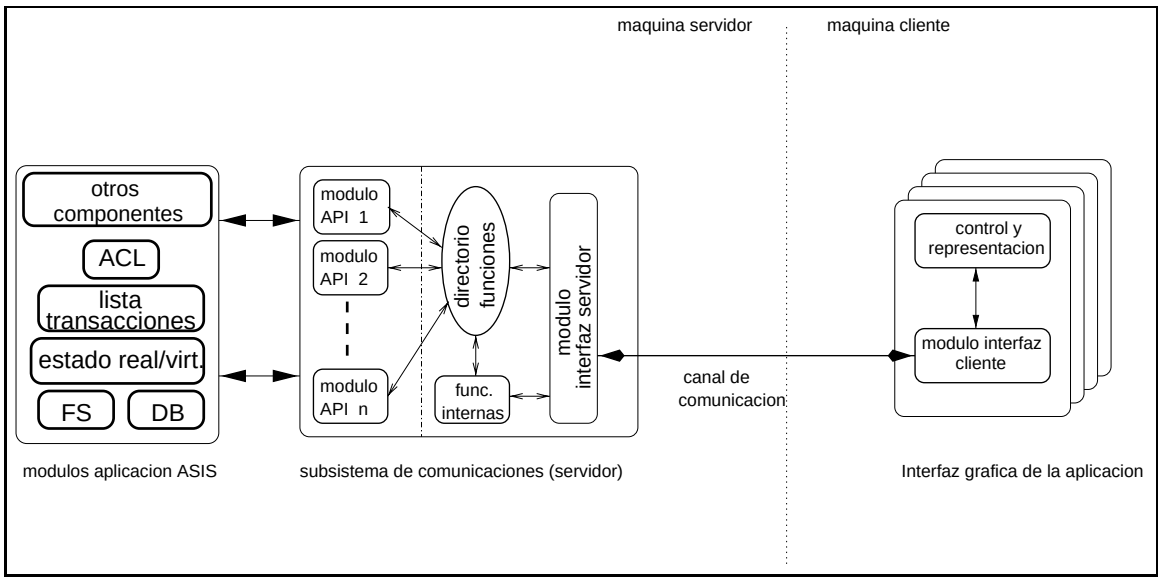

Figura 12: modelo cliente/servidor

Sin embargo, las aplicaciones que no modifican el repositorio, pero que acceden a él (como `ASISinfo` y `ASISwsm`, son ejecutadas directamente en las estaciones de trabajo de los clientes. Estas aplicaciones no son incluidas en el entorno de seguridad. La comunicación con la interfaz gráfica se efectuará asimismo a través del modelo cliente/servidor.

Descripción del modelo

En la figura 12 se esquematiza el modelo propuesto. Las aplicaciones ASIS dispondrán de un módulo de comunicaciones compuesto de una parte cliente y otra servidor. La parte cliente será accedida por el módulo de control de la interfaz gráfica y solicitará servicios remotos al servidor. La parte servidor se compone de un módulo interfaz y componentes o módulos API. La parte servidor se integra en la herramienta con la cual desea comunicar la interfaz, mientras que la parte cliente se integra en la interfaz. La aplicación podrá residir en la misma máquina que la interfaz o en una máquina remota.

Cada uno de los módulos API contendrá un número variable de funciones de interfaz (o funciones API). Cuando una función API es invocada, accede a componentes del sistema para obtener los datos requeridos por la interfaz. Los módulos podrán ser enlazados de manera independiente con la aplicación, en función de las necesidades de ésta.

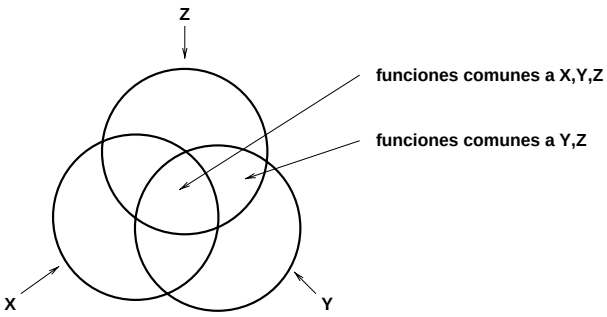

Figura 13: solapamiento del uso de funciones por aplicaciones

El agrupar las funciones en módulos independientes permite aumentar su adaptabilidad y facilita su mantenimiento.

La comunicación entre el módulo principal y las funciones API se hará a través de un directorio de funciones, en el cual se registran las funciones de cada módulo API.

Los módulos se podrán comunicar con el estado virtual y/o real (ver página 74), dependiendo del tipo de repositorio y el tipo de aplicación.

Asimismo existirá un módulo de comandos internos, pensados para el uso interactivo de la aplicación por un usuario.

Fin de las funciones API

El fin de las funciones API no será solamente el de proporcionar información "cruda" a las interfaces gráficas, sino también de llevar *el peso de la transformación de datos*, de tal manera que las interfaces se puedan concentrar en la *representación visual* de la información. Se pretende que el módulo de control de la interfaz sea lo más sencillo posible. De esta manera, se evita tener algoritmos y estructuras de datos repetidas en cliente y servidor.

Agrupación y reutilización de funciones API

Al analizar el tipo de funciones solicitadas por las distintas interfaces se puede apreciar un solapamiento de éstas, como ya se ha indicado. De manera general, se puede dar el

caso de la figura 13: Si el dominio limitado por los círculos son las funciones utilizadas por las aplicaciones X, Y, y Z, se aprecia un solapamiento común a todos ($X \cap Y \cap Z$), y solapamientos en pares ($X \cap Y$, $X \cap Z$). Asumiendo que las zonas no solapadas son funciones que exclusivamente tienen sentido en una aplicación específica, la reutilización de las funciones se logra

- metiendo las funciones en el área común en un módulo al cual todas las aplicaciones tienen acceso
- metiendo las funciones en los áreas solapados a pares (o más) en módulos que sean solamente accedidos por las aplicaciones afectadas
- metiendo las funciones en los áreas no solapados en módulos particulares a la aplicación.

Con ello, cada aplicación tendrá asociada una lista de módulos que pueden ser utilizados por su interfaz. De esta manera, se organizan las funciones según las aplicaciones que puedan sacarlas utilidad.

Por ejemplo, se puede hacer accesible un mismo módulo sobre información del estado real o virtual tanto a una interfaz a la aplicación `ASISinfo` como a la interfaz del gestor de transacciones; las funciones de este módulo no podrán diferenciar si el estado al que están comunicadas es el estado real o el virtual - será la aplicación la que los enlace. En cambio, habrá funciones requeridas concretamente al estado virtual que no serán accesibles a la aplicación ASISinfo, y por lo tanto irán a un módulo de uso exclusivo por el gestor de transacciones.

Protocolo

La comunicación estará basada en el intercambio de mensajes entre cliente y servidor. El cliente envía un *mensaje petición* que contiene una función y sus parámetros, y el servidor le devuelve un *mensaje respuesta* que contiene los resultados obtenidos.

Los protocolos de transporte soportados serán sockets bajo TCP/IP que permitirá que interfaz y aplicación residan en máquinas distintas, y también soportará conexiones por

entrada/salida estándar (para su uso con metaservidores como `inetd` o `asisd`[14]).

Uso interactivo

El usuario podrá acceder interactivamente al módulo de comunicaciones de la aplicación para interrogarla directamente, sin utilizar un interfaz gráfico. El interfaz ofrecido al usuario será basado en el uso de una línea de comandos, de manera similar al de un *shell* de usuario UNIX.

Se dispone de un módulo con comandos internos para el usuario:

- ayuda general, ayuda sobre un comando
- lista de comandos disponibles
- lista de módulos disponibles

Uso por herramientas ASIS

Las siguientes herramientas ASIS utilizarán el modelo cliente/servidor para comunicarse con sus interfaces:

- el editor de dependencias `ASISwdep`
- el gestor de transacciones `ASIStm`
- el gestor de estación de trabajo `ASISwsm`
- el generador de aplicaciones `ASISgen`

Los usuarios de la herramienta `ASISinfo` la accederán interactivamente, con lo cual los usuarios podrán formular sucesivas interrogaciones sin tener que relanzar la herramienta, como es actualmente el caso.

[14] Un metaservidor (o superservidor) centraliza la gestión de servidores. `inetd` es el metaservidor estándar de UNIX. `asisd` será el futuro metaservidor del entorno de seguridad ASIS, que dispondrá de mecanismos para enforzar la seguridad, como autentificación y canales seguros [17]

4.4 Diseño de la arquitectura de componentes

4.4.1 Orientación a Objetos

El paradigma o método de diseño utilizado es el de *orientación a objetos*. Este es un enfoque basado en las ideas fundamentales de *objetos* y *clases*.

Las ideas sobre las cuales se asienta este paradigma son:

- Un objeto se define como la abstracción de un elemento en un cierto dominio. Los datos que reflejan el estado interno de un objeto se llaman *atributos*, y las funciones definidas para consultar y manipular los atributos se llaman *servicios* o *métodos*. Estos pueden ser *privados*, de uso interno al objeto, o bien *públicos*, accesibles externamente. Por lo tanto, los datos se *encapsulan* dentro del objeto.
- Una clase describe un conjunto de objetos con atributos y métodos comunes; un objeto se llama también *instancia* de una clase. Una clase se define por los atributos y métodos que engloba.
- Se permite definir una *jerarquía* entre clases mediante el uso de *herencia*. La clase heredera (o clase hijo) podrá usar como propios los métodos de la clase padre.
- La comunicación entre objetos está basada en el *intercambio de mensajes*. Para ello, un objeto solicitará servicios o métodos de otro objeto. Si un objeto forma semánticamente parte de otro están *agregados*. Dos objetos independientes que intercambian mensajes están *asociados*.
- Se pueden definir clases que marcan un comportamiento general, declarando métodos *virtuales* pero sin implementarlos, tarea que se lleva a cabo en las clases hijo. Estas clases se llaman *abstractas* y no pueden ser instanciadas directamente.
- El *polimorfismo* es la capacidad de enlazar dinámicamente con instancias de distintas clases sin necesidad de conocer el tipo de éstas.

Ventajas

Las ventajas del uso la orientación a objetos para el diseño de los sistemas propuestos se fundamentan sobre todo en la facilidad inherente al diseño orientado a objetos de la *reutilización* de componentes software, al encapsular éstos en clases y permitir mediante la herencia su especialización para fines concretos. Adicionalmente, éstos componentes son fácilmente *extensibles* para su uso posterior.

Cuantos más elementos software sean reutilizados por los sistemas, menor será el esfuerzo de desarrollo. Además, la reutilización deriva en una fiabilidad mayor, ya que un componente probado y usado es menos propenso a fallos que uno diseñado específicamente para una tarea.

Si bien los sistemas a desarrollar cumplen funcionalidades distintas, comparten una base conceptual común que ha de ser identificada y modelizada para permitir su uso compartido.

Un ejemplo evidente para la reutilización es el modelo cliente/servidor, que invita a ser diseñado con especial enfoque a ella.

Pero considerando también el entorno global del proyecto ASIS, en el cual se toma la decisión de rediseñar la totalidad de las herramientas, a la vez de iniciar el desarrollo de otras (entre ellas los sistemas propuestos), se plantea la necesidad de enfocar el diseño de tal manera que se favorezca la reutilización de componentes (y su capacidad de ampliación) entre todas las aplicaciones, y la existencia de un comportamiento y una apariencia común a todas las aplicaciones.

Al ser conceptualmente similar al modelo Entidad/Relación de las bases de datos, la Orientación a Objetos permite directamente modelizar objetos y clases basándose en las entidades, y establecer las relaciones entre éstos en función de las relaciones existentes entre entidades. Esto es un punto de especial relevancia para las aplicaciones ASIS, que acceden a una base de datos común.

Desventajas

El uso de técnicas orientadas a objeto implica una pérdida en eficiencia, al aumentarse habitualmente el consumo de recursos, como la memoria requerida y el tiempo de computación. Además, su uso está condicionado a una formación adecuada del personal de desarrollo, ya que requiere un cambio de mentalidad respecto a técnicas clásicas, como la programación estructurada, a la hora de enfocar un diseño.

top-down y *bottom-up*

Mediante técnicas como los diagramas de flujo de datos se diseña un sistema partiendo de lo general hasta lo detallado (*top-down*). El problema que éste tipo de diseño no se presta a la reutilización dado que su granularidad se establece a nivel de funciones concretas.

En cambio, la unidad mínima de diseño en orientación a objetos, incluye datos, métodos y abstracción al generalizar el comportamiento de objetos de un cierto dominio en clases. Las clases son autónomas y relacionándolas mediante jerarquías y asociaciones se puede construir un sistema partiendo de la unidad mínima (*bottom-up*).

Las técnicas top-down y bottom-up, sin embargo, pueden ser complementarias. En el caso de sistemas como los gestores de transacción y de copia local, se observa que su funcionamiento se describe adecuadamente mediante diagramas de flujo de datos; pero la similitud entre ambos es difícil de formalizar por completo. Sin embargo, su descripción exclusivamente con técnicas bottom-up sería engorrosa.

Haciendo coincidir las técnicas top-down y bottom-up en un nivel intermedio del desarrollo se puede equilibrar la claridad del diseño global con la necesidad de reutilización de componentes.

4.4.2 Diseño de las clases

En el apéndice D (página 134) se presentan las clases diseñadas y las relaciones y jerarquías que existen entre ellas. La estructura de las clases se mostrará según la metodología OMT (*Object Modelling Technique*) descrita por J. Rumbaugh en [30].

La descripción las clases se organiza según el grado de cohesión y su relación lógica. Las partes en las que se divide son las siguientes:

1. clases básicas para el acceso a las entidades principales
2. gestión de mensajes y trazas
3. gestión y ejecución de transacciones
4. estado virtual del repositorio
5. gestión de dependencias
6. esquema cliente/servidor
7. programa principal y control de ejecución de la aplicación.

Las clases básicas, la gestión de mensajes y trazas, el modelo cliente/servidor y el programa principal han sido diseñados con vista a su reutilización y/o ampliación por todas las aplicaciones ASIS.

Las clases diseñadas para gestión y ejecución de transacciones son usadas indistintamente por el gestor de transacciones y el gestor de copia local.

4.5 Diseño detallado e implementación

La implementación de los sistemas diseñados se lleva a cabo en el lenguaje de programación PERL [37]. Una característica de este lenguaje es su alta portabilidad y su fiabilidad, lo cual lo ha convertido en un estándar *de facto* para su uso en tareas de administración de sistemas. Con la versión 5, este lenguaje es capaz de acogerse al paradigma de la orientación de objetos. Otra razón para elegir Perl es la existencia de personal en el entorno de trabajo experimentado con este lenguaje y sus herramientas de desarrollo.

La sintaxis del lenguaje es similar a C. La gestión de datos dinámicos es interna al lenguaje, como en Java, por lo que el programador no ha de preocuparse de reservar o liberar explícitamente memoria.

Una desventaja del lenguaje es su baja velocidad de ejecución. Esto es debido a que para aumentar su portabilidad el lenguaje es compilado en tiempo de ejecución a un código intermedio, que a su vez es ejecutado por una máquina virtual, específica a cada plataforma.

La clase Client (perteneciente al subsistema cliente/servidor), ha sido implantada en TCL [38] , el lenguaje utilizado para la construcción de las interfaces de usuario.

Se utilizaron herramientas estándares de soporte en el desarrollo, como por ejemplo *depuradores* y *perfiladores de ejecución*. Estos últimos sirven para determinar características de la ejecución, como por ejemplo calcular el número de llamadas y tiempo de ejecución por cada método, secuencia de llamadas a métodos, etc. El uso de los perfiladores fue especialmente útil para optimizar la estructura y los métodos de la clase Vfiles, que implanta el sistema de ficheros virtual. Asimismo el editor utilizado (EMACS) dispone un modo de edición específico a Perl, que incluye entre otras funcionalidades un *navegador de clases*, útil para el acceso rápido a métodos y clases.

4.5.1 Diseño detallado

Según el estándar de la Agencia Espacial Europea (ESA) [26], en el diseño detallado "los componentes del menor nivel del diseño arquitectural se descomponen hasta que pueden ser expresados en el lenguaje de programación seleccionado".

En este caso, los componentes de menor nivel son las clases, que son implementadas como clases Perl; cada clase se emplaza en un *módulo* Perl.

La documentación del diseño detallado se integra directamente en el fichero con el código, a través de un mecanismo de documentación integrado llamado POD. Esta documentación se puede extraer del código y convertirse en un documento separado. La ventaja de utilizar este sistema es la facilidad dada para crear y actualizar simultáneamente código y diseño detallado debido a su emplazamiento común.

La integración de código y diseño detallado es recomendada por los estándares de la ESA, en su versión para proyectos software pequeños [25]. Según este estándar, se considera un proyecto pequeño si se requiere menos de dos años/hombre en su desarrollo, o si el equipo de desarrollo es de menos de seis personas.

La documentación de diseño detallado depende del lenguaje de programación, reflejando la sintaxis Perl en los métodos descritos. La ventaja de este enfoque es su rapidez y facilidad, sin embargo *restringe su portabilidad* a otros lenguajes de programación. Junto al hecho de que el diseño detallado y el código son generados simultáneamente (lo cual es recomendado por el estándar de la ESA), se puede considerar la documentación resultante como una *documentación de código*.

En el apéndice E, a partir de la página 172, se muestra, a nivel de ejemplo, la documentación detallada (o de código) de una de las clases desarrolladas.

Asimismo, en el apéndice F, a partir de la página 178, se incluyen algunos sencillos ejemplos de ejecución de las herramientas.

4.5.2 Control de versiones

Se ha utilizado el sistema de control de versiones CVS (*Concurrent Versions System*, sistema concurrente de versiones) [5] para la gestión de los módulos y la documentación. Este sistema de control de versiones permite el acceso simultáneo de varios usuarios a un repositorio de desarrollo común, en el cual se pueden acomodar varios proyectos. En este repositorio se van almacenando e identificando las sucesivas revisiones de los objetos que pertenecen al proyecto desarrollado. El sistema permite gestionar ramas de desarrollo

independientes, además de identificar mediante nombres simbólicos las revisiones de los objetos que conforman una determinada versión de una aplicación.

Los usuarios desarrollarán los objetos sobre un área de trabajo privado, y una vez que un objeto adquiera suficiente madurez, lo enviarán al repositorio, una junto a una descripción de la naturaleza del objeto o en su caso del cambio sufrido.

Sin embargo, CVS no ofrece facilidades de gestión de configuración, como puede ser por ejemplo un control de cambios formal, relaciones de dependencia entre documentos y el código resultante, etc.

El autor del presente trabajo se encargó de iniciar y mantener el sistema de control de versiones para todo el equipo de desarrollo. En este contexto desarrolló una herramienta que permite consultar el diseño detallado correspondiente a los objetos del repositorio, junto a toda la información sobre las distintas revisiones (autor, fecha y hora, líneas afectadas, descripción del cambio, versiones de la(s) aplicacion(es) asociadas).

Esta herramienta es lanzada cada vez que se efectúa una modificación sobre el repositorio de desarrollo. Se llama a un conversor POD a HTML, se obtiene de CVS toda la información sobre el objeto y sus revisiones y se integra el resultado en una página HTML. Esta página se incluye en el servidor "Web" del equipo de desarrollo y se actualiza un directorio con enlaces a todas las páginas correspondientes a los objetos contenidos en el repositorio. Así, mediante un cliente "Web" se puede consultar el estado actual de cada objeto, junto a su diseño detallado y su historial.

Apéndices

Apéndice A

Diseño funcional de los sistemas

A continuación se pasará a realizar una descomposición funcional estructurada (*top-down*) de los sistemas propuestos; para ello se utilizará como técnica de representación los diagramas de flujos de datos (*Data Flow Diagrams*, DFD's).

No se pretende dar una visión completa sobre todos los procesos y funciones implantadas, sino ofrecer una visión general de la estructura del sistema, poniendo el peso en *qué* hace el sistema, no *cómo*. Se ofrecerá un mayor nivel de detalle al describir la arquitectura y el diseño detallado del sistema.

A.1 Procesos del gestor de transacciones

En la figura A.1 se muestra el diagrama de contexto del gestor de transacciones y del gestor de copia local.

En la figura A.2 se detallan los procesos principales del gestor de transacciones.

Comprobación de la transacción (1.1)

Este proceso recibe la transacción enviada por el usuario y comprueba la corrección sintáctica de las operaciones básicas de la transacción, comprobando asímismo la existencia de los productos, versiones y plataformas afectadas por las operaciones que conforman la

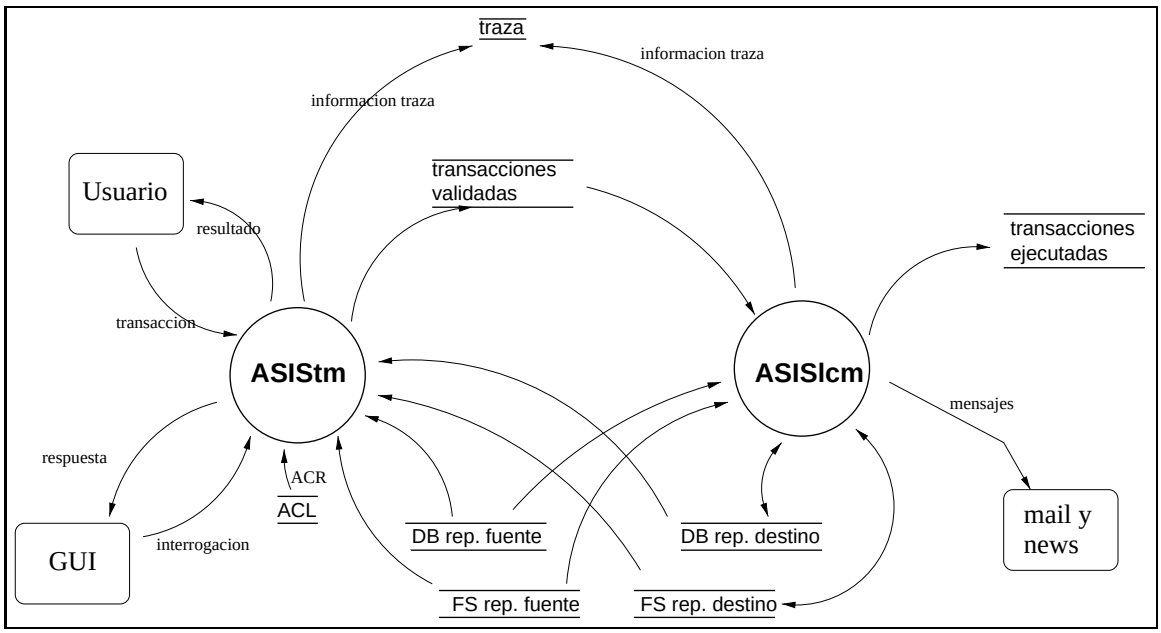

Figura A.1: diagrama de contexto

transacción. Las entradas serán las bases de datos de maestro y esclavo y la transacción. La salida será la transacción validada, añadiéndole la fecha y hora y el nombre de usuario.

