UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

INGENIERÍA

EN INFORMÁTICA

PROYECTO FIN DE CARRERA

Sistema GRID para el Render de Escenas 3D Distribuido: YAFRID

José Antonio Fernández Sorribes

Julio, 2006

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Departamento de Informática

PROYECTO FIN DE CARRERA

Sistema GRID para el Render de Escenas 3D Distribuido: YAFRID

Autor: José Antonio Fernández Sorribes
Director: Carlos González Morcillo

Julio, 2006

Este documento ha sido compuesto con LaTeX. Las figuras que contiene han sido en su mayoría creadas con OpenOffice y El GIMP y los diagramas UML con ArgoUML y Umbrello.

Las figuras que aparecen en la introducción a sistemas distribuidos (Apartado 3.5) han sido sacadas de [Moy05].

Las imágenes de los dragones de la Figura 3.5 han sido cedidas por Carlos González.

TRIBUNAL:

Presidente:
Vocal:
Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	VOCAL	SECRETARIO
Fdo.:	Fdo.:	Fdo.:

Resumen

El último paso en el proceso para la generación de imágenes y animaciones 3D por ordenador es el llamado render. En esta fase se genera una imagen bidimensional (o un conjunto de imágenes en el caso de las animaciones) a partir de la descripción de una escena 3D. Para la obtención de imágenes fotorrealistas se utilizan algoritmos computacionalmente exigentes como el trazado de rayos.

En los proyectos relacionados con la síntesis de imágenes, la etapa de render se suele considerar el cuello de botella debido al tiempo necesario para llevarla a cabo.

Generalmente, el problema se resuelve usando granjas de render que pertenecen a empresas del sector en las que los frames de una animación se distribuyen en distintos ordenadores. El presente proyecto ofrece una alternativa para acometer el render tanto de animaciones como de imágenes basada en computación grid. Para ello, se ha desarrollado un sistema en el que tienen cabida ordenadores heterogéneos tanto en software como en hardware, distribuidos geográficamente y conectados al grid vía Internet.

Abstract

Rendering is the last step of the 3D image synthesis process. This phase consists in generating a bi-dimensional image (or a set of them in animations) from a formal description of a 3D scene.

In order to obtain realistic results, it is necessary to use computationally intensive algorithms like raytracing. Due to the time that those algorithms require, the rendering is often considered to be the bottleneck of the projects related to realistic image synthesis.

A common solution to this problem consists in spread the render of animations among a large number of computers where each one process one single frame. Those sets of computers are often owned by a single organization and are referred as render farms.

This project offers a different approach to solve the problem of rendering both images and animations, an approach based on grid computing. A system like this one can be composed by heterogeneous, in terms of hardware and software, and geographically distributed computers. The only requirement is having an Internet connection.

A mis padres, mi hermana y mi abuela,
por estar siempre ahí aguantándome y
porque a ellos les dedico todo cuanto hago.

Agradecimientos

A Carlos por sus ánimos, su entrega y por tener más fe en mí a veces de la que incluso yo mismo tenía.

A mis padres por la revisión de este documento.

A Raúl por echarme una mano en la implantación del sistema.

A todos los que de una u otra manera han tenido algo que ver con el resultado final.

Índice general

Índice de figuras

Índice de cuadros

Índice de listados

Índice de algoritmos

Terminología

Capa. Término que se usa tanto para nombrar los distintos niveles de una arquitectura multicapa (persistencia, dominio, presentación, etc, ...) como para identificar los niveles de abstracción en la arquitectura de Yafrid (capa de recursos, de servicios, etc, ...).

Cluster. Conjunto de computadores pertenecientes a una misma entidad e interconectados mediante redes de alta velocidad que trabajan como uno solo. Se ha mantenido el original.

Frame. Aunque algunos autores utilizan en castellano marco o fotograma, se ha preferido mantener el original para las imágenes estáticas porque se trata de un término extendido. Se usará fotograma en el caso de las animaciones.

Irradiance cache. Estructura de datos que usan algunos métodos de render para ahorrar cálculos. Se mantiene sin traducir tal y como hacen la mayoría de los autores.

Mapeado de fotones. Traducción del inglés *Photon Mapping*. Se usarán ambos indistintamente.

Middleware. Software que oculta la complejidad de las redes. Se mantiene el original al carecer de traducción aceptada.

Modulo. Se usa para hacer referencia a una parte del sistema como para nombrar un fichero escrito en Python. Para éstos últimos se usará *módulo Python*.

Pathtracing (Método de render). Este término se ha mantenido en su versión original porque la traducción al castellano, *trazado de caminos*, es un término poco utilizado.

Peer-to-peer. Modelo de comunicación en el que todos los participantes poseen el mismo papel y establecen relaciones de igual a igual. Se ha mantenido el término original por estar muy extendido.

Pool. Término utilizado para hacer referencia a aquellos directorios que representan un suministro de cierto tipo de entidades y cuyo uso es compartido. Por ejemplo: *Pool de unidades de trabajo*.

Proveedor. Se utiliza para el usuario que desempeña este papel, para el software y para el módulo dentro de la arquitectura. Cuando se puedan dar equívocos se especificará a cuál se hace referencia. Lo habitual será utilizarlo con mayúscula cuando se haga referencia al módulo o al software (lo mismo ocurre con *Servidor* y *Distribuidor*).

Render. Proceso de generar una imagen 2D a partir de una escena 3D. Es un término aceptado entre autores de habla hispana y que carece de traducción.

Trazado de rayos (Método de render). Traducción del término *raytracing*. En el texto se usan indistintamente ambos ya que término raytracing es común en artículos en castellano.

Unidad de trabajo. Traducción del término *workunit*. Es la unidad mínima de distribución. En el caso del render de imágenes consiste en fragmentos de las mismas. En el caso de las animaciones consiste en un frame.

Notación

- Las referencias bibliográficas aparecen en la sección de *Bibliografía* ordenadas según el primer apellido del primer autor.

  Cuando se citan en el texto, las referencias aparecen encerradas entre corchetes. Se utiliza el siguiente convenio:

  - Cuando el **autor** es **único**, se usan las tres primeras letras del primer apellido, la primera en mayúscula y el resto en minúscula. El conjunto va seguido de las dos cifras correspondientes al año de edición. Un ejemplo puede ser [Lar99].
  - En caso de haber **dos, tres o cuatro autores**, se utilizan las iniciales de sus apellidos seguidas del año, como en [RS02], [FKT02] y [LSSH03].
  - Si hay **más de cuatro autores**, aparecen las iniciales de los tres primeros seguidas del carácter + y el año, como en [LMV+04].

- Las páginas web a las que se hace referencia en el texto aparecen ordenadas en la sección *Referencias web*. Se hace referencia a ellas con el número correspondiente encerrado entre corchetes.

- Las figuras, cuadros, etcétera se referencian con Figura, Cuadro, etcétera, seguidos del número del capítulo y el orden del elemento dentro del mismo, separados por un punto.

- Los diagramas que aparecen están destinados a aclarar el texto al que acompañan y están, habitualmente, simplificados. Pueden omitirse algunos atributos o métodos en una clase e incluso pueden omitirse clases que no son imprescindibles en el ejemplo que tratan de esclarecer.

- Las imágenes se han generado a una calidad de 300 píxeles por pulgada siempre y cuando ha sido posible.

Capítulo 1

INTRODUCCIÓN

En el año 1995 se estrenó **Toy Story** de *Pixar Animation Studios* [25] que fue el primer largometraje íntegramente realizado por ordenador, lo que constituyó un hito en la historia de la animación por computador. Tras este comienzo, raro es el año en el que no se estrena una película de este tipo y las distintas compañías del sector compiten por dotar a sus creaciones de mayor realismo y espectacularidad.

Éste es quizá el aspecto más comercial de un área que está en **continua evolución** con el desarrollo de nuevas herramientas, lenguajes y algoritmos. Sin embargo, no es el único ámbito donde la animación por ordenador y la infografía están siendo utilizadas.

Figura 1.1: Woody y Buzz, los dos protagonistas de Toy Story.

Son numerosas las **producciones cinematográficas** en las que los efectos realizados por ordenador se utilizan con profusión, hasta el punto de que algunas películas parecen simples escaparates para dar a conocer lo que en este campo se puede llegar a conseguir.

Estas técnicas también se usan en **televisión**, introducciones de videojuegos e incluso en **publicidad**, donde el ingenio de artistas y animadores se pone al servicio de incrementar las ventas del producto de moda. Se utilizan también para realizar **simulaciones** de todo tipo, desde aquellas que utilizan los científicos en sus investigaciones y que les ayudan a conocer mejor las leyes que rigen el universo hasta las simulaciones de vuelos militares, pasando por aquellas cuyo único fin es servir de entretenimiento. Últimamente se están llevando a cabo proyectos que tratan de **recrear monumentos** o edificios emblemáticos para realizar **visitas virtuales** o incluso preservar esas obras de forma virtual en caso de que las reales fueran dañadas por algún motivo.

Como se puede observar, es un hecho indiscutible que la generación de gráficos en 3D es un campo en auge en la actualidad y que las áreas que pueden beneficiarse de su uso son muy numerosas y variopintas.

El proceso de síntesis de imagen 3D

La síntesis de imágenes 3D es un proceso que abarca una serie de etapas:

- **Modelado:** Consistente en la construcción de la geometría de los objetos de la escena. La técnica utilizada para representar estos objetos depende del fin para el que están destinados.
- **Texturizado:** Es la fase en la cual se asignan propiedades a los objetos de la escena. Se definen las texturas que se aplicarán, las propiedades de reflexión y refracción de los materiales, etc ...
- **Iluminación:** Se definen las fuentes de luz de la escena. Éstas vienen definidas por propiedades como la orientación, la intensidad o el tono, además de por su tipo. Las fuentes de luz pueden comportarse como puntos que arrojan luz en todas direcciones, como focos, como áreas o incluso como volúmenes.
- **Animación:** Fase que consiste en definir cómo cambian unas determinadas propiedades—por lo general posición, orientación y escala—de un objeto a lo largo de una línea de tiempo. No aparece en los proyectos en los que la salida es una sola imagen estática.

- **Render:** Consiste en la generación de una imagen bidimensional a partir de la descripción de la geometría de la escena junto con las definiciones de las luces, de la cámara y de los materiales.

La etapa de render y su problemática

El último paso del proceso de síntesis de imagen 3D, el de render, introduce una cuestión que se hace crítica especialmente en proyectos grandes. Para generar una imagen en dos dimensiones a partir de una descripción tridimensional de una escena se pueden utilizar diversas técnicas.

Para generar imágenes fotorrealistas—las más demandadas—se utilizan técnicas denominadas de **iluminación global** que tratan de simular la luz dispersa en un modelo 3D ateniéndose a las leyes físicas. Estas técnicas calculan el valor de intensidad en cada punto de la escena teniendo en cuenta las interacciones de la luz con los objetos de la escena.

Figura 1.2: Imagen de prueba (Yafray 0.0.7).

El problema radica en que los algoritmos utilizados para generar imágenes fotorrealistas son **computacionalmente muy costosos**. Este problema es aún mayor cuando la geometría de la escena es compleja—un gran número de polígonos—o cuando se desean obtener determinados efectos como las *cáusticas*.

Para ilustrar la problemática que supone el tiempo que requiere la fase de render en cualquier proyecto relacionado con la síntesis de imagen 3D se ha realizado una sencilla prueba.

En el Cuadro 1.1 se muestran los tiempos empleados en renderizar una misma escena de prueba de poca complejidad (ver Figura 1.2) en distintas máquinas. La imagen resultado que se ha obtenido es de resolución aproximada PAL (720x576). El motor de render utilizado ha

sido *Yafray* [37], uno de los motores libres más conocidos, utilizando una de las técnicas que obtienen resultados fotorrealistas, el *pathtracing*.

A la vista de los datos y tomando el caso más favorable, se pueden establecer una serie de conclusiones. Para dar la sensación de movimiento, las imágenes se tendrían que mostrar a una frecuencia de 25 fotogramas por segundo ante el ojo del espectador. De este modo, para un cortometraje de dos minutos de duración habría que obtener 3000 fotogramas, en cuyo render se invertirían 57000 minutos usando el primero de los equipos del Cuadro 1.1 o lo que es lo mismo . . . *39 días y 14 horas!*

Este problema es aún más crítico teniendo en cuenta que las imágenes de cualquier producción actual de animación son mucho más complejas que la usada como prueba con lo que en realidad el tiempo de render sería muy superior. Como ya se ha apuntado anteriormente, y la prueba que se ha llevado a cabo reafirma, el cuello de botella de este tipo proyectos es claramente el tiempo de procesador.

Procesador	Memoria RAM	SO	Tiempo
Intel Pentium4 2.8 GHz	512 Mb	GNU/Linux Debian	19 m
Intel PIII 667 MHz	256 Mb	GNU/Linux Debian	64 m
MIPS R12000	5 Gb compartida	SGI IRIX 6.5	78 m
AMD Athlon 1 GHz	320 Mb	MS Windows XP	39 m
Intel Pentium4 1.8 GHz	512 Mb	MS Windows XP	38 m

Cuadro 1.1: Tiempos empleados en renderizar una misma imagen de prueba.

Alternativas

La solución clásica para este problema es la de distribuir el render en granjas de ordenadores que están dedicados exclusivamente a obtener frames de animaciones, por lo que reciben el nombre de **granjas de render**. Así, las grandes compañías del sector dedican importantes recursos económicos a mantener estos ordenadores para obtener sus producciones. Desde un punto de vista técnico, estas granjas constituyen un *cluster*, es decir, un conjunto de ordenadores independientes interconectados y pertenecientes a una misma compañía.

Otra de las líneas de investigación que está tomando bastante importancia en los últimos

años consiste en la utilización de granjas de GPU[1]s para acometer tareas de render. La ventajas de éstas últimas con respecto a los habituales *clusters* de ordenadores radica en que es posible conseguir un muy buen rendimiento a un menor coste.

1.1. ENFOQUE DEL PROYECTO

Lo que con este proyecto se pretende es utilizar una aproximación distinta para resolver esta problemática, la que propone la **computación grid**.

La computación grid, un campo en auge en los últimos años, consiste en que una serie de **ordenadores heterogéneos**, tanto en software como en hardware, y **geográficamente distribuidos** sirvan a un mismo propósito como si de **una única maquina** se tratase.

De este modo, el sistema no estará formado por un conjunto de ordenadores pertenecientes a una compañía o universidad. Los componentes del grid podrán ser ordenadores con distintas arquitecturas o sistemas operativos localizados en cualquier lugar del mundo y que posean una conexión a Internet.

Con el proyecto Yafrid se pretende desarrollar un sistema que, sustentado por una arquitectura multisistema, tome ventaja de las características de los grids computacionales aplicadas al render de escenas 3D.

Algunas de las características más importantes del proyecto Yafrid se enumeran a continuación:

- Es **distribuido** lo que permite que el render de una escena se divida entre distintos ordenadores.
- Es **heterogéneo** tanto a nivel de hardware (arquitecturas Intel, SPARC, PowerPC, etc ...) como de software (sistemas operativos MS Windows, GNU/Linux, Solaris, Mac OSX, etc ...).
- Es **independiente del motor de render** y permite fácilmente añadir nuevos motores a los que se ofrecen inicialmente.
- Posee una **interfaz web** que permite a un usuario conocer el progreso de los trabajos que envío al grid en todo momento y desde cualquier parte mediante un navegador web.
- Ofrece una **granularidad variable**. A diferencia de la granjas de render, Yafrid permite distribuir tanto el render de una animación como el de un sólo frame.

[1]Del inglés **G**raphics **P**rocessing **U**nit, es decir, Unidad de Procesamiento Gráfico.

1.2. ESTRUCTURA DE ESTE DOCUMENTO

Este documento se compone de siete capítulos y seis anexos cuyo fin se describe a continuación:

- **Capítulo 1.** Introducción a la problemática que supone la fase de render en los proyectos relacionados con la síntesis de imagen 3D. Enfoque del trabajo abordado y estructura del presente documento.
- **Capítulo 2.** Objetivos, tanto generales como específicos, que se persiguen con la realización de este proyecto.
- **Capítulo 3.** Antecedentes, estado de la cuestión. Estudio de aquellas áreas que guardan relación con el proyecto.
- **Capítulo 4.** Método de trabajo. Se describe el ciclo de vida y la metodología utilizados en el desarrollo del sistema. En un primer momento se presenta también, de forma general, la arquitectura de Yafrid. Más adelante, los distintos componentes del sistema, así como las soluciones a los problemas que se han ido presentando, se tratan con mayor profundidad.
- **Capítulo 5.** Estudio de los resultados obtenidos por el sistema así como de los teóricamente esperables. Estudio de costes de implantación y análisis de posibles alternativas para recuperar la inversión.
- **Capítulo 6.** Conclusiones acerca del sistema con respecto a los objetivos perseguidos. Posibles líneas futuras de trabajo que incluyen mejoras en las actuales funcionalidades y desarrollo de otras nuevas.
- **Capítulo 7.** Bibliografía y referencias web utilizadas para la realización de este proyecto.
- ***ANEXO A.*** Diagramas de casos de uso, de clases, relacionales y de componentes y despliegue.
- ***ANEXOS B, C y D.*** Manuales de Usuario del Proveedor, el Cliente y el Administrador de Yafrid.
- ***ANEXO E.*** Manual de instalación de Yafrid Server 0.0.2.
- ***ANEXO F.*** CD adjunto y estructura del mismo.

Capítulo 2

OBJETIVOS DEL PROYECTO

El objetivo principal del presente proyecto es el de construir un prototipo funcional de un sistema de render distribuido basado en computación grid optimizando el tiempo necesario para llevar a cabo el propio proceso de render.

Además de este objetivo fundamental, existe una serie de objetivos complementarios que son descritos a continuación.

Arquitectura multiplataforma Este sistema debe estar desarrollado sobre una arquitectura que permita su funcionamiento en distintos sistemas operativos y con independencia del hardware. Esto permitirá tener un grid heterogéneo en todos los sentidos ya que cualquier ordenador con conexión a internet es un potencial proveedor de servicio. Es en este aspecto donde se aleja de los clásicos clusters de ordenadores.

Independencia del motor de render Aunque el sistema se orientará a su utilización con dos motores de render en concreto (Yafray y el que posee Blender 3D), el desarrollo se debe realizar de forma que sea posible añadir nuevos motores de render sin excesivo esfuerzo.

Varias granularidades El sistema desarrollado debe permitir la distribución del render tanto de imágenes como de animaciones completas. Las unidades mínimas de distribución, denominadas unidades de trabajo, serán el fragmento de frame y el frame respectivamente.

Interfaz web La mayor parte de la funcionalidad del sistema debe ser controlable vía web. Esto aporta una ventaja clave para ser realmente utilizado por la comunidad de usuarios ya que no es necesario instalar ningún software, cualquier navegador es suficiente.

Grupos de Usuarios Se debe poder permitir la creación de grupos de usuarios (clientes y proveedores) con el fin de ajustar qué proveedores atienden a qué proyectos. Mediante esta utilidad se pueden establecer subgrupos privados de proveedores que proporcionan servicio a los proyectos de determinados clientes.

Prioridades Se establecerá además una sistema de prioridades de los proveedores y los clientes con respecto a los grupos a los que pertenecen.

Diseño Un adecuado diseño del sistema permitirá que en un futuro se añadan nuevas funcionalidades o se mejoren las existentes de una forma sencilla. El uso de una división en capas y patrones de diseño así como la existencia de una completa documentación facilitarán futuros desarrollos.

Usabilidad Tanto el software del proveedor como el interfaz web que son los dos medios principales a través de los cuales el usuario interactúa con el sistema deben ser fáciles de usar. Se tratará de realizar interfaces cómodas e intuitivas.

Instalación sencilla de proveedor El software necesario para que un usuario se convierta en proveedor y pase a formar parte del grid debe ser sencillo de instalar y configurar en las distintas plataformas disponibles. Se intentará reducir el tiempo dedicado a estas dos tareas al mínimo posible.

Inicio automático A fin de que colaborar con el grid requiera el menor esfuerzo posible, se ha de configurar el software de modo que al iniciar el sistema, se inicie el software para que el usuario no tenga que estar pendiente.

Capítulo 3

ANTECEDENTES, ESTADO DE LA CUESTIÓN

3.1. COMPUTACIÓN GRID

Básicamente un grid es una colección de equipos **heterogéneos** tanto a nivel de hardware como de software que están **interconectados** de forma que **operan como uno solo** que tiene unas **mayores capacidades** de memoria, procesamiento y de almacenamiento [FK98].

El término grid proviene de la analogía [Buy02] que se estableció en los primeros años de la computación grid entre ésta y la **red eléctrica**. El parecido está en que ambas tecnologías ocultan el origen de los recursos que ofrecen. En el caso de la red eléctrica, el usuario final no conoce de dónde proviene la energía que se le suministra, ni siquiera el origen de la misma (el tipo de central que la generó).

A lo largo de los años, la computación grid ha ido saliendo de los grandes laboratorios de investigación para acercarse más al gran público en una evolución similar a la que tuvo **Internet**. Hoy en día, la computación grid es un campo en auge tal y como demuestra la gran multitud de trabajos de investigación y proyectos que se están llevando a cabo actualmente. El número de aplicaciones para las que una aproximación tipo grid puede ser útil crece cada día. Incluso el **MIT** posicionó esta tecnología como uno de los diez avances tecnológicos más importantes [16].

Una vez introducido el concepto de computación grid, en las siguientes secciones se pasa a analizar las características de este tipo de sistemas con más profundidad.

3.1.1. Definición de grid

Existen numerosas definiciones de lo que es un sistema de tipo grid pero todas ellas coinciden en los mismos puntos esenciales. Entender el concepto de grid es especialmente importante para entender las diferencias y similitudes entre este tipo de sistemas y otros de similares características (ver Apartado 3.1.4).

Es en 1998 cuando el término de *computación grid* hace su aparición de la mano de Carl Kesselman e Ian Foster que dieron la siguiente definición [FK98]:

> Un grid computacional es una infraestructura hardware y software que proporciona acceso consistente a bajo coste a recursos computacionales de alto nivel.

La evolución de esta definición llevó a una lista de tres puntos propuesta por el propio Ian Foster [Fos02]. De acuerdo con esta lista, un grid es un sistema que:

1. coordina recursos que no están sujetos a un control centralizado ...

   (Un grid computacional integra y coordina recursos y usuarios que están en diferentes dominios, diferentes unidades administrativas de una misma organización o distintas entidades. Este modelo se ilustra frecuentemente usando el término de ***organización virtual*** *que es analizado con profundidad en [FKT02])*

2. ... usando estándares abiertos, protocolos de propósito general e interfaces ...

   (Un grid se sustenta sobre una serie de protocolos de propósito general e interfaces que están dirigidos fundamentalmente a la autentificación, autorización, descubrimiento de recursos y acceso a los mismos. Es importante que estos protocolos e interfaces estén ***estandarizados*** *y sean* ***abiertos****.)*

3. ... para proporcionar calidades de servicio no triviales.

   (Un grid computacional utilizará los recursos que lo constituyen, que deberán ser usados de una manera coordinada, para proporcionar distintas clases de calidad de servicio. Esta calidad de servicio atenderá a parámetros tales como el tiempo de respuesta, la disponibilidad, la seguridad y el rendimiento al procesar. Además, debe manejar correctamente la asignación de múltiples tipos de recursos para satisfacer exigencias complejas de los usuarios. De este modo, se pretende que la utilidad del sistema combinado sea perceptiblemente ***mayor que la suma de sus piezas****.)*

Otra definición de lo que es un grid computacional la aporta Rajkumar Buyya, destacado investigador en computación grid y en clusters de computadores, en un reciente artículo [BV05]. Buyya define un grid como *un tipo de sistema distribuido y paralelo que permite compartir, seleccionar y agregar servicios provenientes de recursos heterogéneos distribuidos a través de múltiples dominios administrativos basado en su disponibilidad, capacidad, eficiencia, coste y los requerimientos de calidad del servicio por parte del cliente.*

Organización virtual

El término **organización virtual**, que ya ha sido introducido, es un término clave en la computación grid. Una organización virtual está formada por aquellas instituciones independientes entre sí que comparten recursos computacionales para alcanzar un objetivo común.

Este acceso compartido a los recursos no se limita al intercambio de ficheros sino que incluye el acceso directo a ordenadores, a software, datos, etcétera. Esto hace necesario una

serie de estrategias colaborativas de planificación de recursos. El acceso a los recursos tiene que estar estrictamente controlado lo que implica que los proveedores y los consumidores de recursos deben definir de forma meticulosa lo que se va a compartir, quién está autorizado a compartir y las condiciones en las que se comparte. Son los individuos y organizaciones definidos por estas reglas los que forman la organización virtual.

Las organizaciones virtuales que se pueden establecer son muy variadas en cuanto a propósitos, alcance, tamaño, duración y estructura. Sin embargo, hay una serie de requisitos y características comunes a todos los casos. Algunos de estos puntos en común son la necesidad de unas relaciones flexibles de intercambio, que van desde la fórmula cliente-servidor al peer-to-peer, el control sobre los recursos o los mecanismos de planificación.

3.1.2. Clasificación de sistemas grid

Se han propuesto diversas clasificaciones de los sistemas grid atendiendo a distintas características. Una de estas clasificaciones, propuesta por IBM [Bro02], categoriza estos sistemas de acuerdo a su funcionalidad. Esta clasificación establece que hay dos variantes principales de sistemas grid, *computacional* y *de recursos*, además de uno que podría considerarse un híbrido de las dos configuraciones básicas, el de tipo *aplicación*.

- El sistema grid de tipo **computacional** está orientado a la resolución de problemas computacionalmente intensivos. Se usa para calcular ecuaciones o procesar tipos específicos de datos. Su objetivo principal es el de hacer uso de tiempo de CPU, y en ocasiones de la memoria de los componentes del grid, con el fin de incrementar la potencia computacional del conjunto.

  Típicamente, el modelo computacional de grid usa un sistema de cola en el que se van colocando los trabajos individuales que esperan ser procesados. De este modo, se tiene un sistema en el que el grid está constantemente trabajando aunque habitualmente de una manera no interactiva y donde cada trabajo puede ser obtenido horas o días después de ser colocado en la cola.

- Otro tipo de grid es el **de recursos**, cuyo objetivo consiste en el manejo y distribución de enormes cantidades de datos. Está caracterizado principalmente por estar orientado no a hacer algún cálculo como en el caso del grid computacional sino a almacenar información en varios equipos. Con esta distribución se busca bien mejorar la eficiencia de la recuperación de esta información o bien incrementar el espacio de almacenamiento potencial.

Estos tipos de sistemas son usados de un modo mucho más interactivo que los anteriores. Desde la perspectiva del cliente, el grid de recursos se usa directamente como si tratara de un único recurso. En un caso típico, el cliente envía al grid un documento para ser almacenado y es el sistema el que asigna una máquina determinada o un grupo de ellas para almacenar esa información.

- Hay un tercer tipo de sistema grid que se denomina híbrido o **de aplicación**. En lugar de proveer el acceso a un recurso específico dentro del grid, estos sistemas hacen uso de conceptos tanto de los grid computacionales como de los de recursos.

  Un ejemplo muy extendido de este tipo de sistema es un servidor de base de datos, en el que una petición de información de la base de datos es distribuida por varias máquinas. Usando esta aproximación, es posible incrementar la eficiencia global haciendo que las máquinas individuales devuelvan cada una sus propios registros resultado que deberán ser mezclados antes de ser devueltos al cliente que realizó la petición.

Otra de las clasificaciones propuestas [LMV+04] establece una distinción entre los sistemas grid en función de la naturaleza de los recursos compartidos que los forman. Estos recursos pueden ser **recursos computacionales** (como ciclos de CPU o capacidad de almacenamiento), **recursos de información** (bases de datos), **sensores** y otros **instrumentos** de medida, **dispositivos de visualización**, etcétera.

Reinefeld y Schintke [RS02] proponen otra clasificación y distinguen tres categorías de sistemas de tipo grid: el *grid de información*, el *grid de recursos* y el *grid de servicios*.

- El **grid de información** equivale a la actual WWW (World Wide Web) que proporciona información de cualquier tipo a cualquier lugar del mundo. La información se puede obtener de manera inmediata por medio de distintas infraestructuras de red, ordenadores personales, dispositivos móviles, etcétera. La compartición de ficheros también es un caso de este tipo de sistemas.

- El **grid de recursos** proporcina mecanismos para el uso coordinado de recursos como ordenadores, archivos de datos, servicios, aplicaciones o dispositivos especiales. Estos sistemas proporcionan acceso a los recursos sin que el usuario se tenga que preocupar del nombre, la localización u otros atributos de éstos. A diferencia de los datos que proporciona el grid de información, los recursos que ofrece este tiempo de sistema sólo son accesibles a usuarios autorizados.

- El **grid de servicio** ofrece servicios y aplicaciones con independencia de su localización, implementación, plataforma hardware, etc. Los servicios están establecidos sobre los recursos concretos que el grid de recursos hace disponibles. Mientras que el grid de recursos proporciona acceso a los recursos en concreto, el grid de servicio ofrece servicios abstractos e independientes de la localización.

3.1.3. Arquitectura OGSA

Para tratar de unificar criterios en la estandarización del grid, se creó la organización Global Grid Forum [14], cuyo principal estándar es la arquitectura *OGSA (Open Grid Services Architecture)* que fue propuesta inicialmente por Foster, Kesselman, Nick y Tuecke en [FKNT02].

OGSA es una especificación que trata de estandarizar el acceso a los servicios presentes en una infraestructura grid. Para ello, define un conjunto de interfaces que deben cumplir los servicios grid más comunes (servicio de gestión de trabajos, servicio de gestión de recursos, servicios de seguridad, etc).

Aunque existen diversas alternativas en cuanto al middleware de grid que utilizar, el que se ha erigido en pocos años como el estándar *de facto* es Globus Toolkit. Se distribuye bajo licencia GPL y ha sido desarrollado por la Globus Alliance [34] bajo la dirección de Ian Foster y Carl Kesselman.

3.1.3.1. Globus Toolkit

Globus Toolkit es una plataforma que implementa un conjunto de servicios de alto nivel necesarios para el desarrollo de aplicaciones grid. Estos servicios (que cumplen con las especificaciones de la OGSA) son, entre otros, los siguientes:

GSI (Grid Security Infrastructure). Proporciona métodos seguros para las comunicaciones entre distintas máquinas garantizando el adecuado intercambio de información entre los componentes del grid.

GRAM (Grid Resource Allocation Management). Proporciona la infraestructura necesaria para permitir la gestión tanto de trabajos como de recursos computacionales mientras que éstos progresan en el grid.

GRIP (Grid Resource Information Protocol). Proporciona una serie de servicios de información.

```
from pyGlobus.gramClient import GramClient,JOB_STATE_ALL
maquina_remota='mimaquina.oreto.inf-cr.uclm.es'
rsl='&(executable="/bin/echo")(arguments="Hello World")'
gramClient=GramClient()
jobContact=gramClient.submit_request(maquina_remota,rsl,JOB_STATE_ALL,'')
```

Listado 3.1: *Hola mundo* con pyGlobus.

GridFTP (Grid File Transfer Protocol). Permite el intercambio de datos entre componentes del grid.

En el Listado 3.1 se muestra un sencillo ejemplo que tiene la finalidad de servir de introducción a los distintos servicios que proporciona Globus para el desarrollo de aplicaciones grid.

El ejemplo hace uso de pyGlobus, el interfaz de Globus para programar en Python, para acceder al módulo GRAM de Globus. Para utilizar este servicio es necesario importar la clase de pyGlobus que sirve de interfaz al mismo. En concreto, esta clase es la *GramClient* incluida en el módulo *pyGlobus.gramClient*.

En el ejemplo, se trata de enviar un trabajo a una máquina del grid para que lleve a cabo una determinada tarea. La máquina a la que se le manda el trabajo corresponde con aquella cuyo nombre es el almacenado en la cadena *máquina_remota*. El trabajo a realizar en este caso consiste en sacar por pantalla el texto *"Hello World"*. Esto último se codifica usando un lenguaje que recibe el nombre de *RSL (Resource Specification Language)* y que fue desarrollado para Globus. De este modo, la cadena *'&(executable=/bin/echo")(arguments="Hello World")'* especificaría que la tarea a realizar es sacar por pantalla "Hello World". Hecho esto, para enviar la petición sólo será necesario crear una instancia de la clase GramClient sobre la que invocar el método *submit_request* con los parámetros adecuados.

3.1.4. Relación entre redes peer-to-peer, clusters y grid

La idea de grid tiene una estrecha relación con otros modelos de computación como son los sistemas distribuidos, los clusters o las redes peer-to-peer tal y como se desprende de [p2p02]. A continuación se apuntan las diferencias y similitudes más notables entre unos y otros.

Un **cluster**, al igual que un grid, está formado por múltiples nodos independientes interconectados, que trabajan juntos cooperativamente como un único recurso unificado. Sin embargo, al contrario que en el caso de los sistemas grid, los recursos pertenecientes a un

cluster son propiedad de una única organización, están conectados mediante redes de alta velocidad y son gestionados de manera centralizada. Los recursos computacionales de un cluster requieren proximidad física y homogeneidad de operación. Por el contrario, un grid puede estar compuesto de múltiples clusters junto con otro tipo de recursos heterogéneos geográficamente distribuidos.

La linea que separa una aplicación grid de una aplicación **peer-to-peer** es muy fina, tanto que algunas aplicaciones son consideradas por cada una de las dos comunidades como propia no llegando a ningún tipo de acuerdo al respecto [LSSH03].

Una aplicación peer-to-peer permite el intercambio de archivos al igual que los sistemas grid. Sin embargo, éstos últimos permiten compartir cualquier tipo de recurso además de ficheros. Además, los fines para los que están orientados son muy diferentes, lo que determina las características de los protocolos que los sustentan.

3.2. SÍNTESIS DE IMÁGENES 3D

La **Síntesis de Imagen** [Gon05] es la disciplina que se dedica al estudio y desarrollo de procesos que permitan sintetizar imágenes raster discretas a partir de modelos vectoriales 3D.

El proceso de síntesis 3D abarca diversas etapas como se puede observar en la Figura 3.1.

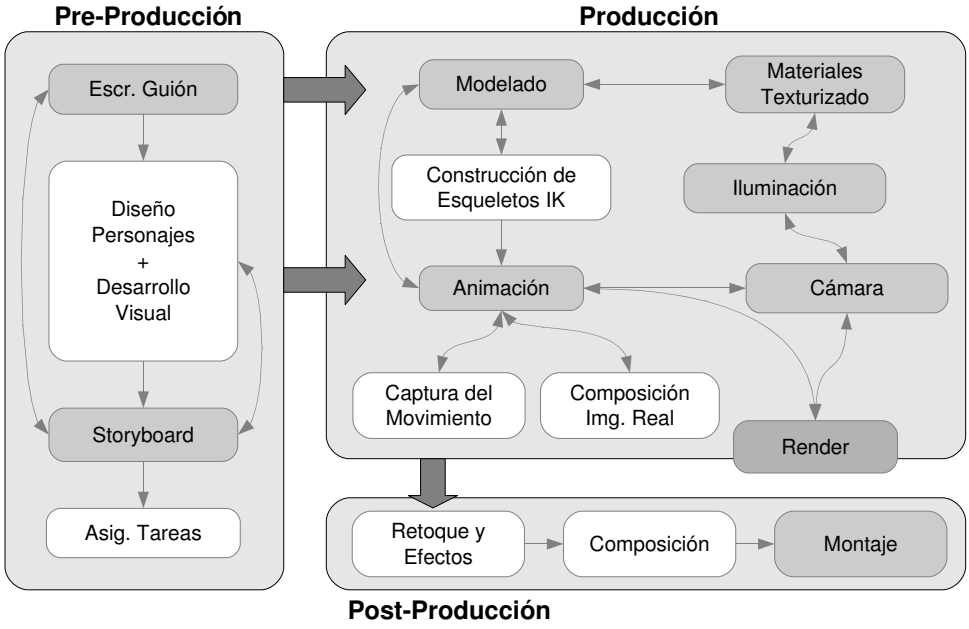

Figura 3.1: Proceso de síntesis 3D.

Las etapas del subproceso de producción son las más relacionadas con la informática gráfica. Los principales pasos para obtener una imagen sintética partiendo de unos bocetos de la escena son:

- **Modelado:** Consistente en la construcción de la geometría de los objetos de la escena. La técnica utilizada para representar estos objetos depende del fin para el que están destinados.

  Hay principalmente dos esquemas de representación:

  - *Modelado de Fronteras* (o *B-Rep*) que define la superficie que limita al objeto. En función de los datos que se almacenen del objeto, se habla de *modelado poligonal* (que almacena información sobre vértices, aristas y caras) o *modelado de superficies curvas* (en el que se guarda información sobre puntos de control, puntos de tangente, etc . . .).
  - *Modelado por Geometría Sólida Constructiva* (o *CSG*) que define el sólido como un conjunto de expresiones lógicas booleanas. Caracteriza implícitamente el interior del objeto y su superficie.

  El más utilizado en los motores de render (y a veces el único) es el modelado poligonal. Proporciona una representación consistente y eficiente (el hardware trabaja bien con vértices y aristas). El inconveniente es que es costoso en almacenamiento.

- **Texturizado:** Es la fase en la cual se asignan propiedades a los objetos de la escena. Se definen las texturas que se aplicarán, las propiedades de reflexión y refracción de los materiales, etc . . . Pueden usarse tanto imágenes como texturas procedurales.

- **Iluminación:** Se definen las fuentes de luz de la escena. Es un paso íntimamente relacionado con el proceso de render debido a que define el modelo de iluminación a utilizar.

  Las fuentes de luz se definen por una serie de componentes:

  - *Posición y Orientación*.
  - *Color*. Para definirlo habitualmente se utilizan modelos aditivos (RGB o HSB).
  - *Intensidad*.
  - *Factor de Caída*. Representa la fuerza de una fuente de luz y hasta dónde llega.

En función de sus características se distinguen varios tipos de fuentes de luz como el foco, la luz omnidireccional, la luz de área, la luz ambiental o la luz infinita.

- **Animación:** Fase que consiste en definir cómo cambian unas determinadas propiedades—por lo general posición, orientación y escala—de un objeto a lo largo de una línea de tiempo. No aparece en los proyectos en los que la salida es una sola imagen estática.
- **Render:** Consiste en la generación de una imagen bidimensional a partir de la descripción de la geometría de la escena junto con las definiciones de las luces, de la cámara y de los materiales. Existen multitud de métodos de render que aplican distintas técnicas y que serán revisados en las siguientes secciones.

3.2.1. Introducción a la síntesis de imagen fotorrealista

El proceso de **síntesis de imagen fotorrealista** se puede definir como aquel cuyo fin consiste en la generación de imágenes sintéticas indistinguibles de las captadas en el mundo real.