Confirmación de autorización (1.2)

Por cada operación básica de la transacción se comprobará la autorización del usuario para llevarla a cabo. Para ello se consultará la ACL (*access control list*, lista de control de acceso) de los usuarios. Esta lista de estará basada en los derechos de acceso por usuario y por producto. Las entradas serán la ACL y el identificador del usuario, además de la transacción correcta, y la salida un mensaje de error caso de producirse.

Ejecutor de transacciones (1.3)

Desde el punto de vista funcional, se puede considerar el motor del sistema. En él se ejecutará la transacción. *Este proceso será compartido por el gestor de transacciones y el gestor de copia local.* La diferencia será que en el gestor de transacciones la información sobre la base de datos y el sistema de ficheros la toma a través del estado virtual, mientras que en el gestor de copia local será conectado al estado real.

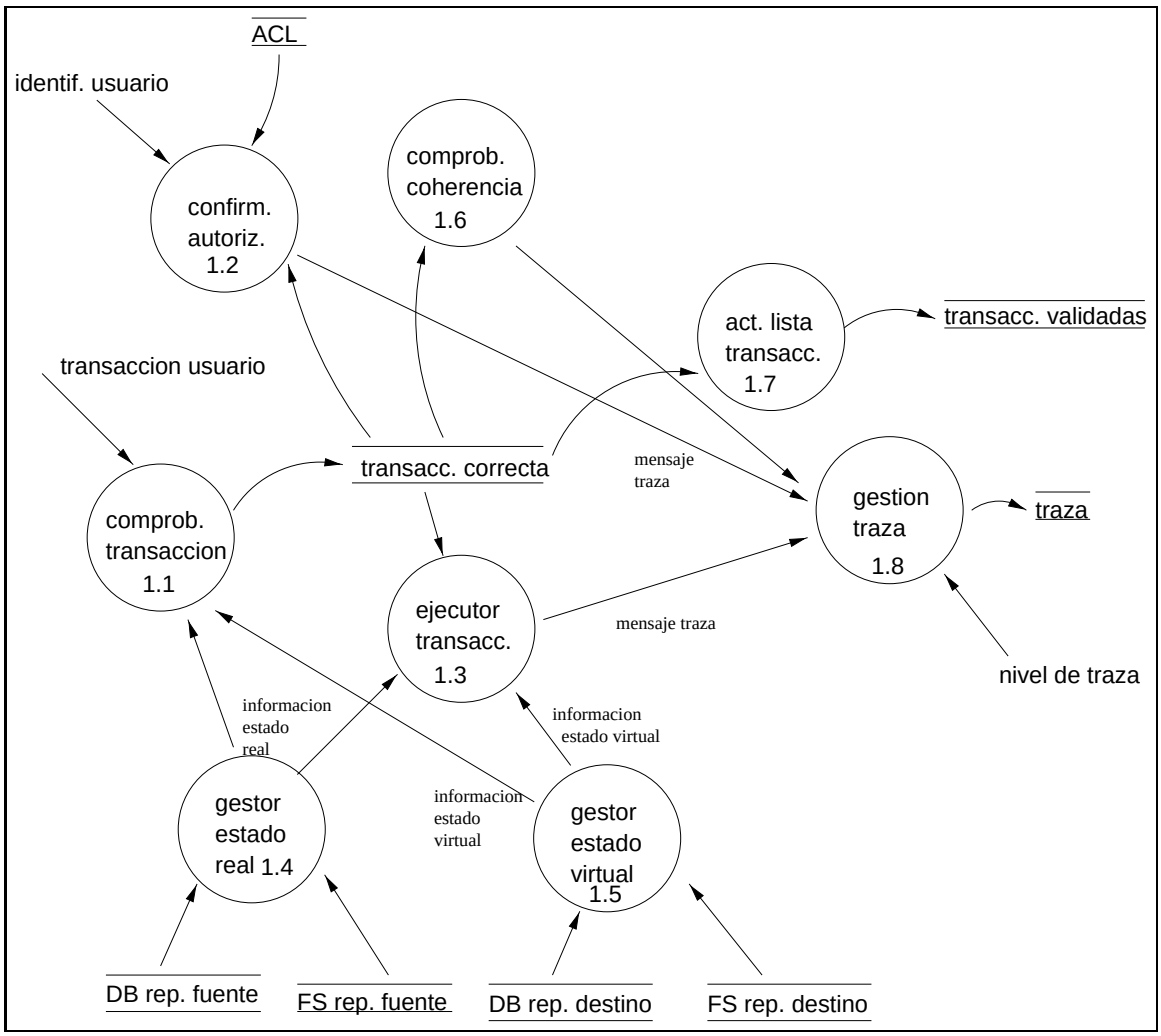

Figura A.2: procesos del gestor de transacciones

El proceso seguido es el mismo en ambos casos, los puntos importantes son:

- verificación de la existencia del paquete o fichero a transferir al repositorio maestro, si la operación lo requiere
- en el caso de operaciones tipo transición, verificar la posibilidad de efectuar la operación dentro del diagrama de estados del modelo de procesado de "software"
- analizar si se generan conflictos de nombres de ficheros en el repositorio
- analizar si habrá problemas de espacio y cuotas en el sistema de ficheros [1]

[1] implica comprobar la cuota por cada partición o volumen afectado del sistema de ficheros

- verificar si existe suficiente espacio temporal para mover/copiar los ficheros
- copia de ficheros, en las operaciones tipo *Introducir En Repositorio*, *Añadir Fichero a Paquete* y *Reemplazar Fichero en Paquete.*
- generar traza de recuperación, basada en registrar las operaciones efectuadas. En el caso de las operaciones *Borrar* y *Retirar de Repositorio*, copiar los ficheros a un directorio temporal.
- actualización de la base de datos destino.

Caso de fallar una operación, se lanzará un subproceso que recorrerá la traza de recuperación, deshaciendo las operaciones hechas hasta el momento, y dejando el estado (real o virtual) como estaba antes de iniciar la transacción. De esta manera, se asegura la ejecución atómica de la transacción.

Gestor estado virtual (1.4)

Aquí se concentra la inteligencia del gestor de transacciones. El estado virtual proveerá al ejecutor de transacciones de la información que necesita sobre la base de datos ASIS, y sobre la información en el sistema de ficheros, pero lo hará teniendo en cuenta las transacciones que han sido validadas y aceptadas desde la última vez que se actualizó físicamente el repositorio. Asímismo, irá almacenando los cambios que se producen sobre la base de datos y el sistema de ficheros, pero sin modificar éstos últimos, a los que accederá sólo para lectura. Las entradas serán el sistema de ficheros y la base de datos del repositorio maestro y las interrogaciones del ejecutor de transacciones, y las salidas las respuestas a éstas.

Gestor estado real (1.5)

Este proceso gestiona el acceso al sistema de ficheros y la base de datos, para su consulta y actualización por el ejecutor de transacciones. Dispondrá del mismo interfaz que el gestor de estado virtual. A través de este proceso se consulta y modifica contenido y estado de paquetes en los repositorios; también será utilizado por el gestor de copia local. En el

caso del gestor de transacciones, es consultado por los procesos 1.1 y 1.3 para acceder al contenido del repositorio auxiliar y del repositorio principal.

Comprobación de la coherencia (1.6)

En este proceso se efectuan las comprobaciones de coherencia. En caso de detectarse alguna de las posibles anomalías descritas, se generará un mensaje de advertencia al usuario. Las entradas serán la transacción y la base de datos del repositorio maestro, mientras que la salida será un mensaje de aviso caso de detectarse incoherencias.

Actualizar lista de transacciones (1.7)

Una vez aceptada la transacción será escrita en la lista de transacciones pendientes del repositorio auxiliar, de lo cual se encarga este proceso. La entrada es la transacción y la salida la lista actualizada de transacciones del repositorio auxiliar.

Generar traza (1.8)

Todos los procesos mandarán información de traza a este proceso, que tomará esta información y el nivel de traza deseado por el usuario como entrada. Escribirá todos los mensajes del nivel de traza solicitado por el usuario en el fichero de traza (*logfile*).

A.2 Procesos del gestor de copia local

En la figura A.3 se muestran los procesos principales del gestor de copia local.

Reducción y simplificación de transacciones (2.1)

Este proceso leerá la lista de transacciones ejecutadas por el repositorio maestro, extrayendo las aún no ejecutadas. Estas serán reducidas aplicando las reglas de reducción descritas

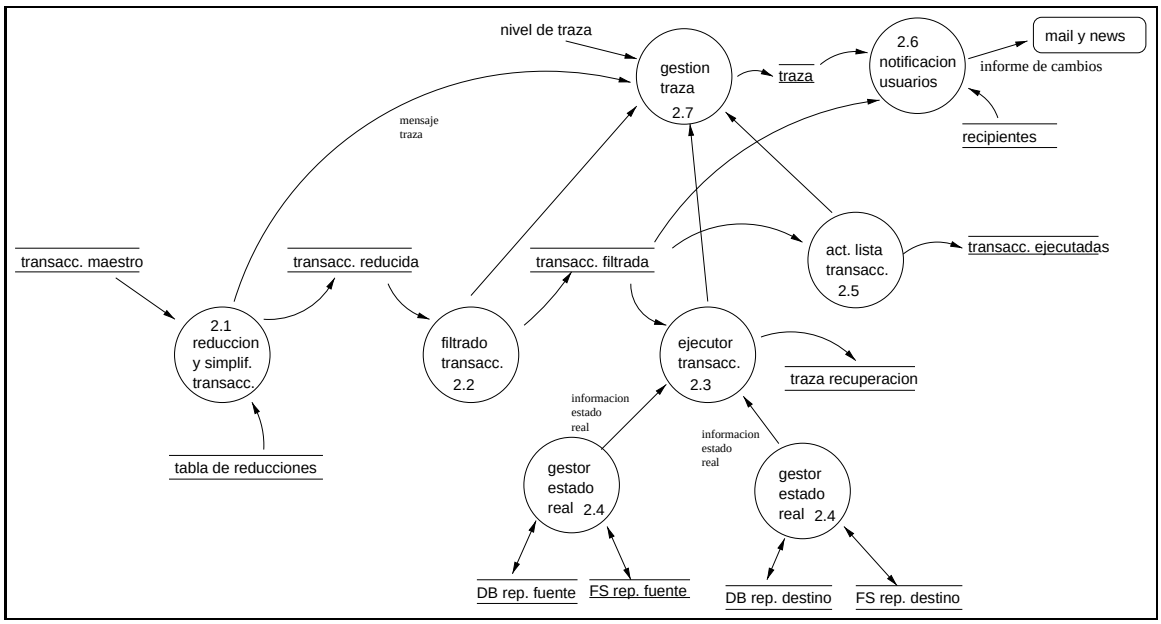

Figura A.3: procesos del gestor de copia local

con anterioridad. La entrada del proceso será la lista de transacciones del maestro y la tabla de reducciones, y la salida del proceso serán las transacciones reducidas.

Filtro de transacciones (2.2)

Este proceso toma como entrada las transacciones simplificadas y las filtra según plataformas y/o familias. La salida del proceso serán las transacciones filtradas.

Ejecutor de transacciones (2.3)

Ejecuta las operaciones de una transacción. Es análogo al proceso (1.3) del gestor de transacciones.

Gestor estado real (2.4)

Este proceso gestiona el acceso al sistema de ficheros y la base de datos; es idéntico al proceso 1.4 del gestor de transacciones.

Actualizar lista de transacciones (2.5)

Este proceso es idéntico al proceso 1.6 del gestor de transacciones; la única excepción es que aquí el proceso actualiza la lista de transacciones del repositorio destino.

Notificación de usuarios (2.6)

Este proceso enviará mensajes, de tipo news y mail, a los recipientes interesados en los cambios generados en el repositorio. Se construirá un mensaje, indicando los cambios, que enviará a(l/los) sistemas de mensajería electrónica. Además, este proceso mandará un mensaje al gestor del repositorio con la traza de ejecución del sistema.

Generar traza (2.7)

este proceso es idéntico al proceso 1.8 del gestor de transacciones.

A.3 Procesos del sistema de dependencias

En la figura A.4 se muestra el diagrama de contexto del sistema de dependencias. El sistema de dependencias dispondrá, al igual que el gestor de transacciones, un modelo cliente-servidor para servir datos a la interfaz gráfica. Este modelo es explicado en la sección 4.3.5.

A.3.1 Procesos para verificación de dependencias

En la figura A.5 se muestran los procesos para verificar las dependencias de las aplicaciones tanto de la configuración de `ASISwsm`, como de los áreas del repositorio.

Se reutilizan procesos definidos en el sistema de transacciones; en concreto el proceso (1.8), que se utilizará para la generación de la traza, y el proceso (1.4), a través del cual se accede a la base de datos ASIS.

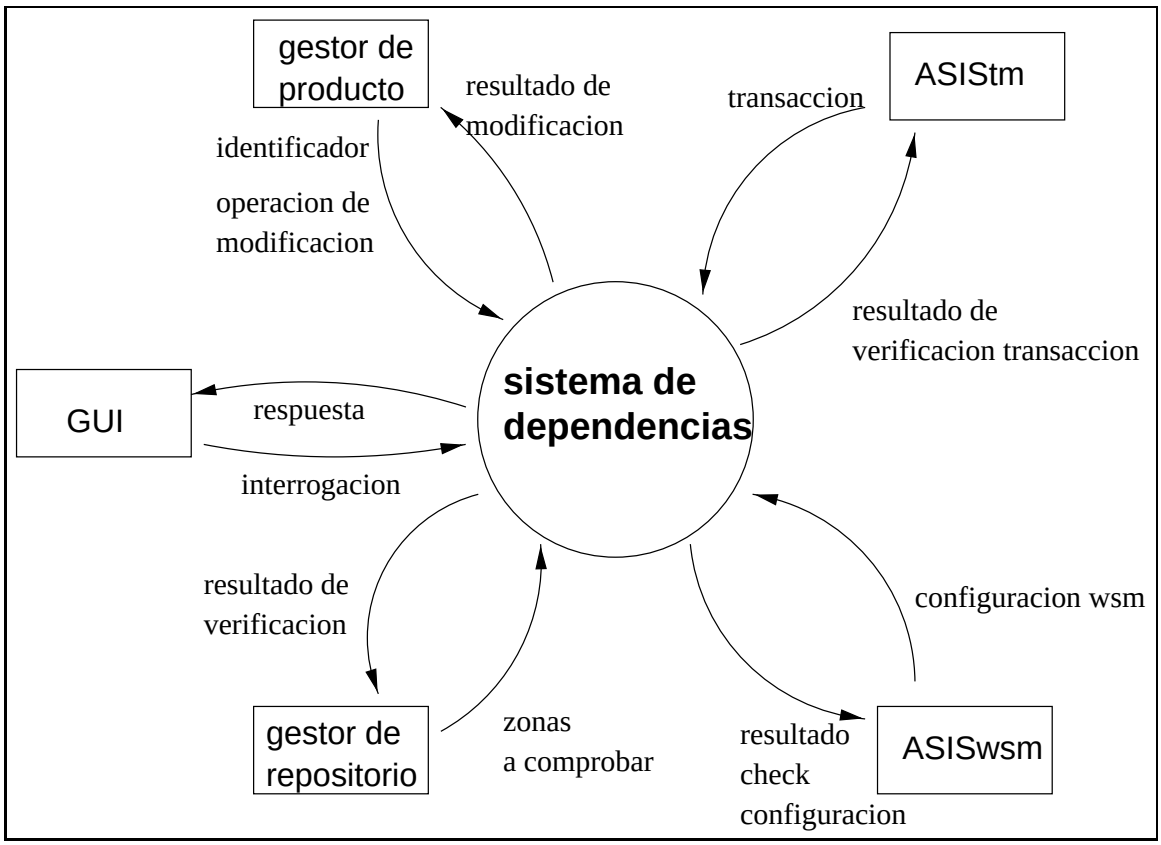

Figura A.4: diagrama de contexto del sistema de dependencias

Los demás procesos se describen a continuación.

Comprobar configuración del gestor de estación de trabajo (3.1)

Este proceso lee la configuración de `ASISwsm` y extrae la información relevante. La parte de la configuración relevante para el sistema de dependencias consiste en el identificador de plataforma y la lista de aplicaciones seleccionadas por el administrador. Para la comprobación de esta lista llamará por cada aplicación al verificador de dependencias (3.6), el cual comprobará tanto las dependencias entre aplicaciones ASIS como las de entorno; la salida del proceso será el resultado de la comprobación por cada aplicación.

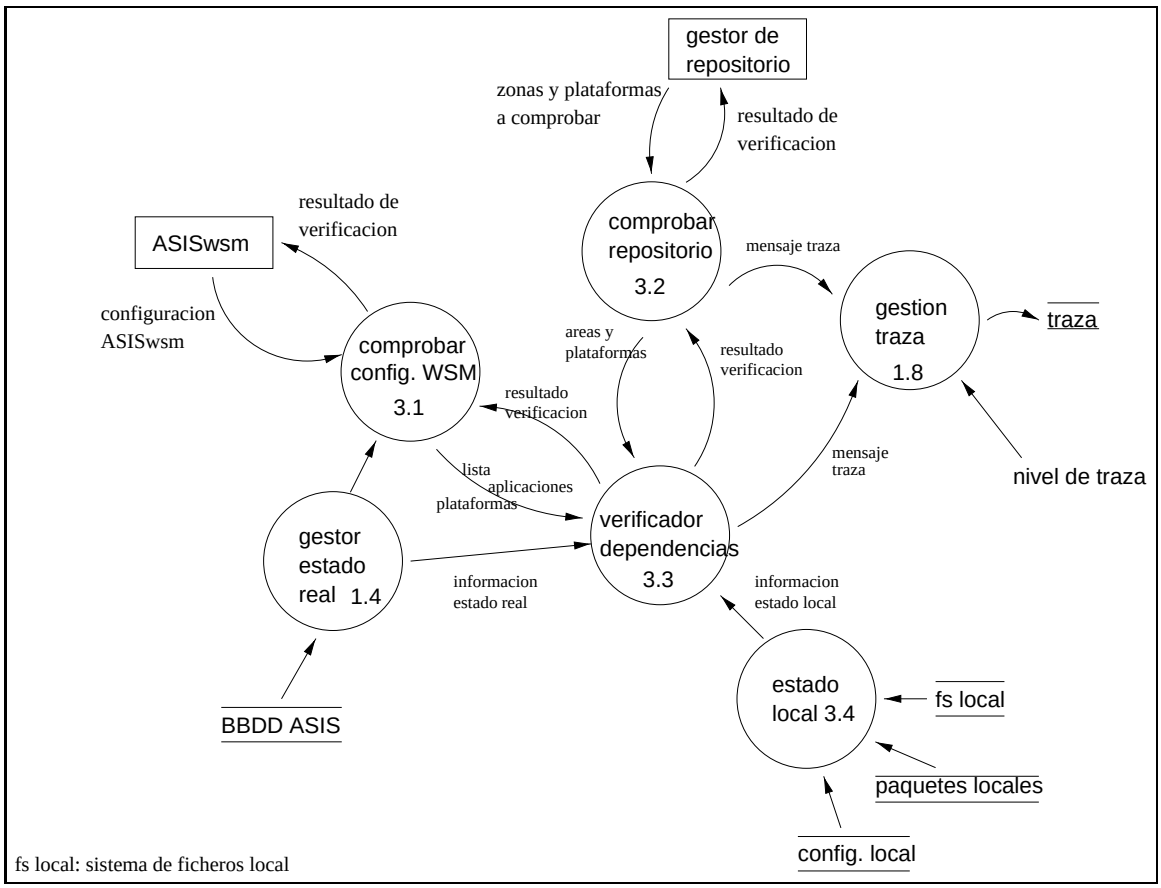

Figura A.5: procesos de verifición de dependencias

Comprobar repositorio (3.2)

Este proceso tomará como entrada el área del repositorio y las plataformas a comprobar. El área podrá ser el compuesto por los estados "Certificado" y "En Producción", o solamente éste último. Para la comprobación se llamará al verificador de dependencias (3.6). Se comprobarán las dependencias entre todas las aplicaciones contenidas en el área, para las plataformas indicadas. La salida del proceso será el resultado de la comprobación por cada aplicación y plataforma.

Verificador de dependencias (3.3)

Es el proceso principal, o el más usado, del sistema de dependencias. Tomará como entrada el nombre de una aplicación, una plataforma, el dominio del repositorio a verificar, y los

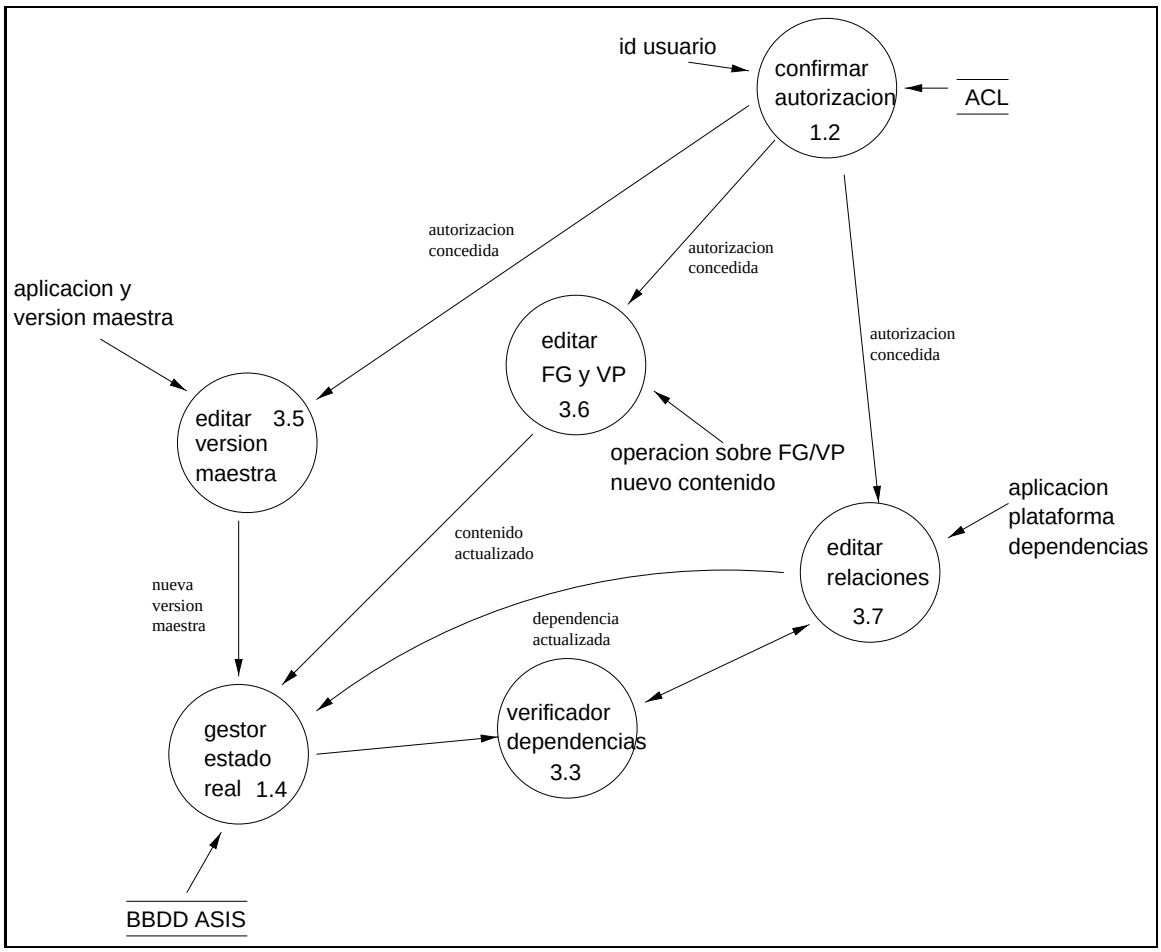

Figura A.6: procesos de edición de dependencias

tipos de dependencias a comprobar; y computará qué dependencias se cumplen y cuales no de la aplicación sobre el dominio dado. El dominio puede ser un área del repositorio o una lista de aplicaciones. La salida será el resultado de la verificación.

Gestor de estado local (3.4)

El verificador de dependencias solicitará a este proceso la información que requiera sobre el entorno de ejecución; es decir la configuración local y el sistema de ficheros local de la estación de trabajo y los paquetes locales instalados en la estación de trabajo.

A.3.2 Procesos para la creación y edición de dependencias

En la figura A.6 se muestran los procesos definidos para la *modificación* de la información del sistema de dependencias.

El acceso de escritura a la base de datos ASIS estará sujeto a la autorización que disponga el usuario. Para ello se consultará la misma lista de control de acceso que en el gestor de transacciones, a través del proceso (1.2).

Editar versión maestra (3.5)

Este proceso permitirá fijar la versión que se quiere utilizar como plantilla para modelizar las dependencias de una nueva versión del mismo producto. La entrada será la aplicación nueva junto a la versión a utilizar como plantilla. Se verificarán los datos de entrada confrontándolos con la base de datos. La salida será la base de datos ASIS actualizada o un mensaje de error caso de producirse.

Editar grupos funcionales y paquetes virtuales (3.6)

Mediante este proceso se podrán crear grupos funcionales y paquetes virtuales, y modificar su contenido. La entrada será el grupo funcional o paquete virtual a crear o editar, y eventualmente el contenido nuevo. Se verificará la entrada confrontándola con la base de datos; la salida será la base de datos ASIS actualizada o un mensaje de error caso de producirse.

Edición de relaciones (3.7)

La entrada a este proceso será el conjunto de dependencias modelizadas para una aplicación, junto al nivel (y plataforma si necesario). Se comprueba la sintaxis y la existencia de los objetos referidos en la base de datos ASIS. La salida será la base de datos ASIS actualizada o un mensaje de error.

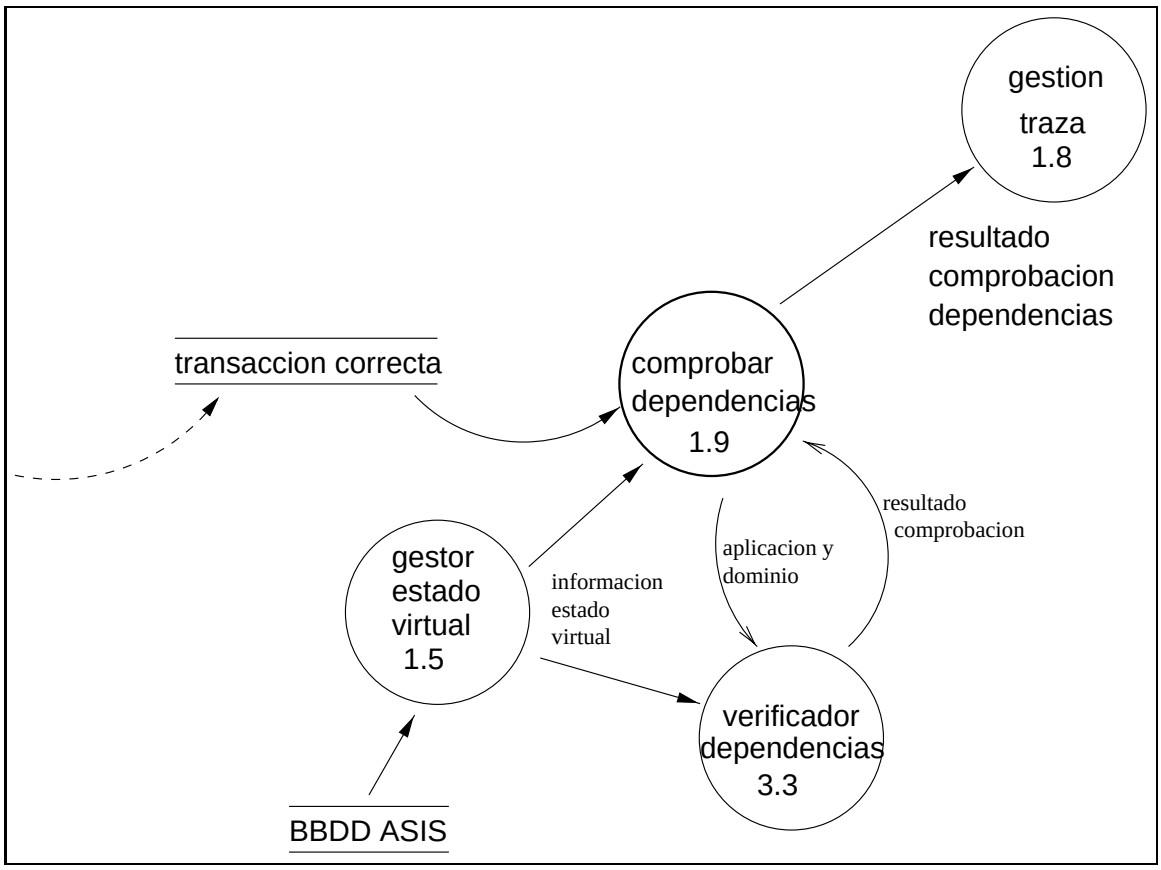

Figura A.7: comprobación de dependencias en el gestor de transacciones

A.3.3 Verificación de dependencias por el gestor de transacciones

El gestor de transacciones deberá verificar, por cada uno de las aplicaciones manipuladas en la transacción, las reglas de la página 95.

Se definirá un nuevo proceso en el gestor de transacciones, que accederá al verificador de dependencias (3.6) y tomará la información del estado virtual, en vez del estado real.

El nuevo proceso se llamará **comprobar dependencias (1.9)**. En él se analizan las reglas de la página 4.3.3. La entrada será la transacción, mientras que la salida será un mensaje de aviso caso de detectarse dependencias incumplidas. Su relación con los demás procesos se muestra en la figura A.7.

A.3.4 Emplazamiento de procesos en aplicaciones

Se aprecia que los procesos definidos para la creación y edición de dependencias han de ser ejecutados en una máquina servidor, ya que el cliente no tiene normalmente derechos de modificar el repositorio desde su estación de trabajo.

Sin embargo, la comprobación de la configuración de la estación de trabajo ha de ser lanzada desde la máquina del cliente, porque requiere acceso a elementos locales (paquetes locales, configuración local, y el fichero de configuración de `ASISwsm`).

La comprobación del repositorio puede ser efectuada desde una máquina cliente o servidor, ya que el acceso en modo lectura no es limitado.[2]

Por lo tanto, se definen dos herramientas:

- `ASISwdep` es la aplicación en la que se emplazan los procesos para la creación y edición de dependencias. Esta aplicación será ejecutada sobre un servidor y con autorización para modificar el repositorio ASIS.
- en `ASISdep` se emplazan los procesos de comprobación, tanto del repositorio como de la configuración de `ASISwsm`. Esta aplicación funciona indistintamente sobre cliente y servidor (los servidores son a su vez clientes del repositorio). No modifica el repositorio.

Resumiendo, el sistema de dependencias se distribuye en tres herramientas: `ASIStm`, `ASISwdep` y `ASISdep`.

[2] no lo es al nivel del repositorio. Pero por ejemplo, las máquinas que sirven y gestionan el repositorio en el CERN no son accesibles desde el exterior.

Apéndice B

Reducción de transacciones

A continuación se describe el método seguido para la reducción de transacciones por el gestor de copia local. No se aplica la reducción de transacciones al gestor de transacciones al aceptar éste las transacciones de una en una; si bien se podrían reducir las operaciones que conforman la transacción enviada por el usuario no se considera ésto necesario.

Para la reducción de transacciones se utiliza una tabla (véase B.1). Esta tabla recoge las reducciones posibles, comparando todas contra todas las operaciones. La reducción se lleva a cabo si dos operaciones afectan *la misma versión de producto para la misma plataforma.*

Así, por ejemplo, al tener las operaciones seguidas

```
IntroduceInProduction GNU.EDIT/emacs-20.2 (share)
RemoveFromProduction GNU.EDIT/emacs-20.2 (share)
```

se aprecia en la tabla que ambas se anulan (''NOP''), ya que el efecto final es que no se ha alterado el estado del paquete. En cambio, con

```
IntroduceInRepository GNU.EDIT/emacs-20.2 (share)
Certify GNU.EDIT/emacs-20.2 (share)
```

no hay nada que simplificar (''-''), ya que forman una secuencia de transiciones válida dentro del diagrama de estados del modelo de procesado de software.

Las operaciones tipo "patch", cuya función no es la de alterar el estado pero sí el contenido (es decir, los ficheros componentes) de un paquete, han de cumplir una condición adicional cuando se comparan entre ellas - la de afectar al mismo fichero; si no afectan al mismo fichero se considera que no son simplificables entre sí (**''-''**).

Las celdas marcadas con **''X''** indican que esa secuencia de operaciones no tiene sentido y que por lo tanto no puede aparecer.[1]

En las celdas cuyo contenido es `'D,REM_FP,REP_FP,ADD_FP'` se reducen las operaciones por la indicada en la celda.

Para la reducción de una lista de transacciones, se aplicarán las siguientes reglas:

1. **si** dos operaciones afectan a una misma versión de producto **y** sobre la misma arquitectura [2] **entonces** serán *candidatas* a ser simplificadas.
2. **si** existen dos *candidatas* en una misma transacción **entonces** se consultará la tabla de reducción y se simplificará si procede.
3. **si** existen dos candidatas en transacciones inmediatamente sucesivas **entonces** se procederá a fusionar las dos transacciones en una sola.

El algoritmo básico utilizado es mostrado a continuación:

Se recorre la lista de transacciones $t_1, t_2, ..., t_n$, transacción a transacción, empezando por la última transacción t_n de la lista como transacción pivote t_i.

1. Si la transacción t_i es fusionable según la regla 3 con la transacción t_{i-1} entonces se fusionan temporalmente en una sola, y se recorren las operaciones $o_1, o_2, ..., o_m$ de la transacción resultante, empezando por la última como operación pivote o_j:
   (a) se compara o_j con o_{j-1}. Caso de que sean *candidatas* según la regla 1, se consulta la tabla de reducciones.
   (b) Si son *candidatas* y existe reducción desaparece la operación o_j y la operación o_{j-1} se sustituye por la reducción indicada en la celda (desaparecerá si la reducción es `NOP`).

[1] el gestor de transacciones no permitiría que se generase una secuencia así, al comprobar la validez de las operaciones dentro del diagrama de estados ASIS.

[2] **y** sobre el mismo fichero, en caso de que ambas operaciones sean de tipo 'patch'

   (c) Si no son candidatas o no se puede reducir, se comparará o_j con o_{j-2}, y así sucesivamente.
   (d) se repite hasta llegar a o_1, momento en el que se toma o_{j-1} de operación pivote y se itera de nuevo.

2. si se produce al menos una reducción durante el bucle anterior se fusionan definitivamente las dos transacciones, en caso contrario se comparará t_j con t_{j-2}, y así sucesivamente.
3. se repite hasta llegar a t_1, momento en el que se toma t_{i-1} como nueva transacción pivote, y se itera de nuevo.

La lista de transacciones resultante es mínima.

Por razones de claridad, este algoritmo básico no tiene en cuenta la posibilidad de la existencia de múltiples plataformas por operación, pero esencialmente significa la existencia de un tercer bucle dentro de él.

Una limitación del algoritmo y de la tabla de reducción es que no se tiene en cuenta que una operación o_1 sobre una aplicación X seguida de una otra operación o_2 sobre otra aplicación Y pero sobre la misma plataforma *sí* pueden tener relación: cuando alguna de las operaciones afecta el área común a todos los paquetes, el área de producción.

El algoritmo genera "saltos" que pueden llevar a que la operación o_2 se ejecute antes que o_1 en la lista de transacciones resultante; el problema en concreto se plantea cuando o_1 es de "Retirar de Producción" y o_2 es de tipo "Introducir En Produción": si comparten algún fichero con idéntico nombre en el mismo directorio, la ejecución de las operaciones no será posible en ese orden. Al no ser eficiente comprobar esta situación se decide impedir la fusión de operaciones dentro de una transacción cuando entre ellas se sitúa una operación tipo "Retirar de Producción"; sin embargo, sí será motivo de fusión de transacción.

op	IIR	C	IIP	RFP	CC	RFR	D	ADD-FP	REM-FP	REP-FP
IIR	X	-	X	X	X	NOP	-	-	-	-
C	X	X	-	X	NOP	X	D	-	-	-
IIP	X	X	X	NOP	X	X	D	-	-	-
RFP	X	X	NOP	X	-	X	D	-	-	-
CC	X	NOP	X	X	X	-	D	-	-	-
RFR	-	X	X	X	X	X	X	X	X	X
D	X	X	X	X	X	X	X	X	X	X
ADD-FP	X	-	-	-	-	-	-	X	NOP	ADD-FP
REM-FP	X	-	-	-	-	-	-	-	X	X
REP-FP	X	-	-	-	-	-	D	X	REM-FP	REP-FP

```
X  : operación imposible
-  : no hay simplificación
NOP: se anulan ambas operaciones
D, REM-FP, ADD-FP, REP-FP: se ejecuta esa operación

IIR:    Introducir en Repositorio (IntroduceInRepository)
C:      Certificar (Certify)
IIP:    Introducir en Producción (IntroduceInProduction)
RFP:    Retirar de Producción (RemoveFromProduction)
CC:     Cancelar Certificación (CancelCertify)
RFR:    Retirar del Repositorio (RemoveFromRepository)
D:      Borrar (Delete)
ADD-FP: Añadir fichero a paquete (AddFileToPackage)
REM-FP: Borrar fichero de paquete (RemoveFileFromPackage)
REP-FP: Reemplazar fichero en paquete (ReplaceFileInPackage)
```

La tabla se consulta leyéndo la fila y luego la columna; así el resultado de RFP seguido de D es "D".

Tabla B.1: tabla de reducción de operaciones

Apéndice C

Requisitos Específicos

Los Requisitos específicos establecen *qué* se va a hacer, sin entrar en los detalles específicos de *cómo* se harán, actividad que se resuelve durante la fase de diseño.

Basándose en el estándar PSS-05 de la Agencia Espacial Europea [26], se identifican dos niveles de prioridad de requisitos, siendo éstos: *Necesario* y *Deseable*. Los requisitos necesarios son de obligado cumplimiento para el sistema mientras que los requisitos de tipo deseable se procederá a su implantación siempre que hayan satisfecho los necesarios y no se incumplan los plazos. Todos los requisitos se considerarán mandatorios a no ser que se indique lo contrario. Los requisitos se estructuran en las siguientes subclases:

Requisitos funcionales (RF)

RF1: El gestor de transacciones garantizará el éxito de la ejecución de las transacciones validadas por él sobre el repositorio principal y encomendadas al gestor de copia local para su ejecución física. En cambio, no comprobará la viabilidad de las transacciones sobre repositorios replicados del principal.

RF2: El gestor de transacciones será utilizado en horas de trabajo, sin afectar a los usuarios.

RF3: El gestor de copia local podrá ser ejecutado en modo inatendido (batch), en horas cuando el acceso al repositorio es mínimo (por la noche y/o los fines de semana).

RF4: El gestor de transacciones comprobará también la coherencia y el buen sentido de las transacciones a procesar, según lo descrito en la página 75. Fallos en la coherencia darán lugar a advertencias o mensajes de error según configuración.

RF5: El gestor de transacciones interactuará con el sistema de dependencias para analizar si se cumplen todas las relaciones impuestas sobre las operaciones y los paquetes en la transacción, efectuándo las verificaciones de página 95. En caso de detectarse dependencias violadas, se generarán mensajes de alerta o de error según configuración.

RF6: El gestor de transacciones y el editor de dependencias podrán ser lanzados en modo batch, y en modo interactivo para su comunicación con una interfaz gráfica.

RF7: Los datos de entrada del gestor de transacciones y del editor de dependencias, podrán venir de un fichero o de la entrada estándar (introducidos por un humano o una interfaz gráfica).

RF8: El gestor de transacciones permitirá mantener informado al usuario de los detalles de la ejecución de la operación y del avance de la misma.

RF9: Las herramientas permitirán la ejecución simulada de todas las funcionalidades.

RF10: La ruta de acceso de todos los objetos (ficheros) consultados, creados y modificados por las herramientas deberá ser configurable.

RF11: Las herramientas permitirán almacenar la traza de ejecución en ficheros. El nivel de traza será ajustable por el usuario.

RF12: Se podrá obtener de las herramientas su configuración actual.

RF13: La interfaz gráfica podrá ser lanzada desde una máquina distinta a la cual se sitúa la herramienta.

RF14: Al iniciar su operación, los gestores de copia local detectarán si el repositorio maestro está disponible. En caso de no estarlo, se quedarán en un bucle de espera, volviendo a contactar al repositorio maestro pasado un intervalo de tiempo aleatorio. Caso de no conseguir acceder al repositorio al cabo de un determinado número de intentos, se enviará un mensaje al gestor del repositorio esclavo.

RF15: Los gestores de copia local no podrán inferir en la operación del repositorio maestro. Si se iniciase una actualización del maestro durante la actualización del esclavo, éste último lo detectará y finalizará su operación.

RF16: En ningún caso un gestor de copia local de un repositorio esclavo podrá modificar datos del repositorio maestro. Se accederá a él en modo de sólo lectura.

RF17: El gestor de copia local simplificará la lista de transacciones pendientes, según lo descrito en el apéndice B, página 125.

RF18: El gestor de transacciones no será operativo durante la actualización del repositorio maestro por el gestor de copia local para evitar inconsistencias.

RF19: Todos los mensajes generados por las herramientas estarán en idioma inglés.

RF20: Las herramientas estarán disponibles y serán operativas 24h/365d.

RF21: El sistema de dependencias dará soporte a las relaciones y dependencias definidas en la sección 4.3: las relaciones tipo *paquetes virtuales* y *grupos funcionales*, y las dependencias entre aplicaciones y entre aplicaciones y *ficheros*, *paquetes locales*, *configuración local*.

Requisitos de Prestaciones (RP)

RP1: El tiempo de respuesta del gestor de transacciones para validar una transaccion consistente en una sola operación (tipo RemoveFromProduction) sobre una sola arquitectura y la parte compartida (share), que implique un paquete compuesto de 1000 ficheros sobre sistema de ficheros AFS, en menos de 60 segundos sobre una estación Sun UltraSparc con 32MB de RAM, bajo carga de trabajo habitual del sistema de ficheros.

RP2: El gestor de transacciones será capaz de procesar una transacción en un espacio de tiempo menor que el gestor de copia local.

RP3: El número máximo de intentos para establecer comunicación entre los interfaces y las herramientas será configurable, asímismo como el tiempo de espera entre intentos.

Requisitos de Interfaz con otros sistemas o componentes (RI)

RI1: Los paquetes a introducir en el repositorio habrán sido creados por el generador de paquetes `ASIShappi`, que los emplazará en un repositorio auxiliar.