La base de la generación de imágenes de este tipo se encuentra en la adecuada simulación de la luz. Un término clave es la **iluminación global** que consiste en simular toda la luz dispersa en un modelo 3D respetando las leyes de la física. Las técnicas de iluminación global persiguen calcular el valor de la intensidad de la luz en cada uno de los puntos de la escena, valor que dependerá de la interacción de las distintas fuentes con los elementos de la escena.

Las técnicas existentes de iluminación global se basan en el **Trazado de Rayos** [Whi80] o en **Radiosidad** [GTGB84] o en ambas a la vez (las llamadas técnicas mixtas).

Los distintos métodos de renderizado pretenden dar solución a la **ecuación de renderizado** propuesta en 1986 por Jim Kajiya [Kaj86]. Esta ecuación (3.1) describe el flujo de energía luminosa en una escena. Se basa en las leyes físicas de la luz y teóricamente proporciona resultados perfectos mientras que las distintas técnicas de render ofrecen aproximaciones a estos resultados ideales.

$$L_o(x, \overrightarrow{w}) = L_e(x, \overrightarrow{w}) + \int_\Omega f_r(x, \overrightarrow{w}', \overrightarrow{w}) L_i(x, \overrightarrow{w}')(\overrightarrow{w}' \cdot \overrightarrow{n}) d\overrightarrow{w}' \tag{3.1}$$

Donde

$L_o(x, \overrightarrow{w})$ es la iluminación saliente en una determinada posición x y con una dirección $\overrightarrow{w}$.

$L_e(x, \overrightarrow{w})$ es la luz emitida desde la misma posición y dirección.

$\int_\Omega \ldots d\overrightarrow{w}'$ es la suma infinitesimal con respecto a un hemisferio de direcciones entrantes.

$f_r(x, \overrightarrow{w}', \overrightarrow{w})$ es la proporción de luz reflejada en la posición x (de dentro hacia fuera).

$L_i(x, \overrightarrow{w}')$ es la iluminación entrante en la posición x desde la dirección $\overrightarrow{w}'$.

$(\overrightarrow{w}' \cdot \overrightarrow{n})$ es la atenuación de la luz entrante debido a su ángulo de incidencia.

La interacción de la luz en cada superficie se expresa mediante la *función de distribución de reflectancia bidireccional*, propia de cada superficie, que corresponde con el término f_r de la ecuación.

Las bases físicas de esta ecuación se sustentan en la ley de la conservación de la energía. En una posición x en particular y con una dirección $\overrightarrow{w}$, la iluminación saliente L_o es la suma de la luz emitida L_e y la luz reflejada. Esta última es la suma infinitesimal de la luz entrante L_i desde todas direcciones, multiplicada por la reflexión de la superficie y el ángulo de incidencia. Debido a que conecta la luz saliente con la luz entrante a través de un punto de interacción, esta ecuación facilita un modelo completo para representar el **transporte de luz** en una escena.

Técnicas de trazado de rayos

Este modelo fue introducido en 1980 por Whitted y consiste en calcular la intensidad de la luz en los puntos de la escena a partir de la información que proporcionan una serie de rayos que se lanzan desde el observador hacia las fuentes de luz atravesando el modelo 3D. Estos rayos son líneas imaginarias que viajan por el entorno 3D recopilando información útil para el render. Los rayos se trazan en sentido inverso, desde el observador a las fuentes, debido a que hacerlo en el sentido real no resultaría práctico. Por ello se habla de *trazado de rayos retrospectivo*.

Algunas limitaciones de este tipo de técnicas consisten en que no se pueden representar efectos como la profundidad de campo, las *cáusticas* o la iluminación indirecta. Para simularlos es necesario extender el trazado de rayos empleando *métodos de Monte Carlo* que lancen rayos adicionales para simular todos los caminos posibles de la luz. Uno de los principales problemas de este tipo de métodos es que introducen ruido y para eliminarlo es necesario aumentar el número de muestras. El modo de distribuir los rayos de forma que el ruido sea menor ha sido abordado en numerosos estudios.

Las técnicas que mejores resultados proporcionan son los *métodos guiados de trazado de rayos* como por ejemplo la técnica de **irradiance cache** [WH92]. Esta técnica almacena

y reutiliza la iluminación indirecta interpolando los valores desconocidos. Permite decidir en qué zonas de la escena es conveniente tomar más muestras y en cuáles menos. En una superficie plana y grande es de suponer que la iluminación cambiará con suavidad, requiriendo menos muestras. En las esquinas la iluminación cambiará más drásticamente por lo que será necesario tomar más muestras. Un ejemplo de irradiance cache que se puede encontrar en la página de Yafray [37] se muestra en la Figura 3.2.

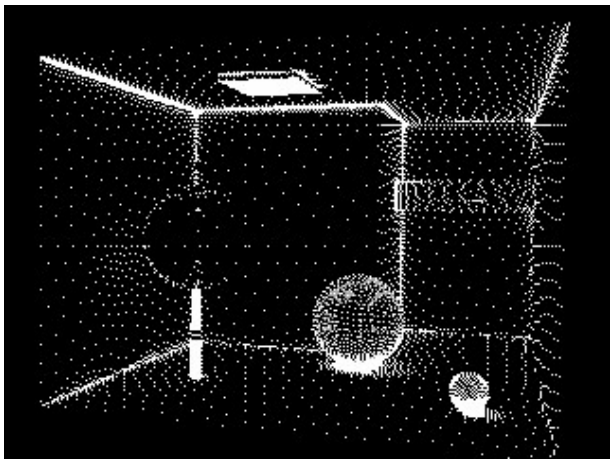

Figura 3.2: Ejemplo de Irradiance Cache.

En las técnicas de trazado de rayos, la geometría de la escena se trata como una *caja negra*. De este modo, los algoritmos gestionan la complejidad de la iluminación pero no la de la geometría. La desventaja es que el número de muestras necesario es muy alto.

Técnicas de radiosidad

Este modelo se basa en el concepto de **intercambio de luz entre superficies** y fue creado por investigadores de la *Universidad de Cornell*.

En la Figura 3.3 se muestra la escena con la que los creadores de esta técnica ilustraron sus investigaciones. En el mundo de la síntesis de imagen 3D, la escena de las *cajas de Cornell* es muy conocida.

Para calcular este intercambio de energía, se subdivide el modelo en pequeñas unidades denominadas *parches*, a partir de las cuales se obtiene la distribución de luz de la escena.

En el modelo de radiosidad, cada superficie tiene asociados dos valores: la intensidad luminosa que recibe, y la cantidad de energía que emite. En el algoritmo de radiosidad se calcula el intercambio de energía desde cada superficie hacia el resto. La complejidad de este algoritmo es $O(n^2)$ considerando que haya n superficies en la escena.

Figura 3.3: Cajas de Cornell.

Una de las características que define a las técnicas de radiosidad es que proporcionan una solución de iluminación **independiente del punto de vista**. Una vez calculada, se puede utilizar la iluminación resultante para renderizar desde diferentes ángulos. Por contra, la solución es costosa tanto en tiempo como en espacio de almacenamiento.

Otras técnicas

Existen **técnicas mixtas** que aúnan las ventajas de los dos tipos de técnicas anteriores. Por un lado, las técnicas basadas en trazado de rayos proporcionan buenos resultados simulando la reflexión especular. En el caso de la radiosidad, los mejores resultados se obtienen simulando las reflexiones difusas. Estas técnicas utilizan la aproximación que proporciona cada algoritmo al tipo de reflexión que manejan mejor. Sin embargo, incorporar las técnicas de radiosidad limita la complejidad de la malla tal y como se ha visto en el punto anterior.

El **mapeado de fotones** [Jen01] utiliza una aproximación diferente que las técnicas híbridas. La idea en la que se basa consiste en cambiar la representación de la iluminación. En lugar de acoplar la información acerca de la iluminación a la información geométrica, esta técnica usa una estructura de datos diferente para almacenarla, el *mapa de fotones*.

Esta estructura se construye lanzando fotones desde las fuentes de luz y almacena la información de los impactos producidos. Estos datos serán utilizados para renderizar la escena de una forma eficiente.

3.2.2. Métodos de renderizado

En esta sección se hará una introducción, que no pretende ser exhaustiva, a los métodos de renderizado **píxel a píxel** más importantes. En la Figura 3.5 se muestra una misma imagen obtenida con distintos métodos de render que se comentarán a continuación.

Scanline

Es quizá uno de los métodos de render más básicos y fue introducido por Bouknight [Bou70] en el año 1970.

Algoritmo 1 Bucle general del Scanline.

```
for all píxel de la imagen do
    línea ← trazar una línea desde la cámara al píxel
    color ← Scanline (línea) {Algoritmo 2}
end for
```

Algoritmo 2 Procedimiento Scanline (línea)

```
punto ← encontrar el punto de intersección más cercano
color ← color de fondo
for all fuente de luz do
    color ← color + iluminación directa
end for
Devolver(color)
```

Una de sus principales ventajas es la rapidez, de hecho, es el método que menos tiempo consume de los algoritmos que se van a tratar en esta sección. Además, trabaja bien con texturas y simula cualquier tipo de sombreado.

Como contrapartida, no permite simular las reflexiones y refracciones de la luz de una manera realista, aunque pueden ser simuladas utilizando técnicas adicionales.

Raytracing

Algoritmo 3 Bucle general del Raytracing.

```
for all píxel de la imagen do
    rayo ← trazar un rayo desde la cámara al píxel
    color ← Raytracing (rayo) {Algoritmo 4}
end for
```

Algoritmo 4 Procedimiento Raytracing (rayo)

```
punto ← encontrar el punto de intersección más cercano
color ← color de fondo
for all fuente de luz do
  rayo_sombra ← trazar un rayo desde (punto) hasta la fuente de luz
  if el rayo no choca con ningún objeto then
    color ← color + iluminación directa
    if el material tiene propiedad de reflexión then
      color ← color + Raytracing (rayo_reflejado)
    end if
    if el material tiene propiedad de refracción then
      color ← color + Raytracing (rayo_transmitido)
    end if
  else
    color ← negro
  end if
end for
Devolver(color)
```

El método de raytracing o *trazado de rayos* fue propuesto por Whitted [Whi80] y proporciona un medio sencillo y recursivo de calcular superficies con reflejos y transparencia.

La idea, tal y como ya se ha comentado, consiste en trazar rayos desde el observador a las fuentes de luz. En realidad, son las fuentes de luz las que emiten fotones que rebotan en la escena y llegan a los ojos del observador. Sin embargo, sólo una pequeñísima fracción de los fotones llegan a su destino, por lo que el cálculo en esta forma directa resulta demasiado costosa. Los rayos que se emiten a la escena son evaluados respecto de su visibilidad trazando nuevos rayos desde los puntos de intersección (rayos de sombra). Una descripción básica del trazado de rayos se muestra en los Algoritmos 3 y 4.

El raytracing permite simular reflexiones especulares y refracciones pero no está pensado para simular iluminaciones indirectas o sombras difusas.

Ambient Occlusion

La técnica de *ambient occlusion* (Figura 3.4) fueron introducidas por Zhurov en 1998 como alternativa a la técnica de radiosidad por su bajo coste computacional [ZIK98].

La *ocultación* $W(P)$ de un punto de una superficie es un valor entre 0 y 1 y responde a la Ecuación 3.2.

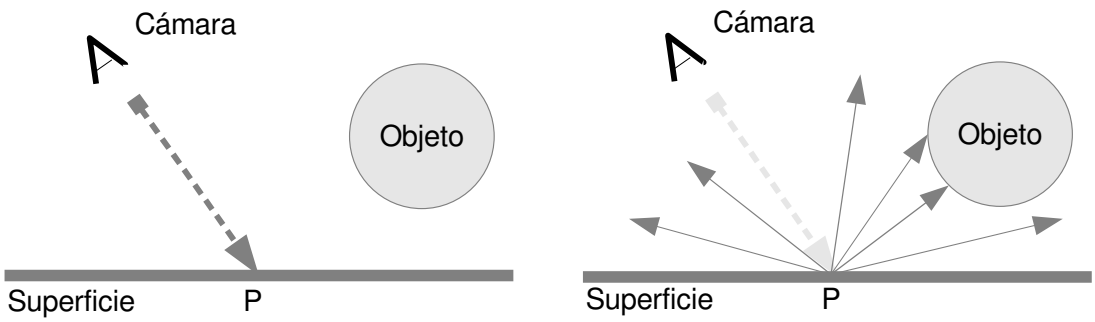

Figura 3.4: Cálculo del método Ambient Occlusion. En un primer paso (izquierda) se calcula el punto P. El valor de la ocultación, $W(P)$, es proporcional al número de rayos que alcanzan el *cielo*, teniendo en cuenta la distancia máxima entre P y el primer objeto de intersección. En el caso de la figura de la derecha, escapa 1/3 de los rayos.

$$W(P) = \frac{1}{\pi} \int_{w \in \Omega} \rho(d(P,w)) cos\theta dw \tag{3.2}$$

Donde $d(P,w)$ es la distancia entre P y la primera intersección con algún objeto de la escena en la dirección de w. El término $\rho(d(P,w))$ es una función con valores entre 0 y 1 que indica la magnitud de iluminación ambiental que viene en la dirección de w. Por último, θ es el ángulo formado entre la normal en P y la dirección de w.

Radiosidad

Las técnicas basadas en *radiosidad* calculan el intercambio de luz entre superficies. Inicialmente fue propuesto por Goral y Greenberg [GTGB84] en 1984. El cálculo del intercambio de luz entre superficies se basa en el concepto de **factor de forma**, que se calcula mediante la Ecuación 3.3.

$$F_{i,j} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{cos\theta_i \, cos\theta_j}{\pi r^2} H_{ij} \, dA_j \, dA_i \tag{3.3}$$

Siendo F_{ij} el factor de forma de la superficie i a la superficie j y $cos\theta_i \, cos\theta_j$ es el ángulo entre las normales de los planos de cada parche. πr^2 mide la distancia entre los parches y H_{ij} es el factor de visibilidad (con valores entre 0 y 1). Por último, el término dA_x corresponde al área de la superficie x.

Pathtracing

Supone una extensión al algoritmo de trazado de rayos y fue formulado por Kajiya en 1986 como solución a la ecuación de renderizado que fue propuesta en el mismo artículo [Kaj86].

El *pathtracing* permite calcular la solución completa de la iluminación global. El mecanismo en el que se basa consiste en lanzar rayos para calcular todos los posibles caminos de donde pueda venir la luz. El muestreo se realiza mediante la integración de Monte Carlo, que consigue crear rayos uniformemente por todos los posibles caminos.

Cuando un rayo choca contra una superficie difusa, en lugar de llamar al procedimiento recursivo con los múltiples rayos que rebotarían en la superficie, se selecciona uno de forma aleatoria.

La utilización de la irradiance cache evita el cálculo de la iluminación global en cada punto del modelo ya que se calculan algunos píxeles y los valores intermedios se interpolan.

Photon Mapping

Propuesto por Jensen en 1996 [Jen96], presenta como novedad que desacopla la representación de la iluminación de la geometría. Esta técnica consiste en dos pasos:

1. Construcción de la estructura del mapa de fotones desde las fuentes de luz al modelo. Para ello, se lanza un gran número de fotones desde las fuentes de luz. Cada uno de estos fotones contiene una fracción de la energía total de la fuente de la que proviene.

2. Se lleva a cabo el proceso de render mediante la utilización de la información contenida en el mapa de fotones.

La recuperación de la información del mapa de fotones tiene una complejidad de $O(\log\ n)$ en el caso medio y $O(n)$ en el peor de los casos. Para agilizar las búsquedas en esta estructura se suelen utilizar *kd-trees*[1].

Pathtracing Bidireccional

Extensión del *pathtracing* introducida por Lafortune y Willems que se caracteriza porque también se calculan los caminos desde las fuentes de luz. Aprovecha la idea de que hay ciertos caminos que son más fáciles de trazar desde las fuentes de luz, como por ejemplo el caso de las cáusticas.

[1] Su nombre proviene de k-dimensional tree, árbol de k dimensiones. Son un caso especial de los árboles BSP y sirven para particionar un espacio de k-dimensiones.

Este método calcula dos caminos, uno que parte desde el punto de observación y otro que parte desde cada fuente. La información obtenida por ambos medios se combina para obtener los valores de iluminación finales.

Aunque requiere menos muestras que el pathtracing sencillo, el cálculo tarda más debido a que el coste de cada muestra es mayor y hay que añadir el tiempo de combinar los caminos. Efectos como las cáusticas se calculan perfectamente.

Transporte de Luz de Metrópolis

Fue propuesto por Veach y Guibas [VG97] en 1997 y pretende utilizar la información de la escena de una forma más eficiente. La densidad de rayos lanzados está en función de la radiancia, concentrando más rayos en las zonas de la escena más brillantes.

El método tiene un punto de partida que consiste en una distribución de muestras aleatorias (pathtracing bidireccional) de todas las fuentes de luz de la escena. Estos caminos son clonados y mutados aleatoriamente. Se descartan los caminos no válidos (atraviesan un objeto sólido por ejemplo) y los válidos son aceptados con una determinada probabilidad.

3.2.3. Casos de estudio en motores de render

Existe una gran variedad en cuanto a motores de render en el mercado. Se diferencian por las técnicas que implementan, por sus características y por si son libres, gratuitos o comerciales.

Existen numerosos **motores de render libres** como *Blender 3D*, *Yafray* o *Toxic*.

Toxic [35] es un motor de render GPL que permite efectos como la profundidad de campo y las cáusticas entre otros, está implementado en C++ y permite importar mallas en numerosos formatos.

Blender 3D [5] es un software distribuido bajo GPL que permite el modelado 3D, la animación, el render, la postproducción, etc ...Cuenta con numerosas características avanzadas y con una comunidad de usuarios muy activa.

Yafray [37] es otro motor de render libre (con licencia LGPL) que permite, entre otros efectos, los de profundidad de campo y cáusticas, cuenta con iluminación *HDRI* y *Skydome*, etc ...

Pov-Ray es un excelente motor gratuito, con el código fuente disponible y compilado para multitud de plataformas. Permite la compilación distribuida en diferentes procesadores mediante PVM. Está basado en el raytracer DKB-Trace.

Rayshade, cuya primera versión data de 1988 y fue desarrollada en Princeton, es un motor muy utilizado en ambientes académicos. Es muy extensible pero no tan potente como Pov-Ray. Su código fuente también es de libre disposición.

También existen varias implementaciones alternativas de RenderMan (la especificación de interfaces desarrollada por Pixar para utilizarla en sus proyectos). Algunas de estas implementaciones son **3Delight**, **Pixie**, **Aqsis** o JRMan, las dos últimas bajo licencia GPL.

BMRT *(Blue Moon Rendering Tools*) también era una implementación gratuita del interfaz RenderMan de Pixar pero su desarrollo se encuentra actualmente detenido.

Además de estas alternativas libres o gratuitas, existen numerosos **motores de render comerciales**. Uno de ellos es la implementación oficial de la especificación RenderMan de Pixar, **PRMan** (Photo Realistic RenderMan).

Brazil [6] es un motor de render comercial desarrollado por la empresa SplutterFish. Permite la integración con 3D Studio Max y Viz y posee una amplia variedad de características como el cálculo de cáusticas.

Mental Ray [18] está incluido en 3D Studio Max y Viz y ha sido desarrollado por la empresa alemana Mental Images. Es un motor de render de uso general que posee capacidad para generar simulaciones físicamente correctas de los efectos de iluminación, como la reflexión y la refracción, los efectos cáusticos o la iluminación global.

V-Ray [8] ha sido desarrollado por la compañía Chaos Group y es un motor de render basado en raytracing. Su uso está muy extendido en visualización arquitectónica y permite generar simulaciones físicamente correctas de los efectos de iluminación.

Maxwell [17] es un motor de render que trabaja con luz físicamente realista y cuyos algoritmos y ecuaciones reproducen todos los comportamientos de la luz. Es uno de los que obtienen mayor realismo en las imágenes renderizadas. Ha sido desarrollado por la compañía Next Limit.

FinalRender [13] está disponible para 3D Studio Max y está desarrollado por CEBAS. Permite, entre otros muchos efectos, emular la dispersión de la luz.

Gelato [19] es un motor de render multiplataforma creado por la empresa *NVIDIA* y en cuyo desarrollo ha participado el creador de BMRT. Utiliza la potencia de cálculo de las tarjetas gráficas de la compañía además de la de la CPU del usuario. Existe una versión gratuita y otra de pago.

Figura 3.5: Escena renderizada con distintas técnicas de render. Los métodos utilizados son de izquierda a derecha y de arriba abajo: *Scanline*, *Raytracing*, *Raytracing* con *Ambient Occlusion*, *Pathtracing*, *Pathtracing* con *IC* y *Pathtracing* con *HDRI*.

3.3. SISTEMAS RELACIONADOS

En esta sección se realizará una introducción a una serie de sistemas actuales cuyo análisis servirá para establecer las bases sobre las que se asentará el sistema a desarrollar.

Se tratarán dos tipos de sistemas:

- Aquellos sistemas que actualmente se usan para resolver la problemática del tiempo de render y que no se basan en una aproximación de tipo grid como sí es el caso del sistema Yafrid.

- Aquellos que no tienen relación alguna con el proceso de render sino que han sido desarrollados para solucionar otros problemas computacionalmente intensivos.

Los primeros proporcionan una visión de un panorama donde Yafrid pretende plantear una aproximación distinta a la habitual. Los segundos servirán para entender cómo se resuelven determinados problemas mediante sistemas grid y cómo esas soluciones pueden ser también aplicadas al problema concreto del render.

Aproximaciones para resolver el problema del tiempo de render

La solución clásica para solucionar esta problemática consiste en la utilización de clusters de ordenadores. Esta solución implica que todos las máquinas pertenecen a una misma organización y poseen niveles bajos de heterogeneidad.

En el ámbito comercial, las grandes empresas del sector como *Pixar Animation Studios*, *Dreamworks* o *Industrial Light and Magic* poseen en sus instalaciones un gran número de máquinas dedicadas exclusivamente al render. Invierten ingentes cantidades de dinero en mantener y actualizar el hardware para tenerlo listo para la próxima producción.

Dos ejemplos de aproximaciones cluster utilizadas en ambientes académicos los constituyen el entorno de render distribuido orientado a la enseñanza desarrollado en la Purdue University [MAB05] y la solución de DeMarle [DPH+03] de raytracing interactivo para la visualización de grandes volúmenes de datos. En esto último, en la visualización interactiva de datos, también trabajan los creadores del proyecto RAVE [GAW04] dedicado a la visualización de datos tomográficos.

Otro interesante proyecto basado en cluster de ordenadores es *OSCAR*, acrónimo de *Open Source Cluster Application and Resources, Aplicaciones y Recursos para Cluster de Código*

Abierto, que es una iniciativa del Servicio de Supercomputación de la Universidad de Castilla-La Mancha [29]. Aunque en principio es un sistema de propósito general, se ha venido utilizando para procesar proyectos de render con excelentes resultados.

Un enfoque totalmente distinto a los anteriores ha dado lugar a una línea de investigación que está tomando bastante importancia en los últimos años. Esta aproximación consiste en la utilización de granjas de GPUs para acometer tareas de render. Un ejemplo de este enfoque lo proporcionan [FQKYS04].

Otra aproximación distinta, basada en la utilización de hardware a medida es YAFRAYNET que pretende llevar a cabo el render de escenas 3D con YafRay en granjas de máquinas equipadas con tarjetas como la de la Figura 3.6.

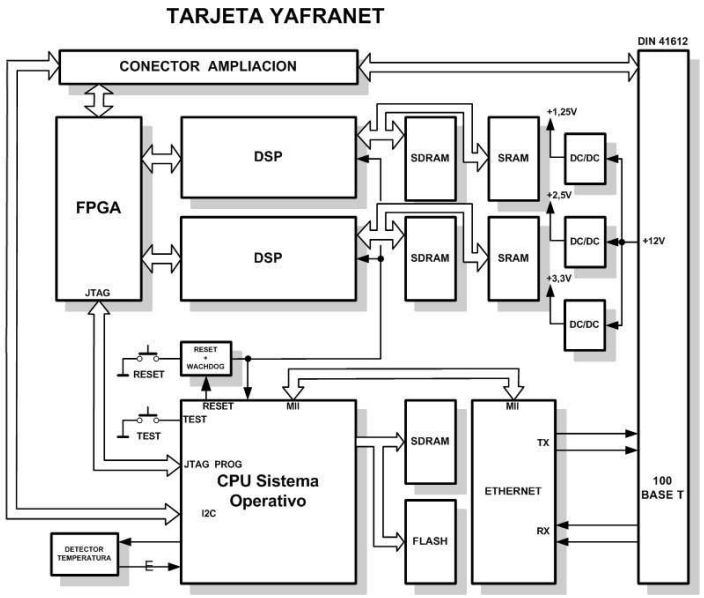

Figura 3.6: Diagrama de una tarjeta YafRayNet.

Durante el tiempo que ha durado el proyecto Yafrid, el panorama en este campo ha ido cambiando y han surgido propuestas que no existían cuando comenzó. Esto es indicador de que la síntesis de imagen 3D es un área en continuo desarrollo.

A lo largo de los últimos años, se ha dado un paso más en el tema de los cluster aplicados al proceso de render. Actualmente existen varios productos que permiten acceder remotamente a servicios de render a través de Internet. El usuario, una vez registrado, puede enviar trabajos de render al sistema que serán procesados por un cluster que pertenece a la empresa en cuestión. La única diferencia reside en que el servicio es accesible a través de Internet, pero quien realiza el trabajo sigue siendo un cluster de computadores. Un ejemplo de este tipo de

servicio es RenderPlanet [28].

Aproximaciones GRID a problemas de ámbito general

La computación GRID se está utilizando en diversas áreas donde se necesita una enorme potencia de cálculo como son las ciencias, la ingeniería o los negocios. Entre los muchos usos que se están dando a estos sistemas se encuentran por ejemplo el desarrollo de modelos moleculares para el diseño de medicamentos, el análisis de los patrones de la actividad cerebral, experimentos en el campo de la física de alta energía o en la búsqueda de vida inteligente fuera de nuestro planeta.

Uno de los ejemplos más conocidos, **SETI@home** [30], se dedica a esto último y constituye el proyecto de computación distribuida a través de Internet más grande y con más éxito [31]. SETI@Home es un proyecto pionero que analiza los datos que provienen del radiotelescopio de Arecibo (Puerto Rico) en busca de patrones. Para ello utiliza el poder de cálculo de cientos de miles de ordenadores conectados a Internet. El funcionamiento es sencillo: cada ordenador recibe un pequeño paquete de datos vía Internet que analizará localmente. Una vez que haya terminado, enviará los resultado de vuelta[2].

Después de este proyecto, muchas aplicaciones han encontrado en la aproximación grid una solución para operaciones muy complejas. Un proyecto científico similar al anterior es **Einstein@home**. Este experimento está diseñado para analizar los datos de los observatorios LIGO en Estados Unidos y GEO 600 en Alemania en busca de ondas gravitacionales, cuya existencia predijo Albert Einstein en su teoría de la relatividad general.

Las infraestructuras grid más importantes a nivel europeo las constituyen los proyectos **EGEE** [33] (Enabling Grids for E-sciencE) y los distintos sistemas grid nacionales (algunos de los cuales también están integrados dentro del propio EGEE). Algunos de estos sistemas son *NorduGRID* (países nórdicos), UK e-Science (Reino Unido) o HellasGrid (Grecia). El EGEE está financiado por la UE y coordinado por el CERN. En Estados Unidos, el equivalente al EGEE es el proyecto **OSG** [21] (Open Science Grid).

Existen también otros sistemas grid más específicos como el *IPG (Information Power Grid)* de la NASA, el *NEES (Network for Earthquake Engineering Simulation)*, o el *ESG (Earth System Grid)*.

[2]Hay que destacar la similitud entre el funcionamiento de este sistema y Yafrid.

3.4. DESARROLLO DE APLICACIONES WEB

Una aplicación web es una aplicación que ofrece un conjunto de funcionalidades que pueden ejecutarse a través de un navegador web que utiliza el cliente de la aplicación. Mediante el protocolo *HTTP*, el cliente envía peticiones al servidor, que las ejecuta y devuelve resultados.

Una de las grandes ventajas de las aplicaciones web es la independencia de la plataforma de ejecución (en el cliente), ya que lo único que se requiere es un navegador que interprete correctamente código HTML.

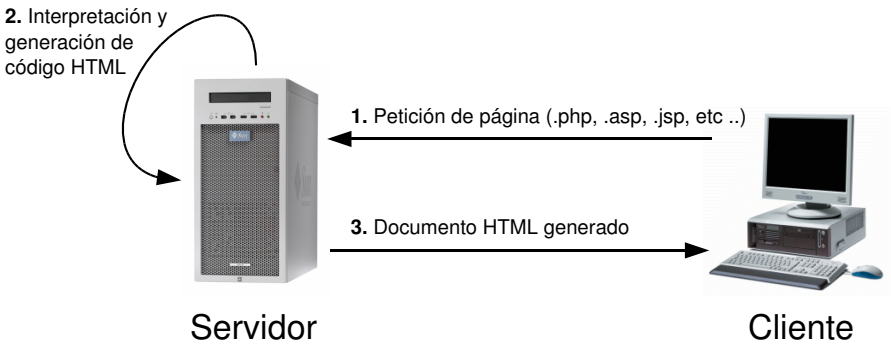

Figura 3.7: Lenguajes de programación del lado del servidor.

Las aplicaciones web se desarrollan mediante páginas dinámicas, es decir, que poseen scripts que se ejecutan del lado del servidor. Estos scripts hacen que el contenido de las páginas que son devueltas al cliente sea dinámico. En lugar de estar prefijado, el contenido de estas páginas puede obtenerse de ficheros almacenados en el servidor o de la información contenida en una base de datos. En la Figura 3.7 se muestra el proceso que se lleva a cabo cuando un cliente hace una petición de una página dinámica.

Existen muchas tecnologías para este tipo de desarrollos entre las que están CGI[3], PHP, JSP, Servlets, ASP, ASP .NET o CSP mediante los cuales, se pueden crear scripts que se ejecuten del lado del servidor.

Por otra parte, también existen lenguajes del lado del cliente como son JavaScript o Visual Basic Script. Para que estos scripts sean ejecutados, el navegador del cliente de la aplicación debe ser capaz de interpretarlo y ejecutarlo.

[3]Del inglés, Common Gateway Interface.

3.4.1. PHP

Según su sitio web [23], *PHP es un lenguaje de scripting embebido en HTML. Mucha de su sintaxis ha sido tomada de C, Java y Perl con un par de características adicionales únicas y específicas de PHP. El propósito del lenguaje es permitir que los desarrolladores web escriban páginas generadas dinámicamente con rapidez.*

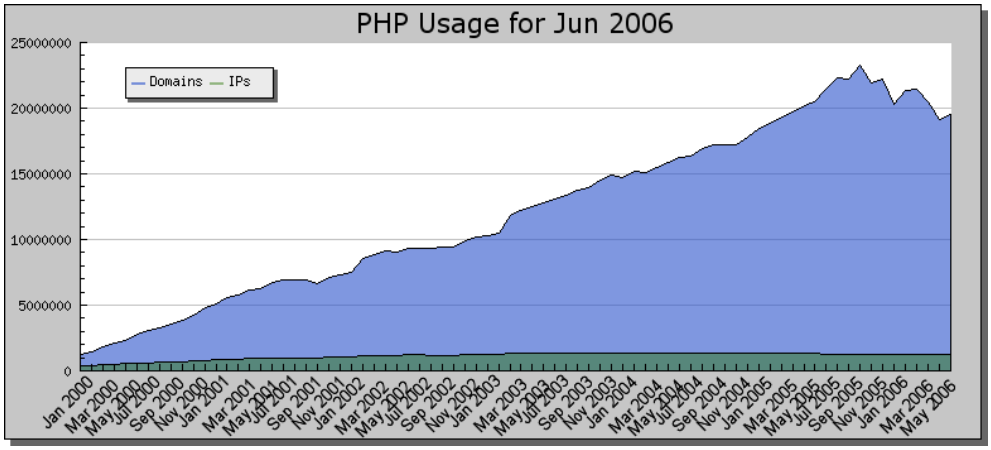

Figura 3.8: El número de sitios que utilizan PHP se ha venido incrementando desde que se tienen datos hasta alcanzar su máximo en el último trimestre de 2005. A partir de ese momento, el crecimiento sostenido que se venía experimentando está sufriendo un cierto retroceso.

En 1995 *Rasmus Lerdorf* quería conocer el número de personas que estaban leyendo el currículum vítae que había colocado en su página web. Para ello, creó un CGI en Perl que mostraba el resultado estadístico en la propia página. Rasmus llamó a ese script PHP, acrónimo de *Personal Home Page* (página de inicio personal).

Durante los dos o tres años siguientes, PHP creció y el número de usuarios se incrementó. Se le añadieron nuevas características hasta convertirse en lo que hoy se conoce como PHP/FI 2.0. Sin embargo, fue en 1997 cuando surgió PHP 3.0 de la mano de *Zeev Suraski* y *Andi Gutmans* quienes reescribieron el código completo dando lugar a un lenguaje parecido al que conocemos hoy. El significado de PHP también evolucionó pasando a ser un acrónimo de los llamados recursivos[4] y significa *PHP: Hypertext Preprocessor* (PHP: Pre-procesador de Hipertexto).

Hoy en día PHP es uno de los lenguajes más utilizados para el desarrollo web y es usado por alrededor de 20 millones de programadores en todo el mundo como se muestra en las

[4]Uno de los más conocidos es GNU que significa *GNU's Not Unix!*

estadísticas de la Figura 3.8 que se pueden consultar actualizadas mensualmente en [12].

Algunas características de PHP son:

- **Multiplataforma** ya que funciona en un gran número de sistemas operativos.
- **Portabilidad**. El código PHP escrito para una plataforma funciona sin cambios en cualquier otra.
- Es un lenguaje **interpretado** de **alto nivel** muy completo con el que es posible desarrollar muy diversos tipos de aplicaciones.
- Proporciona un **alto rendimiento** y no es necesario realizar una gran inversión en hardware.
- Es soportado por **la mayoría de servidores web**, incluido el servidor web libre más extendido, Apache [2].
- Soporte para un gran número de **sistemas de bases de datos** como MySQL, Oracle, Informix o PosgreSQL por citar algunos.
- La última versión hasta la fecha, PHP5, incorpora un completo soporte para **orientación a objetos** que incluye herencia, polimorfismo, tratamiento de excepciones, clases abstractas, interfaces, etcétera.
- Gran cantidad de **librerías** y **extensiones** que van desde el manejo de gráficos a la generación de documentos PDF pasando por la compresión y descompresión de ficheros.
- En parte por ser un proyecto de **código abierto**, PHP goza de una comunidad muy activa de desarrolladores que implementan nuevas librerías y aplicaciones que pueden ser utilizadas para otros desarrollos.
- **Curva de aprendizaje** similar a la de lenguajes como Java o C debido a su sencilla sintaxis, similar a la de los lenguajes mencionados.
- Existe **abundante información** sobre PHP. Existen manuales en más de 25 idiomas, listas de distribución, foros y muchos otros medios de obtener ayuda de otros desarrolladores.

3.4.2. JSP

JSP (*Java Server Pages*) es la tecnología creada por Sun Microsystems [32] para el desarrollo web mediante un lenguaje de script similar a Java.

Permite una sencilla separación de las aplicaciones web en capas ya que es posible integrar la funcionalidad de clases Java en los scripts JSP. Así, las clases Java se pueden encargar de realizar los procesamientos necesarios mientras que los scripts JSP limitan a formatear los resultados.

El uso de Java hace de ésta una tecnología muy versátil ya que puede funcionar en prácticamente todas las plataformas siempre y cuando tengan instalada la máquina virtual de Java, garantizando así la portabilidad de las aplicaciones desarrolladas.

Además, el coste de desarrollo de aplicaciones mediante JSP es bajo ya que la tecnología Java necesaria puede ser adquirida de forma gratuita de la página oficial de Sun Microsystems.

Los JSP son muy similares a los servlets en el sentido de que son compilados la primera vez que son invocados y comienzan a ejecutarse en el servidor como un servlet, lo que proporciona un alto rendimiento. Las aplicaciones que utilizan JSP deben implantarse en un servidor de aplicaciones.

Apache Tomcat [3] (o Jakarta Tomcat) es un servidor de aplicaciones que implementa las especificaciones de servlets y JSP de Sun Microsystems. Tomcat es desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Las versiones más recientes son las 5.x, que implementan las especificaciones de Servlet 2.4 y de JSP 2.0.

Tomcat puede actuar, además de como servidor de aplicaciones, como servidor web independiente. Funciona en cualquier entorno que tenga la máquina virtual de Java instalada ya que está desarrollado en Java. Tomcat es uno de los servidores más utilizados para aplicaciones JSP e incluye el compilador Jasper, que compila los JSP para obtener servlets.

3.4.3. ASP y ASP .NET

ASP *(Active Server Pages)* es un lenguaje de script del lado del servidor creado por Microsoft para el desarrollo de páginas web dinámicas.

ASP ha pasado por cuatro versiones distintas hasta llegar a la actual, ASP .NET que forma parte de la plataforma .NET de Microsoft. Las anteriores a .NET se conocen como ASP clásico y eran distribuidas junto con el servidor de Microsoft, ISS.

IIS *(Internet Information Services o Server)* es un conjunto de servicios que se incluyen en los sistemas operativos de Microsoft entre los que se encuentran el HTTP/HTTPS.

Una de las características más importantes de la versión .NET de ASP reside en su soporte para múltiples lenguajes compilados (como Visual Basic .NET, C++ o C# entre otros) que sustituyen a los lenguajes interpretados.

Como en el caso de los JSP, la primera vez que se solicita una página, ésta es compilada y almacenada en memoria caché. De este modo, las siguientes peticiones serán atendidas de forma más rápida, consiguiendo un incremento del rendimiento.

La posibilidad de utilizar diferentes lenguajes de alto nivel, como Visual Basic .NET, C++ o C# permite el desarrollo de aplicaciones basado en un diseño multicapa que separe dominio y presentación

3.4.4. Hojas de Estilo en Cascada

Las **hojas de estilo en cascada** o **CSS** *(Cascading Style Sheets)* son un medio para especificar el estilo de los documentos HTML y XML mediante un lenguaje formal. Existen dos niveles (o versiones) publicados como recomendación por el *W3C* [36]: *CSS1*, publicado en 1996, y *CSS2*, publicado dos años después.

Mientras que CSS1 está soportado por la mayoría de navegadores, CSS2, que aporta nuevas funcionalidades, no está totalmente implementado en todos los navegadores.

Aunque el uso de hojas de estilo no es imprescindible en el desarrollo de aplicaciones web, proporciona algunas ventajas muy interesantes y su utilización se ha extendido en los últimos años.

Las hojas de estilo sirven para **separar la estructura y el aspecto** de un documento HMTL. Mientras que la estructura se define mediante las etiquetas que proporciona HTML, el aspecto se establece con CSS. De este modo, el código de las páginas web resulta más claro y más sencillo de depurar.