RI2: La comunicación entre los sistemas y sus interfaces se realizará a través del protocolo de comunicación definido en la sección 4.3.5. Se dispondrá de un módulo API común a todos los sistemas, pudiéndose ampliar los módulos API en función de cada herramienta particular, según lo descrito en la sección 4.3.5.

RI3: (Deseable) El gestor de transacciones analizará en su primera ejecución los distintos sistemas de ficheros en el que se sitúa el repositorio maestro de manera automática para explorar su estructura y geometría.

RI4: La herramienta `ASISdep` leerá la configuración del gestor de estación de trabajo, `ASISwsm` para verificar su coherencia.

RI5: El gestor de copia local accederá a la herramienta estándar `sendmail` para enviar correo electrónico.

RI6: Las herramientas `ASISwdep` y `ASIStm` serán integradas en el entorno de seguridad ASIS [17].

Requisitos operacionales (RO)

RO1: Para distinguir las herramientas ASIS, el nombre de los ejecutables estará compuesto por "ASIS", seguido del nombre de la aplicación. Por ejemplo, `ASIStm` será el nombre del gestor de transacciones.

RO2: El gestor de transacciones permitirá incluir una descripción de usuario en las transacciones que éste envíe. Esta descripción será almacenada junto con la transacción.

Requisitos sobre los recursos y portabilidad (RP)

RP1: El sistema de transacciones y el editor de dependencias deberán ser operativos en las plataformas UNIX oficiales en el CERN, debiendo estar equipadas al menos con 32MB de RAM.

RP2: La herramienta `ASISdep` deberá ser operativa en todas las estaciones de trabajo que pertenezcan a las plataformas UNIX oficiales y que tengan acceso al repositorio ASIS.

RP3: El gestor de transacciones y el gestor de copia local deberán ser operativos bajo uso de los siguientes sistemas de ficheros distribuidos: AFS, NFS y DFS bajo cualquier arquitectura UNIX oficial. También deberá ser operativo bajo el sistema de ficheros UNIX normal (UFS).

RP4: El equipo donde se instale el sistema de transacciones y el editor de dependencias deberá tener acceso al protocolo de comunicación TCP/IP para la comunicación entre las herramientas y los interfaces gráficos.

RP5: El equipo donde se instale el gestor de copia local deberá tener acceso a un servidor de correo bajo protocolo SMTP, para la notificación de cambios en el repositorio a los usuarios e notificación del resultado y traza de ejecución al gestor de repositorio.

Requisitos de Seguridad y Protección (RS)

RS1: Se garantizará la integridad de los datos del repositorio y de la base de datos del repositorio, debiendo existir mecanismos para la recuperación de la integridad en caso de cualquier anomalía.

RS2: Se dispondrá de un mecanismo para abortar de manera controlada la ejecución del sistema sin causar daños (por ejemplo, a través del uso de señales(*signals*)).

RS3: El gestor de transacciones será ejecutable sólamente por personal autorizado (gestores de producto o gestor de repositorio). El gestor de copia local sólamente será ejecutable por el gestor del repositorio esclavo. La autentificación de los usuarios se llevará a cabo por el entorno de seguridad ASIS.

RS4: El gestor de transacciones verificará la autorización de un gestor de productos sobre todos los productos que sean modificados en una transacción.

RS5: En el caso de producirse un fallo durante la ejecución de la replicación por el gestor de copia local, el gestor de copia local dejará el repositorio en un estado consistente.

Apéndice D

Diseño de las clases

A continuación, se presentan las clases diseñadas y las relaciones y jerarquías que existen entre ellas. Por cada clase, se muestran sus atributos y los servicios o métodos públicos según metodología OMT. Se mostrarán sólamente los elementos principales, dejándose su ampliación para el diseño detallado (ver sección 4.5, página 108).

Después de las clases se presentan las funcionalidades de los programas principales, y los elementos de configuración de las aplicaciones. Finalmente, se razona el modelo de base de datos utilizado.

D.1 Convenciones

Para la representación gráfica de las clases se utiliza la notación OMT (*Object Modelling Technique*) [30], de J. Rumbaugh.

En esta notación, resumida en la figura D.1, las clases son representadas divididas en tres zonas o niveles: en el superior, se indica su nombre, seguido de sus datos miembros (o atributos), y en el inferior, sus métodos. Una clase también puede ser representada únicamente por su nombre.

Las *relaciones* entre clases se representan uniéndo las clases relacionadas mediante una línea:

- Relación de *herencia*: se representa por el símbolo $\triangle$ (un triángulo) entre padre e hijos, emplazándose la clase padre por encima de las clases hijo.

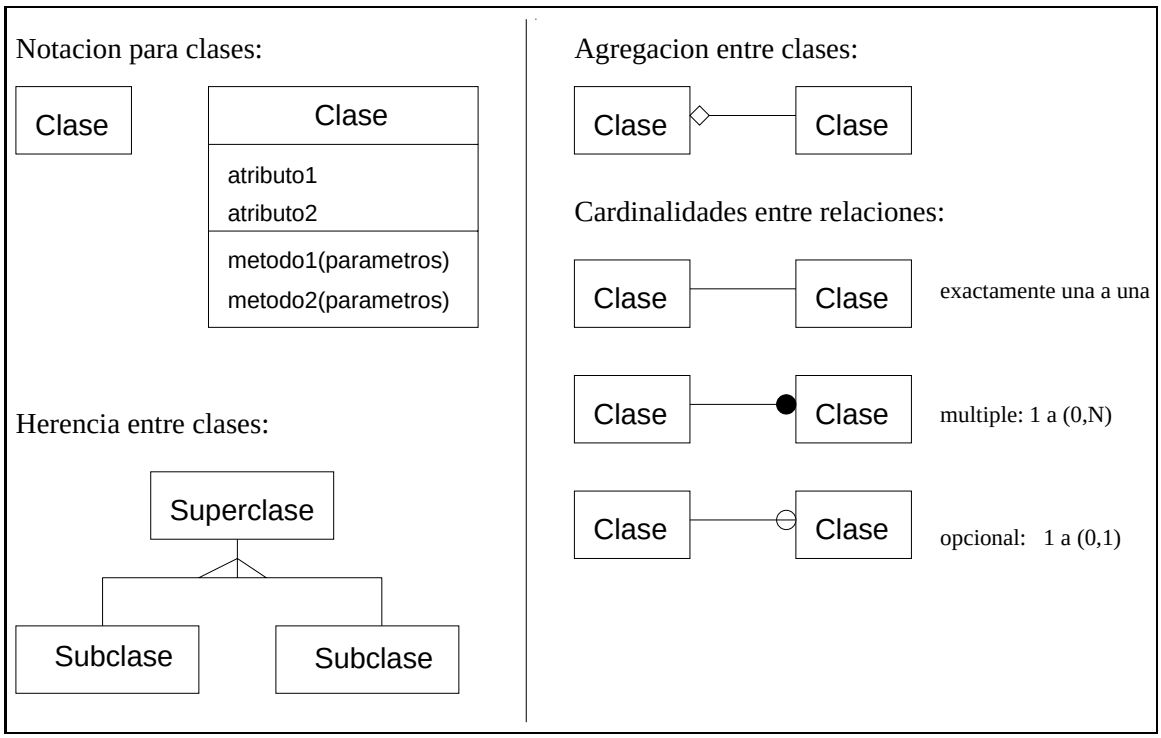

Figura D.1: notación OMT

- Relación de *agregación*: se representa por el símbolo ◇ (un rombo).
- Relación de *asociación*: se representa con la ausencia de los símbolos anteriores.

Las *cardinalidades* de las relaciones de agregación y asociación pueden ser de tres tipos: *uno a uno*, *uno a cero o uno*, y *uno a muchos*, representándose tal y como se muestra en la figura D.1.

Adicionalmente se definen las siguientes convenciones:

- Una "A" antes del nombre de una clase significa que ésta es *abstracta*.
- Una "V" antes del nombre de un método significa que este método es virtual.
- Un atributo tipo lista se representa anteponiéndole el símbolo "@".
- Los métodos para consultar y modificar cada uno de los atributos de los objetos (get-... y set-...) no se detallan en los gráficos por razones de claridad.
- Al instanciarse un objeto, se accederá a la base de datos y a los ficheros de configuración para inicializar sus atributos. Esto se hará en un método constructor que se ejecuta al instanciarse el objeto llamado "initialize".

D.2 Clases básicas

Las clases descritas a continuación se definen alrededor de las entidades principales del repositorio ASIS; es decir, las familias, los productos, las versiones de éstos, y el repositorio mismo.

Repository (Repositorio)

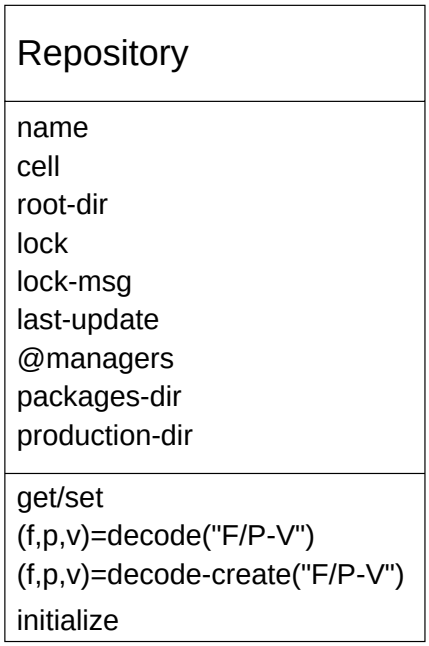

Figura D.2: clase Repository

Existe una instancia de esta clase por cada repositorio. Los atributos relevantes de la clase son el nombre del repositorio (name, por ejemplo `afs`), la célula o dominio del repositorio (cell, por ejemplo `cern.ch`) y el camino de acceso al repositorio dentro del sistema de ficheros distribuido (root-dir, por ejemplo: `/afs/cern.ch/asis`). Asímismo se almacena el camino de acceso al área de paquetes (packages-dir, por ejemplo: `/afs/cern.ch/asis/packages`) y el camino de acceso al área de "En Producción" (production-dir, ejemplo: `/afs/cern.ch/asis`). También se guarda como atributo si el repositorio está siendo actualizado y por quién (lock y lock-msg), la fecha y hora de la última actualización (last-update) y los identificadores de los gestores de repositorio (@managers). Los métodos decode y decode-create servirán para respectivamente instanciar y crear la familia, producto y versión cuyo nombre se pasa como parámetro.

Family (Familia)

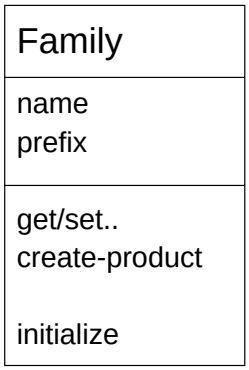

Figura D.3: clase Family

Esta clase es agregada a la clase Repositorio. Los atributos relevantes de una familia son el nombre (name, p.e. `CERNLIB`), y el directorio prefijo usado por las aplicaciones dentro de esta familia (prefix, por ejemplo, `/cern` o `/usr/local`). create-product instanciará el producto cuyo nombre se pasa como parámetro.

Product (Producto)

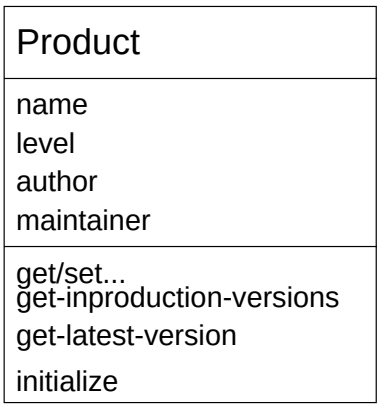

Figura D.4: clase Product

Esta clase es agregada a la clase Familia. Por cada producto existe una instancia de esta clase. Su atributo principal es el nombre (name, p.e. `emacs`). También se guarda información sobre nivel de soporte ofrecido (level), el autor del producto (author) y el gestor del producto (maintainer).

Se disponen de métodos para acceder según criterios a las versiones del producto según la plataforma, éstos son: get-inproduction-versions (obtener versiones "En Producción"), get-latest-version (obtener la versión más reciente).

Version (Versión)

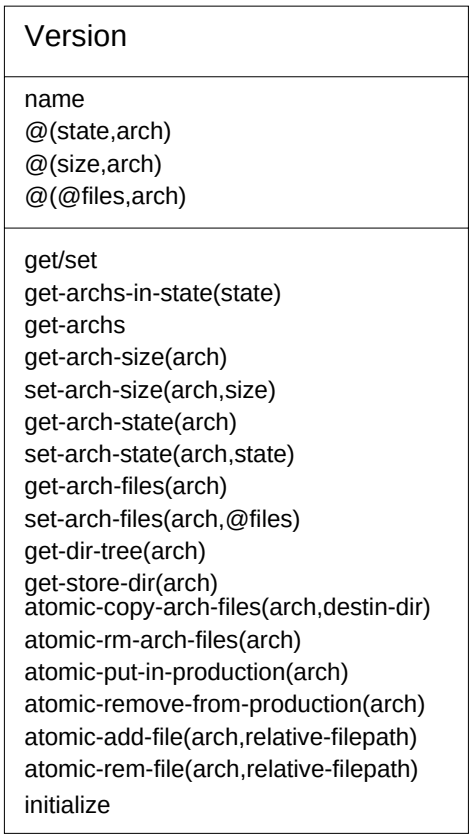

Figura D.5: clase Version

Esta clase, agregada a Producto, contiene los atributos y métodos necesarios para la consulta y manipulación de los paquetes software pertenecientes a una versión.

Si bien lógicamente la versión y los paquetes se pueden considerar entidades distintas, aquí se agrupan en una sola clase para evitar un aumento de complejidad innecesario a la hora de manejar los objetos, y por otro lado la alta cohesión existente entre paquete y versión invita a unirlos.

Un atributo es el nombre de la versión (name, es decir, el número de versión; por ejemplo: 19.34). También se guarda la información de los paquetes por cada plataforma (arch):

El estado dentro del modelo de procesado de "software" (state), el tamaño físico del paquete (size) y la localización relativa de los ficheros del paquete dentro de la jerarquía de directorios (@files).

Los métodos permiten leer y asignar el estado, el tamaño y los ficheros de los paquetes (get-arch-, set-arch-). El método get-archs devuelve una lista con todas las plataformas definidas. Existe tambien un método para obtener los directorios donde se almacenan los ficheros del paquete (get-dir-tree), y otro que devuelve el emplazamiento del paquete dentro del sistema de ficheros (get-store-dir).

Otro método sirve para obtener la lista de plataformas que tienen las plataformas cuyos paquetes estén en un determinado estado (get-archs-in-state).

En esta clase también se localizan los métodos que soportan las operaciones de transacción que involucran modificaciones en el sistema de ficheros; es decir, métodos para modificar el contenido y/o el estado un paquete *de forma atómica*. Estos métodos, en caso de que falle la copia, dejarán el sistema de ficheros igual que al inicio de la operación. Se basan en los métodos atomic-copy-arch-files y atomic-rm-arch-files, que copian y borran atómicamente los ficheros de un paquete:

- atomic-put-in-production: mueve un paquete del área de paquetes al área de producción.
- atomic-remove-from-production: mueve un paquete del área de producción al área de paquetes.
- atomic-add-file: añade un fichero al paquete.
- atomic-rem-file: borra un fichero del paquete.

Arch (Arquitectura)

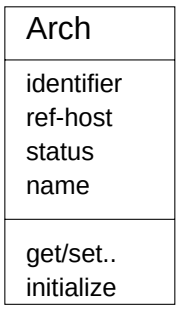

Figura D.6: clase Arch

Esta clase permite el acceso a la información en la base de datos sobre las plataformas (también llamadas arquitecturas) disponibles sobre el repositorio. Como atributos tiene un identificador único para cada plataforma (identifier, por ejemplo, `sun4x-55` para Solaris 2.5 sobre procesador SPARC) y el nombre de la plataforma (name). Los otros atributos son: el soporte dado a la plataforma (status, puede ser "active": activa, "frozen": retirada) y el nombre DNS de la máquina de referencia para la plataforma (ref-host, p.e. `sunsoref.cern.ch`).

Description (Descripción)

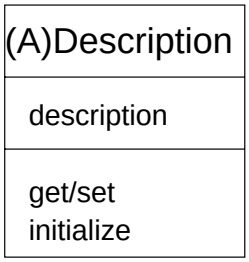

Figura D.7: clase Description

En la base de datos ASIS, tanto la familia, como el producto y la versión disponen de un atributo que es la descripción del objeto en cuestión. La clase abstracta Description es heredada por estas clases. Como atributo tiene la descripción del objeto (description).

Container (Contenedor)

La relación entre dos objetos normalmente da a lugar a que una clase se convierte en atributo de la otra, y que se definan métodos para acceder, crear y borrar este atributo/objeto. En cambio, con relaciones "uno a muchos" es necesario definir una estructura de datos más compleja que permita almacenar y manipular todos los atributos/objeto. Este es el caso de las relaciones entre las clases Repositorio, Familia, Producto y Versión.

Para favorecer la reutilización, se define esta estructura en una clase abstracta que es heredada por las clases que requieran manejar relaciones de este tipo.

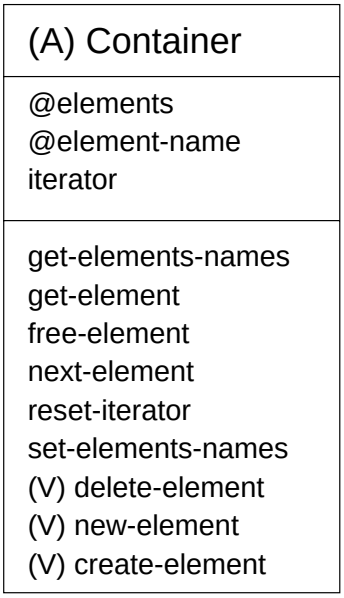

Figura D.8: clase Container

Se definen de manera genérica atributos y métodos para el manejo de listas de objetos (llamados aquí elementos). Los atributos de esta clase serán la lista de nombres de elementos existentes y un "buffer" con punteros a los elementos instanciados (@element-name y @element).

Al instanciarse un objeto cuya clase hereda de Container, llamará al método set-elements-names para indicarle cuáles son los nombres de los objetos contenidos (instancias de las clases agregadas). Por ejemplo, al instanciarse una familia llamará a set-elements-names con la lista de los productos que pertenecen a esta familia.

El acceso a los elementos por la clase ascendente será solicitándolo él o los elementos por su nombre. El Contenedor mirará si el elemento está en la lista de elementos instanciados, si es así devuelve una referencia al elemento. En caso contrario, llamará al método new-element que instanciará el elemento al que corresponde el nombre, y añade un puntero al elemento en la lista de elementos instanciados. Por lo tanto, el control sobre qué elementos ya han sido instanciados y cuales no es llevado por Container y es opaco a la clase heredera. De esta manera, se evita que un mismo elemento sea instanciado más de una vez (lo cual puede llevar a inconsistencias), y que se instancie y destruya un mismo elemento repetidas veces.

Se dispone de un iterador (iterator) para recorrer la lista de elementos, mediante los métodos next-element (devuelve el próximo elemento de la lista, incrementando el iterador) y reset-iterator (asigna el iterador al primer elemento de la lista).

El método new-element es virtual (debe ser implantado por la clase heredera) ya que requiere conocer características específicas del elemento.

Asímismo, existen métodos virtuales para crear y destruir físicamente un objeto (create-element y delete-element), que deberán ser implantados por la clase heredera.

ACL (Lista de Control de Accesos)

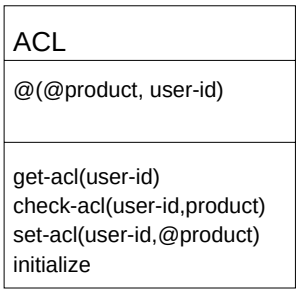

Figura D.9: clase ACL

En esta clase se manejan los permisos que tienen los usuarios para modificar productos. Cada usuario registrado tiene asociado una lista de productos que puede modificar. Se permiten comodínes para dar acceso al repositorio o a familias enteras, por ejemplo: `CERN.LIB/<ALL>` da permiso a modificar todos los productos de la familia CERN.LIB. El método check-ACL comprobará si el usuario tiene derechos de modificar el producto solicitado. El método get-ACL devuelve la lista de productos autorizadas para el usuario.

Las clases vistas se relacionan como sigue:

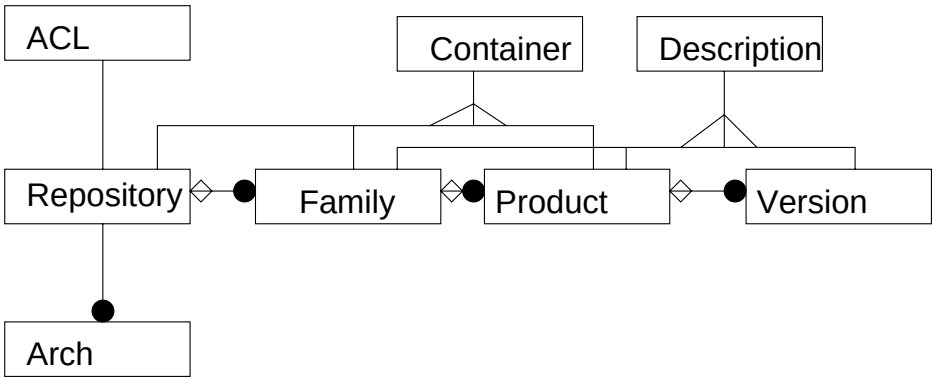

Figura D.10: relaciones entre las clases básicas

D.3 Gestión de mensajes y trazas

Para la generación y gestión de mensajes y ficheros de traza se dispone de dos clases, Reporter y Log. La primera es una clase abstracta y es heredada por *todas* las clases. La segunda se encarga de la generación física del fichero de traza.