Otra de las ventajas de la utilización de hojas de estilo es que se puede cambiar el aspecto de toda una aplicación web sin modificar nada en el código HTML en el caso ideal. Lo único necesario es cambiar la hoja de estilo. Esto puede servir por ejemplo para mostrar una página web según las preferencias visuales de cada usuario o para mejorar la **accesibilidad** de personas con deficiencias visuales.

El estilo de una aplicación web se puede definir en un fichero de texto plano en el que se definen bloques de atributos asociados a los distintos tipos de elementos. Los elementos sobre los que se pueden definir atributos son aquellos que establecen la estructura del documento HTML como las tablas, los enlaces, los encabezados o los párrafos.

Los estilos se pueden aplicar para todos los elementos pertenecientes a un determinado

```
h1{
   color:#800000;
   font-family: Trebuchet MS, Verdana, Helvetica, sans-serif;
   font-size: 18px;
   line-height: 22px;
   margin: 0px 0px 0px 10px;
   padding: 0px;
}
```

Listado 3.2: Definición del estilo de un encabezado con CSS.

```
input.button{
   border: 1px solid #808080;
   background-color: white;
   color: #c30d0d;
   font-size: 12px;
}
```

Listado 3.3: Definición del estilo de un botón con CSS.

tipo como se muestra en el Listado 3.2. En este ejemplo, se define un estilo para todos los elementos de tipo *h1*.

También se pueden establecer distintas clases dentro de un mismo tipo de elementos. Con el Listado 3.3 se establece un estilo para los elementos de tipo *input* que pertenezcan a la clase *button*. Los elementos de esta clase tendrán el fondo blanco, el borde gris y el texto de color rojo y con un tamaño de 12 píxeles. No se aplicará este estilo a los elementos de tipo *input* que no pertenezcan a esta clase.

Para que un elemento de tipo entrada tome las propiedades que define la clase *button* habrá que definirlo como *<input class="button" />* tal y como se muestra en el Listado 3.4.

Una característica interesante de las hojas de estilo consiste en la **herencia** de estilos que permite que un elemento contenido en otro herede el aspecto del padre y lo combine con el suyo propio.

3.4.5. JavaScript

JavaScript es un lenguaje interpretado dedicado al desarrollo web que funciona del lado del cliente y que posee una sintaxis similar a la de Java. Fue creado por Brendan Eich en la empresa Netscape Communications, que fue la que sacó al mercado los primeros navegadores

```
<input type="button" class="button" name="action" value="Login" />
```

Listado 3.4: Asignación de una clase a un botón en HTML.

comerciales y apareció por primera vez en el Netscape Navigator 2.0.

Aunque en sus orígenes se denominó Mocha y más tarde LiveScript, fue rebautizado con su nombre actual en un anuncio conjunto entre Sun Microsystems y Netscape a finales de 1995.

Dos años después, JavaScript fue propuesto para ser adoptado como estándar de la ECMA (European Computer Manufacturers' Association). En junio de 1997 fue adoptado como un estándar ECMA, con el nombre de ECMAScript. Poco después también fue reconocido como un estándar ISO (ISO 16262).

JScript es la implementación de ECMAScript de Microsoft, muy similar al JavaScript de Netscape, pero con ciertas diferencias en el modelo de objetos del navegador que hacen a ambas versiones con frecuencia incompatibles.

Para evitar estas incompatibilidades, el World Wide Web Consortium diseñó el estándar *Document Object Model* (DOM, ó Modelo de Objetos del Documento), que incorporan Konqueror, las últimas versiones de Internet Explorer y Netscape Navigator, Opera versión 7, y Mozilla desde su primera versión.

JavaScript es hoy en día un lenguaje muy potente que permite añadir multitud de características a las aplicaciones web del lado del cliente. Es además uno de los componentes de una tecnología emergente para el desarrollo web, AJAX[5].

3.5. SISTEMAS DISTRIBUIDOS

3.5.1. Introducción a los Sistemas Distribuidos

La complejidad es una de las características inherentes del software distribuido de cualquier tipo. El desarrollo de este tipo de aplicaciones introduce dificultades [Moy05] que no se presentan en aplicaciones no distribuidas:

- **Heterogeneidad** a nivel **de sistema operativo**: MS Windows, GNU/Linux, Unix, MacOS, PalmOS, etc ...
- **Heterogeneidad** a nivel de **hardware**: ordenadores personales, dispositivos móviles, etc ...
- **Heterogeneidad en las redes** y protocolos: Ethernet, ATM, TCP/IP, Bluetooth, etc ...

[5]Del inglés **A**synchronous **J**avaScript **A**nd **X**ML, JavaScript y XML asíncronos. Esta tecnología es usada en algunos productos de Google como su Google Maps.

- **Heterogeneidad** con respecto a los **lenguajes de programación**.

- El **modelo de programación** en el que se basan las aplicaciones centralizadas y las distribuidas no es el mismo por lo que adaptar aplicaciones centralizadas no resulta sencillo.

- Realizar **pruebas** de una aplicación distribuida requiere un entorno más complejo que en el caso de la mayoría de las aplicaciones.

EL problema de la heterogeneidad es insalvable si lo que se pretende es que se llegue a un acuerdo en todos los niveles. Como esto no es posible, la solución se basa en garantizar que los distintos sistemas se puedan comunicar mediante interfaces y mecanismos de interoperabilidad comunes.

Un modelo de programación distribuida consiste en la invocación a métodos remotos o RMI[6] (Remote Method Invocation). Lo que este modelo pretende es acercar el modelo de programación centralizada de modo que invocar un método de un objeto que se encuentre en otra máquina sea similar a hacer una llamada a un método de un objeto local.

Para ello, el cliente utiliza un objeto *Proxy* con la misma interfaz que el objeto destino y que actúa de intermediario con éste último. El servidor por su parte utiliza un *Esqueleto* que traduce los eventos de la red a invocaciones sobre el objeto.

Hay cuestiones que deben ser transparentes para el programador. Un desarrollador debe dedicar sus esfuerzos a modelar el dominio de la aplicación y no a cuestiones tales como la codificación de los argumentos de llamada y de los datos de retorno por ejemplo. Tampoco puede manejar la problemática que plantean las redes.

La solución pasa por la introducción de una capa intermedia que permita al desarrollador abstraerse de este tipo de cuestiones. Esta capa se denomina **middleware** y básicamente ha de proporcionar un núcleo de comunicaciones genérico y un generador automático de proxies y esqueletos.

3.5.2. Middleware para Sistemas Distribuidos

A lo largo de los siguientes apartados se describirán algunos de los middleware más extendidos. Tras esta introducción general, uno en concreto, *Ice (Internet Communications Engine)* será analizado algo más a fondo.

[6]Relacionado con el término RPC, Remote Procedure Call

CORBA

CORBA [20], siglas de *Common Object Request Broker Architecture*, es uno de los middleware más conocidos y un estándar para el desarrollo de sistemas distribuidos que facilita la invocación de métodos remotos usando el paradigma de orientación a objetos. Fue desarrollado por el OMG *(Object Management Group)*, un consorcio formado por numerosas empresas del sector.

Posee su propio lenguaje de especificación, el IDL, que describe cómo serán llamados los servicios que un servidor ofrece a sus clientes. La compilación de esta interfaz producirá código para el cliente (proxy) y para el servidor (esqueleto) que permitirán al primero acceder a la funcionalidad del segundo. Existen implementaciones de CORBA para un gran número de lenguajes de programación.

Los ORB *(Object Request Broker)* son los canales de comunicación a través de los cuales los objetos se solicitan servicios unos a otros con independencia de su ubicación física. Esta comunicación se realiza mediante el IIOP *(Internet Inter ORB Protocol)* definido por OMG y basado en TCP/IP.

Del estándar CORBA existen numerosas implementaciones pero ninguna recoge la especificación completa ya que ésta es muy extensa. Además, y en parte debido a lo anterior, existen ciertos problemas de interoperabilidad debido a las numerosas extensiones propietarias existentes y a la falta de especificación de algunos aspectos clave.

DCOM

DCOM o Modelo de Objetos de Componentes Distribuidos *(Distributed Component Object Model)* es una extensión de un middleware denominado COM que introdujo Microsoft en 1993. DCOM pretendía solventar algunos problemas de COM como el relacionado con mecanismo de recolección de basura.

Presenta numerosos problemas de interoperabilidad con otros middleware, está destinado exclusivamente a sistemas de Microsoft y plantea problemas de escalabilidad.

DCOM fue hace unos años uno de los principales competidores de CORBA pero, desde la aparición de la plataforma .NET, ha ido cediendo interés en favor de ésta última que incluye el paquete *Remoting*.

.NET Remoting

Es un paquete de la plataforma .NET de Microsoft que implementa la invocación remota y que permite además la interoperabilidad con otros middleware como SOAP/Webservices o CORBA.

Sus principales características son que es independiente de la plataforma y del lenguaje de programación, cuenta con una máquina virtual altamente eficiente y su especificación es pública. Creada para sistemas Microsoft, actualmente hay dos proyectos de implementaciones libres (Mono y DotGNU).

Las comunicaciones se realizan a través de canales (análogos a los ORB de CORBA) que se encargan de establecer las conexiones y codificar argumentos y resultados. Estos canales son extensibles e incluso se pueden añadir nuevos que estén orientados a otras redes.

La programación con .NET Remoting es sencilla y su funcionamiento es análogo al de otras plataformas de RMI. Las aplicaciones se pueden desarrollar en cualquiera de los lenguajes de la plataforma .NET.

Java RMI

Tecnología que permite localizar y hacer referencia a objetos remotos creada por Sun Microsystems e implementada en el paquete java.rmi. Proporciona una solución que permite comunicar objetos que se encuentra en distintas máquinas pero a un nivel de abstracción bajo. No proporciona grandes funcionalidades adicionales. Tanto cliente como servidor deben estar implementadas en Java.

EJB

Los EJB *(Enterprise JavaBeans)* son una de las API que forman parte del estándar de construcción de aplicaciones empresariales J2EE de Sun Microsystems. Es, al igual que DCOM, un middleware basado en componentes distribuidos.

Los clientes acceden a los EJB por medio de un contenedor que se ejecuta en el servidor. El contenedor [Pol05] publica dos interfaces por cada componente: una interfaz *home*, que ofrece servicios de localización de instancias de componentes, y una interfaz *remota*, que ofrece métodos de negocio. Existen numerosas empresas que proporcionan contenedores de componentes como la propia *Sun*, *Oracle*, *iPlanet*, *IBM* o *Apache*.

Existen varios tipos de EJBs: *entity beans*, *session beans* y **message driven beans**.

Los primeros, los *entity beans* representan entidades que tienen existencia en algún

mecanismo de almacenamiento persistente. Las llamadas a estos componentes se realizan a través del contenedor.

Los *session beans* ejecutan servicios para el cliente, pero sin consideraciones acerca de su persistencia. Existen dos subtipos, *con estado* y *sin estado*.

Los EJBs de tipo *message driven* se utilizan para el envío y procesamiento de mensajes asíncronos. A diferencia de los otros dos tipos, estos no tienen interfaz *remota* ni *home*.

SOAP / Servicios Web

Mediante los Servicios Web [Pol05], un cliente puede ejecutar un método en un equipo remoto a través de protocolo HTTP. Su principal característica es la portabilidad que se consigue gracias a que todo el intercambio de información entre cliente y servidor se realiza en *SOAP (Simple Object Access Protocol)*, protocolo de mensajería basado en XML. Gracias a esto, cliente y servidor pueden estar implementados utilizando distintas plataformas.

Los servidores ofrecen una descripción de sus servicios web en *WSDL (Web Services Description Language)*, que es una representación en XML del servicio ofrecido. Así, un cliente puede conocer los métodos ofrecidos por el servidor, sus parámetros y los tipos de estos sólo con consultar el correspondiente documento WSDL.

3.5.3. ICE

ICE (Intenet Communications Engine) es un middleware orientado a objetos para entornos heterogéneos desarrollado por ZeroC [38]. Fue desarrollado por algunos de los más influyentes desarrolladores de CORBA como *Michi Henning* y su diseño está inspirado en este middleware.

En comparación con CORBA, estándar del que no existe una implementación completa por ser demasiado extenso, ICE es mucho menos complejo. Esto es en parte debido a que ha sido desarrollado por una pequeña empresa y no por un comité de estandarización. Una amplia comparativa entre ambos middleware se puede consultar en [10].

Algunas de las características de Ice son:

- Es escalable.
- Fácil de aprender y de usar.
- Posee un conjunto rico de capacidades.
- Tiene un buen soporte para la seguridad.

- Se distribuye bajo licencia GPL.
- Está disponible para un gran número de lenguajes y entornos como C++, Java, C#, PHP y Python.
- Soporta multitud de sistemas operativos como MS Windows, Mac OSX, GNU/Linux y Unix.
- Proporciona un conjunto de servicios avanzados como *Freeze*, para manejar la persistencia de los objetos, o *Glacier*, el firewall de Ice.
- Posee extensiones como *Ice-E* (Embedded Ice) para dispositivos móviles o *IceGrid*[7], un conjunto de herramientas para la construcción de sistemas grid.

Terminología de ICE

En ICE existen distintos elementos [Moy05]:

- **Cliente.** Es una entidad *activa* encargada de realizar peticiones.
- **Servidor.** Entidad *pasiva* (sirve peticiones de clientes).
  - Un servidor puede contener cero o más sirvientes proporcionando servicio.
- **Objeto.** Entidad *abstracta* que describe ciertas funcionalidades que ofrece un servidor.
  - Puede instanciarse en varios servidores.
  - Puede estar respaldado por cero o más sirvientes.
  - Implementan al menos un interfaz (el principal).
  - Poseen una identidad.
  - Su tiempo de vida puede exceder el tiempo de vida de los sirvientes que lo respaldan en el caso de los objetos persistentes.
- **Proxy.** Representa a un objeto en el cliente.
  - Puede representarse como una cadena de texto.
  - Puede ser directo o indirecto.
- **Sirviente.** Código que se ejecuta al invocar un método. Un sirviente puede ser instanciado en el momento de realizarse la invocación al método remoto.

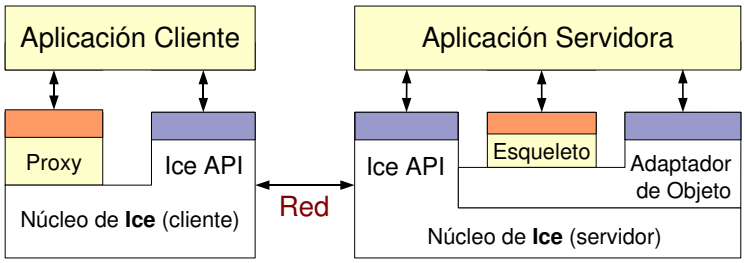

Figura 3.9: Estructura lógica de ICE.

La estructura de una aplicación cliente–servidor realizada con Ice se muestra en la Figura 3.9. La aplicación escrita por el desarrollador accederá al núcleo de comunicaciones de Ice a través de un API. Los proxies y esqueletos se generan automáticamente a partir de una descripción abstracta de los interfaces tal y como ocurría en CORBA.

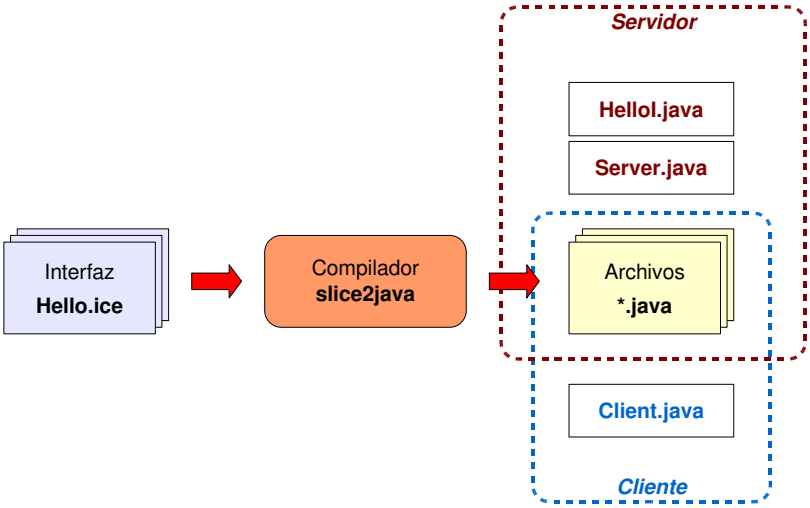

Figura 3.10: Ejemplo de slice2java.

SLICE

Es el lenguaje de especificación de Ice y permite separar interfaces e implementación y establece una especie de contratos entre servidores y clientes (las propias interfaces).

Es independiente del lenguaje de programación de la aplicación y existen compiladores que, a partir de una especificación en SLICE, generan las correspondientes clases en el

[7] Ice incluye IceGrid a partir de su versión 3.0.

```
struct TimeOfDay {
        short hour;
        short minute;
        short second;
};

interface Clock {
        nonmutating TimeOfDay getTime();
        idempotent void setTime(TimeOfDay time);
        void modifyAlarm(TimeOfDay alarm, out TimeOfDay prev_alarm);
        bool synchronize(Clock* my_clock);
};
```

Listado 3.5: Definición de una estructura y un interfaz en SLICE.

lenguaje deseado. Algunos de estos compiladores son *slice2java* (Java), *slice2py* (Python), *slice2cpp* (C++) y *slice2cs* (C#). El funcionamiento de todos ellos es similar y como resultado se obtienen la interfaz del objeto, el proxy, el esqueleto y una serie de clases auxiliares.

En la Figura 3.10 se muestra como, a partir de una definición en SLICE se obtienen las clases comunes a servidor y cliente mediante las cuales éste último podrá acceder a la funcionalidad del objeto Hello que reside en el servidor.

SLICE es puramente declarativo, lo que implica que sólo se pueden definir tipos e interfaces. En el Listado 3.5 se define la estructura *TimeOfDay* que será usada para definir la interfaz del objeto *Clock*.

3.6. LENGUAJES DE SCRIPT Y GLUE-CODE

El término *glue-code* hace referencia a los lenguajes utilizados para unir a modo de *pegamento* (de ahí el término) varias partes de un sistema. En Yafrid se han utilizado tres lenguajes de script para este tipo de tareas adicionales, además de algunos de los analizados en el Apartado 3.4 como JavaScript.

El primero, *Python*, que ha servido para implementar buena parte del sistema, ha servido también como *lenguaje puente* a la hora de acceder a las escenas de Blender. Este motor de render ofrece un API de programación para este lenguaje.

El segundo de ellos, *Bourne Shell*, ha servido para llevar a cabo algunas tareas en sistemas GNU/Linux relativos al control de procesos, a la automatización de tareas repetitivas, etc ...

Por último, se ha utilizado el lenguaje de script que incorpora la herramienta *Inno Setup* para definir las reglas para la creación de asistentes de instalación. El lenguaje en el que se escriben los scripts de Inno Setup es similar a Delphi.

3.6.1. Python

Python [26] es un lenguaje multiplataforma, interpretado, interactivo y orientado a objetos que combina una gran potencia y velocidad con una sintaxis muy sencilla.

Existen numerosos estudios que han comparado Python con otros lenguages con similares ámbitos de aplicación [9]. Las conclusiones con respecto a Python a las que este estudio ha llegado son:

- Escribir programas en Python requiere menos de la mitad de tiempo que en Java, C o C++.
- El código resultante es la mitad de largo.
- No hay diferencias en la fiabilidad de los programas.
- El consumo de memoria es alrededor del doble frente a C y C++. Java consume alrededor del doble que Python.
- El código en Python es más rápido que en Java.
- Los lenguajes de script más rápidos son Python y Perl.
- La variación en rendimiento de los programas se debe más a los programadores que a los lenguajes.

Suele ser usado como *lenguaje pegamento* debido a que permite un prototipado rápido, en parte porque es posible desarrollar operaciones complejas escribiendo poco código.

Las características de Python más importantes son:

- Es un **lenguaje de alto nivel** con la riqueza de lenguajes como Java o C++.
- Es **interpretado** (pseudo-compilado).
- Es un lenguaje **orientado a objetos** (en Python todo es un objeto).
- Es un lenguaje **extensible** que puede enriquecerse con nuevos tipos de datos.
- Es **multiplataforma** tanto a nivel de sistema operativo como a nivel de arquitectura hardware.
- Es un lenguaje **libre** (licencia GNU). Esto hace que sea ampliamente usado y que exista una gran cantidad de librerías disponibles desarrolladas por una comunidad creciente de programadores.

- Es **rápido** en cuanto al desarrollo (productividad) debido en parte a la sencillez de su sintaxis y a la flexibilidad de sus estructuras. También es rápido en términos de eficiencia.

Existen numerosas tecnologías para crear **interfaces de usuario** para programas escritos en Python. Las más importantes son:

- **GTK.** Es la librería de controles gráficos de GIMP. Es multiplataforma y permite la creación de interfaces en varios lenguajes. Posee además un constructor de interfaces (*Glade*) muy completo.
- **wxWidgets.** Tecnología de código abierto que permite crear interfaces de usuario para diversas plataformas (*win32*, *X11*, *Motif*, etc ...) con *look and feel*[8] nativo.
- **Tkinter.** Interfaz de desarrollo de interfaces gráficas de usuario con Tcl/tk.

Figura 3.11: Hola Mundo en Tkinter.

El módulo **Tkinter** (que proviene de *Tk interface*) es el interfaz estándar de Python para el conjunto de herramientas **Tk**. Esta tecnología permite desarrollar interfaces sencillas de usuario en Python con poco código y separando las capas de presentación y dominio. Aporta *look and feel* en las últimas versiones. Es el estándar *de facto* para desarrollar interfaces sencillas en Python (la mayoría de las distribuciones lo incorporan).

En el Listado 3.6 se muestra un sencillo script de ejemplo en Tkinter que tiene como resultado la creación de una ventana con dos botones. Haciendo clic en el que tiene el texto *Hello*, se mostrará por pantalla el texto *hi there, everyone!*. El otro botón sirve para cerrar la ventana. El resultado de la ejecución de este script da lugar a la ventana de la Figura 3.11.

Python, como ya se ha apuntado, posee un gran número de librerías especializadas en cierto tipo de funcionalidades. Así existen varios módulos para, por ejemplo, procesar ficheros comprimidos, para el acceso a base de datos o para el tratamiento de imágenes (como por ejemplo **PIL**, *Python Imaging Library* [24]).

[8] Así se denomina al aspecto global de una aplicación.

```
from Tkinter import *

class App:

    def __init__(self, master):
        frame = Frame(master)
        frame.pack()
        self.button = Button(frame, text="Quit", command=frame.quit)
        self.button.pack(side=LEFT)
        self.hi_there = Button(frame, text="Hello", command=self.say_hi)
        self.hi_there.pack(side=LEFT)

    def say_hi(self):
        print "hi there, everyone!"

root = Tk()
app = App(root)
root.mainloop()
```

Listado 3.6: Hola Mundo en Tkinter

Para facilitar la distribución de las aplicaciones escritas en Python existe el conjunto de herramientas **Distutils**. En concreto, para sistemas MS Windows existe **py2exe**, que usando la librería anterior, compila una aplicación Python generando ejecutables y *dlls*. De este modo, la aplicación funcionará en cualquier sistema Windows sin necesidad de tener instalado Python.

Capítulo 4

MÉTODO DE TRABAJO

4.1. ARQUITECTURA

El tipo de grid más adecuado para implementar la distribución del render de una escena entre múltiples máquinas es el **grid computacional**. Esto se deduce de las propiedades del proceso de render, caracterizado por ser un proceso muy intensivo en términos de consumo de CPU.

Conceptualmente, los componentes mínimos de un grid computacional son los siguientes:

- Distribuidor. Constituye el *corazón* del grid. Su principal tarea es la de tomar los trabajos de la cola y distribuirlos entre los proveedores de la forma más conveniente posible.

- Proveedor de servicio. Esta entidad es responsable del procesamiento real de las peticiones de los clientes.

- Cliente. Un cliente es una entidad externa que no pertenece al sistema en un sentido estricto. Su papel en el funcionamiento del grid pasa por enviar trabajos al sistema para ser realizados por los proveedores. Estos trabajos son almacenados en una cola de la cual el distribuidor irá eligiendo las próximas peticiones en ser procesadas.

Estos componentes interactúan en un ciclo que se muestra esquemáticamente en la Figura 4.1.

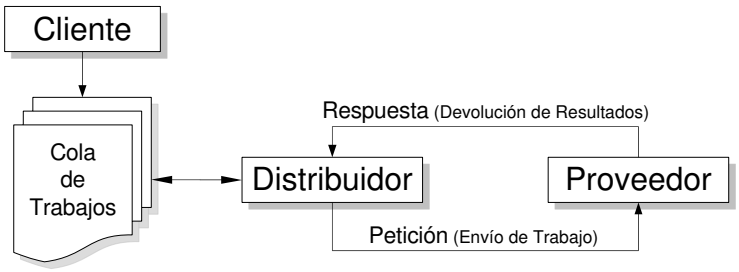

Figura 4.1: Ciclo de interacción en un grid computacional.

Debido a las especiales características del sistema a desarrollar, la arquitectura OGSA se hacía demasiado *pesada* para ser utilizada. Esta arquitectura es especialmente adecuada para sistemas grid formados por equipos que pertenecen al mismo de forma continua.

Si se aplicara a Yafrid, cada Proveedor debería tener implementadas cada una de las capas de la pila de la arquitectura con lo que el software resultante sería grande y difícil de configurar. En el caso de la distribución de trabajos a través de Internet entre máquinas que no están continuamente suscritas, es necesaria una aproximación algo más ligera y flexible. Esto se consigue utilizando un enfoque similar al que se sigue en proyectos como SETI@Home.

Esta decisión ha tenido una fuerte influencia en el desarrollo. Sin embargo, ha sido imprescindible teniendo en cuenta la necesidad de que los Proveedores (personas no vinculadas a ninguna organización y que proveen ciclos de CPU desde sus casas) contaran con un software lo más ligero y cómodo posible.

La principal desventaja es que la utilización de un middleware de más bajo nivel supone la implementación de servicios que usando productos como Globus Toolkit se tendrían resueltos.

La arquitectura desarrollada para soportar el sistema distribuido de render Yafrid se muestra en la Figura 4.2). A continuación se realiza una introducción general a esta arquitectura cuyos componentes serán ampliamente analizados en sus respectivas secciones.

4.1.1. Servidor Yafrid

El Servidor Yafrid es el nodo fundamental alrededor del cual se establece el sistema de render Yafrid. Cada uno de los proveedores se conecta a este nodo para hacer que sus ciclos de CPU puedan ser usados para renderizar las escenas enviadas al grid por los clientes.

El Servidor de Yafrid se ha desarrollado sobre una arquitectura en la que, de forma general, se pueden distinguir cuatro capas o niveles. Esta aproximación está ligeramente

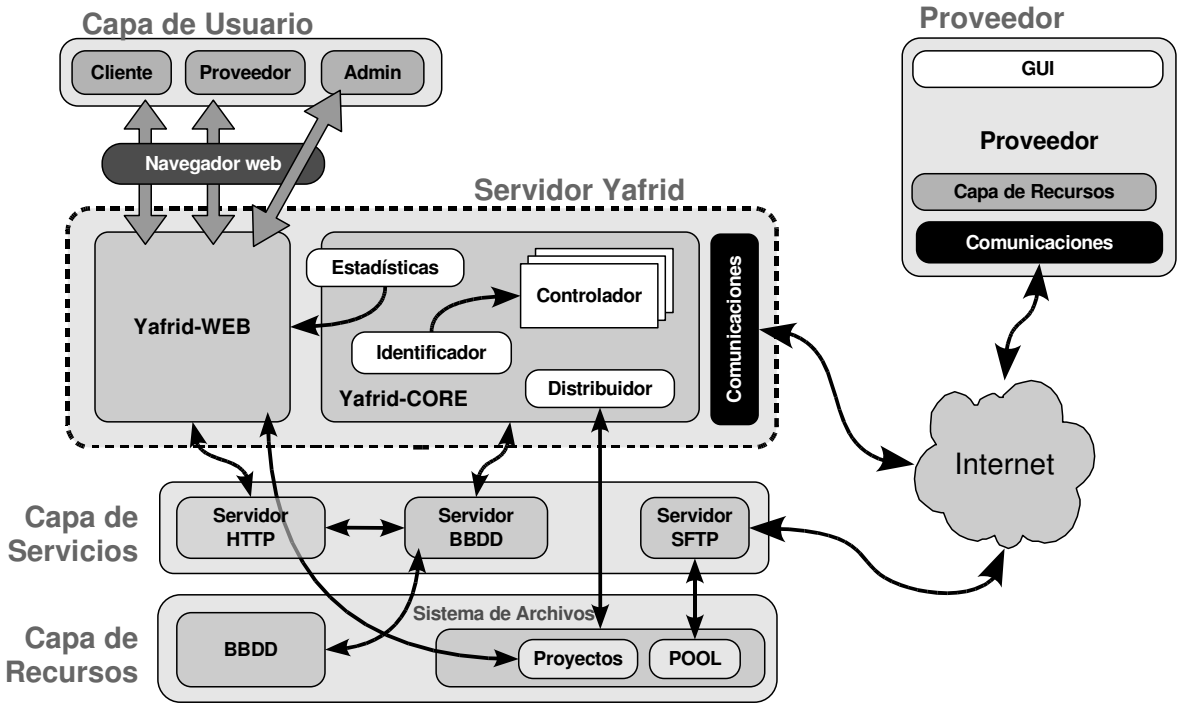

Figura 4.2: Arquitectura general del sistema Yafrid.

inspirada en la arquitectura que aparece en [FKT02]. Estas capas son, de menor a mayor nivel de abstracción, las siguientes:

- Capa de Recursos.
- Capa de Servicios.
- Capa Yafrid.
- Capa de Usuario.

Es en la tercera de estas capas donde el verdadero Servidor Yafrid reside. El resto de capas se consideran capas de soporte pero son igualmente indispensables para la implantación del sistema en un entorno real.

Capa de Recursos

Es la capa que tiene el nivel de abstracción más bajo y la que posee una relación más estrecha con el sistema operativo. La capa de recursos engloba los siguientes componentes:

- Sistema de Base de Datos. Formado por una base de datos en la que están contenidas las distintas tablas para el correcto funcionamiento del sistema. Existen numerosas tablas que se pueden agrupar atendiendo a su utilidad. Mientras que unas se usan para mantener datos a partir de los cuales se obtendrán estadísticas, otras mantienen los datos que ayudan a la gestión de usuarios, grupos, proyecto, etc. Hay un tercer grupo que podríamos llamar *internas* o *de operación* que mantienen datos imprescindibles para el funcionamiento del sistema como por ejemplo los datos de conexión de los proveedores o las características de sus equipos. Los componentes de capas superiores acceden a todos estos datos a través de un servidor de base de datos. La actual implementación de Yafrid utiliza MySQL.

- Sistema de Archivos. En ocasiones es necesario acceder al sistema de archivos desde capas superiores. Básicamente, se pueden distinguir dos tipos de directorios. Por un lado, hay directorios que se usan para almacenar las unidades de trabajo pertenecientes a un proyecto lanzado. Los proveedores accederán a este directorio vía SFTP para recibir los ficheros fuente necesarios para llevar a cabo el render. Estos directorios forman el llamado *POOL* de unidades de trabajo. La otra categoría la forman aquellos directorios que contienen la información sobre los usuarios y sus proyectos.

- Sistema de Red. El módulo dedicado a las comunicaciones que pertenece a la capa principal oculta la utilización de los recursos de red del ordenador por medio de un middleware (la implementación actual utiliza ICE).

Capa de Servicios

Básicamente, esta capa contiene los diferentes servidores que permiten a los módulos de capas superiores acceder a los recursos que pertenecen a la capa por debajo de esta. En este nivel se pueden encontrar los siguiente servidores:

- Servidor HTTP. El módulo Yafrid-WEB se establece sobre este servidor. Como el módulo Yafrid-WEB se ha confeccionado usando páginas dinámicas escritas en un lenguaje de script orientado al desarrollo web (en la implementación actual se ha utilizado PHP), el servidor web debe tener soporte para este lenguaje. También es necesario soporte para la composición de gráficos y para acceso a base de datos.

- Servidor de bases de datos. Los diferentes módulos de Yafrid usan este servidor para acceder a datos imprescindibles para la operación del sistema.

- Servidor SFTP. Accedido por los distintos proveedores de servicio para obtener los ficheros fuente necesarios para llevar a cabo los trabajos de render. Una vez que el render ha finalizado, se usa también el servidor SFTP para enviar de vuelta al sistema las imágenes resultantes.

Capa Yafrid

Constituye la capa principal del servidor y está compuesta básicamente por dos módulos diferentes que funcionan de forma independiente el uno del otro. Estos módulos se denominan Yafrid-WEB y Yafrid-CORE.

- **Yafrid-WEB.**

  Constituye el módulo interactivo del servidor y ha sido desarrollado como un conjunto de páginas dinámicas escritas usando HTML y un lenguaje de script orientado a desarrollo web.

  En términos de acceso al sistema, se han definido tres roles de usuario que determinan los privilegios de acceso de los usuarios a la intefaz web de Yafrid. Estos roles junto con las funcionalidades a las que dan acceso son:

  - **Cliente.** Tener este rol permite a un usuario el envío de trabajos de render al grid. Un cliente es también capaz de crear y gestionar grupos a los que otros clientes y proveedores se pueden suscribir. Cuando un proyecto se crea, se le puede asignar un grupo privado. En este caso, sólo los proveedores pertenecientes al mismo grupo pueden tomar parte en el render del proyecto.

    Además, se generan diversas estadísticas con la información relativa a los proyectos del usuario.

  - **Administrador.** Este usuario es imprescindible para la operación del sistema y tiene privilegios que permiten acceder a información y funcionalidades críticas que incluyen:

    - Acceso a la información de los usuarios del sistema (clientes y proveedores) y de los grupos de render existentes.
    - Creación de conjuntos de pruebas, un tipo especial de proyectos compuestos que están formados por múltiples proyectos individuales con ligeras diferencias entre unos y otros. Estos conjuntos de pruebas constituyen

una herramienta con la que estudiar el rendimiento del sistema y obtener conclusiones a partir de los resultados.

- Gestión de la parte no interactiva del sistema (Yafrid-CORE).

- **Proveedor.** El proveedor es el usuario que tiene instalado el software necesario para formar parte del grid. Los proveedores pueden acceder a su propia información y a las estadísticas de uso.

Los grupos de render mencionados tienen una especial importancia en un grid dedicado a tareas de render como es Yafrid. Con este sencillo mecanismo, los proyectos pueden ser creados dentro de un grupo teniendo la garantía de que los proveedores de ese grupo dedicarán sus esfuerzos al render de ese proyecto y no al de otros.

- **Yafrid-CORE.**

  Esta es la parte no interactiva del servidor. Este módulo ha sido principalmente desarrollado usando Python aunque también existen algunos scripts en Bourne shell para tareas de gestión. Está compuesto por dos módulos:

  - **Distribuidor.**

    Constituye la parte activa del módulo. Implementa el algoritmo principal destinado a realizar una serie de tareas clave, algunas de las cuales se introducen a continuación.

    - **Generación de unidades de trabajo.** Consiste básicamente en lanzar los proyectos activos que existen en el sistema generando las unidades de trabajo que sean necesarias. Las características de esta división dependen de los parámetros que el usuario introdujo al activar el proyecto.
    - **Asignación de las unidades de trabajo** a los proveedores. Esta es la tarea principal del proceso de planificación. El algoritmo tiene que tener en cuenta factores como el software que es necesario para renderizar la escena, el software instalado en los proveedores y la versión específica de éste o el grupo al que pertenecen proveedor y proyecto. Con toda esta información, y alguna más, el Distribuidor tiene que decidir qué unidad será enviada a qué proveedor.
    - **Composición de resultados**. Con los resultados generados por los diferentes proveedores, el distribuidor tiene que componer el resultado final. Este

proceso no es en absoluto trivial, en especial, en los render de escenas estáticas. Debido al componente aleatorio de los métodos de render basados en técnicas de Monte Carlo (como el Pathtracing), cada uno de los fragmentos puede presentar pequeñas diferencias cuando se renderizan en distintas máquinas. Por esta razón, es necesario suavizar las uniones entre fragmentos contiguos usando una interpolación lineal. En la Figura 4.3 (a la izquierda) se pueden observar los problemas al unir los fragmentos sin usar este mecanismo de suavizado.

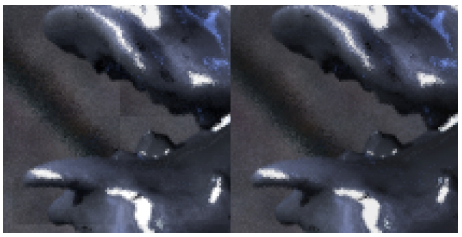

Figura 4.3: Artefactos sin interpolación (izqda) y usando interpolación lineal (drcha).

- **Finalización de proyectos.** Los resultados parciales almacenados por el sistema durante el proceso son eliminados del mismo.
- **Control del timeout**. Cuando el tiempo máximo que está establecido que se puede esperar por una unidad de trabajo es sobrepasado, el sistema consulta el estado de la unidad. Se comprueba si la unidad enviada a un proveedor está siendo aún procesada o no. Si una de las unidades se pierde, ésta tiene que ser reactivada para ser enviada a otro proveedor en un futuro.

- **Identificador.**

  Constituye la parte pasiva del módulo Yafrid-CORE cuya misión consiste en mantenerse a la espera de comunicaciones por parte de los proveedores. Este módulo es responsable del protocolo que se establece entre el servidor y los proveedores para garantizar el correcto funcionamiento del sistema. Se encarga de todas las comunicaciones con los proveedores excepto aquella en la que una unidad de trabajo es enviada a un proveedor, que es realizada por el distribuidor.

  La primera vez que un proveedor contacta con el servidor de Yafrid, el Identificador genera un objeto, el controlador, y devuelve al proveedor un proxy que apunta a este objeto. Cada proveedor posee su propio controlador. Las siguientes interacciones entre el proveedor y el servidor se realizarán a través del

controlador del proveedor. Algunas de estas interacciones representan operaciones como la identificación, la suscripción, la activación, la desconexión, el aviso de que una unidad ha sido terminada, etcétera.

4.1.1.1. Capa de Usuario

Esta capa no forma parte del sistema en un sentido estricto. Básicamente la constituyen los distintos tipos de usuarios que pueden acceder al sistema. Como ya se ha comentado, existen tres roles que caracterizan las tres categorías de usuarios de Yafrid:

- Proveedor de servicio.
- Cliente.
- Administrador de Yafrid.

Estos usuarios pueden acceder a toda la información ofrecida por el módulo Yafrid-WEB concerniente a su relación con el sistema utilizando cualquier navegador.

Esta caracterización de los distintos usuarios da lugar a algo similar a una sociedad en la que cada usuario tiene un papel determinado en la misma. En concreto, existen unos usuarios que proporcionan servicio a otros mientras que unos terceros gestionan el sistema.

Las características de cada uno de estos roles y de su interacción con el sistema serán abordadas en próximas secciones.

4.1.2. Proveedor

El Proveedor es el software que deben instalar en sus sistemas los usuarios que deseen prestar sus ciclos de CPU para que se usen en tareas de render. Este software puede funcionar tanto en modo de línea de comandos como por medio de una interfaz gráfica. El primer paso que un proveedor tiene que dar para formar parte del grid es conectarse al mismo. Una vez conectado y activado, el proveedor se pone a la espera de que el servidor le envíe una unidad de trabajo para ser procesada. Cuando finaliza el render, el proveedor envía los resultados de vuelta al servidor vía SFTP e informa a su controlador de que el trabajo está terminado.

4.2. ASPECTOS DE INGENIERÍA DEL SOFTWARE

En esta sección se identifican algunos de los aspectos generales relacionados con la ingeniería del software que deben ser considerados a la hora de realizar cualquier proyecto.

Las decisiones que han sido tomadas con respecto a estos temas, han sido determinantes para el desarrollo de Yafrid.

4.2.1. Ciclo de Vida

Según la definición que da la ISO 12207-1, el *ciclo de vida* del software es

> Un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto de software, abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso.

Para el desarrollo de Yafrid se ha utilizado el *Modelo de Ciclo de Vida en Espiral* introducido por Boehm [Boe88]. Este modelo establece la existencia de una serie de iteraciones tal y como muestra la Figura 4.4. Al final de cada una de estas iteraciones se obtiene un prototipo del sistema que responde a los objetivos planteados al comienzo de la iteración. Con cada iteración, el producto similar va siendo más similar al sistema final.

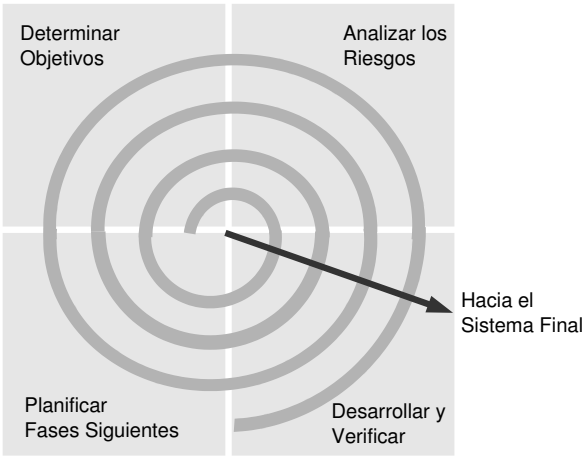

Figura 4.4: Ciclo de vida en Espiral.

Como se puede ver en la Figura 4.4, en cada una de las iteraciones se llevan a cabo cuatro tareas:

1. **Determinar Objetivos.** Además de éstos, se fijan los productos a obtener en la iteración. También se identifican las restricciones y las alternativas existentes y los riesgos asociados.

2. **Análisis de Riesgos.** Este análisis tiene como resultado la elección de una o varias de las alternativas propuestas para minimizar o evitar los riesgos identificados.

3. **Desarrollo y Verificación.** Se desarrolla el prototipo que cumple con los objetivos de la iteración.

4. **Planificación.** Se revisan los productos obtenidos, se evalúan y se planifica la siguiente iteración.

Las características [FM03] que lo hacen interesante y por las cuales se ha optado por este modelo son:

- Es **iterativo.** por lo que en cada iteración se obtiene un producto completo con cierto nivel de funcionalidad que se puede mostrar al cliente, en este caso el director del proyecto.

- Proporciona una fase de **gestión de riesgos.** en cada iteración con lo que los riesgos se identifican desde el principio. Especialmente importante debido a las características de distribución del sistema a desarrollar.

- **Flexible.** Se puede adaptar fácilmente a distintos tipos de desarrollo con la posibilidad de incorporar otros modelos.

4.2.2. Metodología

En Ingeniería del Software, y según [FM03], una *metodología* consiste en:

> Un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar nuevo software.

En Yafrid se ha seguido una metodología basada en el *Proceso Unificado* [JRB99] introducido por Jacobson, Rumbaugh y Booch utilizando UML. Con la utilización de esta metodología se ha permitido acercar el modelo de ciclo de vida al desarrollo orientado a objetos.

Además, tal y como las nuevas tendencias en ingeniería del software sugieren, el proceso se ha aligerado obteniéndose como resultado una metodología algo más *ágil*[1]. Las características de distribución, tolerancia a fallos, etcétera del sistema junto con cuestiones

[1] Ágil, en oposición a las que, como el Proceso Unificado, reciben el nombre de *metodologías pesadas*.

tales como el reducido tamaño del equipo de desarrollo o los cambios en los requisitos a lo largo del proyecto, hacen necesario el uso de estas prácticas más ligeras.

Como consecuencia, el volumen de artefactos generados en cada iteración no ha sido tan grande como en el caso de un Proceso Unificado puro. Los diagramas generados, por ejemplo, se han limitado a los imprescindibles en cada etapa.

Además, se han utilizado algunas prácticas específicas de metodologías ágiles como la *Programación Extrema* o *XP*[2] [Bec00] o el Desarrollo Dirigido por las Pruebas o TDD[3] [Pol05]. Algunas de estas adiciones han sido por ejemplo las *historias de usuario*, la generación de documentación a la vez que se escribe el código o el desarrollo a partir de casos de prueba.

Con respecto a las **herramientas**, que constituyen otro de los elementos de una metodología, se han utilizado de varios tipos. Por un lado, están las herramientas documentales como los generadores automáticos de documentación (*PyDoc* por ejemplo) o los propios diagramas UML que han servido para entender mejor algunas partes del sistema. En cuanto a las herramientas para el desarrollo, se han utilizado principalmente productos multiplataforma de carácter libre. Esto ha sido especialmente importante porque el desarrollo se ha realizado tanto en GNU/linux como en MS Windows, por lo que contar con la misma herramienta en uno y otro era imprescindible. Para el desarrollo se ha utilizado *Eclipse*, con sus extensiones *PHPEclipse* y *PyDev*, para PHP y Python respectivamente.

Entornos de prueba

Mención aparte merece la problemática que han planteado las pruebas del sistema. Como ya se ha comentado, un sistema distribuido plantea muchas complicaciones a la hora de ser probado. En Yafrid se ha contado con tres entornos de pruebas:

- Entorno Yafrid-Localhost (A). En este entorno se han realizado las pruebas más básicas. Entorno formado por un solo PC. No se pueden realizar todas las pruebas necesarias.
- Entorno Yafrid-LAN (B). Red de área local formada por cuatro ordenadores entre los que se podía establecer el sistema completo. Situación bastante ideal ya que no existen problemas debidos al retardo de la red por ejemplo.
- Entorno Yafrid-Internet (C). Entorno más cercano a la realidad. El Servidor se establece en un servidor del laboratorio del Grupo Oreto. Se pueden realizar todo tipo de pruebas

[2] Del inglés EXtreme Programming.

[3] Del inglés Test-Driven Development. Es, además de una metodología ágil, uno de los principios de XP.

en circunstancias realistas.

El proceso seguido consiste en ir pasando desde cada entorno hacia el siguiente en complejidad. En cada *pase* al entorno siguiente es frecuente que salgan a la luz problemas que no se daban en entornos anteriores.

Esto ha hecho que se dedique una gran parte del tiempo de desarrollo a la realización de pruebas unitarias, de sistema, etc, ... haciendo de éste, un tema crítico.

4.2.3. Iteraciones del Ciclo de Vida en Yafrid

Como se ha apuntado en el Apartado 4.2.1, en cada iteración del modelo de ciclo de vida en espiral se establecen una serie de elementos:

1. **Objectivos.** Consistentes en las funcionalidades que se busca implementar en cada iteración.
2. **Alternativas.** Las opciones que se consideran a la hora de cumplir los objetivos establecidos.
3. **Riesgos.** Cada una de las opciones consideradas puede tener asociado un cierto coste o riesgo que servirá a la hora de decantarse por una opción u otra.
4. **Resultados.** Conclusiones obtenidas en el proceso de análisis de los riesgos.
5. **Planes.** Planificación para la siguiente iteración.

El desarrollo del proyecto Yafrid ha pasado por más de diez iteraciones sucesivas a lo largo de las cuales, el sistema se ha ido refinando y completando hasta el estado actual. A continuación se detallan, a modo de ejemplo para ilustrar el proceso, tres de las iteraciones que se han llevado a cabo para desarrollar Yafrid.

- **Iteración 1**
  1. Objetivos
     a) Comunicación entre máquinas remotas.
  2. Alternativas
     a) Middleware específico para sistemas grid.
     b) Middleware para sistemas distribuidos.
  3. Riesgos

   a) Los middleware para grid son potentes y ofrecen una amplia variedad de servicios. Sin embargo, algunos presentan restricciones en cuanto a la plataforma y el lenguaje de programación. Software del proveedor muy pesado.
   b) El middleware para sistemas distribuidos no ofrece facilidades para sistemas grid por lo que hay que implementar los distintos servicios. Alguno de los estudiados son multiplataforma y multilenguaje. Mayor flexibilidad de desarrollo. Software del proveedor más ligero.

4. Resultados
   a) Utilización del middleware para sistemas distribuidos ICE.
5. Plan
   a) Desarrollo de protocolo de comunicación entre Servidor y Proveedor.

- **Iteración 2**

1. Objetivos
   a) Desarrollo de protocolo de comunicación entre servidor y proveedor.
2. Resultados
   a) Diseño de las interfaces que implementarán las clases cuyos objetos comunicarán los proveedores con el servidor mediante el lenguaje de especificación de interfaces SLICE.
3. Plan
   a) Implementación mínima del proceso Identificador.

- **Iteración 6**

1. Objetivos
   a) Internacionalización de la interfaz web del sistema.
   b) Internacionalización de la interfaz del software proveedor.
2. Alternativas
   a) En el caso de la interfaz web, almacenamiento de los textos en base de datos. Varios diseños de tablas son posibles: una tabla por idioma, tabla única en la que se añade la clave del idioma a cada texto, etc.
   b) Para ambos objetivos sería posible mantener los textos a traducir en variables almacenadas en distintos ficheros de texto plano para cada idioma.
   c) En el caso del interfaz del proveedor se puede optar por alguno de los paquetes que automáticamente generan un diccionario a partir de los textos a traducir.
3. Riesgos
   a) Almacenar todos los textos en la base de datos incrementa el tráfico en la base de datos y perjudica el rendimiento ya que para cargar cualquier página es necesario acceder muchas veces a la base de datos.

   b) Usar paquetes especializados como por ejemplo *gettext* para Python (que genera ficheros de traducción compilados) hace que estos ficheros no sean portables ni editables.

4. Resultados
   a) Utilización de ficheros de texto plano para la solución homogénea de ambos objetivos.
5. Plan
   a) Implementación del módulo de creación de proyectos.

4.3. GENERALIDADES SOBRE EL DISEÑO

4.3.1. Diseño Multicapa

A la hora de abordar la construcción de cualquier sistema software se hace evidente la necesidad de establecer una división entre lo que constituye la lógica de la aplicación del resto de elementos que constituyen el sistema. Esto es lo que se conoce como diseño multicapa de un sistema: distinguir una serie de capas a cada una de las cuales se le asigna un conjunto bien definido de responsabilidades.

Según Craig Larman [Lar99], las ventajas más importantes que aporta esta forma de diseño son:

- Facilita la **reutilización** ya que aísla la lógica de la aplicación en componentes separados del resto con lo que estas funcionalidades pueden ser utilizadas en el desarrollo de otros sistemas del mismo ámbito. De igual modo, los elementos de otras capas pueden ser reutilizados en otros desarrollos.

- Mejora la **distribución de los recursos humanos** ya que establece una división del trabajo en distintas áreas de conocimiento. Además, estas tareas se pueden realizar **paralelamente**. Así, los expertos en el dominio de la aplicación pueden dedicar sus esfuerzos a desarrollar la lógica del sistema mientras que los diseñadores gráficos confeccionan el interfaz de usuario.

- Un buen diseño hace que las modificaciones en unos elementos afecten lo mínimo posible a los restantes con lo que hace más fácil el **mantenimiento** de la aplicación.

Una aplicación que siga este tipo de diseño consta al menos de tres capas, cada una de las cuales agrupa una serie de elementos con unas características comunes. A esto se le conoce como *arquitectura de tres capas* y es la arquitectura multicapa más común.

- **Presentación.** Es en esta capa donde residen las ventanas (en el sentido más amplio del término) con las que el usuario interactúa.
- **Dominio.** En ocasiones también se suele hablar de capa de negocio o de procesamiento. Contiene aquellos elementos que representan la lógica necesaria para resolver el problema para el que el sistema se creó.
- **Persistencia.** Es la de más bajo nivel y la encargada de manejar los datos de la aplicación almacenados de forma persistente.

En la Figura 4.5 se muestra esta división y se establecen las dependencias deseables entre las tres capas. Lo ideal es que los elementos de la capa de presentación necesiten a los elementos de la capa de dominio pero no al contrario. Los elementos de las capas de dominio y presentación suelen depender unos de otros para su funcionamiento.

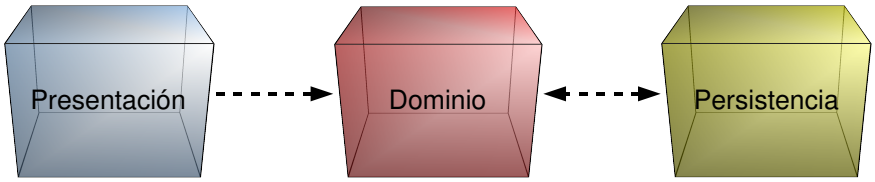

Figura 4.5: Diseño en tres capas.

Este es el esquema general y más básico; sin embargo, las posibilidades van mucho más allá. Estas capas—que en los lenguages orientados a objetos equivalen a paquetes de clases— pueden a su vez tener subcapas que se ocupen de aspectos más específicos. Del mismo modo, este diseño se puede extender con capas adicionales que se encarguen de otras funcionalidades, como por ejemplo de las comunicaciones con otras partes del sistema[4] o con otros sistemas.

Es esta última aproximación al diseño multicapa la que se ha utilizado en el desarrollo de Yafrid. Al ser un sistema formado por varios elementos muy diferenciados, en cada uno de ellos se ha utilizado una arquitectura multicapa adaptada a las necesidades de ese elemento. Así, habrá elementos que carezcan de capa de presentación propiamente dicha como Yafrid-CORE o que tengan alguna capa adicional. Todos estos aspectos serán abordados en las siguientes secciones de este documento.

[4]En Yafrid, se ha utilizado una división en cuatro capas en algunas partes del sistema. Esta cuarta capa se encarga precisamente de eso, de comunicar proveedores y servidor.

4.3.2. Patrones de diseño

En esta sección se hace recopilación a modo de resumen de los patrones que se han usado en el desarrollo de Yafrid. Estos patrones son más ampliamente analizados en las secciones correspondientes. Gamma, Helm, Johnson y Vlissides [GHJV03] establecen una distinción de los patrones por familias que será la que se siga en este apartado.

- Patrones estructurales. Permiten describir la forma de combinar estructuras y objetos con el fin de obtener estructuras más complejas.
  - Patrón **Compuesto** (*Composite*). Utilizado para diseñar la relación de un objeto compuesto y sus componentes, que a su vez pueden ser otros objetos compuestos.
  - Patrón **Proxy**. Un proxy, en el sentido más amplio, es un objeto que sustituye a otro. Las comunicaciones del sistema se sostienen sobre este patrón mediante el cual se puede acceder a la funcionalidad de un objeto remoto.
- Patrones de creación. Permiten abstraer el proceso de creación de instancias independizandolo del resto del sistema.
  - Patrón **Fábrica Abstracta** (*Abstract Factory*). Establece una jerarquía de clases abstractas que permite generar una familia de objetos que poseen una cierta relación.
  - Patrón **Singleton**. Permite garantizar la existencia de una sola instancia de una cierta clase.
- Patrones de comportamiento. Describen cómo se asignan las distintas responsabilidades dentro del sistema.
  - Patrón **Intérprete** (*Interpreter*). Se utiliza en los casos en los que se debe realizar el procesamiento de expresiones en un lenguaje sencillo.
  - Patrón **Estado** (*State*). El *estado* es un objeto en el que otros objetos delegan ciertos aspectos de su funcionalidad. El estado se instancia en una especialización concreta en función de las características del objeto al que da soporte.
  - Patrón **Fabricación Pura**. Las *fabricaciones puras* son clases en las que otras clases de dominio delegan las tareas relacionadas con la persistencia.
- Otros patrones que no pertenecen a ninguna de las familias y que también han sido usados son:

- Patrón **Agente de Base de datos** (*Database Broker*). Hace uso del patrón *Singleton* y se ocupa de la interacción con la base de datos exclusivamente.
- Patrón **Biblioteca** (*Library*). Las clases que implementan este patrón ofrecen una serie de funcionalidades por medio de un conjunto de métodos estáticos y no son instanciables.

4.4. ELEMENTOS BÁSICOS DE YAFRID

Existen algunos elementos que intervienen en diversos módulos del sistema y cuyo papel es importante tener presente antes de abordar el resto de secciones. Algunos de estos elementos como son los proyectos, los lanzamientos y las unidades de trabajo tienen una estrecha relación.

En la Figura 4.6 se muestra que de cada proyecto puede haber múltiples lanzamientos con variaciones en los parámetros de render entre unos y otros. Puede haber incluso un proyecto sin lanzamientos asociados debido a que el proyecto aún no haya sido activado para ser planificado por el sistema en ninguna ocasión. Cada lanzamiento genera un número determinado de unidades de trabajo en función de los parámetros con los que se lanzó.

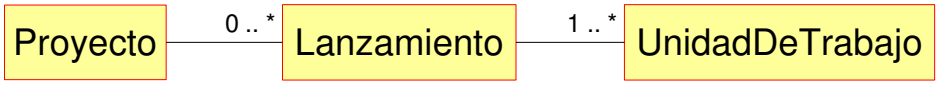

Figura 4.6: Relación entre proyectos, lanzamientos y unidades de trabajo.

Proyecto

Se denomina proyecto a una entidad creada por un cliente que representa un trabajo de render.

Los proyectos son creados a través del **interfaz web** del sistema a partir de un archivo comprimido. Este archivo contiene los llamados **ficheros fuente**. Como mínimo habrá un fichero que describa la escena y que tendrá un formato distinto en función del motor de render. En algunos casos, pueden ser necesarios ficheros adicionales con imágenes con las texturas aplicadas a un objeto por ejemplo. Al crear un proyecto, el cliente también le asigna un nombre, y un grupo.

Nombre	Motor de render	Tipo
yafray	Yafray	estático
blender-static	Blender	estático
blender-animation	Blender	animación

Cuadro 4.1: Tipos de proyectos en Yafrid.

Cuando el proyecto sea procesado en su totalidad, el cliente podrá obtener los resultados desde el propio interfaz web, denominados **ficheros de salida**.

Los proyectos se pueden clasificar por el motor de render con el que se obtendrán los resultados (y que determina el tipo de fichero a subir) o en función si se trata del render de una escena estática o de una animación.

Actualmente, Yafrid soporta tres tipos de proyectos tal y como se muestra en el Cuadro 4.1. El sistema se puede ampliar añadiendo nuevos tipos de proyectos como se verá en secciones sucesivas.

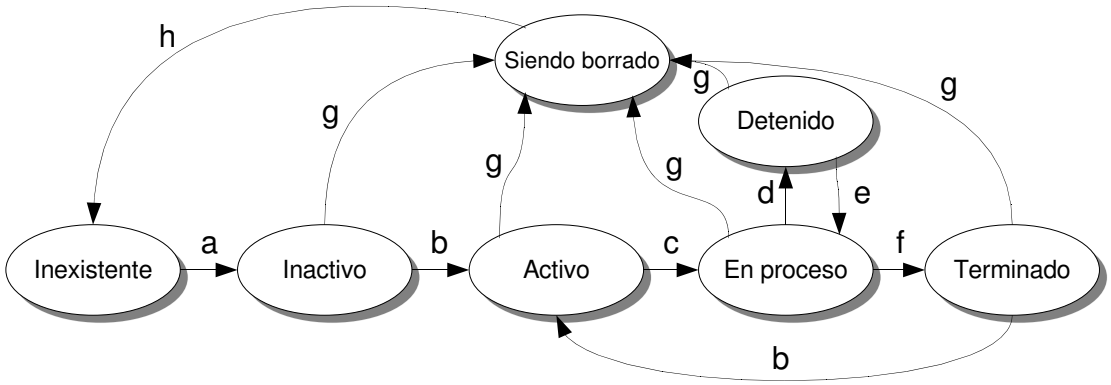

Figura 4.7: Estados de un proyecto en Yafrid.

Los proyectos de un cliente pueden encontrarse en distintos estados dependiendo del momento de su *vida* en el sistema en el que se encuentren. En la Figura 4.7 se muestran las distintas transiciones entre los estados de un proyecto. Estas transiciones se describen a continuación:

- **a, Creación.** Cuando un determinado cliente crea un proyecto desde el interfaz web pasa a estar en estado Inactivo.
- **b, Activación.** El cliente propietario de un proyecto lo activa cuando decide permitir

que el sistema lo procese con lo que se marca como planificable por el grid. En caso de que el proyecto esté en estado Terminado, se habla de reactivación y relanzamiento.

- **c, Lanzamiento.** El sistema ha procesado el proyecto y ha generado las unidades de trabajo necesarias que están listas para ser enviadas a los Proveedores.

- **d, Pausa.** El usuario pausa el proyecto desde el interfaz web con lo que ninguna unidad más será enviada a los Proveedores.

- **e, Reanudación.** Las unidades del proyecto vuelven a ser planificables cuando el usuario reanuda el proyecto.

- **f, Terminación.** Todas las unidades de trabajo del proyecto han sido correctamente realizadas.

- **g, Eliminación.** El usuario puede eliminar un proyecto que esté en cualquier estado. El proyecto no se eliminará hasta que no se determine la localización de las unidades de trabajo.

- **h, Borrado.** Una vez que todas las unidades enviadas han sido recuperadas, el proyecto se borra finalmente del sistema.

Un caso especial de proyecto lo constituyen los proyecto múltiples o *conjuntos de prueba* que han sido creados a partir de los mismos ficheros fuente pero que son diferentes en ciertos parámetros.

Granularidad

La granularidad es la forma en la que los proyectos se dividen para obtener las unidades de trabajo.

Yafrid soporta dos tipos de granularidad:

- **Granularidad fina.** En Yafrid esta granularidad se usa cuando un frame se divide en fragmentos más pequeños, cada uno de los cuales constituye una unidad de trabajo. También se puede denominar *intraframe*.

- **Granularidad gruesa.** Las unidades de trabajo son cada uno de los frames de una animación. También recibe el nombre de granularidad *interframe*.

Lanzamiento

Se denomina lanzamiento de un proyecto a cada una de las entidades que se generan cada vez que se activa o reactiva un proyecto. Se caracteriza por el proyecto del que proviene, del que toma los ficheros fuente, y por los parámetros de lanzamiento como por ejemplo las dimensiones, la calidad o el tamaño de la unidad de trabajo. Estas características son las que distinguen un lanzamiento del resto de lanzamientos del mismo proyecto.

Unidad de trabajo

El concepto de unidad de trabajo (o workunit) es fundamental para el funcionamiento del grid. La unidad de trabajo es la unidad mínima de distribución de Yafrid y constituye una tarea atómica para un Proveedor, es decir, se procesa completamente o no se procesa.

Se caracteriza por el fichero fuente que describe la escena y por unos parámetros que distinguen unas unidades de trabajo de otras. En el caso de proyectos de animación, el parámetro será el número de frame a renderizar mientras que en los proyectos de render estáticos los parámetros determinarán las coordenadas del fragmento a renderizar.

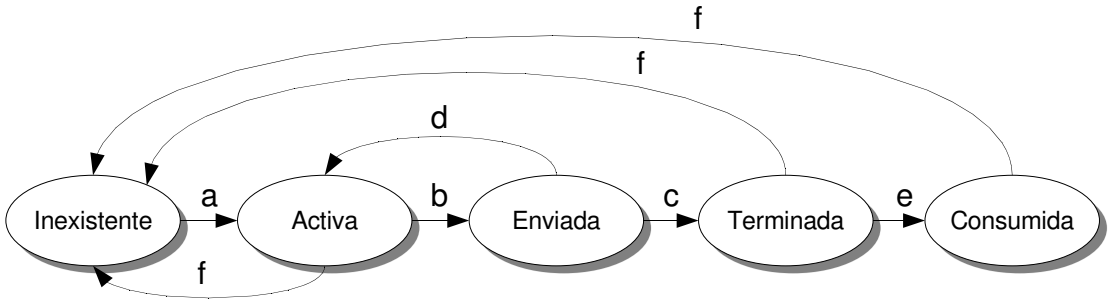

Figura 4.8: Estados de un unidad de trabajo en Yafrid.

En la Figura 4.8 se muestra un diagrama con los estados en los que se puede encontrar una unidad de trabajo a lo largo de su vida en el grid. Las transiciones entre estos estados se definen a continuación.

- **a, Activación.** Cuando un proyecto se lanza—y se genera un objeto lanzamiento como resultado—las unidades de trabajo asociadas se crean en el estado Activa.
- **b, Envío.** El Distribuidor envía a un Proveedor una unidad de trabajo para ser procesada.

- **c, Terminación.** El Proveedor que estaba procesando la unidad de trabajo, la termina correctamente y devuelve los resultados al servidor.
- **d, Reactivación.** Si un Proveedor pierde la unidad de trabajo que le fue enviada, ésta es activada de nuevo.
- **e, Consumo.** Los resultados asociados a la unidad de trabajo han sido procesados por el servidor.
- **f, Borrado.** Todas las unidades de trabajo de un proyecto se eliminan junto a éste.

4.5. PERSISTENCIA

La capa de persistencia se ocupa del manejo de los datos de la aplicación, tanto los que se almacenan en la base de datos como los que se guardan en ficheros. En esta sección se abordarán temas relacionados con el acceso a base de datos comunes a los módulos Yafrid-CORE y Yafrid-WEB (el módulo Proveedor carece de base de datos).

Las particularidades de cada módulo acerca de la persistencia y las cuestiones sobre el acceso a archivos serán tratadas con detalle en las secciones que correspondan.

4.5.1. Construcción de la base de datos

Uno de los principios del paradigma de orientación a objetos establece la unificación del modelado de datos y procesos. Mientras que en el paradigma estructurado se trabajaba por separado, modelando los procesos con *diagramas de flujo de datos* (DFD) y los datos mediante *diagramas E/R* o *E/ER*. En orientación a objetos es común utilizar el diseño de la capa de dominio como esquema conceptual de la base de datos en la que se almacenarán las instancias de las clases persistentes.

Existen varios patrones [Pol05] que facilitan la construcción de una base de datos relacional a partir del diagrama de clases de un sistema. En Yafrid se han combinado tres de ellos para distintas partes del diseño:

- Patrón **una clase, una tabla (1C1T).** Mediante este patrón se construye una tabla por cada clase (y con los mismos atributos que ésta). Las relaciones de herencia se transforman en restricciones de clave externa 1 a 1. Las relaciones de asociación y agregación se traducen en restricciones de clave externa cuya cardinalidad depende de la multiplicidad de la asociación o agregación.

- Patrón **un árbol de herencia, una tabla (1AH1T).** Consiste en representar una generalización de clases con una sola tabla como muestra la Figura 4.9. Lo habitual es utilizarlo cuando hay poca diferencia entre las clases hijas a nivel de datos. En estos casos se utiliza una tabla común para almacenar las instancias de ambas clases.

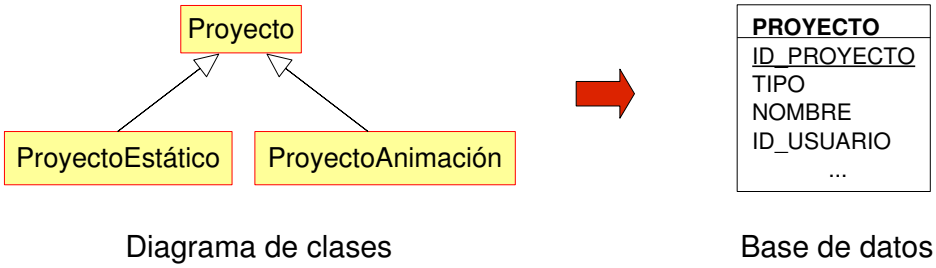

Figura 4.9: Transformación de un árbol de herencia en una tabla.

- Además, para las relaciones denominadas *de muchos a muchos* se ha realizado la transformación que se describe en [CGR05] y que consiste en la creación de una tabla intermedia que asocie las claves primarias de las tablas que representan a las clases relacionadas (Figura 4.10).

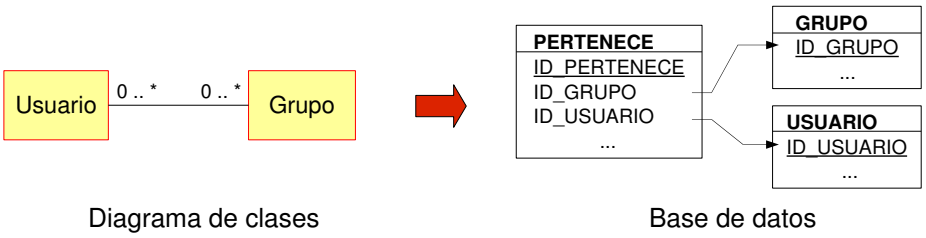

Figura 4.10: Transformación de relaciones de muchos a muchos.

4.5.2. Clases de persistencia

Las clases cuyas instancias se guardan en un sistema de almacenamiento persistente, como una base de datos o un fichero, reciben el nombre de *clases persistentes*. Para gestionar la persistencia en el sistema de las instancias de este tipo de clases se utiliza un conjunto mínimo de operaciones denominadas *operaciones CRUD* [5]:

[5] Otro nombre para el mismo concepto es *BREAD*, **B**rowse **R**ead **E**dit **A**dd **D**elete.

- **C**reate. Operación que almacena una instancia en la base de datos. Al proceso se le denomina *desmaterialización*. Si la base de datos es de tipo relacional, la operación corresponde con un INSERT de SQL.

- **R**ead / **R**etrieve. Al proceso se le denomina *materializar* y consiste en la creación de una instancia de una clase partiendo de la información contenida en la base de datos. En SQL corresponde a una operación SELECT.

- **U**pdate. Se usa para actualizar el estado[6] de una instancia en la base de datos. Corresponde al UPDATE de las bases de datos relacionales.

- **D**elete / **D**estroy. Sirve para eliminar de la base de datos los registros asociados a una instancia. Equivalente al DELETE de SQL.

Además, en las clases persistentes usadas en Yafrid se han incluido otras dos operaciones. Por un lado está la operación estática **search** que devuelve una lista de objetos cuyas características corresponden al patrón que se le pase como primer argumento. El segundo argumento indica el orden en el que se devolverán. Ambos argumentos siguen la sintaxis de las cláusulas *WHERE* y *ORDER* de SQL. Además de ésta, se ha incluido otra operación, **newId**, que obtiene un nuevo identificador para el objeto. Este identificador será el siguiente al máximo de los que haya en la tabla correspondiente.

Para el diseño y construcción de esta capa se han utilizado varios patrones que se describen a continuación:

- Patrón ***Agente de Base de Datos.*** Un agente de base de datos es una clase que se encarga de proporcionar acceso a base de datos a las clases de dominio de una aplicación consiguiendo cierto desacoplamiento. Hay diversas formas de implementarlo pero la más común es crear la clase usando el patrón *Singleton*. De esta forma, un agente poseería una serie de métodos públicos que mapearían las operaciones básicas sobre la base de datos (SELECT, INSERT, UPDATE y DELETE) y una instancia estática que representaría la conexión con la base de datos. La ejecución del constructor del agente o del típico método *getInstance()* devolvería la única instancia de la clase.

- Patrón **Fabricación Pura.** Este patrón relaciona las clases de dominio y persistencia y es una de las muchas formas[7] de que disponemos a la hora de asignar responsabilidades

[6] El conjunto de valores de los atributos de una instancia forman su estado.

[7] Además del patrón Fabricación Pura, existen otros modos de implementar la persistencia como pueden ser el patrón *Experto* o el patrón *RCRUD*.

a las clases del sistema.

Así, para cada clase de dominio *Clase* que necesite ser almacenada persistentemente, existirá una segunda clase *FPClase* que se encargará de la persistencia de la primera a través del agente de base de datos. Un caso real del sistema Yafrid se puede ver en la Figura 4.11.

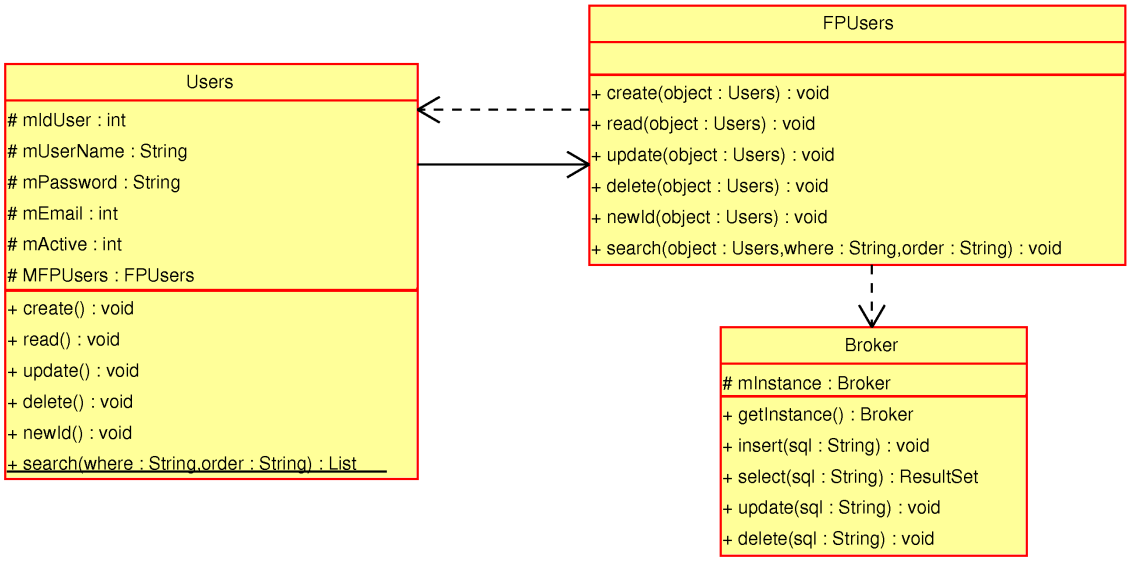

Figura 4.11: Clase *Users* y su fabricación pura, *FPUsers*.

4.5.3. Generación automática de clases

El sistema está desarrollado usando varios lenguajes de programación: Yafrid-WEB eminentemente en PHP y Yafrid-CORE y el Proveedor en Python. Además, la mayor parte de las clases de dominio y persistencia que son necesarias en uno de los módulos también lo son en el otro. Debido a estas circunstancias sería necesario escribir cada clase dos veces, una vez en PHP y otra en Python. A esto hay que unir el problema de mantener sincronizadas ambas clases cada vez que se realiza un cambio en la base de datos o en otra clase de dominio.

Para solventar esta situación en la medida de lo posible, se ha desarrollado una herramienta denominada **gendb**. Esta utilidad genera, a partir de la definición de las tablas, sus respectivas clases de persistencia junto con sus fabricaciones puras (Figura 4.12). Además, estas fabricaciones puras hacen uso de un agente para manejar la persistencia con lo que los productos obtenidos son coherentes con el diseño del sistema.

Con esta utilidad, tenemos un medio para tener sincronizadas las clases de persistencia con las tablas relacionales a las que representan. Además, eliminamos la necesidad de tener

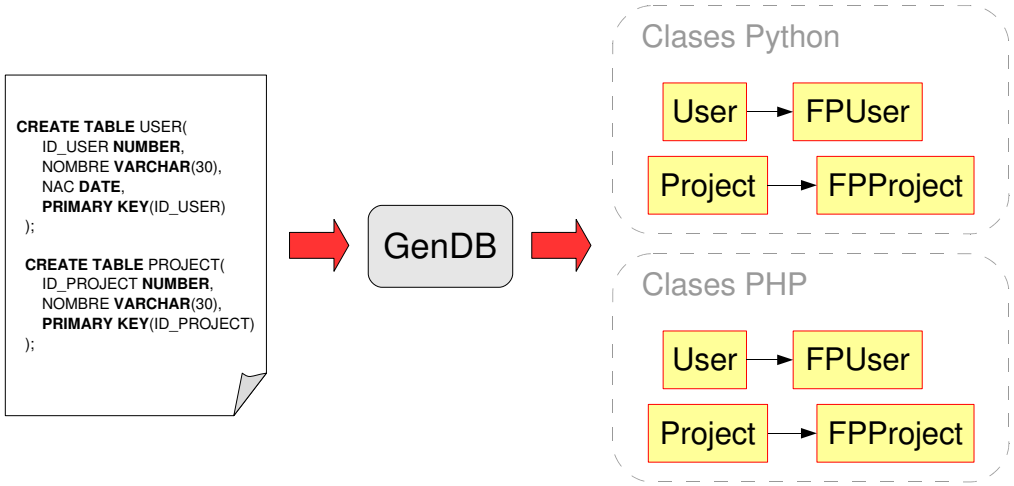

Figura 4.12: Esquema general del generador de persistencia *gendb*.

que escribir dos veces cada una de estas clases. Con gendb ya no es necesario escribirlas con lo que se evita un proceso que de otra forma sería bastante repetitivo.

El diseño de esta herramienta permite dividir el proceso en dos etapas:

- **Análisis.** El generador procesa el conjunto de tablas y crea un *árbol* que representa dicho conjunto. Es una representación independiente del lenguaje de salida.
- **Generación.** El árbol obtenido en la fase anterior se *decora* con los elementos típicos de cada lenguaje. Tras esto se generan los ficheros fuente resultado.

Este generador de clases de persistencia se ha aplicado a aquellas clases para las que el adecuado es el patrón 1C1T.

Para la construcción de esta aplicación se han utilizado algunos patrones, cuya relación y significado se mostrará en el Anexo A, dedicado a los diagramas.

- Patrón **Intérprete.**
- Patrón **Estado.**
- Patrón **Fábrica Abstracta.**

4.5.4. Tecnología de base de datos

La base de datos que se ha utilizado para almacenar la información del sistema es MySQL. Las razones para esta decisión son simples:

- Tanto PHP como Python, poseen mecanismos eficientes y sencillos para comunicar las aplicaciones desarrolladas y las bases de datos MySQL.
- MySQL es una base de datos eficiente, sencilla y multiplataforma.
- La mayoría de servidores tienen soporte para bases de datos MySQL (característica que finalmente carece de importancia a la luz de los resultados obtenidos en el Apartado 5.2).