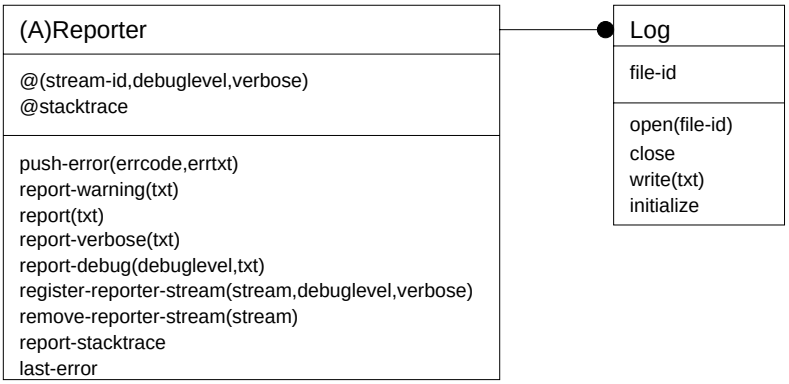

Figura D.11: clases Reporter y Log

Reporter (Informante)

A través de los métodos proporcionados por esta clase, se pueden generar mensajes generales, mensajes de traza, de advertencia (*warning*) y de error. Todas las clases generarán sus salidas únicamente llamando a métodos de esta clase. De esta manera, se puede redireccionar fácilmente todas las salidas, por ejemplo hacia un interfaz gráfico o ficheros de traza.

Si se desea que un dispositivo (por la regla general, un fichero de traza o la salida estándar) recoja los mensajes generados, hay que *registrar* el flujo de entrada (*stream*) de este dispositivo, junto al nivel de traza de referencia, llamando al método register-reporter-stream. Se da de baja un dispositivo mediante remove-reporter-stream. Hay dos tipos de traza: la traza de *debug* (depuración) y la traza *verbose* (explícita).

El método report se utiliza para enviar incondicionalmente un mensaje. El método report-warning genera incondicionalmente un mensaje de advertencia. El método report-verbose se utiliza para enviar mensajes que informan sobre el avance en la ejecución, y el método report-debug se utiliza para informar sobre estados y valores internos de la aplicación para facilitar la detección de errores.

Al llamar a alguno de los métodos de generación de mensajes proporcionados, se envían

a todos los dispositivos registrados. En el caso de report-debug se enviará sólamente a los dispositivos cuyo nivel de referencia es igual o superior al nivel de traza del mensaje. Asi será posible tener por ejemplo dos ficheros de traza, uno con una traza general y otro con una traza muy detallada, además de la información generada en la salida estándar, que puede tener un nivel de traza intermedio.

Los mensajes de advertencia y de traza recibidos son enviados inmediatamente a los flujos registrados, mientras que los errores enviados con push-error son insertados en una pila que puede ser volcada a petición (mediante report-stacktrace). Así, si el método B, que es llamado por el método A, genera un error, entonces A puede o bien recuperarse del error de B, vaciando la pila de errores, o bien volcar la pila de errores, con lo que el error sería enviado a los dispositivos. En caso que A estuviese llamada por otra función, también podría asímismo introducir otro error en la pila y finalizar la ejecución, con lo que la función que llama a A decide qué hacer.

El primer argumento al método push-error es un código de error POSIX. El código del último error introducido en la pila se puede recuperar con el método last-error.

Esta clase abstracta es heredada por todas las clases, siendo sus atributos estáticos y pertenecientes a la clase y no a las instancias, por lo que existe una única lista de flujos y una única pila de errores.

Log (seguimiento)

Esta clase se encarga de escribir físicamente el fichero de traza en el dispositivo. Existe una instancia de esta clase por cada dispositivo de traza. Se abre la traza con el método open y se cierra con el método close. La clase Reporter llama al método write para enviar los mensajes.

D.4 Clases para el manejo, reducción y ejecución de transacciones

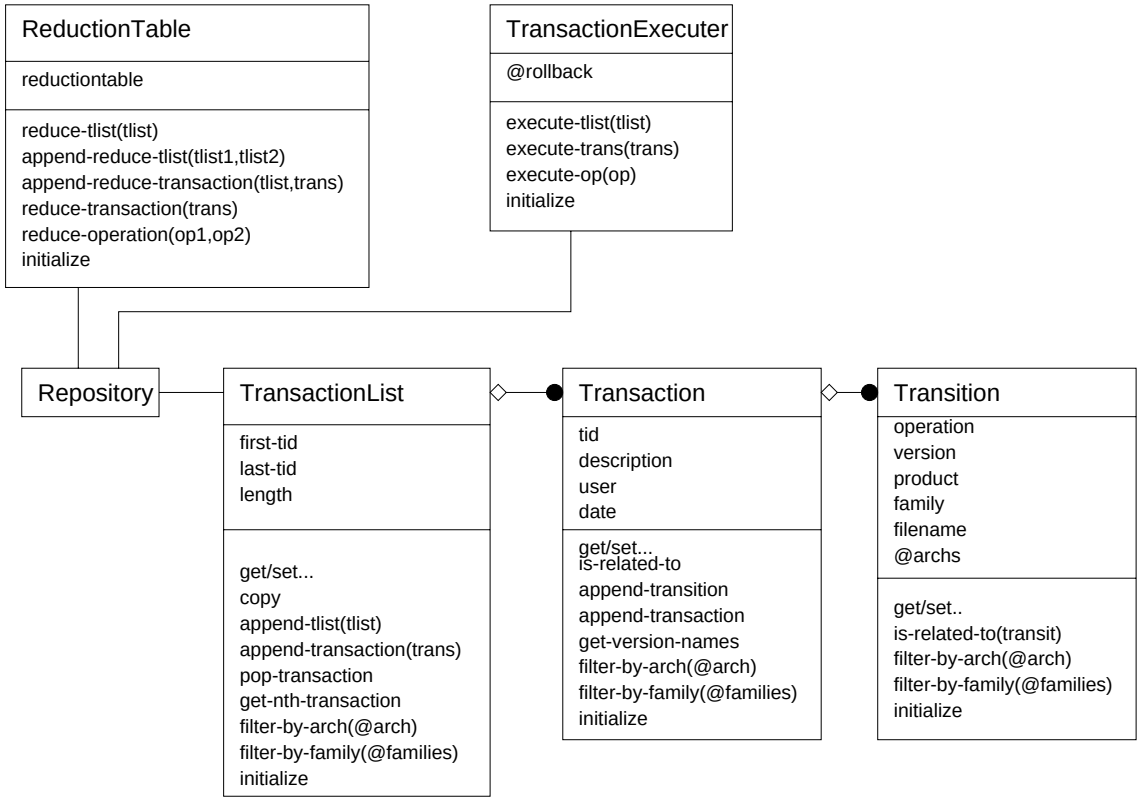

Figura D.12: clases para las transacciones

Las clases TransactionList, Transaction y Transition modelizan las transacciones y la clase ReductionTable ofrece los métodos para la reducción de transacciones.

TransactionList (Lista de Transacciones)

Esta clase contiene los métodos para manejar listas de transacciones. Sus atributos son el identificador de la primera y la última transacción de la lista (first-tid y last-tid), y la longitud de ésta (length). Dispone de métodos para generar una copia de sí mismo (copy), para añadirse una lista (append-tlist) o una transacción (append-transaction), y obtener la última o la enésima transacción de la lista (pop-transaction y get-nth-transaction, respectivamente).

Existe también un método para filtrar la lista de transacciones según las plataformas (filter-

by-architecture), que devolverá una lista con las transacciones que afectan a las plataformas indicadas, ignorando las operaciones de las transacciones que no se aplican a las plataformas en cuestión. Análogamente existe un filtro por familias (filter-by-family), que devuelve una lista con las transacciones que afectan a las familias indicadas.

Transaction (Transacción)

Los atributos de esta clase serán el identificador de la transacción (tid), el autor de la transacción (user) y la hora y fecha (date) de su creación, además de una descripción (description).

Los métodos permiten añadir una operación a la transacción (append-transition), o fusionar dos transacciones (append-transaction) secuencialmente. El método get-version-names devuelve todos los nombres completos de las versiones afectadas dentro de la transacción, en la forma "familia/producto-versión".

Los métodos filter-by-architecture y filter-by-family, de manera similar que en Transaction-List, filtran las transacciones según plataformas y familias, respectivamente.

El método is-related-to devuelve valor lógico verdadero si al menos una operación de la transacción está relacionada con al menos una operación de la transacción con la cual se compara.

Transition (Transición u operación)

En esta clase se guarda el tipo de operación (p.e. `IntroduceInProduction`), la versión de producto afectada (p.e. `GNU.EDIT/emacs-20.2`) y los identificadores de las plataformas sobre las cuales se ejecuta. En caso de ser una operación tipo "patch", contiene también el nombre y directorio del fichero de parche.

Al igual que la lista de transacciones y en la transacción, dispone de un método para filtrar según plataformas (filter-by-architecture) y familias (filter-by-family).

El método is-related-to devuelve valor lógico verdadero si la operación con la cual se compara afecta a la misma versión de producto y además se ejecuta sobre al menos una plataforma en común.

ReductionTable (Tabla de Reducciones)

En esta clase se encuentran los métodos para la reducción de transacciones (ver apéndice B, página 125). El atributo de la clase es la tabla de reducciones.

Los métodos disponibles son:

- reduce-tlist(tlist) toma una lista de transacciones como parámetro y devuelve la lista de transacciones simplificada correspondiente a la primera.
- append-reduce-tlist(tlist1,tlist2) añade la segunda lista a la primera y reduce la lista resultante.
- append-reduce-transaction(tlist1,transaction) añade la transacción a la lista y reduce la lista resultante.
- reduce-transaction(trans1) reduce la transacción
- reduce-operations(op1,op2) consulta la tabla de reducciones para los objetos op1 y op2.

TransactionExecuter (ejecutor de transacciones)

La ejecución de las transacciones se lleva a cabo en esta clase. El método execute-tlist ejecuta las transacciones de la lista. Los métodos execute-trans y execute-op ejecutarán una transacción y una operación, respectivamente. En la ejecución de estos métodos se genera una lista de "rollback" (vuelta atrás), con la lista de operaciones realizadas, para que en caso de fallo de ejecución se pueda deshacer las operaciones ya realizadas, restaurando el estado original. En el caso de las operaciones "RemoveFromRepository" y "Delete", adicionalmente, se genera una copia del paquete en un espacio (directorio) temporal para ser restaurado eventualmente. Para la ejecución de copias y movimientos de ficheros, se instancia la versión de producto afectada (clase Version) y se utilizan los métodos atómicos de esa clase. Así, en caso de que algún método atómico devuelva un fallo, el ejecutor de transacciones sólamente tiene que preocuparse en ejecutar las operaciones inversas por cada operación de la transacción fallada.

D.5 Estado virtual

La clases Repositorio, Familia, Producto y Versión representan una parte del estado *real* de la base de datos ASIS y el sistema de ficheros. Pero para satisfacer el requisito de que el acceso al estado real sea igual al acceso al estado virtual (de tal manera que sea opaco el uso de uno u otro) se define una clase "Repositorio virtual", además de clases Familia, Producto y Versión virtual. Estas clases tendrán los mismos métodos públicos (es decir, accesibles por otras clases) que sus correspondientes clases "reales", pero ejecutarán la operación virtualmente, según lo descrito en la sección 4.2.7 (página 74).

El acceso a las clases Repositorio y sus clases agregadas se realiza polimórficamente a través de *enlace dinámico*. Las clases que soliciten métodos al repositorio no conocerán su tipo, que podrá ser real o virtual.

Estas clases son las únicas que necesitan existir "virtualmente", es decir, que tienen un estado virtual. Todas las demás clases no afectan al funcionamiento *on-line* del repositorio, por lo tanto su estado puede ser modificado durante horas de trabajo. Es vital que no se altere el emplazamiento físico de los ficheros que conforman los paquetes al ser éstos constantemente accedidos por los clientes. Tampoco se deben modificar las entidades de la base de datos que indican donde se encuentran los paquetes; todo esto se gestiona en las clases Repositorio, Familia, Producto y Versión.

El control de la información virtual se lleva a cabo por dos clases: Vfiles y Vstate. La primera de ellas se encarga de la gestión del estado virtual del sistema de ficheros, y la segunda de la gestión del estado virtual de la base de datos (y por lo tanto, del control de las clases virtuales Repositorio, Familia, Producto y Versión).

El principio sobre el cual se basan ambas clases es el mismo. Ambas contienen como atributos sólamente *la diferencia entre el estado real y el estado virtual*. Por lo tanto, cuando un elemento no ha sido manipulado virtualmente, no existe ninguna información respecto a él en las clases Vstate y Vfiles.[1]

El acceso al estado virtual de un elemento (p.ej. una entidad de la base de datos o un fichero del sistema de ficheros) se efectúa siguiendo las siguientes reglas fundamentales:

- (acceso para lectura): **si** existe información virtual sobre este elemento, **entonces** leer esta información virtual. **De lo contrario**, consultar la información del estado

[1] Las ventajas de guardar nada más la diferencia entre los estados virtual y real son la mayor velocidad de acceso y el menor consumo en recursos. En caso del sistema de ficheros, al medirse el tamaño del repositorio clásicamente por Gigabytes, sería simplemente inviable guardar por separado toda la información virtual.

real.

- (acceso para escritura: crear/borrar/modificar elemento): **Si** existe información virtual sobre este elemento, **entonces** modificar esta información virtual. **De lo contrario**, leer la información del estado real (si existe), copiarla al estado virtual y modificar esta información virtual.

De esta manera, nunca se altera el estado real al modificar el estado virtual.

La diferencia entre estado virtual y estado real también se puede expresar en términos de la diferencia entre las transacciones validadas (por el gestor de transacciones) pero aún no ejecutadas (por el gestor de copia local). Para minimizar y evitar redundancias en el estado virtual, se descartan todos los elementos virtuales que hayan sido modificados (virtualmente) con anterioridad a la primera transacción validada pero no ejecutada.[2] Para ello cada elemento virtual tiene como atributo el identificador de la última transacción que lo modificó.

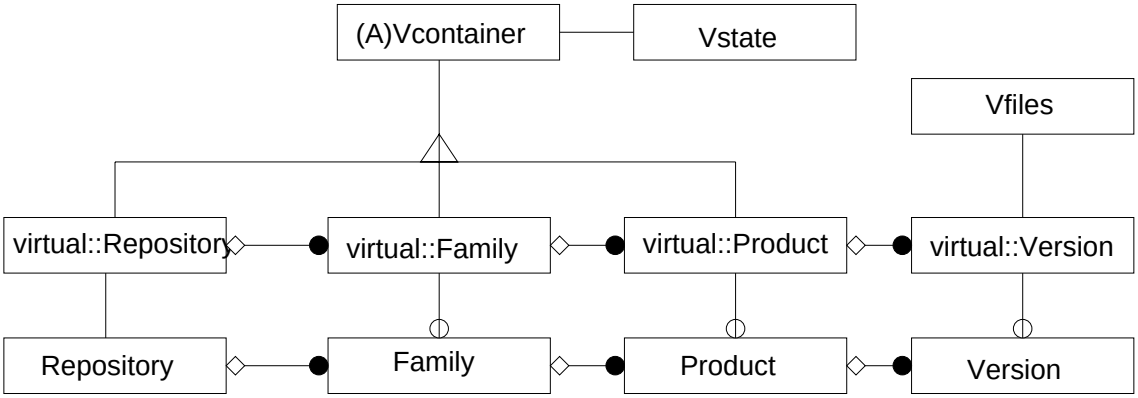

Figura D.13: clases para el estado virtual

virtual::Repository, Family, Product, Version y VContainer

Las clases virtual::Repository, virtual::Family, virtual::Product y virtual::Version disponen de los mismos métodos públicos que sus correspondientes clases reales, y encapsulan el acceso al estado virtual. Por ejemplo, el ejecutor de transacciones recibe un puntero al repositorio real en el gestor de copia local y al repositorio virtual en el gestor de transacciones; solicita objetos familia, producto y versión y ejecuta sus métodos pero en ningún

[2] Hay que tener en cuenta que cuando el gestor de copia local ejecuta una transacción, el estado de los elementos afectados por esa transacción pero por ninguna posterior deja de ser virtual.

momento distingue si trata con clases "reales" o "virtuales". Así se alcanza una máxima fidelidad entre las operaciones efectuadas para validar (por el gestor de transacciones) y al ejecutar (por el gestor de copia local).

Como atributo tendrán un puntero a una instancia de su correspondiente objeto real, si éste existe.

La clase virtual::Version, además tiene una referencia a la clase Vfiles, para el acceso al sistema de ficheros.

Todos los métodos seguirán las reglas indicadas anteriormente para acceder a información virtual.

Al igual que en las clases "reales", se dispone de una clase abstracta contenedor para la gestión de las relaciones "uno a muchos". Esta clase ofrece los mismos métodos que Container, y es heredada por las clases virtuales correspondientes. Esta clase contiene un puntero a la clase Vstate, de donde obtiene la información sobre el estado virtual de los objetos.

Vstate (estado virtual de la base de datos)

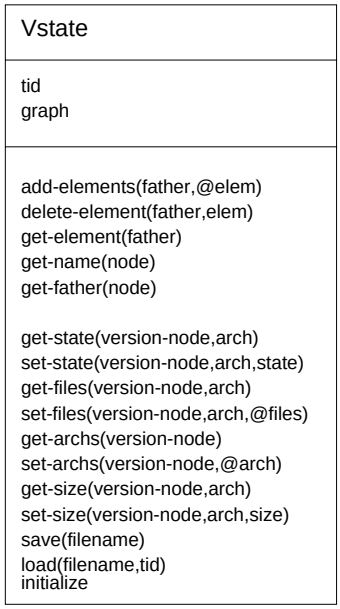

Figura D.14: clase Vstate

Las clases "virtuales" Repositorio, Familia, Producto y Versión interrogarán esta clase para

conocer la información virtual que existe sobre ellos y sus clases descendientes.

La información virtual de Repositorio, Familia, Producto y Versión se almacena en forma de grafo (concretamente, en forma de árbol). Cada nodo del árbol corresponde a una instancia de un objeto con modificaciones virtuales y cada arco indica una relación de ascendencia/descendencia. Así, el nodo raíz representa el repositorio, y los nodos que salen de éste serán las familias; por cada familia hay nodos descendientes que representan los productos; y los nodos descendientes de los productos serán las versiones.

Cada vez que un objeto "real" sea modificado, se crea su nodo correspondiente en el grafo.

Sólamente existirán nodos para los objetos que tienen información virtual, y todos sus nodos ascendentes (tengan o no información virtual).

Cada nodo tiene una serie de atributos. Estos son:

- tid: identificador de la última transacción que modificó virtualmente el objeto
- estado: puede ser
  - nothing(nada): no existe información virtual
  - deleted(borrado): objeto ha sido borrado en el estado virtual (se ha ejecutado el método delete-element de VContainer sobre ese objeto)
  - added(añadido): objeto ha sido creado respecto al estado real (no existe en el estado real, ha sido creado mediante create-element de VContainer)
  - modified(modificado): objeto ha sido modificado en el estado virtual

Los nodos que corresponden a las instancias de la clase Versión tendrán estos atributos adicionales por cada plataforma:

- estado y tamaño del paquete
- lista con los ficheros del paquete

La figura siguiente muestra un ejemplo de representación gráfica del árbol (siendo R: Repositorio, F: Familias, P: Productos, y V: Versiones):

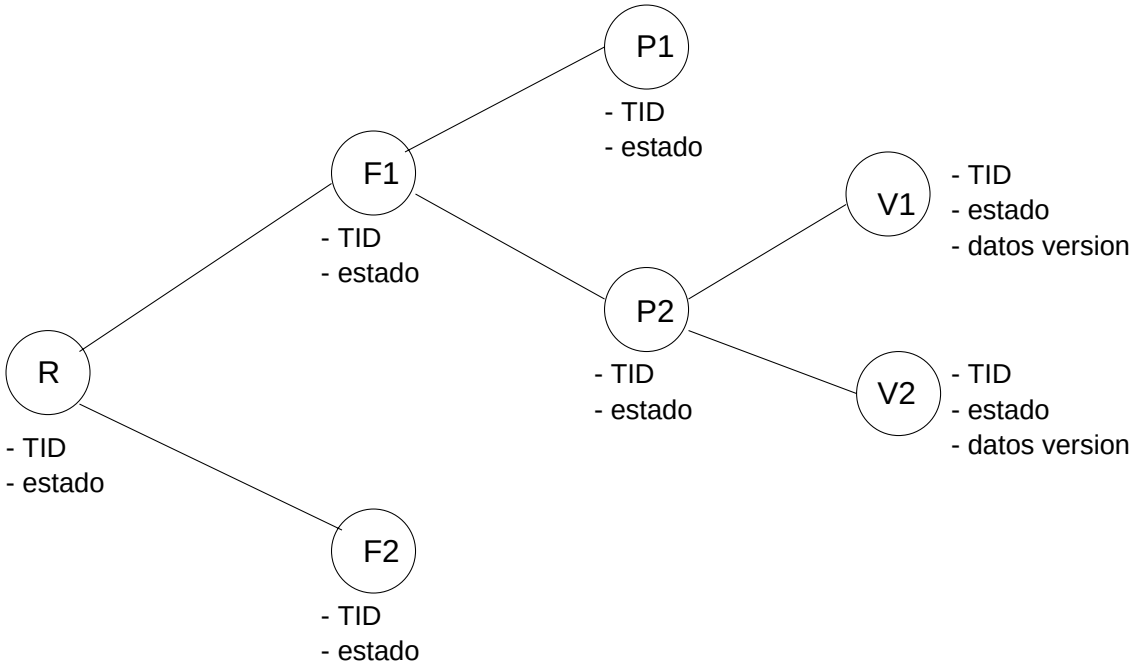

Figura D.15: árbol con información virtual

La clase dispone de métodos para cargar y salvar el estado virtual (load y save, respectivamente). El método load toma como parámetro el identificador de la última transacción ejecutada por el gestor de copia local, y descarta toda la información del árbol que sea más antigua que esa transacción, con el fin de tener sólamente las modificaciones virtuales no ejecutadas aún.