El diseño realizado permitiría, si fuera necesario, adaptar fácilmente el sistema para que usase otras bases de datos. Los cambios que habría que realizar se localizarían en las únicas clases que tratan directamente con la base de datos, es decir, el Broker de Yafrid-WEB y el de Yafrid-CORE.

4.6. COMUNICACIONES

En Yafrid existe una comunicación entre varios componentes del sistema que están localizados habitualmente en máquinas distintas. En concreto, el Servidor y los Proveedores han de comunicarse para el funcionamiento del sistema.

Para implementar las comunicaciones, se ha hecho uso del middleware para sistemas distribuidos ICE (Internet Communication Engine). El cliente de un objeto remoto accede a la funcionalidad de éste a través de un objeto Proxy con la misma interfaz que el objeto y que actúa como intermediario.

El paquete Python que implementa la comunicación entre las distintas partes del sistema se llama *comm* y es compartido por las aplicaciones Servidor y Proveedor.

El middleware para sistemas distribuidos ICE, a partir de su última versión (la *3.0*), incorpora una serie de herramientas para la construcción, mantenimiento y monitorización de sistemas grid denominada IceGrid. Cuando se construyó la parte de Yafrid dedicada a las comunicaciones se utilizó la versión de ICE disponible en ese momento, la *2.1.2*, que no incorpora este nuevo componente para sistemas grid.

4.6.1. Creación de proxies

Las interfaces de las clases cuyos objetos van a ser accedidos remotamente han de ser definidas mediante el lenguaje *SLICE*. En el Listado 4.1 se muestra un ejemplo de

```
module comm{
    interface Provider{
        bool process(WorkUnit wu);
        IntArray getCurrentWorkUnit();
    };
};
```

Listado 4.1: Definición de la interfaz del objeto comm.Provider en SLICE

la definición de una interfaz con SLICE, en concreto de la clase *Provider* del paquete de comunicaciones.

A partir de esta definición de la interfaz, la aplicación *slice2py* [8] generará una serie de clases en Python que implementarán la comunicación con el objeto comm.Provider (Figura 4.13).

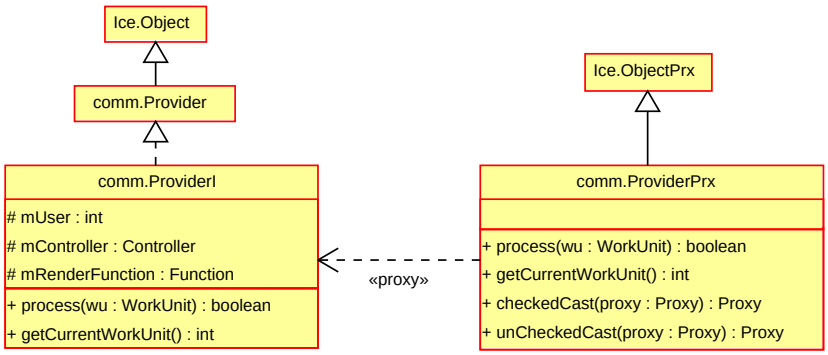

Figura 4.13: Proxy de la clase Provider.

Como resultado se generan las clases comm.Provider, que hereda de Ice.Object, y comm.ProviderPrx, que hereda de la clase Ice.ObjectPrx. La clase comm.ProviderI contiene la implementación del objeto y es la única que ha de ser desarrollada. En Yafrid, es el Distribuidor quien utiliza la funcionalidad de la clase comm.Provider. Para llamar a sus funciones, el Servidor tiene que conocer la clase comm.ProviderPrx—que describe estas funciones—mientras que la clase comm.ProviderI—que es quien las implementa—se encuentra en cada Proveedor. Para instanciar un proxy a un objeto remoto se utiliza el método *checkedCast()* o el *unCheckedCast()* pasándole como argumento la versión textual del proxy. Un proxy para un Proveedor podría ser:

Proveedor -t:tcp -h 127.0.0.1 -p 9995

[8] En la distribución de ICE se incluyen herramientas para generar estas clases en otros lenguajes.

que contiene todo lo necesario para contactar (el identificador del objeto, el protocolo de transporte, la IP y el puerto).

La definición de las interfaces de los objetos de comunicación y de las clases auxiliares se encuentran en el fichero *comm.ice*.

4.6.2. Protocolo de comunicaciones

Las distintas interacciones que se producen entre el Servidor y el Proveedor dan lugar a un protocolo de comunicaciones que se describe a continuación.

- En el primer paso de su interacción con el Servidor, el Proveedor accede a un servicio que posee el Identificador llamado Generador de Controladores que se encarga de asignar un único Controlador a cada Proveedor. Este objeto será con el que el Proveedor realice las sucesivas interacciones con el Servidor. Una vez creado, el Controlador permanece a la espera de las peticiones del Proveedor al que está asignado. El Proveedor se comunicará con este objeto a través de su proxy (devuelto por el Identificador). Comm.ControllerGenerator.GETCONTROLLER.

  La instanciación del proxy del Generador se realiza a partir de la versión textual en la que la IP del Servidor está parametrizada y el puerto es el 9999 que es en el que escucha el Identificador.

- El Proveedor, a través de su Controlador realiza una serie de acciones destinadas a hacer que el Proveedor pase a formar parte del grid. En todas estas comunicaciones, el Controlador actúa como un servidor en el sentido de que ofrece una serie de servicios. Estos servicios son:

  - Identificación (comm.Controller.IDENTIFY).
  - Suscripción (comm.Controller.SUBSCRIBE).
  - Comprobación de unidades de trabajo (comm.Controller.CHECKWORKUNITS).
  - Activación (comm.Controller.ACTIVATE).

- Si todas las acciones de autentificación se han llevado a cabo correctamente, el Proveedor pasa a modo servidor y se pone a la espera de recibir trabajos. En esta fase el Proveedor ofrece dos servicios:

  - Procesar una unidad de trabajo (comm.Provider.PROCESS).

  - Consultar unidad de trabajo (comm.Provider.GETCURRENTWORKUNIT).

- En un momento dado, el Distribuidor selecciona un Proveedor en concreto y le envía una unidad de trabajo para ser procesada. Para instanciar el proxy de un Proveedor utiliza la versión textual almacenada en base de datos en la tabla PROVIDERS.

  En ocasiones también se comunica con el Proveedor para consultar la unidad de trabajo que está procesando. Esto se hace para comprobar si el Proveedor sigue procesando la unidad de trabajo o se ha perdido por algún motivo.

- Cuando termina de procesar la unidad de trabajo avisa a su controlador de que ha acabado (comm.Controller.ENDEDWORK).

- Cuando un Proveedor quiere dejar de procesar trabajos provenientes del grid, se desconecta del mismo (comm.ControllerGenerator.BYE).

4.7. GESTIÓN DE PROCESOS

En Yafrid, en ocasiones es necesario manejar procesos. Sin ir más lejos, los propios Identificador y Distribuidor que funcionan a modo de demonio trabajando en segundo plano constituyen procesos. También existen procesos asociados al Proveedor: cuando se realiza el render, en realidad se lanza un proceso para gestionar esta tarea.

Mientras que para sistemas de tipo UNIX existen funciones en Python que realizan las operaciones básicas sobre procesos, estas funciones no existen o no tienen una funcionalidad completa en sistemas MS Windows.

En sistemas tipo UNIX es posible usar las funciones del paquete *os* de Python destinadas a la gestión de procesos:

- **spawnvp(modo, comando, args)**. Ejecuta *comando* en un subproceso pasándole *args* como argumentos. Si el *modo* es P_NOWAIT devuelve el PID del proceso.

- **waitpid(pid, opciones)**. Espera a la terminación del proceso con identificador *pid*. *Opciones* ha de ser cero para que el proceso se quede bloqueado esperando.

- **kill(pid, señal)**. Mata el proceso identificado por *pid* con la señal *señal*. *Señal* será la constante *signal.SIGTERM*.

En sistemas MS Windows se hace uso de la extensiónn de Python para win32 para el manejo de procesos ya que las anteriores funciones no manejan correctamente los PID y tienen algunas carencias de funcionalidad. Se han desarrollado funciones similares a las anteriores utilizando los módulos win32process, win32con, win32event, win32api y win32pdhutil que proporciona esta extensión.

Para ocultar las diferencias entre unos sistemas operativos y otros a la hora de gestionar procesos se ha implementado un módulo Python, denominado *processes.py* que pertenece al paquete *domain*.

En la Figura 4.14 se describe la estructura del módulo dedicado a la gestión de procesos. Existe una clase común, *Process*, que representa un proceso y que, en función del sistema operativo, usa un tipo u otro de operaciones. Esto se ha conseguido con el patrón *estado* materializado en la clase *Type* y sus especializaciones que es donde realmente residen las diferencias. Cada proceso conoce una instancia de la clase Type que será de una subclase u otra dependiendo del sistema operativo. La clase PidFile del paquete de persistencia se utiliza para mantener temporalmente el PID de los procesos en ficheros. Estos ficheros se encuentran en el directorio *sys/proc* y tienen extensión *pid*.

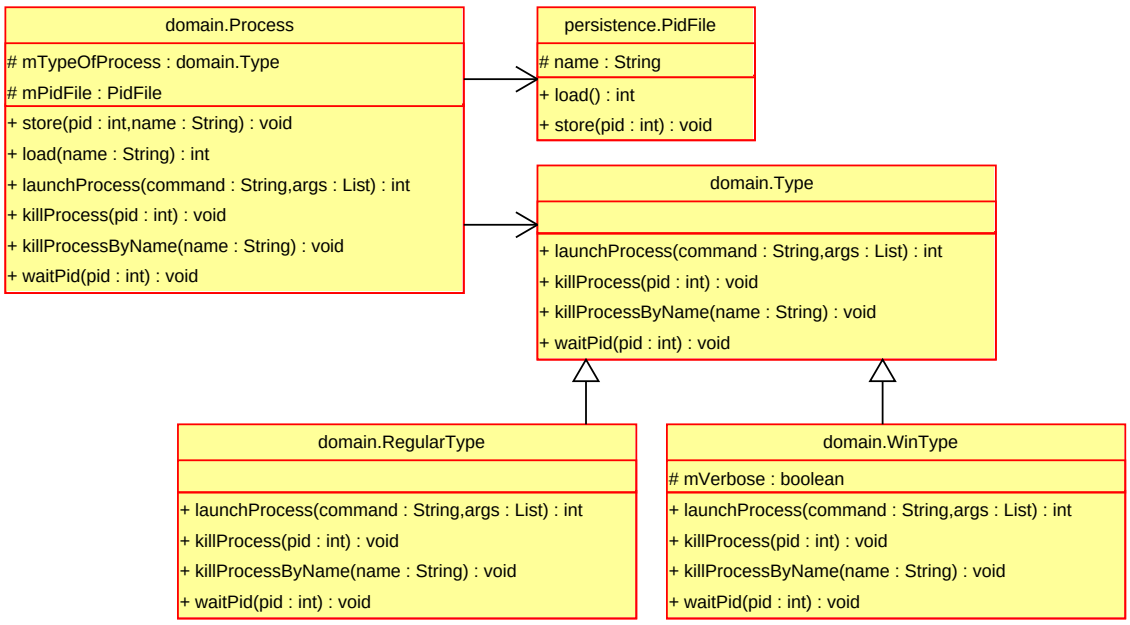

Figura 4.14: Clases del módulo de procesos.

Los procesos creados con la clase Process pueden ser de dos tipos:

- **De depuración.** El parámetro *verbose* está a verdadero con lo que se muestra

información sobre las acciones que va realizando el proceso mediante la salida estándar.

- **Finales.** Carecen de información de depuración. Los errores que puedan surgir serán volcados a un fichero de *log* con el mismo nombre que el proceso.

4.8. PARAMETRIZACIÓN

Con el fin de hacer un software lo más flexible posible, se han parametrizado numerosos aspectos del sistema. La forma en la que se ha implementado esta parametrización ha sido mediante el uso de ficheros de texto plano con un formato determinado que comparten los distintos ficheros de configuración. Para definir una constante se añade al fichero una línea con el formato

NOMBRE_DE_CONSTANTE = VALOR_DE_CONSTANTE

Además, se pueden introducir comentarios definidos como líneas cuyo primer carácter es el *punto y coma*. En realidad, este formato está muy extendido y es el que tiene por ejemplo el fichero de configuración de PHP, php.ini.

La ventaja de usar ficheros de texto plano es que se pueden cambiar características del sistema muy fácilmente con sólo editarlos. Para proteger estos datos sensibles basta con establecer los permisos adecuados para cada fichero.

Otra de las opciones consideradas en un primer momento para la gestión de los parámetros fue el mantenerlos en base de datos. Sin embargo, esto los hace más difícilmente accesibles y constituye una solución inviable para los parámetros del Proveedor, que carece de base de datos. Por lo tanto, y para homogeneizar el tratamiento de los parámetros, se optó por almacenarlos en ficheros de texto plano.

Hay tres ficheros de configuración en Yafrid que serán analizados en profundidad en sus respectivas secciones y que se mencionan a continuación.

- *YAFRID-Web.ini.* Contiene las constantes que definen aspectos del funcionamiento del módulo Yafrid-WEB. Requiere permisos de administrador.
    - *profile.ini.* Define un subconjunto de las constantes de YAFRID-Web.ini que el usuario puede modificar. Estos cambios sólo afectarán al usuario para el que el perfil está definido. Existe un fichero de perfil por cada usuario del sistema.
- *yafrid-distributor.conf.* Define las constantes necesarias para el funcionamiento de Yafrid-CORE. Requiere permisos de administrador.

- *yafrid-provider.conf*. Contiene constantes que afectan al módulo Proveedor y puede ser adaptado por cada usuario.

En **Yafrid-WEB** se manejan de forma distinta los ficheros de configuración del módulo y los de cada usuario. Por un lado, el del módulo, que solamente se va a acceder en modo lectura, se procesa mediante la función *parse_ini_file*. Para acceder a los perfiles de cada usuario se ha optado por desarrollar una clase denominada *PropertiesManager* con los métodos *saveProfile*, *createProfile* y *loadProfile*.

Para el manejo de los ficheros de configuración en **Yafrid-CORE** y en el **Proveedor**, existen una serie de clases Python que pertenecen al módulo configuration.py (Figura 4.15). Las clases *ProviderConfigParser* y *DistributorConfigParser* gestionan el acceso a las constantes de los ficheros yafrid-provider.conf y yafrid-distributor.conf respectivamente.

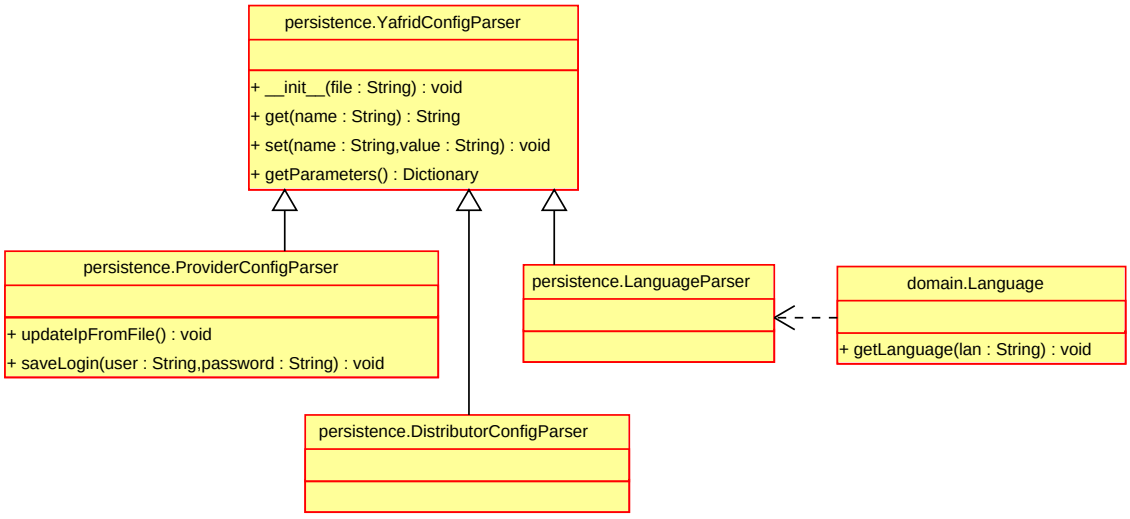

Figura 4.15: Clases del módulo configuration.py.

4.9. INTERNALIZACIÓN

En esta sección se comentan las medidas que se han tomado para hacer el sistema accesible desde el punto de vista de usuarios de otras nacionalidades. También se ha tenido en cuenta que Yafrid puede ser, en un futuro, revisado o mejorado por otros programadores.

Interfaces de usuario multi-idioma

Tanto el interfaz web del sistema como el interfaz de la aplicación proveedora son muy intuitivos y sencillos de usar. A pesar de esto, y para llegar al mayor número de usuarios posible, se ha tenido presente durante el desarrollo del sistema la necesidad de proporcionar al usuario interfaces multi-idioma.

En Python, existe el paquete *gettext* que permite cómodamente crear programas multi-idioma. Esta utilidad se probó con muy buenos resultados, pero finalmente no se utilizó para dotar a todos los componentes del sistema (incluyendo el que no está desarrollado en Python) de un mecanismo común.

La solución más sencilla y portable para dotar a un interfaz de cualquier tipo de la posibilidad de presentarse en distintos idiomas pasa por almacenar los literales del interfaz en **ficheros de texto plano**.

Se ha utilizado el mismo mecanismo tanto para la parte web como para el software Proveedor consistente en sustituir en el programa los literales por variables. Estas variables serán almacenadas en ficheros junto con el texto al que corresponden. Habrá tantos ficheros de idioma como idiomas soporte la aplicación.

Las ventajas de esta solución radican en que es **portable** (gracias al uso de ficheros de texto plano) y **sencilla de implementar y extender**. Para añadir un idioma basta con crear un nuevo fichero de texto que defina cada una de las variables y les asigne el texto adecuado en el idioma correspondiente.

Yafrid-WEB

Los ficheros de idioma tanto de Yafrid-WEB como del software Proveedor se encuentran en los directorios *presentation/languages/* correspondientes. El nombre de cada fichero corresponde con las dos letras que identifican al idioma que representa.

En la parte web se importan directamente las variables a usar en el programa. Para que las variables de traducción no colisionen con el resto de variables, se ha seguido un convenio de nombrado. Las de traducción comienzan con los tres caracteres que identifican el módulo al que pertenecen y acaban en _t indicando que serán objeto traducciónion.

Para que los mensajes JavaScript que se muestran en ocasiones al usuario, se presenten en el idioma adecuado se ha optado por generarlos en tiempo de ejecución sustituyendo las variables de traducción por valores.

Para textos con cierta relación, se han utilizado diccionarios como por ejemplo en el caso de las traducciones de las descripciones de los estados de un proyecto.

En la parte web, si el usuario se encuentra fuera de su cuenta—por ejemplo, en la pantalla de login o creando la propia cuenta—el idioma que muestra la aplicación web es el que está definido en el fichero de configuración. Una vez dentro, se aplica el que determine el perfil del usuario.

Proveedor

En el software Proveedor no existe el problema de la colisión ya que las variables de traducción están contenidas en un solo diccionario. Existe una clase en el modulo configuration.py (Figura 4.15) que sirve para el manejo de los ficheros de idioma.

Internalización dirigida a futuros desarrolladores

Por otro lado, a la hora de escribir el código del sistema, se ha tenido en cuenta en todo momento la potencial proyección del sistema. Yafrid está dirigido a una serie de usuarios interesados en el diseño 3D por medio de software libre.

La comunidad potencial a la que va dirigido será la encargada de probar el sistema, ver qué aspectos son mejorables y, en definitiva, decidir si el sistema demuestra ser verdaderamente útil.

Esta comunidad también puede mejorar el sistema implementado nuevas funcionalidades o mejorando las que están ya desarrollas. Por todo ello, es imprescindible que el código sea claro, intuitivo y escrito con una nomenclatura estándar además de estar adecuadamente comentado. Todo ello, utilizando el inglés que es el idioma más extendido en informática. Todos estos objetivos que han sido llevados a cabo.

4.10. MOTORES DE RENDER

Las clases denominadas *motores de render* constituyen abstracciones de lo que un motor de render es. Únicamente se tienen en cuenta los factores que interesan desde el punto de vista del sistema Yafrid que, por lo general, trata un motor de render como una caja negra.

El paquete Python que implementa los motores de render se denomina *render_engines* y es compartido por las aplicaciones Servidor y Proveedor. Este paquete contiene los siguientes módulos Python:

- **supported_engines.py.**

Es el fichero en el que se definen los motores de render que serán cargados dinámicamente en el sistema. Aunque podría tratarse simplemente de un fichero de texto plano se ha optado por que también sea un archivo escrito en Python.

Contiene tantas definiciones como motores estén soportados. Cada definición consiste en una lista de atributos que definen un motor de render desde el punto de vista de Yafrid.

Los elementos que deben contener cada una de estas listas son en orden

RE_NAME = [engine_name, default_name, constant_path,
constant_version, class_name, (workunit_type, ..)]

1. **RE_NAME**. Identificador de la definición del motor. Tiene que empezar por RE_ (de *Render Engine*) para que el cargador dinámico lo tenga en cuenta.
2. **engine_name**. Identificador con el que se conocerá al motor dentro del sistema.
3. **default_name**. Nombre por defecto del ejecutable del software.
4. **constant_path**. Variable que contendrá la ruta del ejecutable en caso de que el ejecutable no esté en la variable PATH del sistema del proveedor.
5. **constant_version**. Variable que contendrá la versión que el Proveedor tiene del software de render, que será enviado al Servidor en el proceso de suscripción y utilizado a la hora de planificar el procesado de unidades.
6. **class_name**. Nombre del módulo / clase de Python que implementa el motor de render.
7. **workunit_types**. Tupla que contiene los tipos de unidades de trabajo que corresponden al motor de render.

La definición del motor de render de Blender 3D sería como se muestra a continuación:

RE_BLENDER = ['blender', 'blender', 'BLENDERPATH', 'BLENDER',
'BlenderRenderEngine',('blender-static', 'blender-animation')]

- **engines_manager.py**

  Módulo Python que contiene las clases necesarias para gestionar los distintos motores de render existentes.

En la Figura 4.16 aparece la clase EnginesManager que *conoce* varias instancias de la clase EngineDescription. Esta última clase se genera a partir de las definiciones de los motores soportados.

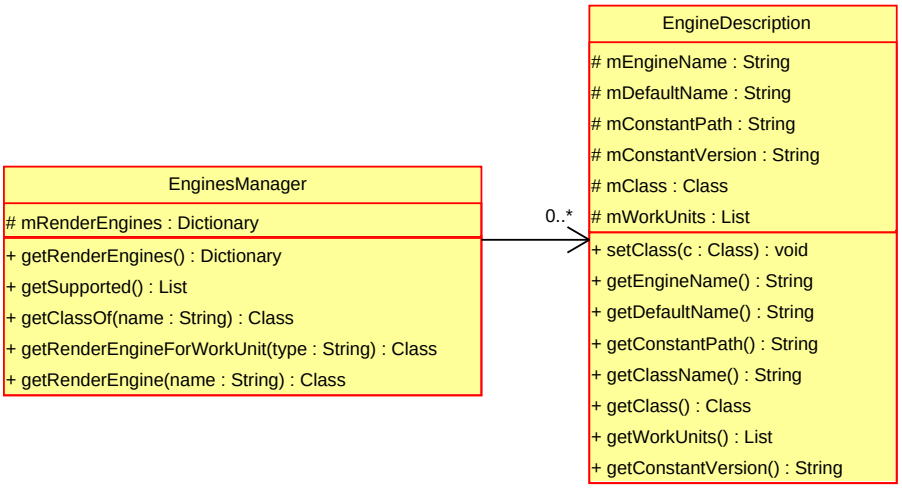

Figura 4.16: Clases en el módulo Python engines_manager.py.

Hay una serie de funciones globales a este módulo que analizan el fichero supported_engines.py y crean una clase por cada una de las definiciones. Asocian además el objeto Clase[9] de nombre *class_name* (que tiene que estar implementada y pertenecer al paquete render_engines) al objeto de clase EngineDescription correspondiente. Este paso se denomina *registro*.

De este modo, al instanciar la clase EnginesManager, se genera un diccionario que contiene las descripciones de los motores (junto con los objetos Clase que los implementan) y que tiene como claves los nombres de los motores. Se accederá a la funcionalidad de los distintos motores mediante la clase EnginesManager.

No existen sentencias *import* que importen explícitamente las clases que implementan los módulos. La importación se hace en tiempo de ejecución y en función de las definiciones en supported_engines.py. Esto se ha hecho así para que incluir nuevos motores no suponga modificar código existente.

Esto es en el caso general, sin embargo, en el caso de la distribución ejecutable de Windows hay que importar los motores explícitamente ya que el compilador py2exe que genera el .exe no detecta esa *dependencia dinámica*.

[9]No hay que olvidar que en Python *todo es un objeto* incluso los módulos, las clases o las funciones.

- **RenderEngineTemplate.py**

  Clase abstracta de la que heredan las distintas implementaciones de los motores de render. Un motor de render tiene que implementar los siguientes método:

  - **GenerateParameters.** Método utilizado por el Distribuidor para generar los parámetros que configurarán las distintas unidades de trabajo en las que se divide una escena o animación.
  - **Render.** Utilizado por el Proveedor para procesar la unidad de trabajo que le ha sido asignada.
  - **PostProcessing.** Realiza un conjunto de acciones sobre los resultados del render. Es utilizado por el Proveedor de servicio.
  - **GetVersion.** Contiene los comandos necesarios para obtener la versión del software de render que el Proveedor tiene instalada. Suele consistir en ejecutar el software con cierta opción (por ejemplo, *yafray -v*) y procesar la salida estándar para obtener la versión.

  Hasta ahora se han implementado en Yafrid los motores de render Yafray y Blender 3D tal y como muestra la Figura 4.17. Para añadir un nuevo motor de render sólo sería necesario añadir una nueva definición al fichero supported_engines.py y crear la clase Python que herede de la clase RenderEngineTemplate y que implemente dicho motor.

4.10.1. Generación de parámetros

La generación de parámetros en el caso de las animaciones es muy sencilla ya que se genera una unidad de trabajo por cada frame de la animación. La salida de este proceso, en este caso, consistiría en una lista cuyos elementos serían los números de orden de cada uno de los frames.

En el caso del render de una imagen estática, la salida de esta fase sería una lista formada por los parámetros que representaran los límites de cada uno de los fragmentos a renderizar. Así, partiendo del tamaño de la escena y de las características de las unidades de trabajo que se desean obtener, se generarán una serie de parámetros que definirán cada unidad de trabajo. Estos parámetros dependen de cada motor de render y se le enviarán al Proveedor encargado de procesar la unidad de trabajo.

La generación de parámetros consta de dos fases sucesivas:

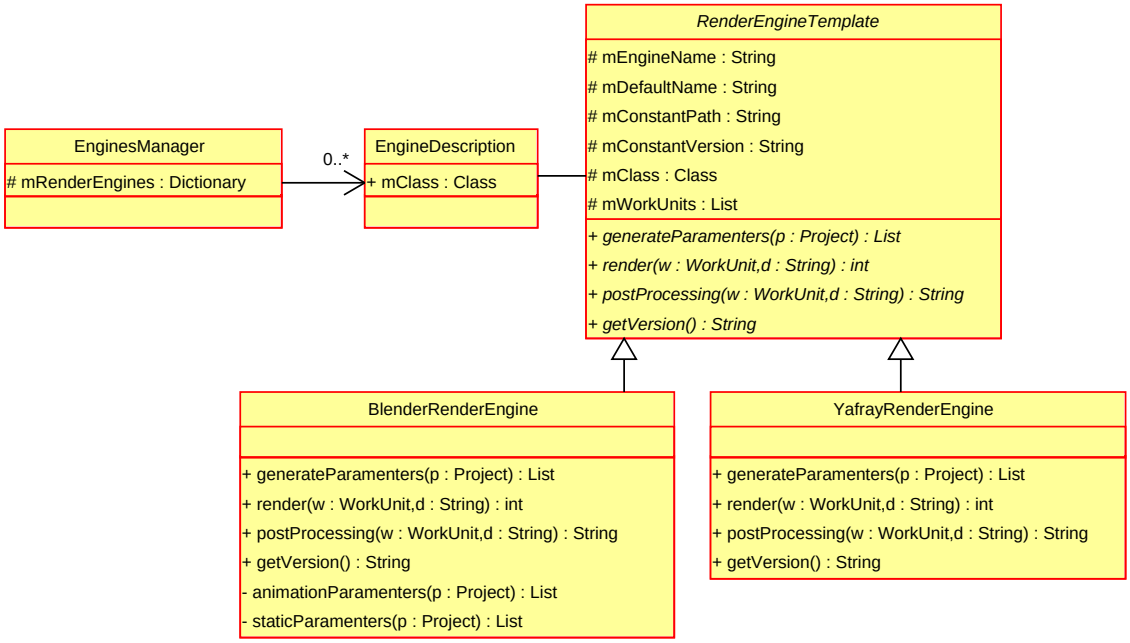

Figura 4.17: Motores de render actualmente implementados en Yafrid.

1. **Transformación del espacio.** Consiste en pasar de las dimensiones *reales* de la escena (que están en píxeles) a las dimensiones correspondientes en el *espacio de render* que son relativas. Otros parámetros de la generación como son la anchura de la banda de interpolación o el propio tamaño de la unidad de trabajo han de ser también normalizados ya que el usuario (o el sistema automáticamente) los especifica en píxeles.

   En la Figura 4.18 se muestra este paso desde el espacio real al espacio de render. En el ejemplo se ha utilizado concretamente el espacio de render que define Yafray, es decir, en el que $-1 \leq X, Y \leq 1$.

   En el caso de Blender tanto la X como la Y van de 0 a 1 con lo que la coordenada $(0, 0)$ corresponde con la esquina inferior izquierda de la imagen y la coordenada $(1, 1)$ con la esquina superior derecha.

2. **División de la escena.** Una vez que las dimensiones de la escena se han hecho corresponder con las del espacio de render, se ha de dividir la escena en fragmentos con un tamaño determinado. En el tamaño de estos fragmentos interviene tanto la anchura de la banda de interpolación que hay que dejar entre las unidades de trabajo adyacentes como el tamaño de unidad de trabajo seleccionado.

   La banda de interpolación es un área extra que se añade a cada fragmento de imagen.

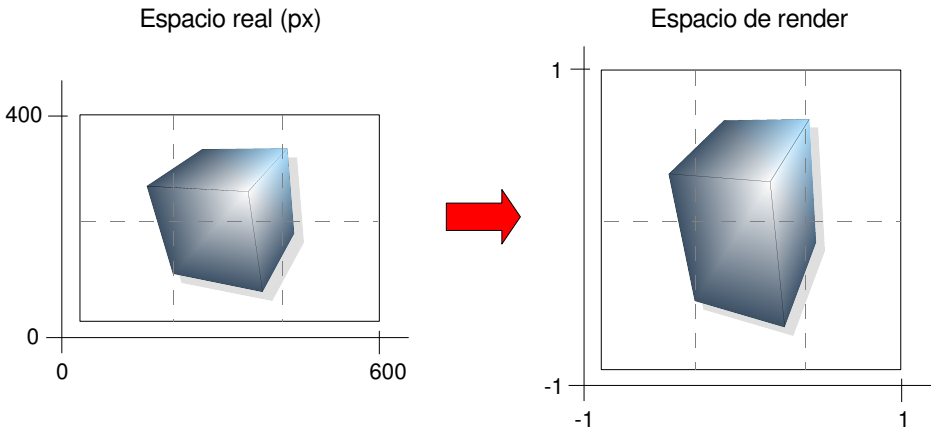

Figura 4.18: Transformación del espacio real al espacio de render (Yafray).

Esta banda es información redundante ya que se calcula en una unidad de trabajo y en su adyacente (la suma de las áreas de los fragmentos es mayor que el área total de la imagen de partida). A pesar de esto, es necesaria debido a que el render en distintas máquinas puede generar resultados algo distintos debido a características de la propia máquina y a la aleatoriedad inherente a algunos métodos de render. Al unir dos fragmentos adyacentes, la zona común se trata aplicando una interpolación lineal entre las dos versiones obtenidas (una de cada unidad de trabajo). Así, en la zona de unión de los fragmentos se pasa de uno a otro con suavidad. En el caso de que la anchura de la zona de interpolación sea cero, no se interpola entre fragmentos consecutivos y la suma de las áreas de los fragmentos coincide con el área de la imagen.

En la Figura 4.19 se puede ver el área extra que se añade a cada uno de los fragmentos A y B formando la banda de interpolación (el espacio sombreado entre los puntos P_1 y P_2).

Algunos problemas que el algoritmo de generación tiene que contemplar son:

- División de la escena en unidades de distintos tamaños en los casos en los que las dimensiones de la unidad de trabajo no sean múltiplo de las de la escena.
- La banda de interpolación que es común a dos unidades adyacentes tiene que coincidir perfectamente en una y otra ya que un fallo de sólo un píxel haría que la escena final fuera inservible.
- Cada unidad tiene cuatro bandas de interpolación con cuatro unidades adyacentes

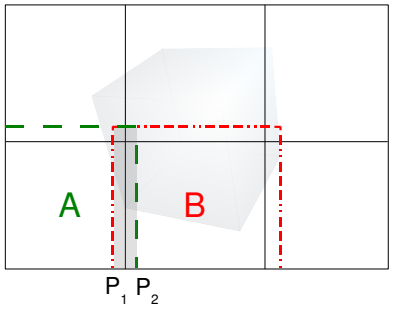

Figura 4.19: Banda de interpolación entre los fragmentos A y B.

en el caso general (no siendo así si la unidad se encuentra en los extremos de la escena o en una esquina).

El proceso es recursivo y se aplica una vez para los fragmentos de una linea y otra a las lineas resultantes de aplicar el proceso la primera vez. En un primer paso se unen los fragmentos para formar una linea. En el segundo, se unen las lineas, que se tratan como fragmentos individuales, para formar la imagen completa.

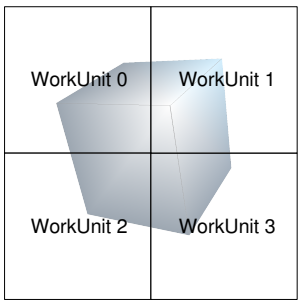

Figura 4.20: División de una escena de 400x400 en unidades de 200px de lado.

Con la división de la Figura 4.20, ignorando las bandas de interpolación, se generarían los siguientes parámetros que definirían otras tantas unidades de trabajo:

- **Yafray (x_min, x_max, y_min, y_max)**.

  Donde $-1 \leq x\_min, x\_max, y\_min, y\_max \leq 1$.

  - WorkUnit 0: (-1, 0, 0, 1)
  - WorkUnit 1: (0, 1, 0, 1)

- WorkUnit 2: (-1, 0, -1, 0)
- WorkUnit 3: (0, 1, -1, 0)

- **Blender 3D (x_min, y_min, x_max, y_max)**.

  Donde $0 \leq x\_min, y\_min, x\_max, y\_max \leq 1$.

  - WorkUnit 0: (0, 0.5, 0.5, 1)
  - WorkUnit 1: (0.5, 0.5, 1, 1)
  - WorkUnit 2: (0, 0, 0.5, 0.5)
  - WorkUnit 3: (0.5, 0, 1, 0.5)

4.10.2. Render

Cada motor de render posee un método mediante el que el Proveedor realiza el render de la unidad de trabajo que se le pasa como parámetro. Los datos que son necesarios para el render son una combinación de los que contiene la unidad de trabajo y los que posee el motor de render:

- De la unidad de trabajo se obtienen el nombre del fichero fuente, el tipo de unidad de trabajo (si es estática o una animación en el caso de Blender 3D) y los parámetros que determinan las coordenadas de la unidad de trabajo (Figura 4.21).
- Del motor de render se obtiene el nombre del ejecutable o la ruta hasta éste.

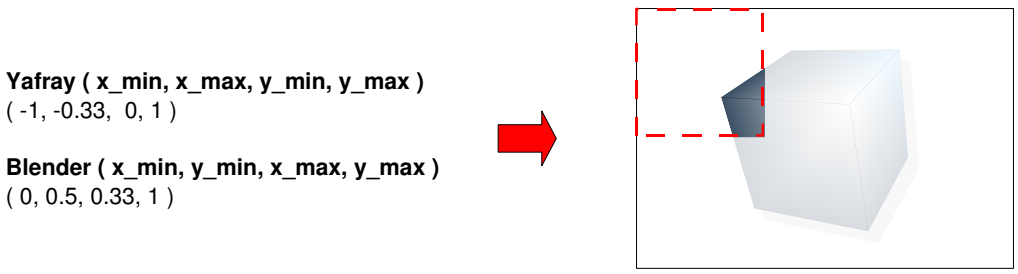

Figura 4.21: Una unidad corresponde con distintos parámetros en función del motor.

Con esta información se creará una instancia de la clase Process (ver Figura 4.14) que constituirá el proceso que se encargará del render (el software de render se usa como una caja negra llamándolo por línea de comandos).

```
import Blender
from Blender import *
from Blender.Scene import Render

scn = Scene.GetCurrent()
context = scn.getRenderingContext()

# Obtain coordinates from stdin
(x_min, y_min, x_max, y_max) = [float(p) for p in sys.argv[5:9]]

# Set the border to render
context.enableBorderRender(1)
context.setBorder(x_min, y_min, x_max, y_max)
```

Listado 4.2: Propiedades de una escena Blender por medio de Python

Mientras que Yafray sí acepta como parámetros los límites entres los cuales se debe renderizar, en Blender 3D esto es algo más problemático. No se le pueden pasar las coordenadas al ejecutarlo por línea de comandos pero finalmente se encontró un modo de establecer estos límites. La solución pasó por hacer uso del API de programación en Python que posee Blender para establecer estos límites en la propia escena tal y como muestra el Listado 4.2.

Para modificar la escena a renderizar se aprovecha que uno de los parámetros que puede aceptar Blender al ser ejecutado es la ruta de un script Python que procesará antes de hacer el render. Este procedimiento también se ha utilizado para variar algunas otras propiedades de la escena como pueden ser el tamaño de la imagen resultado o la calidad final. Estos ajustes vienen definidos en el fichero *project.info* que acompaña al fichero *blender* que recibe el Proveedor.

4.10.3. Post-procesado

Ejecutado por el Proveedor al finalizar el render. Se encarga de pequeños ajustes que hay que realizar a los resultados obtenidos antes de ser enviados de vuelta al grid.

En proyectos de imágenes estáticas, tanto en Yafray como en Blender 3D, es necesario eliminar la parte de la imagen que no contiene información para quedarse sólo con lo relevante. Normalmente cuando se renderiza un fragmento, se genera una imagen del tamaño total de la original pero sólo una parte tiene información relevante, la que corresponde a la unidad de trabajo. Para ello, se hace uso de la función *cutImage* de la clase *ImageLibrary* que pertenece al módulo Python *image.py*. Esta función y las demás de esta clase se analizan con

profundidad en el Apartado 4.11, dedicado al tratamiento de imágenes.

En proyectos de animación no es necesario realizar ningún tipo de ajuste.

4.11. TRATAMIENTO DE IMÁGENES

Para el tratamiento de imágenes necesario tanto en el Proveedor como en el Servidor, se ha desarrollado un módulo Python denominado *image.py* que pertenece al paquete de dominio. Este módulo contiene una sola clase, ImageLibrary, que se describe en la Figura 4.22. Las distintas funciones estáticas que contiene esta clase hacen uso de los distintos paquetes y clases de PIL, la librería para el manejo de imágenes de Python.

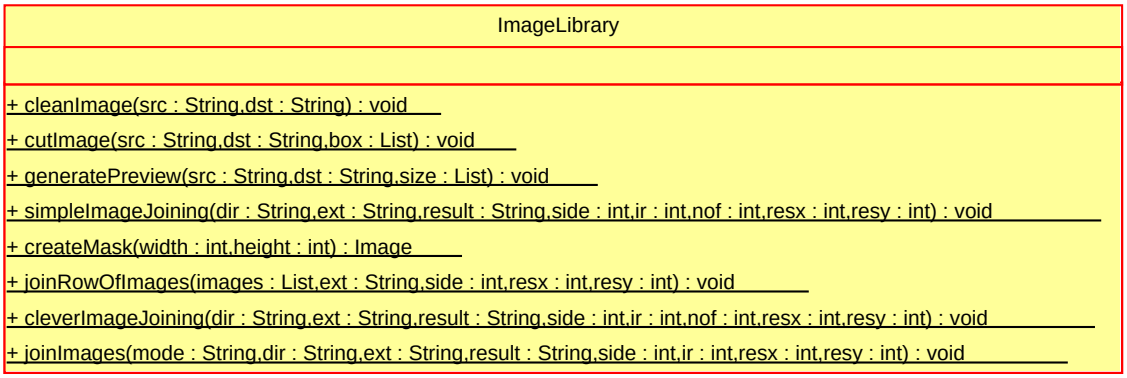

Figura 4.22: Clase ImageLibrary para el tratamiento de imágenes.

La clase ImageLibrary contiene las siguientes funciones:

- **cleanImage.** Elimina de una imagen las zonas sin información relevante.
- **cutImage.** Devuelve la imagen que resulta de extraer de otra el área cuyas coordenadas se le pasan por parámetros. Esta función y la anterior se utilizan en el Proveedor.
- **createMask.** Crea una imagen que será usada como máscara para la interpolación entre los fragmentos. Variando las características de esta imagen se podrían obtener distintos tipos de interpolación.
- **generatePreview.** Utilizada por el Distribuidor para generar la imagen que será mostrada en el interfaz web de la aplicación.
- **simpleImageJoining.** Función que une los distintos fragmentos que estén disponibles de una imagen sin realizar ninguna interpolación entre ellos. Así, mientras que se tengan

resultados parciales, se realizará una composición más burda pero menos costosa que la que hará cuando se hayan procesado todos los fragmentos.

- **cleverImageJoining.** Función que une los distintos fragmentos de una imagen ya terminada llevando a cabo una interpolación lineal entre ellos.

- **joinRowOfImages.** Función utilizada a la hora de unir fragmentos. Se aplica una primera vez con los fragmentos de cada fila obteniendo tiras de imágenes y una segunda con el resultado rotado de lo anterior con lo que se obtiene una tira de columnas que corresponde con la imagen completa.

- **joinImages.** En función del parámetro *mode* se realiza un tipo de composición u otra. El proceso de unión de las imágenes se tratará con más profundidad en el Apartado 4.11.

Algoritmo 5 Composición de imágenes.

```
R ← Create Row
I ← Create Row
Width ← 0
for i = 0 to Images.Length do
   R.Append(Images[i])
   Width ← Width + Images[i].Width
   if Width ≥ XRES then
      Width ← 0
      I.Append(R)
      R.Reset()
   end if
end for
I ← I.Reverse()
C ← Create Row
Img ← Create Image
ImgResult ← Create Image
for i = 0 to I.Length do
   Img ← JoinRowOfImages(I[i])
   Img ← Img.Rotate(-90)
   C.Append(Img)
end for
ImgResult ← JoinRowOfImages(C)
ImgResult ← ImgResult.Rotate(90)
```

Una restricción que ofrece PIL es que no maneja de forma completa ciertos tipos de ficheros gráficos. Así, en el caso de los ficheros de tipo *TGA (Targa)* por ejemplo, PIL puede

interpretarlos pero no puede escribirlos. Esto supone un problema ya que es necesario hacer algunas modificaciones en los resultados de los render estáticos—en las animaciones no es necesario ya que el frame se envía tal cual—para eliminar ruido. Así, mientras que en el caso de las animaciones el sistema trata los resultados en el formato que especifica el usuario, en los render estáticos se utiliza el formato sin pérdida *PNG*.

Composición de imágenes

Mientras que el paso de composición en los proyectos de animación es trivial, esto no ocurre en el caso de los proyectos de render estáticos. El proceso en este caso tiene tener en consideración cuestiones similares a las que se plantean al generar los parámetros (ver Apartado 4.10.1). Hay que prestar especial atención a los casos en que el tamaño de la unidad de trabajo no es múltiplo de las dimensiones de la imagen, con lo que habría fragmentos de distintos tamaños, o cuando faltan fragmentos intermedios.

Algoritmo de composición

Para unir los fragmentos de imagen que han generado los Proveedores ImageLibrary proporciona dos métodos. Como ya se ha comentado, uno se utiliza para unir resultados parciales y el otro para obtener la imagen final. El funcionamiento es básicamente el mismo sólo que en el primero no se calcula la zona de interpolación. Este proceso de composición se describe en el Algoritmo 5.

El método que realiza la unión de los fragmentos es joinRowOfImages, que compone la lista de imágenes que se le pasa por parámetros horizontalmente. En un primer paso (Figura 4.23) se aplica el método a los fragmentos de agrupados por su coordenada Y obteniendo una serie de *imágenes fila*. El paso siguiente consiste en rotar -90º las imágenes generadas. Estas imágenes se vuelven a pasar como entrada al método (Figura 4.24) que las trata de nuevo como fragmentos de una fila. Ahora, la *fila* resultante es la imagen completa sólo que rotada -90º con lo que para obtener la imagen inicial sólo es necesario deshacer la rotación.

De este modo, el problema de unir e interpolar en las direcciones X e Y a la vez se ha reducido a hacerlo dos veces en un sola dimensión. Esto ha sido posible ya que el caso de unir e interpolar un fragmento y el que tiene a su derecha es análogo al caso de unir e interpolar una fila y la que tiene encima.

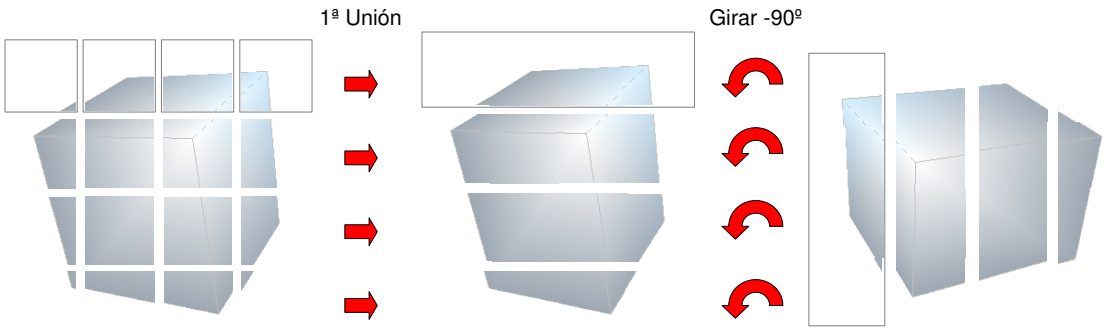

Figura 4.23: Algoritmo de composición (I).

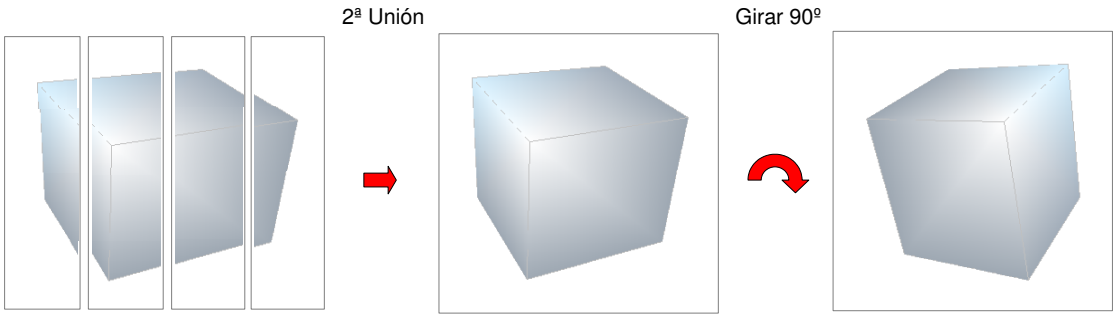

Figura 4.24: Algoritmo de composición (II).

Banda de interpolación

Los fragmentos de imagen realizados por los Proveedores se separan en fragmentos más pequeños según pertenezcan sólo a una unidad de trabajo o a varias (es decir, forman parte de la banda de interpolación). A la hora de unir los fragmentos que forman la imagen final se colocan en sus posiciones correspondientes los fragmentos que pertenecen sólo a una unidad de trabajo.

Con respecto a los demás, se realizarán interpolaciones lineales entre cada fragmento y su correspondiente la unidad de trabajo contigua. Esto se puede ver en la figura 4.25. Antes del punto P_1, el valor del resultado será el calculado en la unidad de trabajo A. A partir del P_2, la imagen final será la calculada en la B. Entre ambos puntos se realizará la interpolación entre lo calculado en A y lo calculado en B. En el punto P_1 se comenzará con el valor de A y progresivamente irá teniendo más importancia el valor de B hasta que en el punto P_2 el resultado estará íntegramente formado por la imagen obtenida en B.

Una descripción simplificada del proceso de unión e interpolación llevado a cabo por la

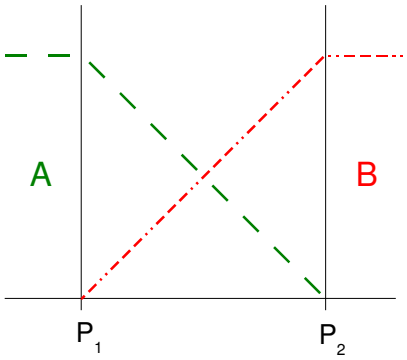

Figura 4.25: Interpolación entre los fragmentos A y B.

función joinRowOfImages se muestra en el Algoritmo 6. Cada unidad de trabajo se divide en tres partes. Una parte central, *FragmentCenter*, se pasará tal cual a la imagen final. Las bandas que quedan a ambos lados, de anchura *INT*, se tratan antes de pasar a la imagen final. Cada parte izquierda (*FragmentLeft*) se interpola linealmente con la parte derecha del fragmento inmediatamente a su izquierda (el que tiene índice $i - 1$). Esto se realiza en todos lo casos excepto en el primero de cada fila que no tiene un fragmento más a la izquierda.

Algoritmo 6 Unión e interpolación de imágenes.

```
FragmentLeft ← Create Image
FragmentCenter ← Create Image
FragmentBefore ← Create Image
Mask ← CreateMask()
Interpolation ← Image
for i = 0 to Images.Length do
  {If it is the first workunit of the row}
  if i = 0 then
    FragmentCenter ← Images[i].Cut(0,Images[i].Width-INT)
  else
    FragmentBefore ← Images[i-1].Cut(Images[i-1].Width-INT,Images[i-1].Width)
    FragmentLeft ← Images[i].Cut(0,INT)
    FragmentCenter ← Images[i].Cut(INT,Images[i].Width-INT)
  end if
  ImgResult.Write(FragmentCenter)
  Interpolation ← Interpolate(FragmentBefore, FragmentLeft, Mask)
  ImgResult.Write(Interpolation)
end for
```

4.12. PRIORIDADES

Cada cliente y cada proveedor tienen una prioridad en cada grupo al que pertenecen. Sin embargo, los significados de estas propiedades varían según el rol que el usuario desempeñe en el grupo.

En el caso de la prioridad de los clientes, ésta vendrá representada por un número del 0 al 100 e indicará la importancia del cliente en el grupo. Los proyectos de un cliente se crean con una prioridad que corresponde a la que tenía el cliente en el momento de la creación. Esta prioridad representa el tanto por ciento máximo de recursos de un determinado grupo que puede utilizar en cada momento un usuario. Así, si un determinado cliente tiene una prioridad de 50 en un determinado grupo, podrá utilizar la mitad de los proveedores disponibles de ese grupo. En caso de que sólo haya un proyecto siendo ejecutado, podrá hacer uso de todos los recursos del grupo. Cuanto mayor sea la prioridad de un cliente en un grupo, mayor número de proveedores se podrán dedicar a sus proyectos a la vez y, por lo tanto, tardarán menos en finalizarse.

La prioridad inicial de los clientes la establecerá el sistema y vendrá definida en una constante de configuración. La prioridad aumentará con el número de proyectos de render finalizados correctamente.

En cuanto a los proveedores, la prioridad va del 0 al 10 y es establecida por el propio usuario. Define un orden entre los grupos a los que está suscrito el proveedor. Así, atenderá antes a los proyectos pertenecientes al grupo para el que mayor prioridad tenga definida el proveedor. Si para un grupo A tiene prioridad 6 y para un grupo B tiene prioridad 5, sólo atenderá a los proyectos del grupo B si no existen proyectos del grupo A.

4.13. YAFRID-CORE

Representa la parte no interactiva del sistema ya que, una vez que se inicia, funciona sin intervención del usuario a modo de demonio.

Está formado principalmente por dos submódulos, el Distribuidor y el Identificador, independientes el uno del otro y desarrollados en Python. Mientras que el primero constituye la parte activa del sistema en el sentido de que toma la iniciativa para comunicarse con otras partes del sistema, el Identificador y sus componentes permanecen a la espera de comunicaciones.

4.13.1. Parametrización

La parametrización de este módulo consiste en una serie de variables almacenadas en el fichero *yafrid-distributor.conf*.

Las variables definidas en este fichero son:

- **POOL.** Directorio donde las unidades de trabajo serán almacenadas para su adquisición por parte de los Proveedores.
- **USER_PROJECTS.** Directorio raíz a partir del cual se creará el árbol de directorios con los proyectos de los clientes.
- **TMAX.** Tiempo máximo que se espera a que una unidad de trabajo sea procesada antes de consultar sobre su estado (en segundos).
- **TSLEEP.** Tiempo que marca la frecuencia de las iteraciones del Distribuidor (en segundos).
- **MYSQL_HOST.** Máquina para acceder al servidor MySQL.
- **MYSQL_USER.** Usuario de MySQL.
- **MYSQL_PASSWD.** Contraseña para conectar con el servidor MySQL.
- **MYSQL_DB.** Base de datos MySQL en la que se encuentran las tablas del sistema Yafrid.
- **WORK_ORDER.** Orden en el que serán procesadas las unidades de trabajo Los dos valores posibles son *ALEA* y *ORDERED*. El primero hace que las unidades se seleccionen aleatoriamente y el segundo que estén ordenadas en función de su posición en la imagen final.
- **TIMEOUT.** Parámetro interno del sistema. Tiempo en milisegundos tras el que caducarán los paquetes TCP/IP enviados a los Proveedores.

4.13.2. Distribuidor

El Distribuidor es el módulo que constituye la parte activa de Yafrid-CORE y cuya principal labor consiste en enviar unidades de trabajo a los Proveedores para ser renderizadas

y procesar los resultados obtenidos. Además de estas tareas principales hace muchas otras que son imprescindibles para el funcionamiento del grid.

Todas estas actividades se realizan periódicamente cada cierto tiempo. Dicho tiempo está parametrizado en la constante *TSLEEP* que se encuentra en el fichero *yafrid-distributor.conf*.

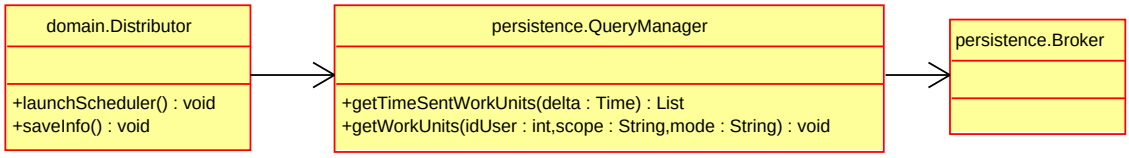

Figura 4.26: Diagrama de clases del módulo Distribuidor.

La única instancia de la clase Distribuidor *conoce* mediante una relación de asociación a la única instancia de la clase *QueryManager* (Figura 4.26) que a su vez conoce a la instancia de la clase Broker. El Distribuidor conoce además al resto de clases de dominio por medio de relaciones de dependencia. El Distribuidor debe manejar objetos de tipo Proyecto, Proveedor, Unidad de Trabajo, etc.

Las distintas actividades que lleva a cabo el Distribuidor implican el manejo de la mayoría de entidades del sistema. En casi todos los casos, estas acciones se realizan mediante las clases persistentes. Existen un par de casos en los que esta aproximación resulta muy costosa por el gran número de objetos que habría que instanciar. Es para resolver estos casos para lo que existe la clase QueryManager, que contiene varios métodos que directamente realizan consultas a la base de datos a través del agente.

Creación de las unidades de trabajo.

El Distribuidor selecciona todos aquellos proyectos que el usuario ha activado y realiza una serie de acciones que tienen como objetivo que el proyecto sea dividido en unidades de trabajo para su planificación. Los puntos más importantes del proceso se describen en el Algoritmo 7.

Por un lado, se crea un lanzamiento que se caracterizará por las propiedades del proyecto del que proviene y por los parámetros de render que haya establecido el cliente. Los ficheros fuente del proyecto se copian al directorio de POOL desde donde serán accedidos por los distintos Proveedores del sistema.

Tras esto, se generan los parámetros de render (ver Apartado 4.10.1) y las unidades de

Algoritmo 7 Creación de las unidades de trabajo.

```
for all P / IsProject(P) and P.State = ACTIVE do
  L ← Create Launching(P)
  Copy L.Sources to POOL.Sources[L]
  Engine ← EnginesManager.getEngine(P.Type)
  Parameters ← Engine.GenerateParameters(P)
  for all x in Parameters do
    W ← Create WorkUnit(x, L)
    W.State ← ACTIVE
  end for
  P.State ← EXECUTING
  P.Count, L.Count ← |Parameters|
  L.InitialTime ← NOW
end for
```

trabajo correspondientes (que se crearán en estado ACTIVA). Finalmente se asocia el número de unidades generadas al proyecto, se inicializa el tiempo del lanzamiento y se procede a la ejecución del proyecto cambiando su estado.

Distribución de las unidades de trabajo entre los Proveedores.

Este proceso es una de las tareas clave que tiene que realizar el Distribuidor. Para cada Proveedor disponible en el sistema, selecciona una de las unidades de trabajo adecuadas y se la envía para que la procese.

Este proceso es algo complicado pero básicamente se llevan a cabo una serie de pasos que se comentan simplificados a continuación. Estos pasos se realizan para cada proveedor disponible.

- Se obtienen de una cola las unidades de trabajo que pertenecen a los grupos a los que el proveedor esté suscrito. Estas unidades estarán ordenadas en función de la prioridad del proveedor para esos grupos.

  Dentro de un mismo grupo, las unidades de trabajo se pueden devolver ordenadas aleatoriamente o de modo secuencial con respecto a su posición en la imagen final. Esta característica está definida por la constante de configuración *WORK_ORDER*.

  En caso de que el proveedor sea público, también estarán disponibles las unidades de trabajo públicas pero siempre detrás de las de los proyectos privados.

- De las unidades seleccionadas se toman en orden aquellas para las que el proveedor tenga el software necesario (incluyendo la versión).

- Antes de enviar la unidad de trabajo se comprueba si el propietario de la unidad de trabajo ha excedido su límite de utilización del grid (indicado por su prioridad).

- Si todos los requisitos se satisfacen, el Distribuidor envía al Proveedor la unidad de trabajo, actualizando las estructuras de datos e instancias necesarias. Para enviar un trabajo a un proveedor, se instancia el objeto adecuado mediante su proxy textual y se invoca al método remoto process, al que se le pasa como argumento la unidad de trabajo.

- En todo momento se comprueba si el Proveedor sigue estando conectado.

Unión de los resultados obtenidos.

Este paso consiste en la composición de las imágenes generadas por los Proveedores con el fin de generar un producto para el usuario final, el cliente. El proceso que se sigue se detalla en el Algoritmo 8.

Algoritmo 8 Unión de los resultados obtenidos.

```
for all W / IsWorkUnit(W) and W.State = FINISHED do
  L ← Read Launching(W)
  P ← Read Project(L)
  Copy POOL.Results[L][W.Order] to P.TemporalImages
  Delete POOL.Results[L][W.Order]
  W.State ← CONSUMED
  L.FinalTime ← NOW
  P.Count ← P.Count - 1
  if P.Count = 0 then
    P.Result ← JoinResults(P.TemporalImages)
  else
    P.Result ← JoinPartialResults(P.TemporalImages)
  end if
  P.Preview ← GeneratePreview(P.Result)
end for
```

Para cada una de las unidades de trabajo procesadas por los Proveedores existe un fichero de resultados en el POOL. Cada uno de estos ficheros queda identificado por medio del identificador del lanzamiento y el orden de la unidad en el mismo. En esta fase se extraen[10] las imágenes y se almacenan en el directorio del proyecto. En este punto se dice que la unidad de

[10]Las imágenes son enviadas comprimidas por los Proveedores.

trabajo ha sido *consumida*. También se actualiza el tiempo del lanzamiento y los fragmentos que faltan por ser recibidos.

Finalmente, en función de si se han recibido todas las unidades o no se lleva a cabo un proceso de unión u otro. Esta distinción existe debido a que al generar resultados parciales (aún no han llegado todos los fragmentos) se lleva a cabo una unión más tosca que no invierte tanto tiempo como el mecanismo que dará lugar a la imagen final. En ambos casos se genera una previsualización de los resultados de menor tamaño que los propios resultados para ser mostrada en el interfaz web.

El proceso de composición de resultados es sencillo para las unidades de trabajo que forman parte de un proyecto de animación. Tan sólo consiste en comprimir los fotogramas de la animación en un fichero conjunto para su descarga por parte del cliente.

En el caso del render de una imagen individual, el proceso es algo más complejo. En los algoritmos con una componente aleatoria (métodos de Monte carlo), como el Pathtracing, se pueden distinguir pequeñas diferencias entre los fragmentos generados por distintos Proveedores. Por ello es necesario realizar una suavizado en las uniones entre cada fragmento y sus vecinos. Todo lo relativo a la composición de imágenes se trata en el Apartado 4.11.

Finalización de proyectos.

La finalización de un proyecto es el proceso que abarca las acciones que se realizan sobre los proyectos cuyas unidades de trabajo han sido todas correctamente procesadas. El proceso se describe en el Algoritmo 9.

Algoritmo 9 Finalización de proyectos.

```
for all P / IsProject(P) and P.Count = 0 do
    Priority_{P.Client} ← Priority_{P.Client} + 1
    L ← Read Launching(P)
    L.FinalTime ← NOW
    for all W / IsWorkUnit(W) and W.Launching = L do
        Delete W
    end for
    Delete POOL.Sources[L]
    Delete P.TemporalImages
    P.State ← FINISHED
end for
```

Esta tarea consiste básicamente en eliminar los elementos auxiliares del proceso de render distribuido de los proyectos que hayan finalizado con éxito. Estos elementos son los ficheros fuente que se encuentran en el POOL y los resultados parciales obtenidos.

Además, también se eliminan las unidades de trabajo generadas, se actualiza la duración del lanzamiento y se incrementa la prioridad del cliente cuyo proyecto ha terminado con éxito.

Comprobación de unidades enviadas.

Esta tarea consiste básicamente en comprobar periódicamente si las unidades enviadas siguen procesándose o por el contrario se han *perdido*. La razón principal por la que una unidad de trabajo puede haberse perdido es que el usuario proveedor haya desconectado el software antes de completar con éxito el render de la misma. El Algoritmo 10 muestra en pseudocódigo el desarrollo de esta tarea.

Algoritmo 10 Comprobación de unidades enviadas.

```
for all W / IsWorkUnit(W) and W.State = SENT do
  if NOW - W.TimeOfRequest ≥ TMAX then
    P ← Create Provider(W.ProxyOfProvider)
    CW = P.getCurrentWorkUnit()
    if IsNull(CW) or W != CW then
      W.State ← ACTIVE
      W.TimeOfRequest ← NULL
      GridUsage_{W.Client} ← GridUsage_{W.client} - 1
    else
      W.TimeOfRequest ← NOW
    end if
  end if
end for
```

En primer lugar, el Distribuidor, invoca el método getTimeSentWorkUnits del QueryManager pasándole como parámetro el valor del parámetro TMAX del fichero de configuración. Este parámetro corresponde con el tiempo que se permite que un Proveedor procese una unidad antes de ser consultado. El método devolverá una lista con las unidades de trabajo para las que haya pasado un tiempo mayor o igual a TMAX desde que fueron enviadas.

Tras esto, el Distribuidor instanciará los objetos Proveedores a los que fueron enviadas cada una de estas unidades de trabajo a partir de sus proxies (recuperados de la base de datos).

Después, se invocará el método getCurrentWorkUnit de cada Proveedor con lo que obtendremos la información sobre la unidad de trabajo que está procesando. Pueden darse tres casos:

1. Que el Proveedor siga procesando la misma unidad de trabajo y aún no haya acabado porque sea costosa. Si en el sistema se da este caso muy frecuentemente, sería recomendable incrementar el tiempo TMAX en el fichero de configuración.

2. Que el Proveedor no esté procesando ninguna unidad de trabajo. Esto puede ocurrir si el usuario Proveedor reinicia el software en medio del proceso.

3. Que el Proveedor esté procesando otra unidad de trabajo. Puede ocurrir por algún fallo en las comunicaciones o porque el usuario haya reiniciado el software y haya recibido una nueva unidad de trabajo sin haber finalizado la anterior.

En el caso 1, se actualiza el tiempo de la unidad de trabajo con lo que será consultada al pasar otras TMAX unidades de tiempo. En los casos 2 y 3 se activa de nuevo la unidad para ser planificada, se elimina el tiempo que indica cuándo se envió y se decrementa el uso que del grid hace el cliente propietario de la unidad de trabajo.

Borrado de proyectos.

Cuando un usuario decide borrar uno de sus proyectos, esta acción no se realiza de forma instantánea. El proyecto queda en un estado previo a ser eliminado denominado *Siendo borrado* (Figura 4.7). Un proyecto permanece es este estado mientras que haya Proveedores procesando sus unidades de trabajo. Una vez que estas unidades hayan sido terminadas, el proyecto podrá ser borrado. Esto se ha hecho así para que no sea necesario avisar a todos y cada uno de los Proveedores que poseen fragmentos de proyectos que van a ser borrados. De este modo, se les permite terminar y el Servidor de Yafrid no tiene que realizar este sobreesfuerzo.

Borrar un proyecto implica:

- Eliminar de la base de datos el registro de la tabla PROJECTS correspondiente mediante la operación Delete de la clase persistente.

- Borrar el directorio del proyecto.

- Anular el campo ID_PROJECT de los lanzamientos asociados con el proyecto. El registro no se elimina para garantizar que los identificadores de los lanzamientos sean únicos a lo largo de toda la vida del sistema.

- Borrar los ficheros fuente contenidos en el POOL de unidades asociados con el último lanzamiento del proyecto.

- Eliminar los requisitos de software asociados al último lanzamiento del proyecto por medio de la clase ProjectRequirements.
- Eliminar de la base de datos las unidades de trabajo asociadas con el último lanzamiento del proyecto.

4.13.3. Identificador

El Identificador es el módulo del Servidor que constituye el componente pasivo del mismo. Constituye lo que en la terminología de ICE se denomina un servidor, es decir, un proceso que se mantiene a la espera de que otros objetos del sistema (los Proveedores) realicen peticiones a sus servicios. Estos servicios están gestionados por dos objetos, el Controlador y el Generador de Controladores.

La Figura 4.27 muestra el diagrama de clases que componen el módulo Identificador. En él se puede ver como el Identificador *conoce* a una instancia de la clase ControllerGeneratorI que es la implementación de la clase comm.ControllerGenerator que hereda de Ice.Object. A su vez, la instancia de la clase comm.ControllerGeneratorI maneja varias instancias de la clase comm.ControllerI que es la implementación de comm.Controller.

Los Proveedores del sistema accederán al Generador de Controladores y después a su propio Controlador a través de sus respectivos proxies que son especializaciones de la clase Ice.ObjectPrx.

La clase de dominio IdentificatorHelper es una clase con método estáticos con la que las distintas clases del módulo establecen relaciones de dependencia.

4.13.3.1. Generador de Controladores

Es el objeto con el que el Proveedor realiza su primera interacción para integrarse en el grid y la última una vez que quiere dejar de formar parte del sistema.

En esta primera comunicación, el Generador crea un nuevo objeto Controlador y lo añade al adaptador de objetos creado por el Identificador para que pueda ser accedido remotamente. La versión textual del proxy a este objeto Controlador es devuelto al Proveedor para que se pueda conectar al mismo.

Además, se asocia al Proveedor con su Controlador almacenando la información en la tabla CONTROLLERS (Cuadro 4.2). Esta inserción se realizará de la forma habitual ya perfilada en la sección dedicada a la persistencia. La clase IdentificatorHelper conocerá a las instancias de la clase Controllers del paquete de dominio que manejarán su persistencia

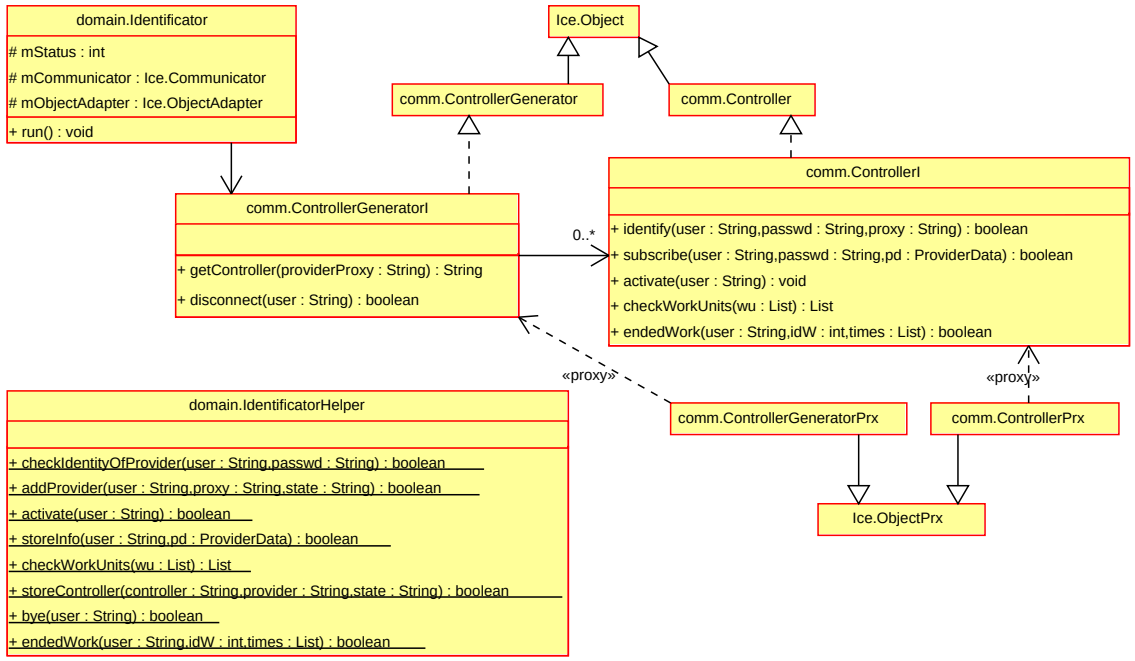

Figura 4.27: Diagrama de clases del módulo Identificador.

CONTROLLERS		
ID_CONTROLLER	INT	PK
CONTROLLER_PROXY	VARCHAR(150)	NOT NULL
PROVIDER_PROXY	VARCHAR(150)	NOT NULL
STATE	VARCHAR(20)	NOT NULL

Cuadro 4.2: Tabla que almacena la asociación entre Proveedor y Controlador.

mediante su respectiva clase de fabricación pura FPControllers. Esta última clase dejará el manejo de la base de datos a la clase Broker.

Este objeto también se encarga de gestionar la desconexión de un Proveedor. Este proceso consiste en eliminar de la base de datos las referencias al Proveedor (Cuadro 4.3) y a su Controlador así como eliminar éste último del adaptador de objetos de ICE.

4.13.3.2. Controlador

Es el objeto con el que se comunica el Proveedor durante la mayor parte de su vida en el sistema. Cada uno de estos objetos está asignado a un Proveedor al que ofrece una serie de servicios:

PROVIDERS		
ID_PROVIDER	INT	PK
ID_USER	INT	FK USERS
PROXY	VARCHAR(150)	NOT NULL
STATE	VARCHAR(20)	NOT NULL

Cuadro 4.3: Tabla que almacena la información de los Proveedores conectados.

- Identificación (IDENTIFY). Es invocado por el Proveedor al conectarse al grid. En él se comprueba que el nombre de usuario y la contraseña enviados corresponden con un proveedor válido. En caso afirmativo, se almacena el proxy del objeto Proveedor asociándolo con el usuario proveedor (tabla PROVIDERS).

- Suscripción (SUBSCRIBE). Este método almacena o actualiza la información que el proveedor envía sobre su sistema y sobre el software instalado en las tablas PROVIDER_DATA y PROVIDER_SOFTWARE respectivamente. Se almacena la información sobre el software en otra tabla ya que un proveedor puede serlo para varios motores de render.

- Comprobación de unidades de trabajo (CHECKWORKUNITS). El Proveedor envía una lista con los lanzamientos que posee en su directorio local. El Controlador procesa la lista y devuelve al Proveedor los identificadores de aquellos que ya no son necesarios y puede borrar. No serán necesarios aquellos lanzamientos que pertenezcan a un proyecto que no está ya en ejecución.

- Activación (ACTIVATE). El Proveedor invoca este método cuando está listo para recibir peticiones por parte del servidor. El resultado es el cambio de estado del Proveedor en el sistema.

- Notificación de fin de tarea (ENDEDWORK). Cuando un Proveedor ha terminado de procesar la unidad de trabajo que tenía asignada avisa al Servidor por medio de su controlador. Esta llamada desencadena una serie de acciones:

  - El estado del Proveedor cambia y éste pasa a estar disponible para nuevos trabajos.
  - La unidad de trabajo se marca en la base de datos como finalizada.
  - Se elimina la unidad de la tabla de unidades enviadas.

- Se decrementa el uso que del grid está haciendo el cliente propietario de la unidad de trabajo.
- Se almacenan los tiempos invertidos en procesar la unidad de trabajo que han sido enviados por el Proveedor.

4.14. YAFRID-WEB

Es el medio principal a través del cual el usuario de Yafrid puede acceder a la funcionalidad del mismo. Constituye la parte interactiva del servidor y está formada por un conjunto de páginas web dinámicas escritas en PHP.

4.14.1. Tecnologías web

Algunos de los lenguajes que permiten desarrollar sitios web dinámicos que se mencionan en el Apartado 3.4 son **PHP**, **ASP**, **ASP .NET** y **JSP**.

En un primer momento se descartan ASP y ASP .NET debido a que no son multiplataforma y un sistema MS Windows es imprescindible, con el consiguiente incremento en los costes debido a la adquisición de licencias. Además, los costes derivados del servidor necesario también son mayores con sistemas MS Windows (ver Apartado 5.2). En cuanto a PHP y JSP, el coste de desarrollo e implementación con estos lenguajes es en principio nulo.

Entre JSP y PHP, la decisión se ha basado en que el número de los servidores que ofrecen PHP es mayor que el de los que ofrecen JSP. Además, el conocimiento que se tenía iniciamente sobre PHP era mayor que sobre JSP.

Una vez seleccionado el lenguaje de programación también se estableció el servidor HTTP necesario. La elección de **Apache** no supone costes, además de proporcionar un alto rendimiento y ser multiplataforma.

Como resultado de estas decisiones, junto con las descritas en otras secciones, se tiene que Yafrid-WEB se asienta sobre una *solución WAMP/LAMP*, es decir, es una combinación de:

- (GNU/)**L**inux o **W**indows, como sistema operativo.
- **A**pache, como servidor web.
- **M**ySQL, como sistema gestor de bases de datos.
- **P**erl, **P**HP o **P**ython, como lenguajes de programación.

Además del uso PHP, se ha utilizado el lenguaje **JavaScript** del lado del cliente. El uso de scripts en este lenguaje se ha limitado a lo imprescindible. JavaScript se ha utilizado para validar los datos introducidos por el cliente y para realizar alguna pequeña aplicación como el reloj que aparece en la ventana principal del usuario.

Se han utilizado además hojas de estilo **CSS** para dotar a la aplicación web de un aspecto homogéneo y hacer más cómodo el mantenimiento.

4.14.2. Generalidades sobre Yafrid-WEB

En esta sección se abordan generalidades sobre el diseño del interfaz web de Yafrid que es conveniente tener presentes a la hora de pasar a los apartados siguientes.

Mensajes de usuario

En Yafrid-WEB aparecen dos tipos de mensajes que avisan al usuario de los resultados de sus acciones sobre el interfaz.

Ambos tipos de mensajes toman el texto que muestran de los ficheros de idioma situados en *presentation/languages* en función de la configuración del sistema o en función del perfil del usuario si éste se ha identificado.

- **infoMessage.php**

  Por un lado están los mensajes que se presentan al usuario como una página web que son los más usados en la aplicación.

  Para estos tipos de mensajes se ha definido una plantilla, infoMessage.php que, en función de la clave que se le pase como parámetro, muestra una configuración u otra. Se puede configurar el título del mensaje, el texto explicativo que aparece y el link al que ofrece regresar (que puede aparecer o no).

  En la Figura 4.28 se muestran tres ejemplos del uso de infoMessage.php. En el caso a), la clave es *forbidden* y sólo se muestra el título y el texto explicativo. En b) y c), el parámetro que se le pasa a infoMessage.php por GET es *loginFail* y *existingUser* respectivamente.

- **Mensajes JavaScript**

  Se utilizan para avisar al usuario de que los datos introducidos no pasan las validaciones necesarias o para informar sobre el resultado de alguna acción sencilla.

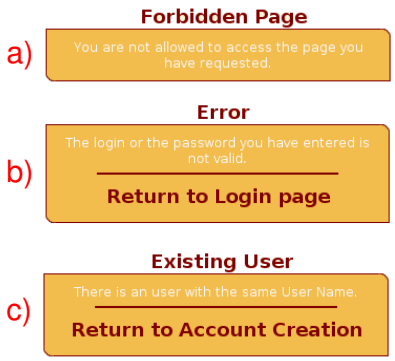

Figura 4.28: Ejemplos de infoMessage.php.