Los métodos add-element y delete-element crearán y borrarán respectivamente un nodo del árbol.

Los métodos get/set-archs, get/set-state, get/set-files y get/set-size permiten consultar y asignar a los nodos versión las plataformas existentes, y por cada una de ellas, el estado, los ficheros y el tamaño del paquete correspondiente.

El métododo initialize llama al método load para cargar el estado virtual nada más instanciarse el objeto.

Vfiles (estado virtual del sistema de ficheros)

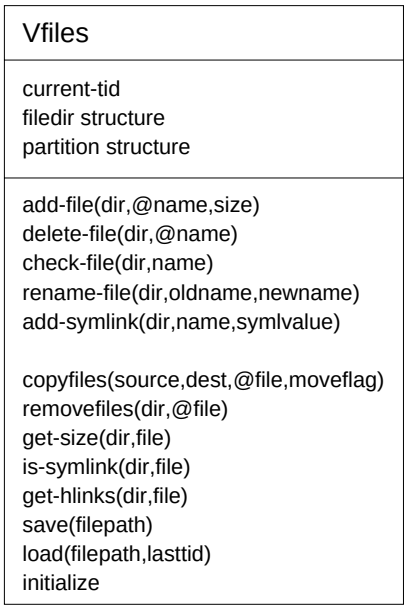

Figura D.16: clase Vfiles

La clase virtual::Version utiliza los métodos de esta clase para manipular "virtualmente" el sistema de ficheros.

La estructura del modelo está basada en el sistema de ficheros UNIX, el cual debe emular "virtualmente", descartando los aspectos y elementos no relevantes para la ejecución virtual.

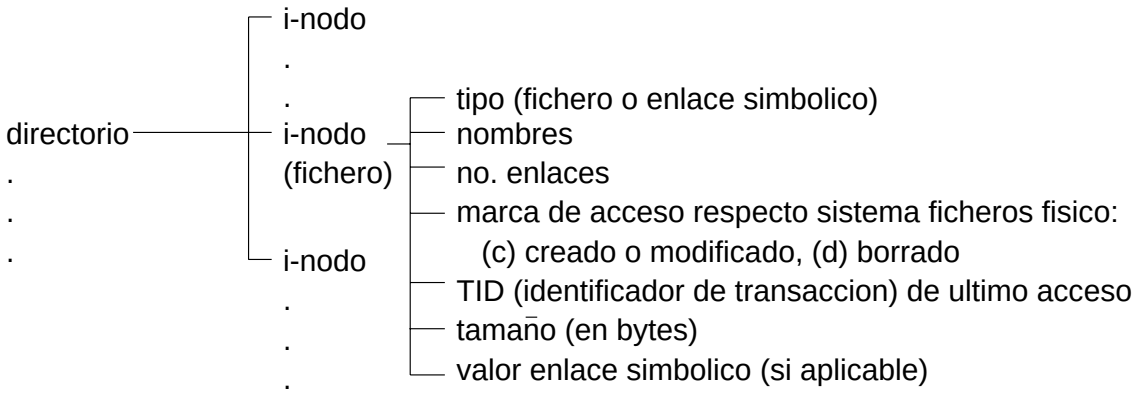

Figura D.17: sistema de ficheros virtual

En la figura anterior se muestra el modelo adoptado. El sistema de ficheros está formado por directorios y ficheros. Cada fichero puede tener uno o varios nombres (llamados *hard links*

o enlaces duros), o bien ser un fichero de tipo *symbolic link* (enlace simbólico). Sólamente se guardan los ficheros que han sido creados, modificados o alterados respecto al sistema de ficheros real.[3]

A su vez, cada directorio pertenece a una partición:

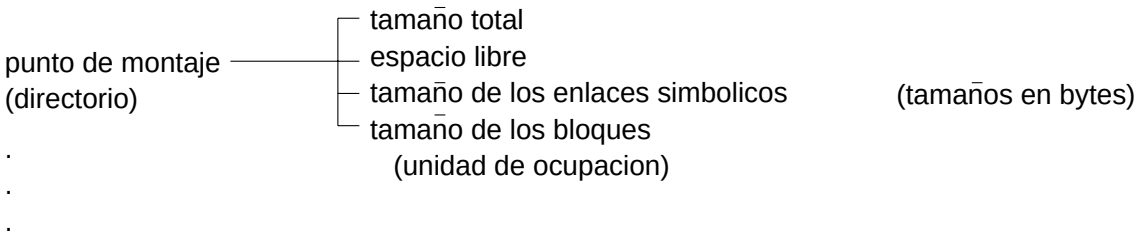

Figura D.18: particiones virtuales

Por cada partición se guarda el tamaño total y el espacio libre (real y virtual) en ella. Además, se almacena el tamaño ocupado por un fichero de tipo enlace simbólico, y el tamaño de bloque (unidad mínima y atómica de almacenamiento). La información inicial sobre la geometría de las particiones es leída de un fichero de configuración.

Las limitaciones funcionales más importantes respecto al sistema de ficheros UNIX es que un fichero no puede tener enlaces duros en directorios distintos, y que un enlace simbólico no puede ser de tipo directorio. Además, si un fichero tiene múltiples nombres, éstos no pueden ser manipulados por separado por una operación tipo "patch".

Estos supuestos carecen de importancia en el contexto del repositorio ASIS al no estar permitidos, pero son detectados por Vfiles, en ese caso se genera un error.

Hay métodos para añadir, borrar y verificar la existencia de un fichero (add-file y add-symlink, delete-file y check-file). Además existen métodos para obtener las características de un fichero (get-size obtiene el tamaño, is-symlink si es un enlace simbólico, get-hlinks el número de enlaces duros). Mediante el método rename-file se puede cambiar de nombre un fichero, siempre que éste tenga un sólo nombre.

Se pueden copiar y borrar grupos de ficheros con los métodos copyfiles y removefiles.

Todos estos métodos comprueban que las operaciones son factibles, por ejemplo, al copiar ficheros se comprueba que los ficheros origen existen y que no se sobreescribe ningún fichero al copiar.

[3]No se almacenan datos como las fechas de acceso, permisos, o propietarios al carecer de importancia en este contexto.

Los métodos save y load respectivamente almacenarán y recuperarán la información sobre el sistema de ficheros virtual. Al igual que en la clase Vstate, el método load toma como parámetro el identificador de la última transacción ejecutada por el gestor de copia local, y descarta toda la información del sistema de ficheros que sea más antigua que esa transacción.

D.6 Dependencias

A continuación se describen las clases definidas para modelizar el sistema de dependencias.

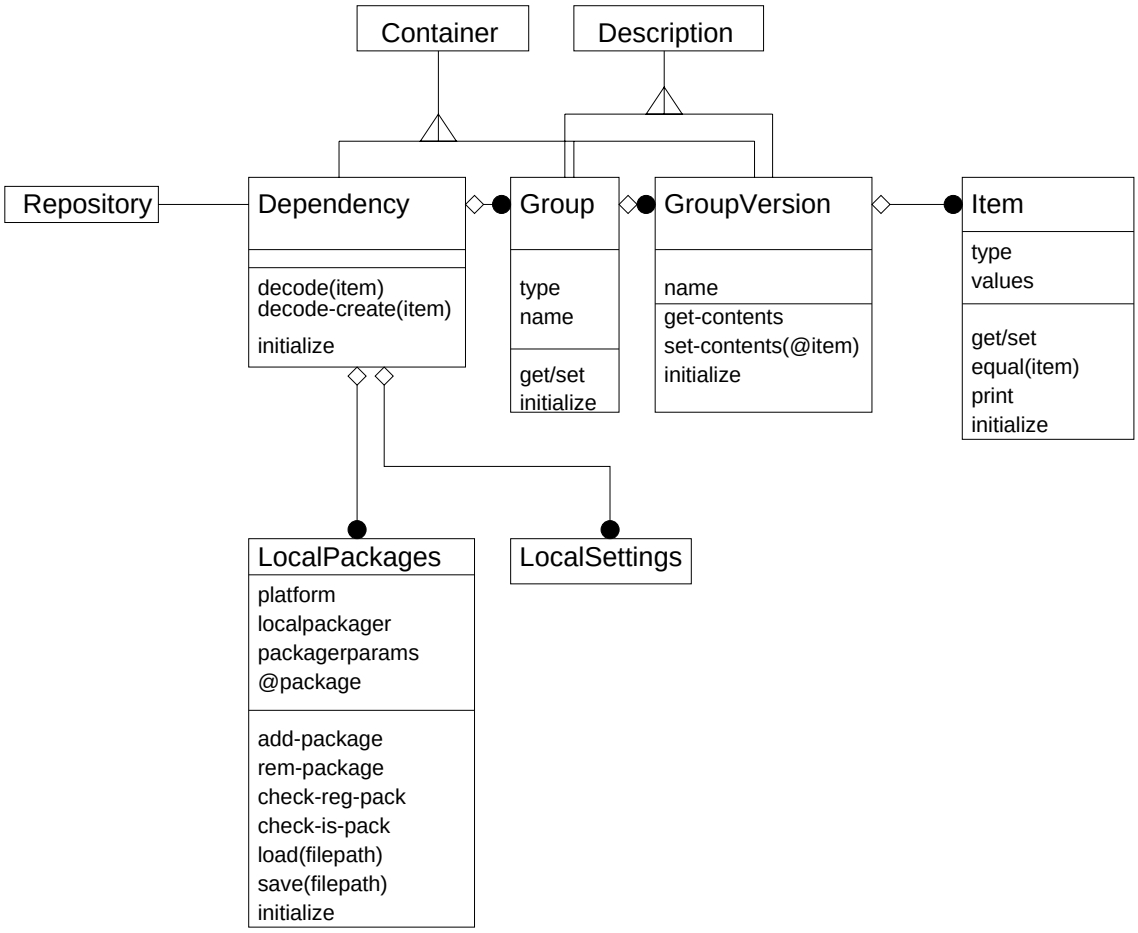

Figura D.19: clases para el sistema de dependencias

Item

La clase Item modeliza los objetos que forman las dependencias. Los tipos y formato de las dependencias se muestran en la página 87).

Por cada objeto que forma parte de una dependencia se genera una instancia de esta clase. Los atributos de la clase son el tipo (que puede ser grupo funcional, grupo virtual, versión de producto, fichero, paquete local o configuración local), y los valores asociados al tipo. Estos valores varían según el tipo; para una versión de producto es el nombre de la familia, del producto y de las versiones; para un paquete virtual es el nombre del paquete y la versión etc.

El método equal (igual) comprueba si un objeto es igual a otro, y print devuelve una tira con el tipo y los valores en el formato adecuado.

Dependency (Dependencia)

Tanto los grupos funcionales y los paquetes virtuales, los paquetes locales y los elementos de configuración local se obtendrán desde la clase Dependency. Esta clase hereda de Container para el manejo de la clase Group. Análogamente a la clase Repository dispone de los métodos decode y decode-create para instanciar y crear, respectivamente, grupos virtuales, grupos funcionales, paquetes locales y configuraciones locales.

Esta clase es controlada por la clase Repository.

Group y GroupVersion (Grupo y Versión de Grupo)

Aunque la funcionalidad de los grupos funcionales y paquetes virtuales es distinta, comparten la misma estructura. Ambos están compuestos de versiones de producto ASIS, y ambos son divididos en versiones.

Las clases Group y GroupVersion representan los grupos funcionales y los paquetes virtuales y sus versiones. Group contiene como atributo el tipo (grupo funcional o paquete virtual) y el nombre, mientras que el atributo principal de GroupVersion es el nombre de la versión. Ambas clases heredan de Description para gestionar la descripción del objeto. Group hereda de Container para el manejo de las versiones.

Los métodos get-contents y set-contents sirven para consultar y asignar, respectivamente, el contenido de GroupVersion. Al ser el contenido de una versión (ya sea de grupo funcional

o de paquete virtual) siempre versiones de producto, el contenido de GroupVersion es una lista de instancias de la clase Item de tipo Versión de Producto.

LocalPackages y LocalSettings (Paquetes locales y Configuración local)

La clase LocalSettings no ha sido definida aún; se requiere un análisis más detallado sobre qué tipos de configuración local han de ser modelizados, cuáles deben ser comunes a todas las plataformas y cuales serán específicos por plataforma.

Existe una instancia de la clase LocalPackages por cada plataforma que disponga de un sistema propio de gestión de paquetes. Aparte del identificador de plataforma, se tiene como atributo una lista con los nombres y la descripción de los paquetes conocidos[4]. Otro atributo es el camino de acceso a la aplicación gestora de paquetes (localpackager, por ejemplo, `/usr/bin/pkginfo` en Solaris, `/usr/sbin/setld` en Digital Unix) en y los argumentos de esta aplicación para listar los paquetes instalados sobre la estación de trabajo.

Relation y RelType (Relación y Tipo de Relación)

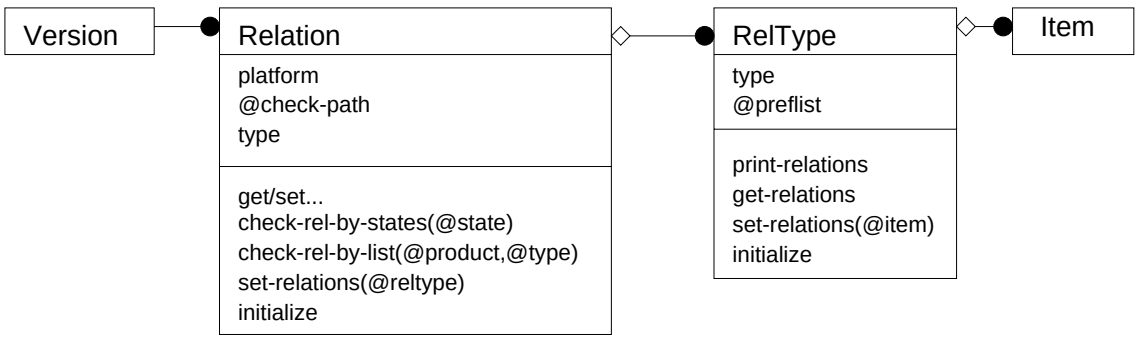

Figura D.20: clases Relation y RelType

Las clases Relation y RelType guardan la información sobre las dependencias definidas sobre una versión de producto. Existe una instancia de Relation para las dependencias generales y una instancia por cada plataforma que tenga definida dependencias de sistema. A su vez, existe una instancia de RelType por cada tipo de dependencia. Los componentes de RelType serán instancias de la clase Item, que representan los objetos sobre los cuales se establecen las dependencias.

[4]esta lista sirve para ayudar al gestor de producto a la hora de incluir un paquete local en las dependencias.

La clase Relation cuelga de Version y cada instancia de Relation pertenece a una versión de producto particular.

Atributos de la clase Relation serán el tipo (type, que puede ser general o de sistema); en caso de tratarse de una dependencia de sistema se tiene también el identificador de plataforma (platform) y una lista de directorios en los que típicamente se almacenan los ejecutables en esa plataforma (@check-path). Esta lista de directorios se utiliza al comprobar las dependencias hacia objetos de tipo fichero ejecutable.

El método check-relations-by-list comprueba si se cumplen las dependencias dentro del dominio expresado por la lista de versiones de producto pasado como parámetro. El parámetro @type es una lista con los tipos de objetos a comprobar. Devuelve una lista con las dependencias incumplidas, además por cada dependencia incumplida indica:

- el tipo de dependencia
- el objeto u objetos que provocan el fallo
- la razón del fallo:
  - Dentro del dominio no se encuentra el objeto, en caso de una dependencia positiva (o que se encuentra, en caso de una dependencia negativa) hacia el cual se ha expresado una dependencia.
  - dentro del dominio no se satisface un grupo funcional o un paquete virtual hacia el cual se ha expresado una dependencia. En este caso se indican qué objetos del grupo funcional o paquete virtual no se encuentran.

El método check-relations-by-states hace las mismas comprobaciones. La diferencia está en que el dominio serán todas las aplicaciones dentro de la lista de estados que se pasa como parámetro.

Las dependencias se salvarán con el método set-relations. Antes de salvarlas se comprueba que son correctas (por ejemplo, que no haya grupos funcionales ni paquetes virtuales en dependencias negativas), que todos los objetos son instanciables y que no aparecen autoreferencias (de tipo A requiere A).

La clase RelType almacena la información por cada tipo de dependencia. El atributo principal de la clase es el tipo (que puede ser requiere, recomienda, choque, conflicto). Los objetos que componen cada dependencia individual son las alternativas (que no son más que listas de instancias de la clase Item). Si solo existe una alternativa, esta lista contendrá nada más que el objeto en cuestión.

D.7 Subsistema de comunicaciones

La cooperación entre cliente y servidor está basada en el intercambio de mensajes entre ambos. El cliente envia un mensaje petición que contiene una función y sus parámetros, y el servidor le devuelve un mensaje respuesta que contiene los resultados obtenidos.

El cliente podrá ser un humano o un interfaz gráfico. Todos los intercambios se realizarán en código ASCII, de esta manera se garantiza la portabilidad entre sistemas.

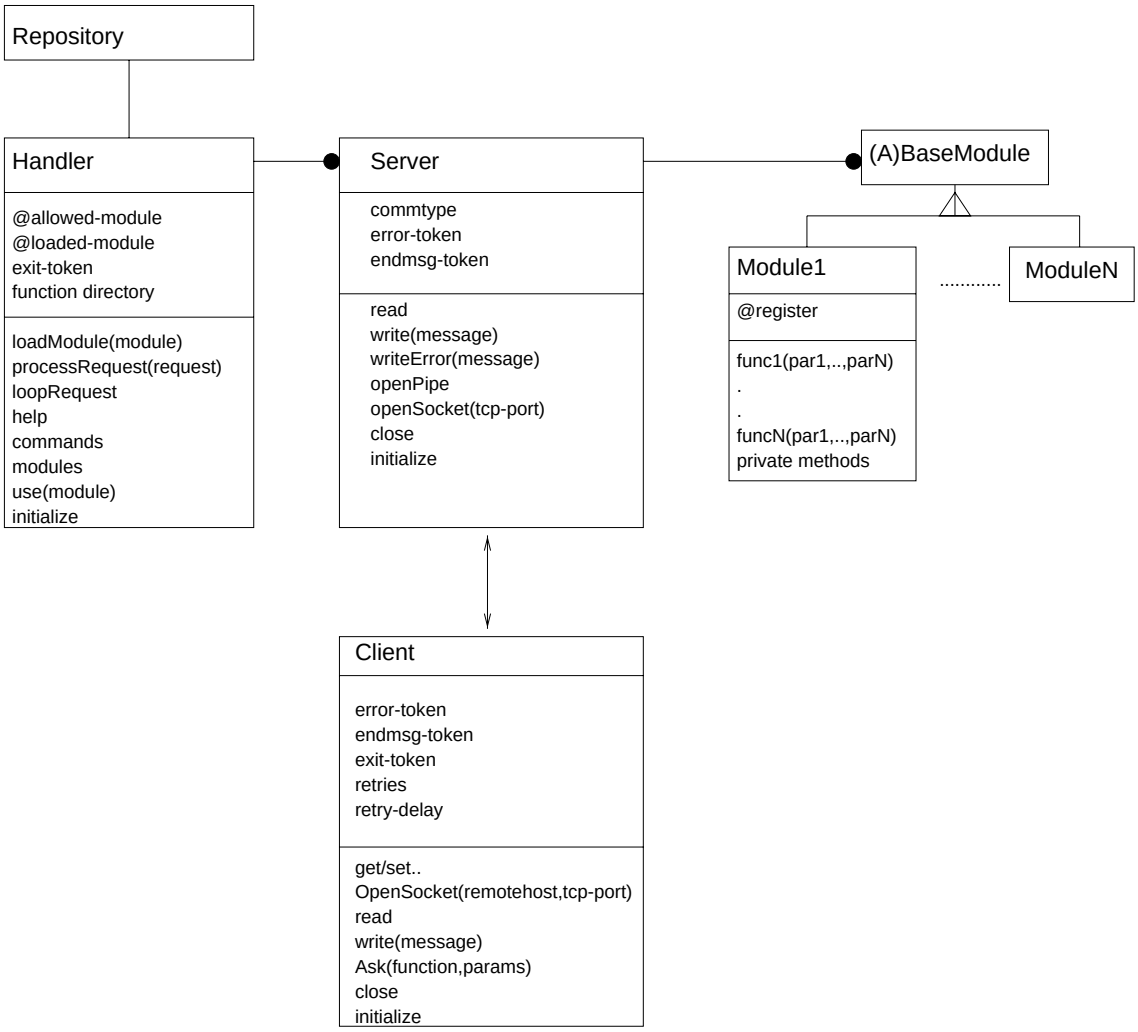

Figura D.21: clases del subsistema de comunicaciones

Las clases Handler (manejador), Server (servidor) y los módulos API formarán parte de la aplicación mientras que la clase client esta integrada en la interfaz.

La clase Handler opera a nivel de sesión pero no conoce las características del canal de comunicación, que son encapsuladas en la clase Server.

Handler (manejador)

En la clase Handler reside la lógica de control de la comunicación por parte de la aplicación. Sus atributos serán la lista de módulos API cargados (loaded-modules), la lista de módulos API autorizados (allowed-modules) para ser usados por la aplicación y el directorio de funciones API. El directorio de funciones es una estructura de datos que contiene el nombre de la función, el módulo API al que pertenece y un texto con la descripción y sintaxis de la función.

El método loadModule carga el módulo API cuyo nombre se pasa como parámetro, siempre que este módulo esté autorizado para ser utilizado. El método processRequest recibe como parámetro una petición; su salida es la respuesta a la petición. Este método es usado por otro, loopRequest, que va recibiendo peticiones por el canal de comunicación y procesándolas hasta que recibe por el canal un comando de finalización.

Los métodos help, commands, modules y use implantan las funciones internas. El método help proporciona la descripción y sintaxis de un comando; el método commands, la lista de comandos (funciones) disponibles por cada módulo cargado; el método use llama a loadModule para cargar un módulo API. El método modules devuelve la lista de módulos cargados y la lista de módulos autorizados.

Para procesar una petición se consulta el directorio de funciones y se solicita el método correspondiente a la función al módulo en cuestión (o a las funciones internas en su caso).

Server (servidor)

La clase Server encapsula el canal de comunicación y la implantación de los mensajes. El canal puede ser de tipo socket TCP/IP o *pipe* (tubería) sobre la entrada/salida estándar.

Los mensajes son delimitados por *tokens* (señales) de fin de mensaje (*prompt token*), y los errores del servidor al cliente se identifican por comenzar por un *token* de error.

El método open espera la conexión del cliente. En caso de canal tipo socket, recibe como parámetro el puerto donde ha de esperar la conexión; al producirse la conexión guarda el nombre, dirección y puerto de la máquina cliente. Nada más efectuarse la conexión manda un *prompt token* al cliente.

El método close cierra la conexión con el cliente.

El método read espera un mensaje del cliente, que será el valor de retorno. Los mensajes del cliente irán terminados por un token de fin de línea. El método write envía un mensaje al cliente. Este mensaje puede consistir en múltiples líneas (separadas por tokens de fin de línea) y es finalizado por un token de fin de mensaje. El método writeError envía un mensaje de error al cliente; el formato de un mensaje de error es idéntico al de un mensaje normal salvo que adicionalmente se envia un token de error.

Client (cliente)

Esta clase es la que se utiliza por parte de la interfaz para iniciar la conexión, realizar consultas al servidor y cerrar la conexión. El método OpenSocket abre una conexión socket TCP/IP con el servidor en la máquina y el puerto dados como parámetro. La dirección de la máquina puede ser dado en formato DNS o IP. El número máximo de reintentos de contactar el servidor en caso de fallo se fija como atributo (retries), al igual que el intervalo temporal entre intentos (retry-delay). El método write envia un mensaje al servidor mientras que el método read espera a recibir un mensaje del servidor. El método Ask (pregunta) envia un mensaje al servidor y espera la respuesta (este método combina read y write). El método close cierra la conexión con el servidor, enviándole un comando de finalización y cerrando el canal.

BaseModule y clases *módulo*

En estas clases se emplazan los módulos API con las funciones que enlazan la aplicación con su interfaz (ver sección 4.3.5, página 101).

A cada módulo API le corresponde una clase. Todas las clases "módulo" heredan de la clase abstracta BaseModule. El método de inicialización de esta clase registra los métodos de la clase "módulo" que servirán como funciones API basándose en un atributo que facilita el nombre, sintaxis y descripción de éstas (@register). Los métodos no registrados serán privados y para uso interno dentro de la clase.

Las funciones tendrán acceso a la los objetos de la clase Repository y todas sus clases agregadas y/o asociadas, a las que solicitarán sus servicios. El resultado de la función será entregado a la clase Handler a través del valor de retorno del método.

Los módulos API son cargados y enlazados dinámicamente *on demand* (bajo demanda) por la clase Handler en tiempo de ejecución, con lo que se minimiza el tamaño de la aplicación,

al estar cargados nada más que los módulos estrictamente necesarios por cada aplicación en cada momento.

El problema de la agrupación de funciones (ver página 101) se hubiese podido solucionar también definiendo una jerarquía de todos los módulos utilizando herencia, en vez de usar listas de módulos autorizados. Pero al ser la herencia un concepto estático, todas las clases deberían ser cargadas desde el principio, independientemente de si van a ser usadas o no. Esto aumentaría considerablemente el tamaño de la aplicación.

D.8 Programa principal y control de ejecución

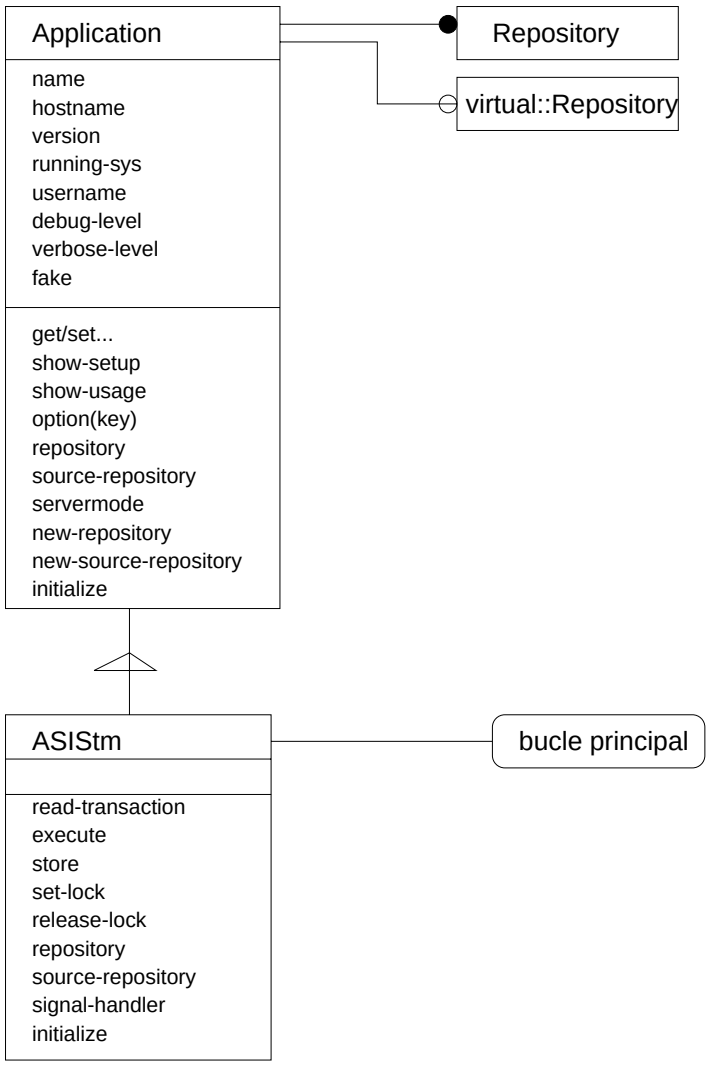

Figura D.22: clase Application y programa principal

Application (Aplicación)

Las aplicaciones ASIS deben tener una estructura y un manejo uniforme. Se define una clase abstracta, Applicacion (aplicación), de la cual cada aplicación individual hereda los métodos definidas en ella.

Como atributos de la clase Application formarán parte el nombre de la aplicación (name), la versión de la aplicación (version), el nombre de usuario (username), el nombre de la máquina sobre la cual se ejecuta (hostname), el identificador de la plataforma de ejecución (running-sys) y los niveles de traza *verbose* y *debug*. La ejecución de la aplicación será fingida (*faked execution*) si el atributo *fake* tiene valor lógico verdadero.

A través de esta clase se accede a las opciones de la línea de comando mediante el método option, que devuelve el valor de la opción cuyo nombre se pasa como parámetro. El método show-setup muestra algunos valores típicos de configuración, como el nombre del repositorio, directorio raíz, etc. El método show-usage muestra un mensaje de ayuda para el uso de la aplicación.

El método servermode lanza la aplicación en modo servidor o interactivo, pasándo el control de ejecución al método loopRequest de la clase Handler.

Los métodos source-repository y repository devuelven una referencia al objeto repositorio fuente y destino, respectivamente. En caso de aplicaciones que instancian un sólo repositorio (como `ASISdep` y `ASISwdep`) éste es el repositorio destino. A través de los métodos virtuales new-source-repository y new-repository se instancian estos repositorios. Estos métodos han de ser implantados en la clase derivada al ser ésta la que conoce que tipo de repositorio (real, virtual o ninguno) corresponde a cada uno de ellos.

Programa principal y bucle principal

El programa principal es asímismo una clase, que hereda de Application. En ella se definen los métodos principales necesarios para la ejecución de la aplicación.

El *control* de flujo de la aplicación se lleva a cabo en el *bucle principal*, que deberá ser mínimo. El bucle principal tiene una referencia a la instancia del programa principal y ejecuta los métodos necesarios y en orden.

Antes de entregar el control al bucle principal, Application lee la línea de comandos, y en caso de solicitarse la configuración de la aplicación, la ayuda, o el modo servidor, entrega el control de ejecución a los métodos respectivos. A continuación finaliza.

Ejemplo: Gestor de transacciones

En el caso del gestor de transacciones `ASIStm`, los métodos relevantes del programa principal son:

- new-source-repository: instancia el repositorio fuente (que será el repositorio auxiliar), de tipo "real".
- new-repository: instancia el repositorio destino (que será el repositorio principal), de tipo "virtual".
- read-transaction: lee una transacción y devuelve un objeto de la clase Transaction.
- set-lock: bloquea (*lock*) atómicamente ambos repositorios, fuente y destino.
- release-lock: desbloquea los repositorios fuente y destino.
- execute: comprueba la ACL, ejecuta la transacción, comprueba la coherencia de la transacción (implantando las reglas descritas en la página 75) y el cumplimiento de las dependencias (implantando las reglas descritas en la página 4.3.3).
- store: almacena el nuevo estado virtual, y añade la transacción a la lista de transacciones validadas.
- signal-handler: encargado de abortar de manera segura la aplicación en caso de recibir ésta una interrupción asíncrona solicitándo su terminación.
- reset-virtual-repository: este método se llama si los cambios en el repositorio virtual han de ser cancelados por los gestores de repositorio en caso de existir algún problema. Esencialmente vacía el estado virtual y anula las transacciones correspondientes; siempre y cuando el gestor de copia local no haya ejecutado alguna de ellas.

El flujo de control definido por el bucle principal sigue los siguientes pasos, para ejecutar una transacción:

1. instanciar programa principal
2. leer transacción (read-transaction)
3. bloquear repositorios (set-lock), si *fake* es falso
4. ejecutar transacción (execute)
5. almacenar nuevo estado virtual (store), si *fake* es falso
6. desbloquear repositorios (release-lock), si *fake* es falso

Modo servidor

El gestor de transacciones `ASIStm` puede ser lanzado en modo servidor. El modo servidor se usa primordialmente para la comunicación con una interfaz gráfica, si bien es accesible directamente por un humano. La apariencia, en ambos casos, es la misma; el token de fin de mensaje es `''ASIStm% ''`, por lo que tiene la apariencia de un "shell" de usuario. Aparte del módulo API "básico", el gestor de transacciones utilizará un módulo API específico, llamado "tm". Este módulo permitirá consultar la lista de transacciones, comprobar la validez de transacciones, el estado de los repositorios, etc. Las funciones de este módulo serán definidas a medida que se diseñe la interfaz.

Gestor de copia local

El funcionamiento del gestor de copia local es muy similar al del gestor de transacciones. Una diferencia es que ambos repositorios instanciados son de tipo "real". Se bloquea sólamente el repositorio esclavo, ya que se accede al repositorio maestro en modo lectura. Sin embargo, si el repositorio maestro está bloqueado (por estar siendo actualizado, por ejemplo), se esperará un número determinado de veces durante un tiempo variable aleatorio. Si el repositorio sigue bloqueado, se enviará un mensaje a los gestores del repositorio y se finaliza la ejecución.

Adicionalmente se filtra (mediante los métodos filter-by- de TransactionList) y reduce (mediante ReductionTable) la lista de transacciones del maestro. Por cada transacción resultante se llama al método **execute** de TransactionExecuter y se actualiza la lista de transacciones ejecutadas. Entre ejecución de transacciones se comprueba si el repositorio maestro ha sido bloqueado, en ese caso se finaliza la ejecución. Finalmente se envia correo electrónico (llamando a la utilidad `sendmail`) notificando las transacciones ejecutadas. Al igual que en el gestor de transacciones, se dispone de un mecanismo encargado de abortar de manera segura la aplicación en caso de recibirse una interrupción.

El momento y la frecuencia de ejecución del gestor de copia local es controlado por el gestor de repositorio, que incluirá su ejecución en la configuración general del servidor.[5]

Por el momento no se prevée el uso de un interfaz gráfico para `ASISlcm`, por lo que no tiene módulo API propio.

[5] por ejemplo, a través de la utilidad estándar `cron`.

Ejemplo: sistema de dependencias

Aplicación `ASISwdep`

`ASISwdep` es la aplicación para crear y editar dependencias. Su programa principal dispone de los siguientes métodos:

- new-repository: instancia el repositorio de tipo "real".
- set-master-version: asigna la versión maestra de una versión.
- create-group: crea un grupo funcional, paquete virtual, o una versión.
- set-group-description: asigna una descripción a un grupo funcional o paquete virtual, o a una versión de éstos.
- set-contents: asigna el contenido a un grupo funcional o paquete virtual, o a una versión de éstos.
- set-relations: asigna las dependencias generales o de sistema a una versión de producto.

El bucle principal evalúa las opciones de la línea de comandos y según la operación deseada, entrega el control de ejecución a alguno de los métodos indicados.

Modo servidor y módulo API

`ASISwdep` dispone de un modo servidor, que ha sido diseñado para interactuar con su interfaz gráfico, `tkdep`, actualmente en desarrollo. Para su comunicación con este interfaz se ha definido un módulo API (llamado "dep"), que contiene las siguientes funciones API:

- group-names(type): devuelve la lista de nombres de los grupos funcionales o paquetes virtuales, dependiendo de type.
- group-version-names(type,name): devuelve la lista de las versiones del grupo funcional o paquete virtual name.
- group-description(type,name[,version]): devuelve la descripción asociada a un grupo funcional o paquete virtual (o una versión de éste).

- group-contents(type,name,version): devuelve la lista de componentes de la versión del grupo funcional o paquete virtual.
- master-version(family,product,version): devuelve la versión a utilizar como plantilla para la aplicación family/product-version.
- dependencies(family,product,version,platform,reltype): devuelve las dependencias del tipo reltype definidas para la aplicación identificada por family/product-version y la plataforma platform.

Adicionalmente, se utilizan algunas funciones del módulo API "Info", común a las herramientas ASIS que acceden al repositorio:

- families: devuelve la lista de familias del repositorio.
- products(family): devuelve la lista de productos pertenecientes a la familia family.
- versions(family,product): devuelve la lista de versiones de productos pertenecientes a la familia family y el producto product.
- archs: devuelve la lista de plataformas definidas.

Aplicación `ASISdep`

A través de `ASISdep` se comprueba la coherencia desde el punto de vista de las dependencias del repositorio y de la configuración del gestor de estación de trabajo `ASISwsm`.

Los métodos relevantes del programa principal son:

- check-wsconfig: lee la configuración de `ASISwsm` y comprueba las dependencias de las versiones de producto de la configuración.
- check-repository: comprueba la coherencia del repositorio.

Esta aplicación no dispone de modo servidor.

D.9 Configuración

Las aplicaciones permiten ser configuradas por sus usuarios. Esta configuración se efectúa en ficheros de configuración por parte de los gestores de repositorio. Los valores de los

ficheros de configuración pueden ser sobreescritos en la línea de comandos.

Configuración del repositorio

Esta configuración se establece por cada repositorio. Los elementos importantes son:

- nombre del repositorio y nombre de la célula
- nombre e identificador de los gestores de repositorio
- nombre de la máquina gestora para manipulación del repositorio

Configuración general

Esta configuración es general a todas las aplicaciones ASIS:

- la localización del repositorio usado, dentro del sistema de ficheros distribuido
- valores para determinar el tipo de información de salida generada (*debug* y *verbose*) (por defecto, nulos)
- ejecución simulada o no (*fake*) (por defecto, ejecución real)
- el directorio donde guardar la información de traza

Configuración del módulo de comunicaciones

Se fijan determinadas características para el módulo de comunicaciones:

- puerto TCP al que se debe conectar el cliente
- *token* de error y *token* de fin de mensaje (*prompt*)
- número máximo de intentos de contactar el servidor
- tiempo de espera entre intentos de contacto con el servidor

Configuración del gestor de transacciones

Los valores de configuración son:

- localización del repositorio auxiliar dentro del sistema de ficheros distribuido
- valor booleano para indicar si se desea efectuar las comprobaciones de coherencia (por defecto, se comprueba)
- valor booleano para indicar si se desea efectuar las comprobaciones de verificación de dependencias (por defecto, se comprueba)
- valor booleano para indicar si las dependencias fallidas (de tipo "choca" y "requiere") deben de generar un error y causen el rechazo de la transacción (por defecto, generan una advertencia)
- valor booleano para aceptar o no la inclusión de plataformas no activas en las transacciones (por defecto, no se acepta)

Adicionalmente, existe un fichero de configuración para la definición inicial de la geometría de las particiones, usado por la clase Vfiles.

Configuración del gestor de copia local

Sus valores de configuración son:

- localización del repositorio maestro dentro del sistema de ficheros distribuido
- las plataformas a replicar (por defecto, todas)
- las familias a replicar (por defecto, todas)
- lista de recipientes del resultado de ejecución

Configuración del gestor de dependencias

- directorio donde se localiza el fichero de configuración de `ASISwsm` (por defecto, `/usr/local/adm`)
- los áreas del repositorio a comprobar (por defecto, los áreas Certificado y En Producción)

- las plataformas a comprobar (por defecto, todas)
- lista de productos a comprobar (por defecto, todos).

D.10 Modelo de la base de datos ASIS

La base de datos ASIS actual es una base de datos jerárquica, basada en directorios y ficheros con datos. Una base de datos jerárquica se puede concebir como una base de datos relacional en la cual se limitan la existencia de relaciones a entidades agregadas.

El acceso se realiza creando, borrando y modificando ficheros y directorios, que representan las entidades y relaciones de la base de datos. Esta modelización resultaba correcta en un principio, cuando las únicos datos almacenados era la información sobre las familias, productos y versiones, que se organizan jerárquicamente por agregación.

Se evalua la posibilidad de sustituir este modelo por una base de datos que permita modelizar relaciones más completas y encapsular la implementación física de las entidades y relaciones (comúnmente llamadas bases de datos relacionales), dado que con los conceptos nuevos (como las transacciones y las dependencias), la estructura lógica de la base de datos aumenta en complejidad, apareciendo nuevas entidades y relaciones no jerárquicas.

Para ello se evaluaron distintas bases de datos relacionales (en concreto, Postgres, mSQL y Oracle).

El mayor problema es cuando se produce un cambio en el repositorio, y las estaciones de trabajo empiezan a actualizar sus enlaces hacia el repositorio. En el CERN, la base de datos tiene que soportar un pico de hasta 2000 estaciones de trabajo que lanzan `ASISwsm` en una misma franja horaria. Una base de datos requiere un servidor central al que se conectan los clientes solicitándole las interrogaciones (*queries*); este servidor ha de resolverlas y devolver los resultados para cada uno de los clientes. Parte del esfuerzo computacional, por lo tanto, es realizado en los servidores.

Mediante el esquema actual, el único acceso central es al sistema de ficheros, ya que la computación se lleva a cabo en cada estación de trabajo.

La compra de una base de datos comercial, tipo Oracle, se descarta dado que se quiere mantener el carácter de libre distribución del software, además de minimizar las dependencias hacia otros sistemas. El uso de mSQL y Postgres se descarta al no existir implantaciones suficientemente fiables y/o potentes.

Por lo tanto, por el momento se mantiene el uso de la base de datos jerárquica, extendiéndola en los conceptos necesarios:

- la lista de transacciones se implanta como un fichero que contiene todas las transacciones
- la lista de control de acceso se implanta como un fichero único
- los grupos funcionales y los paquetes virtuales se implantan como una jerarquía de directorios, donde sus atributos y componentes serán almacenados en ficheros
- las dependencias se guardan como ficheros junto a las versiones de producto correspondientes.

Sin embargo, al encapsularse el acceso a la base de datos ASIS dentro de las clases, el modelo elegido es opaco desde el exterior de las clases.

De esta manera, se deja abierta la posibilidad de poder cambiar, a un coste de modificación mínimo, de modelo de base de datos - en el momento que exista una alternativa aceptable.

Apéndice E

Ejemplo de documento de diseño detallado

A título de ejemplo se presenta un documento de diseño detallado, junto a su información del sistema de control de versiones CVS. En concreto, se corresponde a la clase Vfiles del estado virtual.

Todos los documentos tienen la misma estructura. Se empieza por el nombre de la clase, seguido de la sinopsis de su uso, el nombre de las clases de las que hereda y una breve descripción de su función. A continuación, se presentan los métodos públicos y los métodos privados de la clase. Se muestra en cada método su nombre, parámetros y valor de retorno junto a una descripción de su funcionalidad. También se indican las estructuras de datos utilizadas. La descripción de métodos y datos es específica al lenguaje de programación (en este caso, PERL), lo cual limita la aplicación del diseño detallado a otros lenguajes, pero permite ganar en detalle y exactitud.

La información de control de versiones muestra el nombre y camino de acceso al fichero dentro del repositorio de desarrollo, los nombres simbólicos, y la lista de revisiones. Por cada revisión se indica la fecha y hora, su autor, una pequeña descripción del cambio efectuado y las líneas modificadas del fichero.

NAME

Vfiles : ASIS Transaction Manager Virtual File System Class

SYNOPSIS

```
use virtual::Vfiles;
$vf=virtual::Vfiles->new($repository_root,$partition_info_file);
$vf=virtual::Vfiles->load($repository_root,$filename,$lasttid,$currenttid);  ....
$vf->save([$filename]);
```

INHERITANCE

asis::Reporter

DESCRIPTION

Manages the files, directories and quotas of the Virtual Repository. As a matter of fact, only the differences to the real repository are stored.