Los scripts que se envían al navegador del usuario se generar en tiempo de ejecución definiendo los textos que van a aparecer mediante variables de los ficheros de idioma.

Envío de correos

Esta funcionalidad está implementada en las clases EMail y EMailHelper (Figura 4.29) del paquete de dominio. Sólo se aplica si la constante ACTIVATION_REQUIRED así lo indica.

En función del objetivo que persigue, cada tipo de email se crea con un texto u otro. Hay dos tipos de correos en cuanto a la activación de las cuentas de usuario:

- **Activación.** Se envía un email cuando se crea una cuenta de usuario inactiva.

- **Confirmación.** Una vez que el usuario ha respondido, se le envía un nuevo correo para confirmarle la activación de la cuenta.

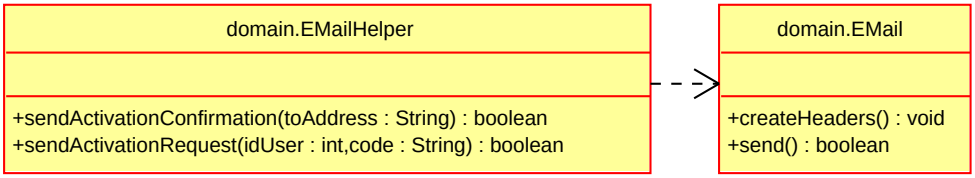

Figura 4.29: Clases para el envío de correo electrónico.

Clases Helper

Las clases *helper* son clases que se han desarrollado para incrementar la funcionalidad de otras clases de dominio. Todos sus métodos suelen ser estáticos por lo que constituyen un ejemplo de *clases biblioteca*.

Así, la clase UserHelper, por ejemplo, contiene métodos relacionados con los objetos de la clase Users pero que no se han incluido en ésta última para no sobrecargarla de responsabilidades. Además de crear un nuevo usuario, la clase UserHelper crea el directorio asociado y su fichero de perfil, envía un correo de activación y asocia los roles al usuario. Existen clases helper para los proveedores, los usuarios, etc.

En el Anexo A, dedicado a los diagramas, aparece la clase UserHelper y su relación con el resto de clases como ejemplo.

Páginas de consulta

Yafrid-WEB posee varias características que lo hacen más cercano a una aplicación clásica de gestión que el resto del sistema, que está más cercano a las aplicaciones en tiempo real.

En la aplicación web hay numerosas pantallas que simplemente muestran los datos bien de un usuario o de un proyecto. Todas estas ventanas usan la funcionalidad de las clases de presentación desarrolladas para mostrar una representación de los objetos de dominio. Así, existirá una clase *FUSer* con único método público, *show*, que permitirá mostrar de forma homogénea en toda la aplicación los datos de un usuario.

Por otra parte, también es habitual mostrar un listado de elementos de cierto tipo y tener la posibilidad de relizar filtrados sobre los resultados existentes. Para ello también se han utilizado mecanismos comunes para incrementar la reutilización del código.

Generación de números aleatorios

Se ha desarrollado la clase *RandomManager* que, a partir de unos parámetros sobre los resultados a obtener, obtiene una cadena de caracteres aleatorios. Esta funcionalidad se utiliza por ejemplo para obtener la imagen de verificación de la ventana de creación de cuentas. También se usa para generar el código mediante el cual un usuario podrá activar su cuenta.

Clases Parser

Para obtener los atributos de los proyectos y poder modificarlos (esto último es necesario para el lanzamiento de proyectos) se han creado dos clases que implementan la interfaz *GeneralParser*. La primera de estas clases es *Parser* que permite analizar y modificar los ficheros xml que definen un proyecto Yafray (en realidad cualquier xml). La otra se llama *ParserInfo* y analiza el fichero de texto plano en el que se almacenan los atributos de los proyectos Blender.

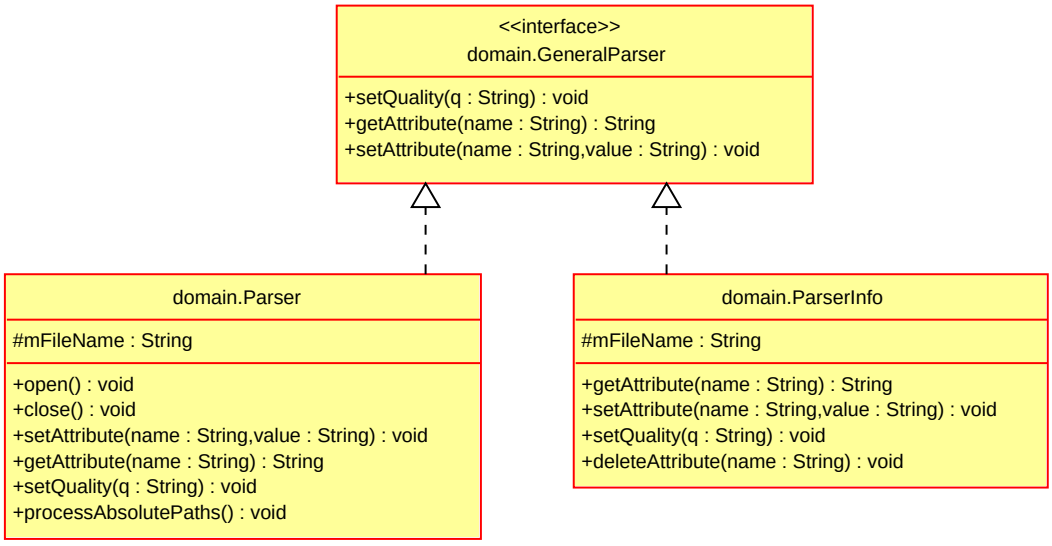

Figura 4.30: Clases para manejar los atributos de los proyectos.

4.14.3. Creación de cuentas

Los usuarios que deseen entrar a formar parte de Yafrid, bien como clientes o bien como proveedores de servicio, necesitan obtener una cuenta de usuario.

Para crear una cuenta se necesitan cuatro pasos que están modelados en otras tantas páginas PHP:

1. **Selección del tipo de cuenta.** El usuario puede adquirir una cuenta de cliente, de proveedor o de ambos (*step1.php* de la capa de presentación).

2. **Introducción de datos.** Implementada en *step2.php*. En ambos tipos de cuenta se piden unos datos mínimos a introducir (Figura 4.32) que son el nombre de usuario, la contraseña y la dirección de correo. Estos datos se almacenarán en la tabla USERS

Figura 4.31: División de la pantalla de login.

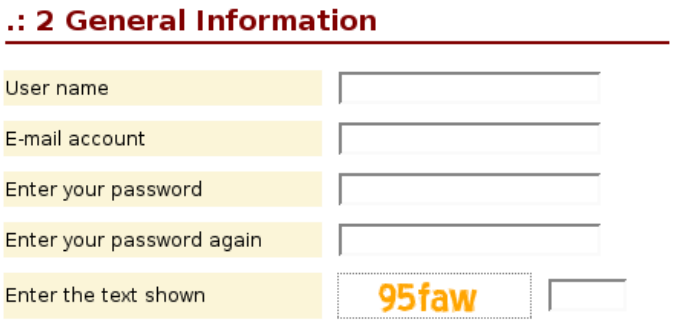

Figura 4.32: Creación de cuenta. Datos generales.

(Cuadro 4.4). Se pide además la introducción del texto que se muestra en una imagen, de lo que se hablará más adelante.

3. **Suscripción a grupos.** La sección de *Suscripción a Grupos* se configurará en función de los parámetros que *step1.php* le pase a través de GET. Lo mostrado en la Figura 4.33 corresponde al caso de una cuenta de usuario mixta (de proveedor y cliente). En esta sección se eligen los grupos para los que se va a estas suscrito y las prioridades en cada uno (ver Apartado 4.12).

4. **Creación de la cuenta.**

   Los datos introducidos se validan mediante JavaScript en el cliente y si son correctos se envía la petición al servidor. Si no hay ningún usuario con el mismo nombre, se procede a la creación de la cuenta. Para ello, se llama a los métodos *createAccount* de la clase *UserHelper* y *subscribeToGroup* de la clase *GroupHelper*. Ambas clases pertenecen al paquete de dominio.

USERS		
ID_USER	INT	PK
USER_NAME	VARCHAR(100)	NOT NULL
PASSWORD	VARCHAR(100)	NOT NULL
EMAIL	VARCHAR(100)	NOT NULL
ACTIVE	VARCHAR(1)	NOT NULL

Cuadro 4.4: Tabla que almacena la información de los Usuarios.

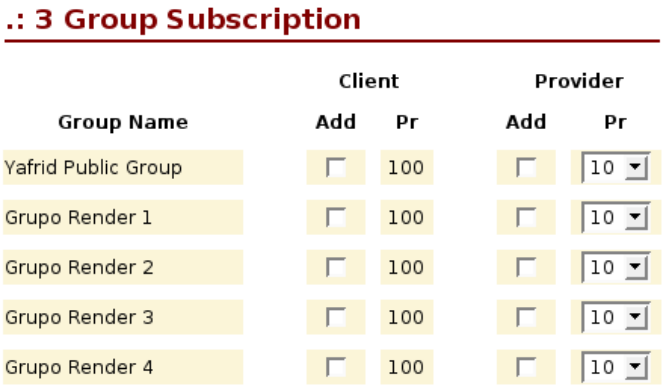

Figura 4.33: Creación de cuenta. Suscripción a grupos.

La creación de la cuenta implica la creación del usuario y del rol o roles asociados. Además, se crea el directorio de proyectos del usuario y se añade un fichero de perfil inicial.

Además, si la constante de configuración ACTIVATION_REQUIRED está a true, la cuenta se crea como inactiva con lo que será necesario un cuarto paso de activación. Al crear una cuenta inactiva, se crea un registro en la tabla ACCOUNT_ACTIVATION (Cuadro 4.5) que relaciona esa cuenta con una cadena aleatoria servirá más tarde para

ACCOUNT_ACTIVATION		
ID_ACCOUNT_ACTIVATION	INT	PK
ID_USER	INT	FK USERS
CODE	VARCHAR(64)	NOT NULL

Cuadro 4.5: Tabla que almacena la información de las cuentas por activar.

activar la cuenta.

5. **Activación de la cuenta.** En caso de que la cuenta de usuario haya sido creada como inactiva, el usuario recibirá un correo en su cuenta, informándole de los pasos a seguir. Para activar una cuenta, el usuario sólo tendrá que hacer click en el enlace que se indica en el correo (Figura 4.34).

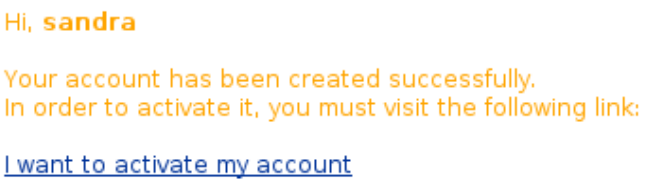

Figura 4.34: Correo para la activación de una cuenta.

En enlace llevaría a la página *activateAccount.php* con un código determinado como por ejemplo:

activateAccount.php?code=y3umqdb231oz6fgzsibgf2skonbb7yth

Esta página comprobaría mediante AccountActivationHelper si el código es de un usuario aún inactivo. En ese caso procedería a su activación poniendo a verdadero el campo ACTIVE de la tabla de usuarios. También se enviaría un correo al usuario informándole de que su cuenta ha sido correctamente activada y se eliminaría el registro de la tabla ACCOUNT_ACTIVATION.

Evitar registros automáticos

Con el fin de evitar los registros automáticos, se ha utilizado la aproximación basada en la tecnología *CAPTCHA*[11] [7] que ya aparece en la mayoría de sitios web y que fue introducida por la Universidad Carnegie Mellon.

Esta tecnología consiste en realizar una prueba que sólo un humano sabría responder y se usa en aquellas situaciones en las que se requiere garantizar que la entidad con la que se interactúa es humana y no un programa software. La versión más sencilla de esta

[11] Acrónimo de **C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part (Prueba de Turing pública y automática para diferenciar a máquinas y humanos).

prueba utiliza un texto que no puede ser *comprendido* por medio del *reconocimiento óptico de caracteres* porque está, por ejemplo, distorsionado.

En Yafrid, al crear una cuenta de usuario, hay que introducir el texto que aparece en una imagen que se genera de forma aleatoria. El texto de la imagen se envía encriptado con el algoritmo MD5 desde el servidor al cliente. La verificación de si el texto introducido coincide con el que generó la imagen se lleva a cabo mediante JavaScript. De este modo, y al no aparecer en el código html, el texto no puede ser introducido por un mecanismo de registro automático. Distintos niveles de seguridad se obtendrán si se varía la fuente que se usa en la generación de la imagen.

Esta funcionalidad es implementada en el fichero *image.php* del paquete de presentación que hace uso además de la clase *RandomManager*.

4.14.4. Usuarios de Yafrid

Cada uno de los usuario está caracterizado por una serie de datos que introduce al crear su cuenta y que se almacenarán en base de datos.

Asociado a esa cuenta también existe un directorio del que colgarán los directorios de los distintos proyectos que el usuario vaya creando. En este directorio, se creará un fichero denominado profile.ini que contendrá algunos parámetros de configuración específicos de cada usuario.

Como ya se ha comentado en varias ocasiones, los usuarios de Yafrid se pueden dividir en tres categorías en función de su papel en el sistema. Esto también condiciona su nivel de acceso a las funcionalidades del sistema.

Desde la página principal, *home.php*, se realiza la comprobación de los permisos que posee cada usuario mediante la clase *SecurityHelper* que posee tres métodos, *checkPermission*, *checkIdentity* y *getRoles*. Se comprobará en primer lugar que el usuario esté registrado y su cuenta activada. Una vez verificado esto, se verifica que el módulo al que intenta acceder (y que determina el parámetro *sec* pasado mediante *GET*) se encuentra dentro de sus posibilidades (si es accesible desde alguno de los roles que desempeña).

Yafrid-WEB está dividido en tres módulos (uno por rol) y éstos a su vez en submódulos. Estos submódulos implementan una funcionalidad única y corresponden con los requisitos funcionales descritos en el Anexo A.

A continuación se realiza una descripción de los distintos submódulos agrupados por módulo.

- [**Administrador**]

H_TIMES		
ID_H_TIMES	INT	PK
ID_LAUNCHMENT	INT	FK PROJECT_LAUNCHMENTS
ORD	INT	NOT NULL
ID_USER	INT	FK USERS
T_FETCH	LONG	NOT NULL
T_RENDER	LONG	NOT NULL
T_SEND	LONG	NOT NULL

Cuadro 4.6: Tabla que almacena los tiempos consumidos por cada unidad de trabajo.

- **Gestión de Usuarios.** Pantalla de consulta (Apartado 4.14.2) que muestra los usuarios del sistema. Se pueden realizar filtros sobre los resultados y consultar los detalles de los usuarios.
- **Gestión de Grupos.** Pantalla de consulta que muestra los grupos del sistema. Se pueden modificar sus datos y las prioridades de los usuarios suscritos.
- **Gestión de Proveedores.** Pantalla de consulta que muestra los proveedores, sus datos y su estado con respecto al sistema. Se puede acceder al detalle de su información de conexión.
- **Estadísticas.** Con la información contenida en las distintas tablas, en especial en la tabla *H_TIMES* que almacena los tiempos enviados por los proveedores (Cuadro 4.6), se muestran distintas estadísticas que relacionan proyectos, conjuntos de pruebas y proveedores.
- **Activación.** Pantalla a través de la cual el administrador puede activar grupos.
- **Pruebas.** Permite crear múltiples proyectos a partir de un mismo fichero fuente que se diferencian en los parámetros de calidad y división.
- **Gestión de Procesos.** Pantalla mediante la cual el administrador de Yafrid puede conocer el estado de los procesos que forman Yafrid-CORE. También es posible iniciarlos o detenerlos desde el propio interfaz web.

- [**Cliente**]
  - **Gestión de Proyectos.** Pieza central en torno a la cual se establece el papel del cliente en el sistema. Permite crear nuevos proyectos y consultar y modificar el

estado de los proyectos creados.

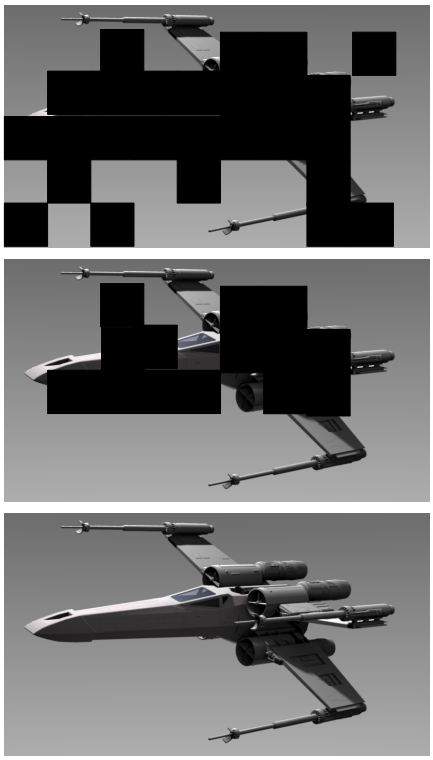

Figura 4.35: Distintos grados de progreso de un proyecto Yafrid.

- **Gestión de Grupos.** Pantalla que permite crear nuevos grupos, editar aquellos de los que se es administrador y suscribirse a grupos existentes.
- **Estadísticas.** Estadísticas relacionadas con los proyectos creados por cada cliente.

- [**Proveedor**]

  - **Información General.** Permite conocer el estado y los datos de conexión y del sistema del proveedor. También sirve para que el proveedor se suscriba a alguno de los grupos existentes para prestar sus servicios.
  - **Estadísticas.** Pantalla en la que un proveedor puede conocer su desempeño en el sistema.
  - **Software.** Esta sección se mantendrá actualizada con las distintas versiones del Proveedor para su descarga.

4.14.5. Parametrización

Como el resto de módulos del sistema, Yafrid-WEB también está parametrizado. La parametrización de Yafrid-WEB se realiza a mediante ficheros de texto plano. Como ya se ha comentado, existen dos tipos de ficheros. Por un lado, está YAFRID-Web.ini que se encuentra en el directorio */domain/ini/* y que determina la configuración de todo el sistema web. Por otra parte, cada usuario posee su propio fichero de configuración local a su cuenta. Este último contiene un subconjunto de las constantes definidas en YAFRID-Web.ini, en concreto la sección *DEFAULT* que se aplicarán a nivel de usuario.

A continuación se muestra un listado completo de las constantes que contiene este fichero de configuración organizadas por secciones.

- [GENERAL]
  - **YAFRID_ROOT.** Contiene el directorio base de Yafrid-WEB.
  - **YAFRID_URL.** Dirección web de Yafrid-WEB.
  - **ACTIVATION_REQUIRED.** Indica si es necesario activar vía correo electrónico las cuentas de usuario y los grupos después de crearlos o no.
  - **SUB.** Cadena de sustitución. Sirve para parametrizar ciertas cadenas de traducción con mensajes comunes.
  - **PASS_MIN.** Tamaño mínimo en caracteres de una contraseña válida.
  - **LOGIN_MIN.** Tamaño mínimo en caracteres de un nombre de usuario válido.
  - **INITIAL_CLI_PR.** Prioridad inicial con la que se crearán las cuentas de los clientes.
- [EMAIL]
  - **FROM_NAME.** Parámetro para el envío de correos.
  - **FROM_ADDRESS.** Parámetro para el envío de correos.
  - **STYLE.** Cadena en notación CSS que determina el aspecto del texto de los mensajes de correo.
- [MYSQL]
  - **HOST.** Host de MySQL.

- **DB.** Base de datos de MySQL.
- **USER.** Usuario de MySQL.
- **PASSWORD.** Contraseña de MySQL.

- [PATH]
  - **CHECKSERVER.** Ubicación del script YAFRID_checkServer.sh.
  - **MANAGESERVER.** Ubicación del script YAFRID_manageServer.sh.
  - **BLENDER.** Ubicación del ejecutable de Blender 3D.
  - **USER_PROJECTS.** Ubicación del directorio de proyectos.
- [SOFTWARE]
  - **BLENDER.** URL de descarga de Blender 3D.
  - **YAFRAY.** URL de descarga de Yafray.
- [DEFAULT]
  - **LANGUAGE.** Lenguaje por defecto de la aplicación.
  - **PREVIEW_SIZE.** Tamaño por defecto de la previsualización de los resultados de los proyectos.
  - **REFRESH_FREQUENCY.** Tasa de refresco de la previsualización de los resultados de los proyectos.
- [PROJECT]
  - **SIZE.** Tamaño de la unidad de trabajo por defecto.
  - **OFFSET.** Ancho de la banda de interpolación por defecto.

4.15. PROVEEDOR

El Proveedor es el software que se encuentra instalado en las máquinas de los usuarios que desean ceder sus ciclos de CPU para ser utilizados en tareas de render.

Puede usarse a través de un interfaz gráfico tanto en GNU/Linux como en MS Windows. Sin embargo, en cada sistema cuenta con unas peculiaridades. Mientras que en el primero

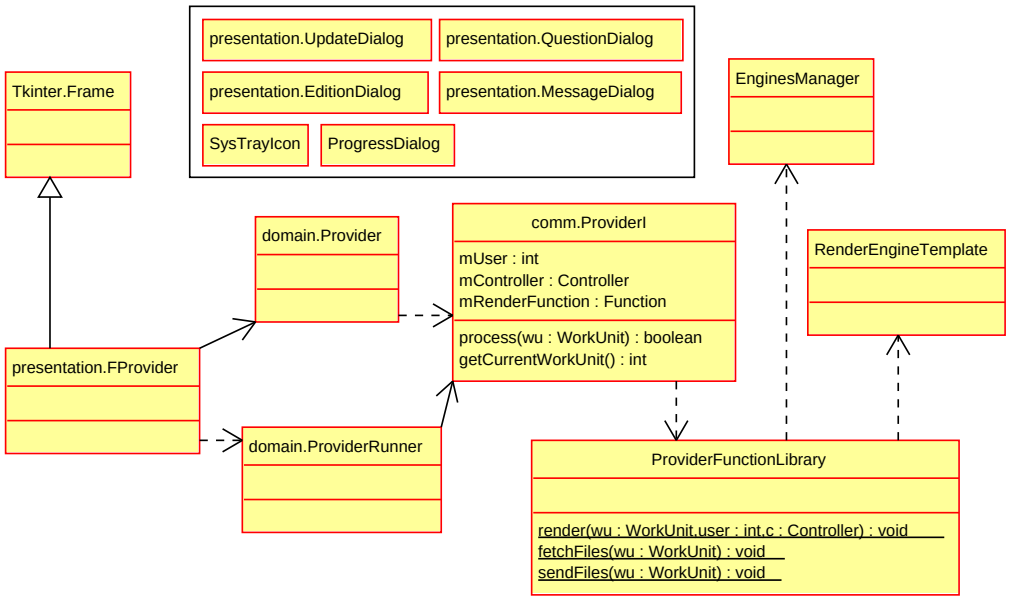

Figura 4.36: Diagrama de clases general del Proveedor.

puede utilizarse también por línea de comandos, en el segundo se puede ocultar el programa siendo sustituido por un icono en la barra de tareas.

Otra singularidad es que la del software para sistemas MS Windows se distribuye en la forma de un asistente de instalación. Mediante la combinación de las herramientas *py2exe* e *Inno Setup* se ha empaquetado el software en un solo fichero que, al ejecutarse, presenta un asistente que irá guiando al usuario en el proceso de instalación.

4.15.1. Interfaz gráfica de usuario

De las opciones disponibles para desarrollar la interfaz gráfica de usuario (GUI) de un programa escrito en Python, se eligió Tkinter. Tkinter, como ya ha sido mencionado, es el interfaz que posee Python para Tcl/Tk y está considerado como un estándar *defacto* en los desarrollos en Python.

Se optó en parte por esta alternativa debido a que Tkinter es multiplataforma y la mayor parte de las distribuciones de Python lo llevan incorporado. Otras razones fueron la brevedad de los programas escritos en Tkinter y la robusted que ha ido consiguiendo a lo largo de los años[12]. Bien es cierto que tiene algunas desventajas como por ejemplo que carece de programas de desarrollo rápido de aplicaciones y de objetos gráficos avanzados como árboles

[12]Tkinter fue lanzado en 1990.

o barras de progreso[13]. Esto último pierde importancia debido a la existencia de multitud de extensiones que incorporan este tipo de objetos.

Una opción que se tuvo en consideración fue GTK, que además ya se había utilizado en otros desarrollos. Sin embargo, se descartó debido a que el tamaño de las distribuciones de software que se obtendrían iba a ser excesivo, sobre todo en sistemas MS Windows. Una opción igual de buena podría haber sido el uso de wxWindows.

La interfaz de este módulo es bastante sencilla (Figura 4.37) pero cumple perfectamente con la finalidad para la que fue ideada.

En la parte superior de la figura (**A**), se tiene el menú típico de cualquier aplicación. Desde ahí se podrá acceder a algunas funcionalidades secundarias que se describen más adelante, en el Apartado 4.15.5.

En la zona del interfaz que en la Figura 4.37 aparece etiquetada con la letra **B** se encuentran los parámetros de conexión que el usuario tiene que introducir. El campo *puerto* corresponde con el puerto en el que el Proveedor permanecerá a la espera de peticiones por parte del Servidor una vez conectado. En los campos *usuario* y *password* se mostrarán los valores de la última conexión realizada correctamente (almacenados en el fichero de configuración).

En **C** se encuentran los botones que realizarán las tareas básicas de conectar, desconectar y salir de la aplicación.

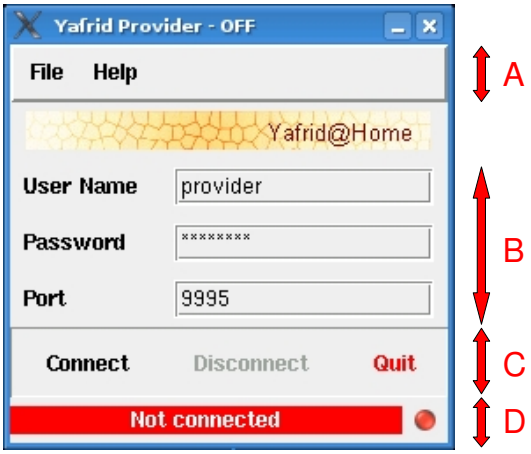

Figura 4.37: Interfaz del Proveedor en GNU/Linux.

En la parte inferior (**D** en la Figura 4.37) se muestra de modo visual el estado del Proveedor. Por un lado, la barra de estado muestra el estado del Proveedor con respecto

[13] De hecho, se ha tenido que desarrollar una para usarla en el interfaz del Proveedor.

del grid. El color rojo indica que no está conectado, el amarillo que está dando los pasos necesarios para conectar y finalmente el verde indica que el Proveedor está conectado al grid.

A la derecha de la barra de estado existe una pequeña bombilla que cambia de color en función de si el Proveedor está renderizando alguna unidad de trabajo o no. Del color rojo inicial se pasará al amarillo cuando el Proveedor esté ocupado.

Icono en la barra de tareas (en MS Windows)

Para los usuarios de MS Windows existe una característica que no ofrece el Proveedor cuando funciona en sistemas GNU/Linux. En los sistemas de Microsoft aparece un botón con el texto *Ocultar* (Figura 4.38) que hace que el interfaz se minimize y desaparezca quedando representado por un icono en la barra de tareas (el primero por la izquierda en la Figura 4.39).

Figura 4.38: Interfaz del Proveedor en MS Windows.

Las acciones que permite este icono son volver a mostrar el interfaz, con lo que el icono desaparece, y salir del programa. La acción por defecto, que se ejecuta al hacer doble clic, es la de maximizar el interfaz.

Figura 4.39: Icono del Proveedor en la barra de tareas.

La implementación de esta funcionalidad se ha llevado a cabo desarrollando un módulo *sys_tray_icon.py* que se encuentra en el paquete de presentación y que contiene una clase

SysTrayIcon (Figura 4.40). Esta clase hace uso de la extensión de Python para win32, en concreto de los módulos win32api, win32gui_struct y winxpgui (o win32gui en caso de que el sistema no sea Windows XP).

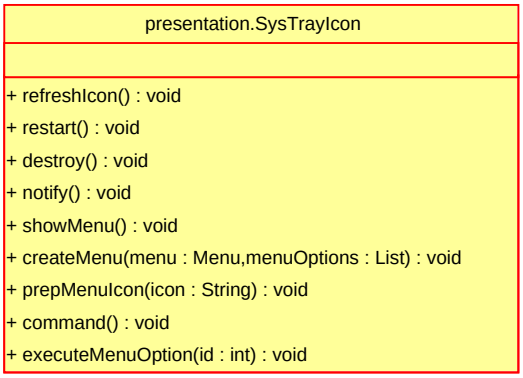

Figura 4.40: Clase SysTrayIcon.

Arranque automático

En MS Windows, al instalar el Proveedor, por defecto se crea un acceso directo en el Menú Inicio con lo que el software se ejecuta al iniciar el sistema. En este caso, se le pasa al constructor de FProvider un parámetro que hace que se intente un máximo de tres ocasiones la conexión con el grid. En caso de que el Proveedor consiga conectarse, el programa se ocultará en la barra de tareas. En caso contrario, el programa terminará y se creará un fichero de log con la información sobre los errores encontrados durante la conexión.

En el caso de sistemas GNU/Linux se incluye en la distribución un script en Bourne shell que permite, con permisos de superusuario, instalar el Proveedor de modo que se inicie al arrancar el sistema.

4.15.2. Conexión con el grid

La conexión con el grid comienza cuando el usuario hace clic en el botón con el texto *Conectar* e implica una serie de pasos que se ejecutan de forma secuencial:

1. **Adquisición del controlador.**

   En primer lugar, el Proveedor crea un proxy al objeto ControllerGenerator que reside en el Identificador. Esto se hace de la forma habitual, es decir, por medio de la clase

comm.ControllerGeneratorPrx y la versión textual del proxy que se puede crear a partir de la información del fichero de configuración.

Una vez instanciado el Generador, se invocará su servicio getController (Figura 4.41) como si en realidad el objeto estuviera en la máquina local. Será con este Controlador con quien el Proveedor se comunique en las sucesivas interacciones.

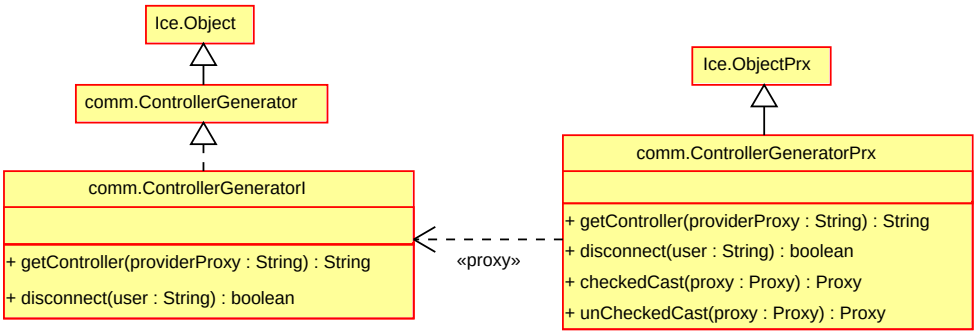

Figura 4.41: Proxy de la clase ControllerGenerator.

2. **Identificación.**

   Básicamente se ejecuta el método identify de la clase domain.Provider que llama a su vez al identify del Controlador que le ha sido asignado.

3. **Suscripción.**

   Se obtiene la información del sistema del usuario por medio de la clase SysInfo (ver Apartado 4.15.6). Esta información se almacena en un fichero cuyo nombre está formado por el nombre del usuario y la extensión *data*. Este fichero se guarda en el directorio *sys/users*.

   Si la información obtenida del sistema no coincide con la almacenada en el fichero, es decir, algo en el sistema ha cambiado desde la última conexión, se realiza la suscripción.

   La suscripción consiste en llamar al método SUSCRIBE del Controlador asignado enviándole los datos del sistema en una estructura ICE denominada comm.ProviderData. Los campos de esta estructura son:

   - **mProc** (ICE string). Frecuencia del procesador.
   - **mProcType** (ICE string). Modelo de procesador.
   - **mMem** (ICE string). Memoria RAM.

- **mOs** (ICE string). Sistema operativo.
- **mSoftwareVersions** (ICE dictionary<string, string>). Diccionario cuyas claves son las constantes definidas en supported_engines.py para cada motor. El diccionario contiene las versiones de los motores instalados o *None* si no lo están.

4. **Actualización de la configuración.**

   Si los pasos anteriores se han realizado correctamente, se guardan en el fichero de configuración el nombre de usuario y la contraseña.

5. **Lanzamiento.**

   Se lanza el proceso *ProviderRunner* encargado de instanciar el objeto comm.ProviderI que se mantendrá a la espera de comunicaciones por parte del Distribuidor. El PID del proceso se almacena para que éste pueda ser manejado. Desde este momento, el Proveedor pasa a modo pasivo y se comporta como un servidor en el sentido de que proporciona un servicio.

6. **Comprobación de unidades de trabajo.**

   El Proveedor recorre su directorio de render (el definido por la constante POOLDIR) en busca de los ficheros de lanzamiento que tiene almacenados. Una vez hecho esto, invoca el método CHECKWORKUNITS de su Controlador pasándole la lista de los identificadores de estos lanzamientos. El Controlador le devuelve los lanzamientos que ya no son necesarios en el sistema y se pueden borrar.

7. **Activación.**

   Una vez que el objeto comm.ProviderI está listo para recibir peticiones, el Proveedor llama al método ACTIVATE de su Controlador indicándoselo al Distribuidor.

Si algo falla en cualquiera de los pasos, aparece un mensaje de error describiendo el problema encontrado.

Barra de progreso

Durante el proceso de conexión aparece un diálogo que muestra, mediante una barra de progreso, los pasos de los que se compone este proceso (Figura 4.42).

La clase FProvider conoce a una instancia de la clase ProgressDialog. Un vector con la descripción de las acciones que se llevan a cabo durante la conexión se le pasa al constructor

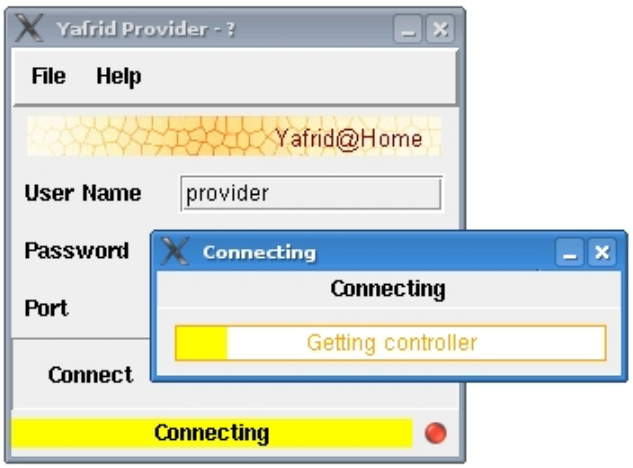

Figura 4.42: Barra de progreso durante la conexión.

de esta clase al iniciar el proceso. Cada vez que se alcanza uno de estos hitos, FProvider se lo notifica a su instancia de ProgressDialog que hace avanzar la barra de progreso una posición y muestra la descripción del hito siguiente.

4.15.3. Render de unidades de trabajo

Una vez que el Proveedor ha pasado a estar en modo pasivo, puede ocurrir que el Servidor le envíe una petición de servicio. Para ello, el Servidor instancia un proxy al objeto comm.Provider a partir de la clase comm.ProviderPrx y el proxy textual que almacena en su base de datos. Una vez instanciado el objeto, llama a su método *process* pasándole la unidad de trabajo a realizar. Esta unidad de trabajo viene descrita por una estructura ICE, comm.WorkUnit, que posee los campos siguientes:

- **mType** (ICE string). Establece el tipo de la unidad de trabajo tal y como aparece en supported_engines.py. De ello depende el tratamiento que le de el proveedor y, en concreto, qué motor de render se utilice.
- **mId** (ICE int). Identifica el lanzamiento del proyecto del que proviene la unidad de trabajo.
- **mNum** (ICE int). Es el orden de la unidad dentro del lanzamiento. Junto con el mId, supone la identificación única de la unidad dentro del sistema.
- **mMainFile** (ICE string). Nombre del fichero fuente que se le pasará al motor de render.

- **mResultFile** (ICE string). El nombre del fichero resultante del proceso de render.
- **mParameters** (ICE string). Cadena con los parámetros que describen la unidad y que la diferencian de las demás de un mismo lanzamiento. La cadena tiene un formato distinto dependiendo del tipo de unidad de trabajo (ver Apartado 4.10.1).
- **mResX** (ICE int). Resolución horizontal de la imagen.
- **mResY** (ICE int). Resolución vertical de la imagen. Ambas resoluciones se utilizan para el post-procesado de los resultados.

El envío de esta estructura desencadena un proceso que tiene una serie de pasos, que corresponden con cada una de las funciones estáticas de la clase ProviderFunctionLibrary:

- **Adquisición de ficheros fuente.** El proveedor se conecta al servidor SFTP del Servidor de Yafrid con los parámetros de configuración (la IP de la máquina, el nombre de usuario y la contraseña). Una vez conectado, adquiere el fichero del lanzamiento *projectfile.zip* que está en el directorio cuyo nombre es el identificador del lanzamiento (mId) y que contiene los ficheros fuente. Este fichero será almacenado en el directorio local del Proveedor destinado al render. Las siguientes unidades de trabajo de un mismo lanzamiento no requerirán la adquisición de los ficheros fuente.

  De este fichero comprimido serán extraídos los ficheros fuente necesarios para realizar el render. Para el manejo de los ficheros zip se ha desarrollado un módulo llamado zip.py que hace a su vez uso del módulo de Python zipfile.

- **Render.** En función del tipo de la unidad de trabajo, el EnginesManager devuelve el motor de render (instancia de una de las clases hijas de RenderEngineTemplate) adecuado para renderizarla. Una vez obtenida la instancia del motor, se ejecuta el método *render* del motor que devuelve el PID del proceso creado. El proveedor se queda entonces esperando a que termine dicho proceso con la función *waitpid* de la clase Process.

  En el caso de Blender, que no permite pasarle como parámetros las coordenadas, se ha de usar la técnica introducida en el Apartado 4.10.2. Los scripts escritos en Python que modifican las características de la escena se encuentran en el directorio *scripting*. Uno de ellos, *cropBlender.py*, redefine las coordenadas, los parámetros de calidad y la resolución de la escena. Esta información la obtiene, parte por línea de comandos, parte

del fichero project.info que acompaña al fichero de Blender. El otro, *normBlender.py*, sirve para modificar el directorio de salida de Blender.