Public methods

add_file($to_dir, $size, $force,$name [$name2,$name3...]):boolean

add_file stores the $named filenames in $to_dir of the virtual fs $vf. If more than one $name is given, it is considered that the names are hard links to the same file. The size of the file (in bytes) is specified in $size. Returns undef if the file does already exist or if $size exceeds the remaining space on disk. If $force is !=0 then a eventually existing file is deleted (overwritten).

delete_file($dir,@filenames):boolean

Deletes the files (or symlink) in $dir named @filenames. All names of the file have to be given. If the file doesnt exist, undef is returned, else 1.

check_file($dir,$filename):boolean

Checks for the file in $dir named $filename. If the file doesnt exist, undef is returned.

rename_file($dir, $old_name, $new_name):boolean

rename_file renames the filename from $old_name to $new_name in the directory $dir. $old_name cannot be hardlinked with another filename. Returns undef on failure, else 1.

add_symlink($to_dir , $name , $force,$value):boolean

Adds a symlink in the specified directory $to_dir, being the symlink value equal to $value. If $force !=0 the operation overwrites the previous file, if existing. If $force is not set, a filename clash will return undef.

copyfiles ($src_dir,$dest_dir,$@files,$force,$move):boolean

"Copies" $@files from $src_dir to $dest_dir. $@files is a reference to an array with the filename and relative paths. Preserves the hard links found. $force !=0: overwrites existing files. $move=1: removes source files. If $force is not set, a filename clash will return undef.

removefiles ($basedir,$@files):boolean

Virtually removes $@files, starting from base directory $basedir. A file may not be hardlinked to another outside $@files. Returns undef if a file is not found.

set_currenttid($tid):true

sets the current tid (transaction ID) to $tid.

get_currenttid:$tid

returns the current tid.

save([$file]):boolean

Saves the virtual fs to disk. Has to be used after validating and commiting a transaction. If $file is defined it will be used for storing the virtual fs, else the last used load file will be used. Returns undef if a failure arises during saving.

general format for the generated file:

```
directory1 (eg. /afs/.cern.ch/asis/packages/GNU.EDIT/usr.local/bin/emacs)
file11
file12
....
directory2
file21
file22
....
```

format for each file:

```
name1(*name2..)|mark|size|tid|[symlvalue]|
```

load($rep_root,$file,$lastTID,$currenttid):$vf

Loads the virtual FS from disk and creates a new Vfiles instance. $lastTID is the last executed

transaction by ASISIcm. FS entries with a TID smaller than or equal to $lastTID are ignored. $currenttid is the tid of the transaction being processed. $rep_root is the current repository root, and $file the path to the Vfiles file. Returns undef if file cannot be read or is corrupted.

get_names($dir,$name):@names

returns the names associated with the inode $name belongs to being @names an array with all the names (hardlinked directory entries) to this file. This method is slow in execution and should be avoided.

get_size($dir,$file):$size

returns the size, in bytes, of $file in $dir. Returns undef if the file does not exist.

is_symlink($dir,$file):boolean

returns the link value if $file in $dir is a symlink, undef otherwise.

get_hlinks($dir,$file):$hlinks

returns number of hard links to the file pointed to by $file. undef is returned if the file does not exist.

is_virtual($dir,$file):boolean

returns 1 if file $dir/$file is virtual (has a entry in the VFS) - even if the file is marked as "d". Otherwise, undef is returned.

get_mountpoint($dir):$mountpointdirectory

returns the (virtual) mountpoint directory of the partition on which $dir is "mounted" on. The last accessed directory is cached internally for faster access.

get_partition_info($dir):($free,$total,$symlsize,$blocksize,$mountpointdir)

Returns complete virtual information about the partition $dir is mounted on: $free=current free size, $total=total size, $symlsize=symlink size, $blocksize=block unit size, $mountpointdir=mount point directory.

load_partition_info($file):boolean

Loads the partition information from the file $file. Returns undef if the file cannot be loaded. File format:

```
directory total-size free-space symlink_size block_size
directory total-size free-space symlink_size block_size
...
```

Example:

```
/ 100000000 5000000 1024 1024
/afs/.cern.ch/asis/packages 6553600 6553600 512 1024
```

Private methods

_init_dir($dir,$lasttid,$@filelist):boolean

initializes a directory with all his files after reading from disk. Used by the load method.

_add_partitions($part1,[$part2...]):boolean

Adds the partitions described as $part1 $part2.. to the partition list. Used by the load method.

_modify_space($dir,$diff[,$mountpointdir]):boolean

Adds $diff to the free space on the partition $dir is "mounted" on. If $diff is negative, the free space is decremented. Returns undef if free space is insufficient, 1 otherwise. The minimal amount of space allocated is the block size of the partition, and the real space allocated is the amount of blocks used (computed in bytes). $mountpointdir is calculated if not specified.

_initialize

Initializes the object. This method is called by asis::Object.

Internal Data

```
directories and files:

-----directory/filename----  ---inode---

$self->{'VF'}{$dir}{$name}->{'HLINKS'}  # how many hard links
                           {'MARK'}    # "c" or "d"
                           {'SIZE'}    # in bytes
                           {'TID'}     # modification TID
                           {'SYML_TO'} # string (symlink value)
                           {'INODE'}   # vfs inode id

partition info:

$self->{'PARTITIONS'}{$dir}{'TOTAL'}    # total size in bytes
                          {'FREE'}     # free        "
                          {'SYMLSIZE'} # size of a symlink
                          {'BLOCKSIZE'}# block size, in bytes (minimum alloc)
```

SEE ALSO

virtual::*

CVS Revision Information

```
RCS file: /afs/.cern.ch/project/asis/cvs/share/asistm/virtual/Vfiles.pm,v
Working file: Vfiles.pm
head: 1.5
branch:
locks: strict
access list:
symbolic names:
       v3_97_2: 1.2.0.2
       v971014: 1.2
       v971007: 1.1
keyword substitution: kv
total revisions: 5;    selected revisions: 5
description:
----------------------------
revision 1.5
date: 1997/11/18 11:09:43;  author: gcancio;  state: Exp;  lines: +2 -2
more verbose info added
----------------------------
revision 1.4
date: 1997/11/17 20:19:56;  author: gcancio;  state: Exp;  lines: +35 -31
some bugfixes in removefiles method
----------------------------
revision 1.3
date: 1997/11/17 15:49:57;  author: gcancio;  state: Exp;  lines: +57 -19
some bugfixes
----------------------------
revision 1.2
date: 1997/10/10 10:02:18;  author: gcancio;  state: Exp;  lines: +2 -19
inherits now from asis::Object, new method removed
----------------------------
revision 1.1
date: 1997/09/25 16:08:51;  author: gcancio;  state: Exp;
first version in CVS
=============================================================================
```

Apéndice F

Ejemplos de ejecución

A continuación se presentan algunos sencillos ejemplos de ejecución de los sistemas diseñados.

sistema de transacciones

Ejecutando el gestor de transacciones con la opción `-help`, se obtiene un mensaje de ayuda:

```
ASIStm: the ASIS transaction manager
Usage: ASIStm [options]

--help
        Displays this help message.
--verbose
        Verbose mode, prints performed operations.
--fake
        Do not actually execute the operations.
--version
        Prints current version and exits.
--repository <dir> (default is command location)
        Uses repository with given root.
--siteconfigdir <dir> (default is deduced from repository)
        Directory where to find site configurations
--debug {0|1|2|3|4|5}
        Set the debugging level to the given value.
--showsetup
        Describes the setup environment of the command and exits.
--interactive
        Switches to interactive or server mode.
--socket <port#>
        Use TCP/IP connection on the given port (server only).
--errortok <string> (default ERROR:)
```

```
        Token to insert with an error message (server only).
--prompt <string> (default is Application name and version)
        Prompt value (server only).
--file <file>
        Reads transaction from file <file> instead of standard input
--RESET
        (restricted) resets virtual repository to real repository (drop virtual state)
--master <dir>
        root of the scratch repository
--mastersiteconfigdir <dir> (default is deduced from master root dir)
        Directory to find configurations for the master repository.
--logdir <dir> (default adm/com/asismgr)
        Directory to store the log of ASIStm ops.
--partdir <dir> (default adm/share/config)
        Dir where to find partitionsrc file of the destination repository
--tmdir <dir> (default adm/com/asistm)
        directory where to find the vfiles and vstate files
--checkcoherence {yes|no} (default yes)
        Perform coherence checks on transaction
--checkdep {yes|no} (default yes)
        Perform dependency check
--forcedep {yes|no} (default no)
        generate always warnings in dependency check, never errors
--frozen {yes|no} (default no)
        Accept frozen architectures in transactions

This is ASIStm version 1.03
```

Mediante `ASIStm` validamos ahora una transacción para introducir una aplicación en el repositorio:

```
IntroduceInRepository GNU.EDIT/lyx-0.10.7 (sun4x_55)
```

El resultado de enviar la transacción es la siguiente:

```
Retrieving transaction
executing transaction

Executing IntroduceInRepository GNU.EDIT/lyx-0.10.7 for sun4x_55

Coherence checks on transaction...

WARNING in GNU.EDIT/lyx-0.10.7:
    share has not been introduced in repository but present on scratch
WARNING in GNU.EDIT/lyx-0.10.7:
    those archs are present on scratch, but have not been introduced
    into repository: i386_linux2

transaction is OK, 2 warnings found
```

```
Storing transaction
ASIStm: Terminated successfully
```

Las comprobaciones de coherencia indican que nos hemos olvidado de la parte compartida (share) y de una plataforma, que están presentes en el repositorio auxiliar (scratch).

Más adelante deseamos pasar la aplicación al área de producción en una transacción:

```
Certify GNU.EDIT/lyx-0.10.7 (share,sun4x_55,i386_linux2)
IntroduceInProduction GNU.EDIT/lyx-0.10.7 (sun4x_55,i386_linux2)
```

El resultado de ejecución es:

```
Retrieving transaction
executing transaction

Executing Certify GNU.EDIT/lyx-0.10.7 for i386_linux2,share,sun4x_55

Executing IntroduceInProduction GNU.EDIT/lyx-0.10.7 for i386_linux2,sun4x_55

Coherence checks on transaction...

WARNING in GNU.EDIT/lyx-0.10.7:
    states of those archs are higher than share(Certified):
    sun4x_55(InProduction), i386_linux2(InProduction)

WARNING GNU.EDIT/lyx-0.10.7:
    share is alone on state Certified

Dependency checks on transaction...

WARNING in GNU.EDIT/lyx-0.10.7:
     no dependencies defined

transaction is OK, 3 warnings found

Storing transaction
ASIStm: Terminated successfully
```

En este caso, se ha olvidado pasar la parte compartida al estado de "En Producción", además no se han definido las dependencias para la aplicación.

Supongamos que queremos introducir una nueva versión de la aplicación en producción, que ya se encuentra en estado "Certificado":

```
IntroduceInProduction GNU.EDIT/lyx-0.12.0 (share,sun4x_55,i386_linux2)
```

Obtenemos el siguiente resultado:

```
Retrieving transaction
executing transaction

Executing IntroduceInProduction GNU.EDIT/lyx-0.12.0 for i386_linux2,share,sun4x_55

ERROR in GNU.EDIT/lyx-0.12.0:
     file InProduction/i386_linux2/usr.local/bin/lyx
     does already exist
     Failure performing transition IntroduceInProduction for i386_linux2

Failure during transaction
transaction has been REJECTED, errors found
ASIStm: Terminated with errors
```

Dado que ambas versiones comparten un nombre de fichero en el mismo directorio (`/usr/local/bin/lyx`), se produce un conflicto lo cual lleva al rechazo de la transacción. Cabe recordar que este conflicto se ha detectado *virtualmente*. Dado que no se ha ejecutado aún el gestor de copia local sobre ninguna de las transacciones validadas, todavía no existe físicamente el fichero "conflictivo".

La transacción correcta a ejecutar será

```
RemoveFromProduction GNU.EDIT/lyx-0.10.7 (share,sun4x_55,i386_linux2)
IntroduceInProduction GNU.EDIT/lyx-0.12.0 (share,sun4x_55,i386_linux2)
```

que no producirá ningún conflicto.

Si ahora ejecutamos el gestor de copia local `ASISlcm` sobre el conjunto de transacciones validadas, el resultado de ejecución es el siguiente:

```
ASISlcm: Started Fri May  8 14:57:59 1998
Updating pending transaction list
Processing transactions

Reducing transactions...

executing transactions

Executing IntroduceInRepository GNU.EDIT/lyx-0.10.7 for i386_linux2,share,sun4x_55

Executing Certify GNU.EDIT/lyx-0.10.7 for i386_linux2,share,sun4x_55

Executing IntroduceInRepository GNU.EDIT/lyx-0.12.0 for i386_linux2,share,sun4x_55

Executing Certify GNU.EDIT/lyx-0.12.0 for i386_linux2,share,sun4x_55

Executing IntroduceInProduction GNU.EDIT/lyx-0.12.0 for i386_linux2,share,sun4x_55
```

```
Writing current date into last update file
Finished Fri May  8 15:01:10 1998
Sending mail...
ASISlcm: Terminated successfully
```

A partir de este momento, el repositorio ha sido actualizado, y los usuarios han sido notificados.

sistema de dependencias

Supongamos que definimos con `ASISwdep` las siguientes dependencias generales para la aplicación GNU.EDIT/lyx-0.10.7:

```
REQUIRES:
X11/xforms-(0.81,0.92)
VP:tex_base-1
FG:ps_visualizer-2 OR X[E]:ghostview OR X[E]:gv
TeX/dvips-5.58e OR X[E]:dvips

RECOMMENDS:
CERN/xprint-(2.1,2.3,2.5,2.6)
GNU.EDIT/ispell-3.1.20 OR X[E]:ispell OR X[E]:spell

CONFLICTS:
MISC/xemacs-19.14
```

El grupo funcional `ps_visualizer-2` se compone de:

```
GNU.MISC/ghostview-(1.5.2,1.91)
GNU.MISC/gv-(2.7b5,2.7.6,3.4.3,3.5.2)
```

y el paquete virtual `tex_base-1`:

```
TeX/teTeX_base-0.4.1
TeX/teTeX_bin-0.4.1
TeX/teTeX_tex_latex-0.4.1
```

Si ahora queremos comprobar la coherencia de la configuración de una estación de trabajo, utilizaremos `ASISdep`, que ofrece, entre otras, las siguientes opciones:

```
--check_wsconfig
        checks workstation config coherence (pakfile)
--pakfiledir <dir> (default /usr/local/adm)
        directory where to find ASISwsm.pak file
```

```
--check_repository {Certified|InProduction}
        checks repository coherence in given area
--products <F/P-V,F/P-V,...>
        check coherence only on this product version(s)
```

Comprobemos la coherencia de nuestra estación de trabajo, sólamente sobre el paquete `GNU.EDIT/lyx-0.10.7`:

```
ASISdep --check_wsconfig --products GNU.EDIT/lyx-0.10.7
```

El resultado es:

```
ASISdep: checking ASISwsm cfg file in /usr/local/adm:

ERROR in GNU.EDIT/lyx-0.10.7: required
    TeX/dvips-5.58e or executable 'dvips'
    but not found

WARNING in GNU.EDIT/lyx-0.10.7:
    conflict detected with MISC/xemacs-19.14

ASISdep: dependency check completed, 1 errors, 1 warnings found
```

No se cumplen la dependencia establecida hacia `TeX/dvips-5.58e` o el ejecutable `dvips`, y se detecta un conflicto debido a que `MISC/xemacs-19.34` forma parte de la configuración actual. Todas las demás dependencias se verifican.

Comprobemos ahora que pasa si se modifica el estado de una aplicación sobre la cual se establece una dependencia por parte de otra. En el ejemplo anterior, si retiramos de producción X11/xforms-0.81, obtenemos:

```
Retrieving transaction
executing transaction

Executing RemoveFromProduction X11/xforms-0.81 for i386_linux2,share,sun4x_55

Coherence checks on transaction...

WARNING Product X11/xforms:
    has now no InProduction version for archs:
    i386_linux2, sun4x_55

Dependency checks on transaction...

WARNING in X11/xforms-0.81:
    dependency broken for GNU.EDIT/lyx-0.10.7 type: REQUIRES
    affected archs:
    i386_linux2(InProduction), sun4x_55(InProduction)

transaction is OK, 2 warnings found
```

```
Storing transaction
ASIStm: Terminated successfully
```

Esto se debe a que GNU.EDIT/lyx-0.10.7 depende de esta aplicación, y se encuentra en el área de producción, al igual que X11/xforms-0.81. Si ésta última es retirada del área de producción (sin ser sustituida por una nueva versión), entonces se incumple la autocontención del área de producción debido a la aparición de dependencias violadas.

Otra comprobación de integridad es la efectuada con los paquetes virtuales. Así, si intentamos retirar del repositorio componentes de VP:tex_base-1 sin haberlos eliminado de este paquete, obtenemos:

```
...
Executing Delete TeX_teTeX/bin-0.4.1 for i386_linux2,share,sun4x_55
...
WARNING in TeX/teTeX_bin-0.4.1:
    still member of group VP:tex_base-1.0
    affected archs:
    i386_linux2(Deleted), sun4x_55(Deleted)
...
```

Bibliografía

[1] NFS: The networked file system. Technical report, Sun Microsystems, Inc., 1985.

[2] The AFS file system in distributed computing environments. Technical report, Transarc Corporation, 1996.

[3] Paul Anderson. Managing program binaries in a heterogeneous unix network. In *LISA V Proceedings*, pages 1–9. Usenix, 1990.

[4] M. Bach. *The Design of the UNIX Operating System.* Prentice Hall, 1986.

[5] Brian Berliner. CVS: Parallelizing software development. Technical report, Prisma, Inc., 1994.

[6] Shirley Browne, Jack Dongarra, Eric Grosse, et al. NetLib services and resources. Technical report, National Science Foundation, ATT Bell Labs, 1997.

[7] Ronald R. Cooke. *Network Centric Computing: A Survival Strategy for Enterprise CIOs.* Network Client Business Group (NCBG), Octubre 1997.

[8] Attachmate Corporation. *NetWizard Product Profile.* Attachmate Website, http://www.attachmate.com, 1997.

[9] Microsoft Corporation. *NetPC specifications document.* Microsoft Website, http://www.microsoft.com, 1997.

[10] Microsoft Corporation. *Microsoft SMS user's manual.* Microsoft Website, http://www.microsoft.com, 1998.

[11] Susan Dart. Spectrum of functionality in configuration management systems. Technical report, SEI Software Engineering Institute, Carnegie Mellon University, Diciembre 1990.

[12] Philippe Defert, Germán Cancio, Eusebio Fernández, Stéphane Gouache, et al. ASIS: Product maintainer's guide. Technical report, CERN - IT Division, Genève, 1997.

[13] Philippe Defert, Germán Cancio, Stéphane Gouache, et al. ASIS: User's and reference guide. Technical report, CERN - IT Division, Genève, 1997.

[14] Free Software Foundation, Boston. *GNU General Public License*, 1991.

[15] N. Friedman, D. Mackenzie, et al. *Autoconf: Generating automatic configuration scripts*. Free Software Foundation, Abril 1994.

[16] Bob Glickstein. *Stow: Managing the installation of software packages*, 1996.

[17] Stéphane Gouache. Designing a security environment for the ASIS system. Technical report, CERN - IT Division, Genève, 1997.

[18] Per Hagen and Alberto Pace. The Windows 95 and NT administration guide at CERN. Technical report, CERN IT-DCI, Genève, Marzo 1997.

[19] A. Heavey, L. Carpenter, E. Berman, et al. UPS / UPD Product Methodology at Fermilab. Technical report, Fermi National Accelerator Laboratory, EE.UU., Agosto 1997.

[20] J. Herman, T. Forbath, and C. Leighton. *Investing in Desktop Management Productivity, white paper*. Northeast Consulting Resources,Inc., Agosto 1997.

[21] André Hoek, Richard Hall, Dennis Heimbigner, et al. Software Release Management. In *6th European Software Engineering Conference, LNCS 1301, Berlin*. Springer, 1997.

[22] Ian Jackson. Debian Dpkg programmer's manual. Technical report, Debian GNU/Linux project team, Mayo 1997.

[23] David Lebel, Duncan Fraser, Michel Dagenais, et al. *LUDE: Logithéque Universitaire Distribuée et Extensible*. Université de Montreal, Canada, Octubre 1995.

[24] Kenneth Manheimer, Barry Warsaw, Stephen Clark, and Walter Rowe. The Depot: A framework for sharing software installations across UNIX platforms. In *LISA VII Proceedings*, pages 37–46. Usenix, 1990.

[25] C. Mazza, J. Fairclough, B. Melton, and D. De Pablo. *Guide to applying the ESA software engineering standards to small software projects*. ESA-ESTEC. European Space Agency, Mayo 1996.

[26] C. Mazza, J. Fairclough, B. Melton, D. De Pablo, et al. *Software Engineering Standards*. ESA-ESTEC. European Space Agency, 1991.

[27] Microsoft Corporation, Redmont. *The Zero Administration Initiative for Windows*, Marzo 1997.

[28] Sun Microsystems. *JavaStation white paper*. Sun Website, http://www.sun.com, 1997.

[29] Yvan Pannatier. Test du logiciel SMS. Technical report, Centre informatique, Université de Lausanne, 1996.

[30] James Rumbaugh, M. Blaha, W. Premerlani, et al. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[31] John Sellens. Software maintenance in a campus environment: the Xhier approach. In *LISA VI Proceedings*, pages 21–44. Usenix, 1991.

[32] Steven Shafer and Mary Thompson. The SUP Software Upgrade Protocol. Technical report, Carnegie Mellon University, 1990.

[33] Andrew Tanenbaum. *Operating Systems: Design and Implementation*. Prentice Hall, 1987.

[34] Andrew Tanenbaum. *Modern Operating Systems*. Prentice Hall, 1992.

[35] Rainer Többicke and Lionel Cons. SUE: Standardized Unix Environment. Technical report, Unix Workstation Support, CERN IT Division, Genève, Mayo 1995.

[36] Rainer Többicke and Philippe Defert. Recommended standard for UNIX workstation environment setup. Technical report, CERN IT, Genève, 1990.

[37] Larry Wall and Randal Schwartz. *Programming Perl*. O'Reilly and Associates, Inc., 1994.

[38] Brent B. Welch. *Practical Programming in Tcl and Tk*. Prentice Hall, 1995.