- **Envío de resultados.** Una vez el render ha terminado, se realiza el post-procesado de los resultados mediante la función adecuada del motor de render. Estos resultados se envían comprimidos al Servidor vía SFTP y los ficheros temporales se eliminan.

  Cuando el envío de los ficheros ha finalizado correctamente, el Proveedor, por medio de su Controlador, avisa al Servidor de que la unidad de trabajo ha sido procesada correctamente invocando el método ENDEDWORK del mismo.

Se almacenan también los tiempos que requieren cada uno de los tres pasos, tiempos que serán enviados al servidor. Las distintas acciones que se realicen serán registradas en un fichero de log manejado por la clase log.Log.

Durante el tiempo que dure el proceso, el fichero cwu.info (en el directorio sys/proc/) contendrá el identificador de la unidad que está siendo procesada. Este dato podrá ser consultado por el Servidor para saber si una unidad de trabajo sigue siendo procesada o bien se ha perdido invocando al método getCurrentWorkUnit del objeto comm.Provider. La clase que gestionará esta información se denomina CurrentWorkUnit y se encuentra en el módulo Python current_workunit.py.

Los módulos zip.py, log.py y current_workunit.py pertenecen al paquete de persistencia y las clases que lo componen se muestran el la Figura 4.43.

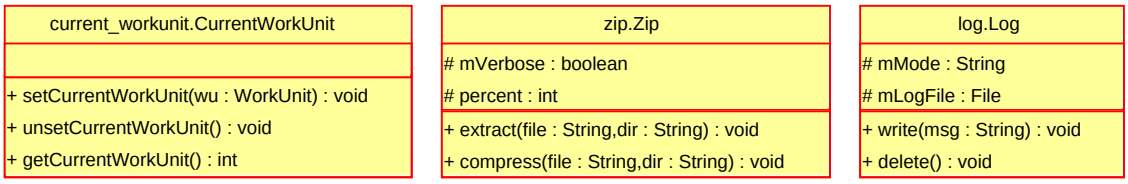

Figura 4.43: Otros módulos del paquete de persistencia.

4.15.4. Desconexión del grid

En cualquier momento, el usuario puede finalizar su participación en el grid. Para ello, puede desconectarse o bien salir del programa. En ambos casos las acciones que se realizan son:

- Se elimina la información sobre la unidad de trabajo que se estaba procesando en caso de que la hubiera.
- Se ejecuta el comando BYE del controlador asociado. Esto hace que el grid deje de contar con el Proveedor a partir de ese momento.
- Se finaliza el núcleo de comunicaciones de ICE.
- Se finaliza el proceso ProviderRunner.
- Se finaliza el proceso creado para gestionar el render de la escena si lo hubiera.

4.15.5. Diálogos adicionales

Como se puede ver en la Figura 4.44, se ha desarrollado una pequeña jerarquía de clases para manejar los diálogos que aparecen en el sistema.

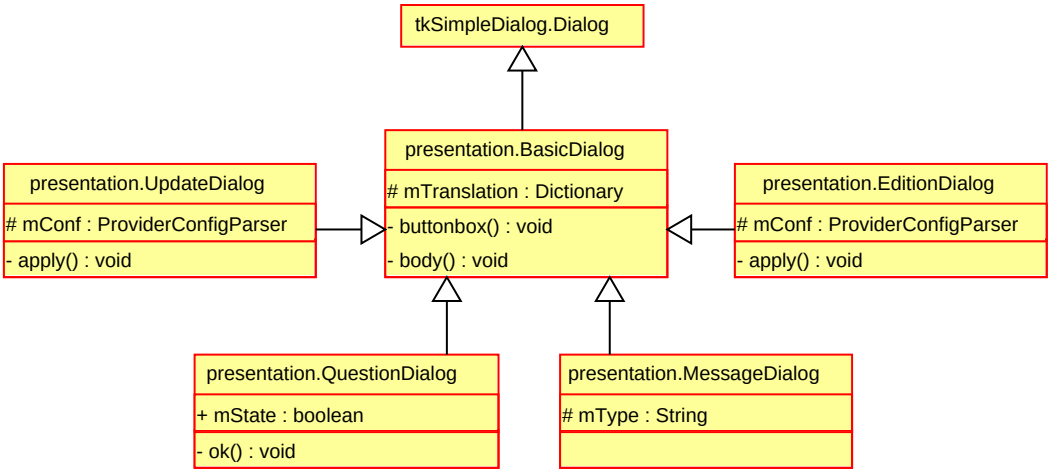

Figura 4.44: Jerarquía de diálogos (GUI).

La raíz de esta jerarquía la constituye la clase tkSimpleDialog.Dialog de la que hereda presentation.BasicDialog. Esta segunda clase sobreescribe algunos de los métodos de la primera para impedir que se puedan modificar las dimensiones de los diálogos, para que muestren los componentes con el aspecto visual del resto de la aplicación y para que tomen los textos del diccionario de traducciones.

De este diálogo básico heredan el resto:

- EditionDialog. Accesible desde *Archivo → Editar configuración*. Muestra los valores de las constantes que encuentre en el fichero de configuración del proveedor permitiendo su edición. Utiliza para ello la clase ProviderConfigParser del módulo configuration.py.

- UpdateDialog. Accesible desde *Archivo → Actualizar IP Servidor*. Permite automatizar la adquisición de la ip del servidor Yafrid. Ésta se obtiene de un fichero denominado *ip* que se encuentra en el servidor. Para obtener dicho fichero se utiliza *wget* pasándole como parámetro la url que se introduzca en el diálogo. Por defecto se toma la definida por la constante YAFRIDURL. Una vez adquirido este dato, se actualiza el fichero de configuración mediante la clase ProviderConfigParser.

- QuestionDialog. Muestra un mensaje con una pregunta que el usuario debe responder afimativa o negativamente para lo que tiene los típicos botones *aceptar* y *cancelar*.

- MessageDialog. Muestra un mensaje al usuario. Puede ser de dos tipos, un mensaje de información o un aviso, de lo que dependerá el símbolo que se muestre en el diálogo. Sólo posee un botón y el pulsarlo no tiene ningún efecto aparte de cerrar el diálogo.

Todos son diálogos modales ya que suspenden la ejecución del programa hasta que el usuario responde haciendo clic en un botón del diálogo.

La funcionalidad de los dos últimos la proporcionaban los métodos tkMessageBox.askokcancel, tkMessageBox.showinfo y tkMessageBox.showwarning de Tkinter. La razón de que se hayan desarrollado otros diálogos es que los mencionados no permiten cambiar el idioma del texto de los botones ya que éste se obtiene a partir del idioma del sistema operativo. Para lograr que se pudiera parametrizar el idioma de los botones, se desarrollaron otros diálogos a los que se les pasa un diccionario de traducciones.

4.15.6. Obtención de las características del sistema

Con el fin de caracterizar a los proveedores a partir de sus equipos es necesario enviar cierta información al Servidor en el momento de la suscripción. Del sistema se recuperan la memoria RAM, el sistema operativo, el modelo y la frecuencia del procesador y las versiones de los motores de render instalados.

Para obtener esta información se ha desarrollado un módulo Python denominado *sys_info.py*. Este mismo módulo es el que también se usa en el Distribuidor para obtener las características de la máquina donde reside el Servidor de Yafrid. El módulo sólo contiene

una clase, *SysInfo*, con un método *get* que devuelve un diccionario con la información del sistema. Estos datos tienen distintos orígenes:

- **Modelo de procesador.** Se procesa el fichero de sólo lectura */proc/cpuinfo* y se toma el valor de la clave *model name*.
- **Frecuencia del procesador.** Se obtiene de la clave *cpu MHz* del fichero de sólo lectura */proc/cpuinfo*.
- **Versión de sistema operativo.** Se obtiene de la función *platform.platform()* de Python.
- **Memoria RAM.** Se procesa el fichero de sólo lectura */proc/meminfo* y se utiliza el valor de la clave *MemTotal*.
- **Versiones de los distintos motores de render instalados.** Para cada uno de los motores soportados por el Proveedor (supported_engines.py) se ejecuta el método *getVersion*.

Para que esta aproximación funcionara tanto en GNU/Linux como en MS Windows, en éste último fue necesario utilizar recursos del proyecto Cygwin [27]. Así, en la distribución del proveedor de Yafrid para MS Windows se incluye el ejecutable *cat.exe* y las librerías necesarias para que *cat /proc/cpuinfo* y *cat /proc/meminfo* devuelvan los mismos valores en ambos sistemas operativos.

Una vez que el Proveedor se ha suscrito, la información del sistema puede ser consultada desde el interfaz gráfico del proveedor desde *Ayuda* → *Sistema*.

4.15.7. Parametrización

En el fichero *yafrid-provider.conf* se encuentran una serie de variables que sirven para parametrizar el software. Estas variables son:

- **IDENTIFICATORIP.** Determina la IP de la máquina donde se encuentra el proceso Identificador (Puede obtenerse automáticamente con la herramienta de actualización).
- **POOLDIR.** Ruta del directorio local donde se van a guardar los ficheros recibidos del grid. También será el directorio de trabajo a la hora de llevar a cabo el render.
- **YAFRIDURL.** URL a la que se hará referencia para actualizar la IP del Identificador y realizar otras adquisiciones.

- **DIRINSTALL.** Directorio donde está instalado el software Proveedor. Se actualiza automáticamente.
- **USER.** Nombre de usuario que se introdujo la última vez que se produjo una conexión correcta con el grid.
- **PASSWORD.** Contraseña que se introdujo la última vez que se produjo una conexión correcta con el grid.
- **FTP_USER.** Nombre de usuario del protocolo utilizado para acceder a los ficheros fuente de las unidades de trabajo asignadas.
- **FTP_PASSWD.** Contraseña del protocolo utilizado para acceder a los ficheros fuente de las unidades de trabajo asignadas.
- **LANGUAGE.** Lenguaje en el que se mostrará la interfaz de usuario.
- Además, puede haber otras variables que representen la ruta en la que se encuentran los ejecutables de los programas de render. El nombre de estas variables viene definido en el fichero *supported_engines.py* como *constant_path*. En el caso de Yafray y Blender se denominan, YAFRAYPATH y BLENDERPATH respectivamente. En caso de ser nulas, se supone que su ruta aparece en el PATH del sistema y se intenta utilizar el programa como aparece en *default_name*.

Capítulo 5

RESULTADOS

5.1. RENDIMIENTO DEL SISTEMA

En diversas etapas del desarrollo del sistema se han realizado pruebas con los distintos prototipos que se iban obteniendo. Mientras que en las primeras fases se realizaban pruebas sencillas en una sola máquina, cuando el sistema estuvo mínimamente maduro se comenzaron a realizar pruebas en un entorno distribuido y controlado. En esta sección se muestran algunos de los resultados obtenidos en estas pruebas así como los resultado esperados según los análisis previos.

5.1.1. Resultados analíticos

Para el siguiente estudio se suponen conexiones simétricas entre Servidor y Proveedores y capacidades de cálculo similares de los distintos Proveedores.

Como vemos en la Ecuación 5.1, al utilizar Yafrid para renderizar una escena se añaden dos factores al tiempo de render, T_{trans} y T_0, que no aparecen en un sistema no distribuido y con un solo ordenador.

El T_{trans} es el tiempo de transmisión y depende en parte de la velocidad de la red que une Servidor y Proveedores. T_0 es un tiempo que el sistema utiliza en operaciones tales como la unión de las unidades de trabajo y que se puede acotar en función del tamaño del proyecto.

$$T_{total} = T_{render} + T_{trans} + T_0 \tag{5.1}$$

Tanto en Yafrid como en un sistema centralizado, el tiempo de render responde a la Ecuación 5.2 donde fps corresponde a los frames por segundo de la animación, t_{al} es la duración de la animación y t_{rf} es el tiempo medio en renderizar un frame.

$$T_{render} = \begin{cases} fps \times t_{al} \times t_{rf} & \text{en el caso de animaciones} \\ t_{rf} & \text{en el caso de frames} \end{cases} \tag{5.2}$$

El tiempo de transmisión se calcula mediante la Ecuación 5.3 en la que n_r es el número de Proveedores (o nodos de render), S_{pet} es el tamaño en bytes del fichero fuente que el Servidor envía a los Proveedores, S_{res} es el tamaño en bytes del fichero resultado que cada Proveedor envía al Servidor, W_i es el número de unidades de trabajo realizadas por el Proveedor i y $Vtrans_i$ es la velocidad de transmisión (ancho de banda) entre Servidor y Proveedores en bytes/s. El número de Proveedores es lo que se denomina tamaño del grid.

$$T_{trans} = \sum_{i=0}^{n_r} \frac{(S_{pet} + (S_{res} \times W_i))}{Vtrans_i} \tag{5.3}$$

Así, en un sistema con un sólo ordenador no aparecen los tiempos extra T_{trans} y T_0 que sí aparecen en un sistema como Yafrid. Sin embargo, el proceso se desarrolla en paralelo en distintas máquinas con lo que, en un caso ideal, el tiempo total se divide equitativamente entre los distintos Proveedores del sistema (Ecuación 5.4).

$$T = \frac{T_{total}}{n_r} \tag{5.4}$$

En el caso de que las redes que conectan los componentes del grid sean de alta velocidad, el término T_{trans} se puede despreciar, con lo que la ganancia sería del orden del número de Proveedores n_r (n_r sería el valor máximo de la ganancia).

En futuras mejoras del sistema se pretende optimizar la generación de unidades de trabajo en proyectos de render estático de forma que éstas se rendericen con distintas calidades. Esto permitiría obtener ganancias por encima de n_r ya que la suma de los tiempos en renderizar

Método	Calidad	Tiempo
Pathtracing	128 Muestras	02:56:56
Pathtracing	512 Muestras	07:35:28
Pathtracing	1024 Muestras	14:23:07
Pathtracing	2048 Muestras	23:16:12
RT Clásico	1 Rayo/Píxel	00:17:50
RT Clásico	8 Rayos/Píxel	00:20:39
RT Clásico	16 Rayos/Píxel	00:26:47

Cuadro 5.1: Resultados obtenidos al renderizar en una sola máquina

cada uno de los fragmentos sería menor que el tiempo en renderizar la imagen completa debido a que algunos de los fragmentos se renderizan con una calidad menor que la inicial.

5.1.2. Resultados empíricos

Con el fin de sacar conclusiones sobre el comportamiento del sistema en un entorno real, se ejecutaron varios conjuntos de pruebas (test sets). Estos conjuntos de pruebas se pueden generar desde el interfaz web con permisos de administrador y consisten en una serie de proyectos de prueba (tests). Cada uno de los conjuntos de pruebas posee sus propios parámetros de calidad (como son por ejemplo el número de muestras de luz o el número de rayos por píxel) y su propio método de render. Dentro de un mismo conjunto de pruebas, los distintos proyectos de prueba se diferencian en el valor de los parámetros de calidad y en el tamaño de las unidades de trabajo en las que se dividirá el frame.

Para hacer que los datos que se obtuvieran pudieran ser comparables, se optó por configurar un grid homogéneo en el que todos los proveedores tuvieran similares características. Así, los proyectos de prueba mencionados se lanzaron en Yafrid mientras el grid tenía un tamaño de 16 proveedores idénticos en capacidades (procesador Pentium IV a 2.80 GHz, 256Mb de RAM y el mismo sistema operativo). Este tamaño de grid se mantuvo constante durante toda la prueba.

Para realizar las pruebas se seleccionó una imagen bastante compleja. La escena contiene más de 100.000 caras, 4 niveles de recursión en el algoritmo de trazado de rayos en superficies especulares (el dragón) y 6 niveles en superficies transparentes (el cristal de las copas). Además, se lanzaron 200.000 fotones para construir el *mapa de fotones*.

Para generar la imagen se utilizaron dos métodos de render:

- **Pathtracing** con **mapeado de fotones** [Jen01] e **irradiance cache** (Figura 5.3).
- **Trazado de rayos** clásico [Whi80] con una implementación de la técnica **ambient occlusion** (Figura 5.4).

Antes de presentar los resultados obtenidos con Yafrid (con un tamaño de grid de 16 ordenadores), en la Tabla 5.1 se muestran los tiempos de render de cada conjunto de pruebas obtenidos usando un solo proveedor (render local).

Como se puede observar en la Figura 5.2, el tiempo de render en el mejor de los casos es más de ocho veces mejor utilizando Yafrid. En el peor de los casos es tan sólo dos veces mejor.

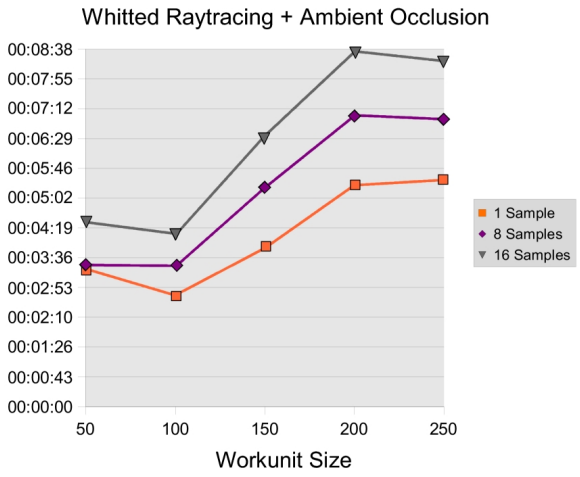

Figura 5.1: Resultados usando Raytracing clásico.

Del mismo modo, utilizando el algoritmo de trazado de rayos de Whitted (Figura 5.1), el tiempo de render usando Yafrid es, en el mejor de los casos, siete veces menor y tan sólo tres veces menor en el peor de los casos.

También se lanzó una sencilla animación de prueba consistente en un movimiento de cámara en torno a los objetos principales de la escena usada en las pruebas anteriores. La duración de la animación se estableció en 50 fotogramas y se utilizó el algoritmo de trazado de rayos clásico.

Tal y como se muestra en la Figura 5.2 en los proyectos de animación se consiguen ganancias más importantes que en el caso del render estático. La ganancia está, en este caso, más cercana al límite teórico que suponía el número de proveedores. El tiempo obtenido con Yafrid es alrededor de catorce veces menor.

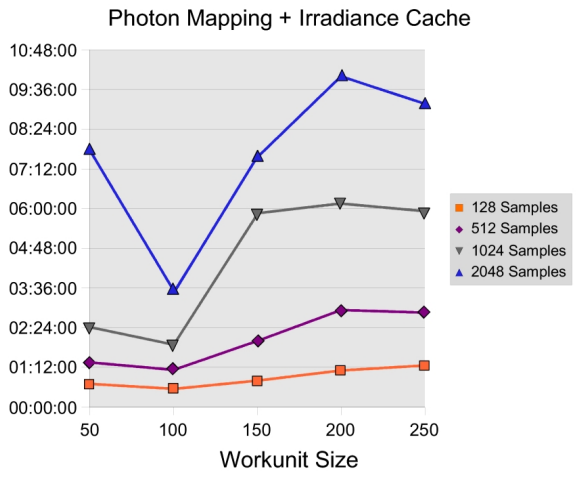

Figura 5.2: Resultados usando Mapeado de Fotones.

5.1.3. Análisis de los resultados

Con la observación de los resultados empíricos obtenidos en el apartado anterior, se pone de manifiesto la importancia que tiene la elección del tamaño de la unidad de trabajo en los proyectos de render estáticos. Para cada escena hay un tamaño óptimo de unidad de trabajo que permite obtener el mejor rendimiento posible.

En esta escena en concreto hay algunas unidades de trabajo que tardan mucho más que el resto (como las cáusticas de las copas de cristal). Esto hace que el proyecto se mantenga en proceso gran parte del tiempo con únicamente algunos fragmentos por terminar. Esto enmascara la ganancia obtenida perjudicando el tiempo que se ha dedicado al proyecto.

En el método usado por Yafray para realizar el render (Pathtracing), los fragmentos a calcular no son totalmente independientes unos de otros. Para obtener las unidades de trabajo, cada Proveedor tiene que calcular la *irradiance cache* completa cada vez. La mejora aumentaría mucho si un Proveedor calculara la caché una vez y se la enviara a los demás

Calidad	Tiempo Yafrid	Tiempo 1 PC
1 Rayo/Píxel	01:03:50	14:20:55
8 Rayos/Píxel	01:15:12	16:33:10
16 Rayos/Píxel	01:22:36	20:12:02

Cuadro 5.2: Resultados obtenidos al renderizar una animación.

Figura 5.3: Imagen generada utilizando Pathtracing (512 muestras).

Figura 5.4: Imagen generada usando Raytracing clásico (1 muestra).

Proveedores (mecanismo aún no implementado en Yafrid).

La ganancia es mayor en el método de trazado de rayos de Whitted debido a que no se hace uso de la *irradiance cache*.

Además existe otro factor por el que la ganancia es menor que la teórica. A la hora de dividir una imagen en fragmentos se añade un área extra, la banda de interpolación, que servirá para suavizar las uniones entre unos fragmentos y otros. Usar una banda de interpolación menor—en los ejemplos tiene una anchura de 10 píxeles—mejoraría la ganancia.

Con respecto a las animaciones, la situación es bastante distinta. La ganancia obtenida experimentalmente está más cerca de ser proporcional al tamaño del grid que en el caso del

render de un solo frame, es decir, está más cerca de la ganancia teórica. Una de las causas es que, habitualmente, el tiempo de render es más homogéneo entre un frame y el siguiente que entre fragmentos de un mismo frame. Además, el tiempo invertido en unir los frames de una animación es menor que el invertido en unir e interpolar los fragmentos de un frame. Otra razón importante es que no hay que renderizar un área extra ya que no existe banda de interpolación.

5.1.4. Conclusiones sobre el rendimiento del sistema

De acuerdo con los resultados empíricos examinados con anterioridad, el sistema proporciona claramente un mejor rendimiento que en el caso del render en un sistema con un sólo procesador. Esto es así incluso cuando la elección del tamaño de la unidad de trabajo es la peor de las posibles.

Yafrid ofrece una significativamente mejor **productividad** (número de entidades renderizadas, frames o fragmentos de un frame, por unidad de tiempo) que un único ordenador porque las distintas unidades de trabajo se generan paralelamente en distintos Proveedores. Esta productividad depende en gran medida del tamaño de la unidad de trabajo y del tamaño del grid.

Por otro lado, la **latencia** de cada una de las unidades (el tiempo que tardan en ser generadas) se incrementa porque hay que tener en cuenta factores que no existen en el procesamiento no distribuido. En sistemas distribuidos de cualquier tipo, además del tiempo dedicado a renderizar la escena, el tiempo de transmisión es también importante. Este tiempo de transferencia depende de la cantidad de datos a transferir, de la velocidad de la conexión y del estado de la red.

El aumento de la latencia no es importante si obtenemos una mayor productividad, que es lo que en realidad va a percibir el usuario final del servicio.

Por regla general, cuanto mayor sea el número de fragmentos en los que se divide una escena, mayor paralelismo se obtendrá y menos tiempo se necesitará para renderizar la imagen completa. Si analizamos el tema con más detalle, esto no es tan simple. Si una escena se divide en fragmentos demasiado pequeños es posible que el tiempo dedicado a la transmisión y al procesamiento de los resultados sea mayor que el tiempo dedicado al render. En este caso, los resultados obtenidos por un sistema distribuido de cualquier tipo serán peores que los obtenidos en una sola máquina. Una elección incorrecta del tamaño de la unidad de trabajo puede perjudicar el rendimiento del grid.

5.2. RESULTADOS ECONÓMICOS

En esta sección se aborda una estimación de los requisitos del sistema para su implantación en un entorno real. Con esta información se podrá elegir, de entre las distintas alternativas que ofrece el mercado, la que más se ajuste a las necesidades estimadas.

Una vez tomadas las decisiones necesarias para la implantación de Yafrid, será necesario plantear distintas opciones para sufragar los costes resultantes de esas decisiones.

5.2.1. Estimación de los requisitos del sistema

Para cuantificar los costes derivados de la implantación de Yafrid en un entorno real es necesario realizar un análisis previo estimando las necesidades del sistema en cuanto a recursos software, hardware y de transferencia de datos.

A continuación se lleva a cabo una estimación de los requisitos del sistema basada en una serie de hipótesis que bien podrían darse en la realidad.

Suponemos en primer lugar que el **espacio** medio que el sistema utiliza para un proyecto Yafrid es de 11 MB, tal y como se detalla a continuación.

- Proyecto Render Estático. Total: 5,3 MB
  - Fichero ZIP subido por el cliente: 2,5 MB.
  - Fichero ZIP replicado en el POOL: 2,5 MB.
  - Imágenes parciales / Imagen final: 0,3 MB.
- Proyecto Animación. Total: 60 MB
  - Fichero ZIP subido por el cliente: 2,5 MB.
  - Fichero ZIP replicado en el POOL: 2,5 MB.
  - Fotogramas parciales / Animación final (1 minuto, comprimida): 60 MB.
- Si el número de proyectos estáticos representase el 90 % del total de proyectos y los de animación el 10 % restante, el tamaño medio de un proyecto Yafrid sería 10,77 MB. Se toma el valor de 11 MB para redondear y para tener en cuenta otros gastos de espacio (otro ficheros auxiliares, datos asociados, etc).

  Tamaño Proyecto = 0,90 x 5,3 MB + 0,10 x 60 MB = 10,77 MB

Si a cada cliente se le establece una cuota máxima de 10 proyectos simultáneos en el sistema, se obtiene que cada cliente supondría un gasto en almacenamiento de 110 MB como media. Para que este valor también sea un máximo, se puede establecer una cuota de 110 MB en proyectos, o bien, un máximo de 10 proyectos, lo que antes de rebase.

Espacio 1 cliente = 11 MB/proyecto x 10 proyectos = 110 MB

Estimando el número de clientes de Yafrid en 200, se obtiene que el espacio de almacenamiento necesario para mantener los 2000 proyectos de dichos clientes tendría que ser al menos de 22 GB.

Espacio 200 clientes = 110 MB/cliente x 200 clientes = 22000 MB = 22 GB

Si además tenemos en cuenta que en cualquier momento dado el 25 % de los proyectos en el grid estarían finalizados, los ficheros del POOL no existirían en el caso de estos proyectos. Esto produciría una disminución en las necesidades de almacenamiento en un 5,7 %.

Espacio 2000 proyectos = 2000 proyectos x 0,75 x 11 MB/proyecto +
2000 proyectos x 0,25 x (11 - 2.5) MB/proyecto = 20750 MB = 20,75 MB

Otro factor básico a tener en cuenta es la necesidad de **transferencia de datos** que requiere el sistema. Partiendo de las estimaciones anteriores y estableciendo el número de proyectos en 2000 (200 clientes con un máximo de 10 proyectos cada uno) y el de proveedores en 100, se obtienen los siguientes resultados.

- Ficheros fuente (Servidor → Proveedor) Si en cada proyecto participan el 25 % de los proveedores:

  Transferencia fuentes = (2,5 MB/proyecto x 2000 proyectos)/proveedor
  x 25 proveedores = 125 GB

- Imágenes parciales (Proveedor → Servidor) = 0,3 MB/proyecto x 2000 proyectos = 0,6 GB

- Otras comunicaciones = 100 MB

- Navegación web en general = 100 MB

- Subida de ficheros = 2,5 MB/proyecto x 2000 proyectos = 5000 MB = 5 GB

- Descarga de ficheros = Imagen Final x Num.Proyectos = 300 KB/proyecto x 2000 proyectos = 600 MB = 0,6 GB

- Total = 131,4 GB

En cuanto a las necesidades de **procesamiento**, sería conveniente un procesador de gama media/alta ya que hay que gestionar numerosos servicios. Dichos servicios irían desde servidores de bases de datos, SFTP o HTTP a los propios servicios de Yafrid, lo que supondría un consumo de CPU a tener en cuenta. Esto también es aplicable en cuanto a las necesidades de memoria RAM de la máquina a utilizar.

5.2.2. Costes de la implantación del sistema

Esta estimación inicial de los requisitos del sistema junto con las decisiones tomadas a lo largo del desarrollo servirán como guía a la hora de decidir qué alternativa escoger para implantar el sistema. Esta decisión determinará en gran medida los costes asociados a la implantación de Yafrid para su uso real.

Las opciones más comunes que se podrían plantear inicialmente son:

- **Contratar un servicio de hosting web.**

  En la actualidad, son muchas las compañías que ofrecen este servicio a través de sus páginas web. Se trata de un servicio muy demandado y con unos precios más que asequibles. Sin embargo, no es una solución viable para implantar el sistema Yafrid debido a varias razones.

  Este tipo de servicios está orientado a otros tipos de sistemas cuyas necesidades distan bastante de las que un sistema como Yafrid tiene. Están destinados principalmente a desarrolladores de aplicaciones web. Yafrid tiene una parte web que perfectamente podría ser implantada con un servicio de hosting. Sin embargo, esto es sólo una parte del sistema. El intercambio de información entre el Servidor y los Proveedores supone un ancho de banda considerable que supera las tasas de transferencia ofertadas por los proveedores de hosting. Esto también ocurre con el espacio de almacenamiento que también es insuficiente.

  Además, algunas acciones necesarias para hacer funcionar un sistema como Yafrid no son posibles cuando se contrata un servicio de hosting. En Yafrid, por ejemplo, es necesario instalar software con permisos de administrador y establecer procesos que

escuchen en ciertos puertos. Un servicio de este tipo no proporciona tanto control sobre las aplicaciones y el sistema en general.

Además, algunos servicios de valor añadido que se ofrecen en otras opciones de implantación serían prácticamente imprescindibles y no son incluidas en las ofertas de hosting.

- **Alquiler de un servidor dedicado.**

  Otra de las opciones que se pueden contemplar para poner en explotación el sistema consiste en el alquiler de un servidor dedicado lo que proporciona un control total con respecto a las tecnologías a utilizar

  En el Cuadro 5.3 aparecen algunos sitios de la red que ofertan servidores dedicados. Las características del servidor vienen acompañadas de las tasas de transferencia ofrecidas y el precio tanto en plataformas GNU/Linux como en MS Windows.

  Es interesante destacar lo siguiente sobre las ofertas de los sitios analizados:

  - Aunque no aparezca en el Cuadro 5.3 por falta de espacio, las compañías que ofertan servidores dedicados ofrecen una serie de servicios de valor añadido muy útiles. Entre estos servicios se cuenta el backup automatizado, soporte técnico 24x7, direcciones IP, parada y arranque, instalación y puesta a punto, monitorización y un largo etcétera.
  - Son varias las compañías que carecen de servidores con MS Windows pero todas poseen servidores GNU/Linux.
  - En los casos en los que se ofertan servidores con sistemas operativos GNU/Linux y MS Windows, los precios de estos últimos son más altos. Esto, unido al uso de tecnologías libres, hace que la utilización de un servidor GNU/Linux para la implantación sea mucho menos costosa.

    La media del coste de un servidor dedicado con GNU/Linux es de 1525,60 €/año mientras que si el servidor tiene MS Windows, el coste se eleva a 1734,15 €/año, es decir, un 14 % más.
  - Las características ofrecidas por la mayor parte de las ofertas analizadas satisfacen los requisitos que se han estimado para el sistema.

- **Adquisición de un servidor.**

	Servidor	Transferencia	GNU/Linux	MS Windows
ferca.com	P4 3 GHz 512 MB RAM 80 GB	1000 GB/mes	1668 €/año	1848 €/año
arsys.com	P4 3 GHz 512 MB RAM 80 GB	1000 GB/mes	1188 €/año	1788 €/año
acens.com	P4 3,2 GHz 512 MB RAM 80 GB	30 GB/mes	3000 €/año	3588 €/año
comalis.com	Celeron 2,4 GHz 256 MB RAM 40 GB	660 GB/mes	588 €/año	1068 €/año
digitalvalley.com	Xeon 2,8 GHz 1 GB RAM 36 GB	500 GB/mes	3384 €/año	Carece de máquinas MS Windows
interdominios.com	P4 3 GHz 512 MB RAM 80 GB	1200 GB/mes	1069,20 €/año	1393,20 €/año
hostinet.com	Dual P4 3 GHz 2 GB RAM 250 GB	2000 GB/mes	2100 €/año	2100 €/año
espaweb.com	Celeron 2,4 GHz 512 MB RAM 160 GB	10 GB/mes	1379,40 €/año	Carece de máquinas MS Windows
argored.com	P4 3,2 GHz 512 MB RAM 80 GB	Consultar	1068 €/año	1500 €/año
ran.es	Intel 2,4 GHz 256 MB RAM 80 GB	1024 GB/mes	468 €/año	588 €/año
simbionet.com	P4 1,6 GHz 512 MB RAM 80 GB	700 GB/mes	869 €/año	Carece de máquinas MS Windows

Cuadro 5.3: Alternativas de servidores dedicados. Como se puede observar, existe una gran variedad en este tipo de servicios. También existe mucha diferencia en cuanto al precio, que puede ser debido a cuestiones tales como la calidad de los equipos o los servicios de valor añadidos ofertados.

Servidor	Procesador	Mem. RAM	Almacenamiento	Precio (€)
HP ML110G3	P4 3 GHz	512 MB	80 GB	429
IBM x206m Express	P D 820 2,8 GHz	512 MB	78 GB	1.175,57
IBM Express x346	Xeon 3,2 GHz	1GB	3 x 73,4 GB	3.182,60
IBM TS eServer X100	P4 3 GHz	512 MB	80 GB	569
Fujitsu Siemens Primergy Ec. 100	P4 630 3 GHz	1 GB	160 GB	816,08
HP Proliant DL140 G2	Xeon 2,8 GHz	1 GB	80 GB	929
AIRIS Atrio SR210	Xeon 2,8 GHz	512 MB	80 GB	1857,26

Cuadro 5.4: Ofertas de servidores

La última de las opciones consideradas para la implantación de Yafrid consiste en la adquisición de un servidor propio para alojar el sistema. En el Cuadro 5.4 se muestra el precio de algunos servidores que podrían satisfacer los requisitos necesarios.

Además del coste de adquisición de la máquina habría que tener en cuenta otras muchas cuestiones como el lugar físico donde disponer el servidor, los costes asociados al mantenimiento del mismo, el coste de la línea, el de la energía eléctrica, etcétera.

La solución más adecuada sería el alquiler de un servidor dedicado por su mayor comodidad frente a la opción de la adquisición y porque ofrece los recursos necesarios para el sistema. Los servidores con un sistema operativo GNU/Linux son más económicos y en concreto los sitios *comalis.com* y *ran.es* ofrecen las ofertas más interesantes.

Además, la decisión de utilizar sistemas GNU/Linux para servidores de todo tipo es la más utilizada por las características de estabilidad y robustez de estos sistemas.

5.2.3. Alternativas de amortización

Para hacer frente a los costes derivados de la implantación y operación del sistema es necesario pensar en algún mecanismo que permita recuperar la inversión realizada e incluso hacer de Yafrid un sistema rentable.

Algunas posibles opciones que podrían tenerse en cuenta se enumeran a continuación. La mayoría no son excluyentes entre sí con lo que podrían usarse varias simultáneamente para conseguir unos mayores beneficios.

- **Publicidad tradicional.** Una de las prácticas más comunes consiste en incluir en una parte del interfaz web una o varias imágenes publicitarias anunciando algún producto o servicio. Al hacer clic sobre el anuncio, el usuario sería redirigido a la página

del anunciante. Debido a la temática del proyecto, entidades dedicadas al área de la generación de gráficos 3D podrían estar interesadas en anunciarse en Yafrid.

Si se optara por la oferta de servidor dedicado más económica, la de ran.es de 468 €/año, bastaría con obtener por publicidad unos 39 € al mes para amortizar los costes generados por la implantación del sistema

- **Google Adsense.** Es un sistema desarrollado por Google [1] y que permite a los desarrolladores web incluir pequeños anuncios de una forma fácil en sus páginas. Lo interesante de este servicio es que los anuncios que ofrecen están ya segmentados por áreas con lo que corresponden con la temática de la propia página. El algoritmo en el que se basa Google para pagar a los desarrolladores no ha sido detallado pero se basa en dos conceptos, los clics y las impresiones por clic. Básicamente, cuanta más gente visite la página, más probable es que hagan clic en alguno de los anuncios y mayor es la cantidad que se obtiene.

  Tanto en esta alternativa como en la anterior, la publicidad tradicional, es importante atraer al mayor número de usuarios posibles. Para ello habrá que cuidar la presentación de la página y actualizar los contenidos con frecuencia. También es importante colocar los anuncios en posiciones estratégicas con el fin de maximizar los beneficios. En la Figura 5.5 aparece un *mapa de calor* en el que se muestran en colores más cálidos las zonas que se supone obtienen mejores resultados. Esto es sólo orientativo, ya que los mejores lugares para colocar la publicidad dependerán en gran medida de los contenidos de cada página

- **Autofinanciación.** Esta aproximación consiste básicamente en establecer una especie de **mercado de ciclos** en el que los proveedores ofrecen un producto (ciclos / segundos dedicados a tareas de render) que los clientes adquieren a un determinado precio. Así, a la hora de lanzar un proyecto, los clientes podrían especificar un precio máximo por ciclo o el máximo precio que están dispuestos a pagar por el render completo. En ambos casos, el sistema tendría que informar de si es posible o no satisfacer los requisitos que introduce el usuario en cuanto a precio.

  Aunque se ha hablado del término mercado de ciclos, el ciclo de CPU no sería una unidad adecuada sobre la que establecer el sistema por ser muy pequeña y poco intuitiva para el potencial cliente. En realidad, la unidad mínima sobre la que se establecerían las tarifas sería el segundo dedicado al render, *SoR* (del inglés *Second of Rendering*).

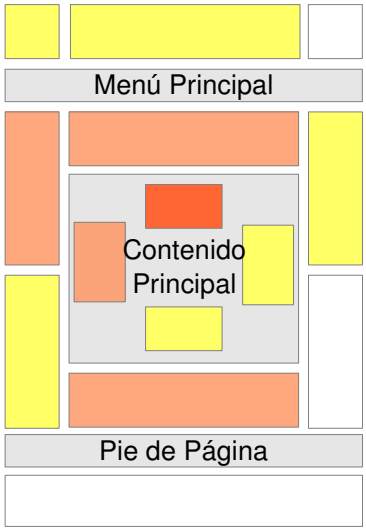

Figura 5.5: Mapa de calor de las zonas más adecuadas para insertar publicidad.

En cuanto al precio por SoR, *CSoR* (del inglés, *Cost per Second of Rendering*) se podría establecer de distintas formas:

- **Por parte del sistema**

  El CSoR se establece automáticamente cuando un proveedor se conecta por primera vez al grid en función de parámetros del propio proveedor como pueden ser la memoria RAM, el modelo de procesador, la frecuencia del mismo o la velocidad de conexión. En función de estas características se establecería una clasificación de proveedores donde cada categoría tendría un CSoR distinto. Así, los proveedores con equipos más potentes y conexiones más rápidas podrían pedir más por sus SoR.

  Además, este coste podría variar en función del comportamiento del proveedor durante su vida en el sistema. El CSoR de un proveedor que genere unidades de trabajo erróneas o directamente las pierda bajará paulatinamente, mientras que el de un proveedor que vaya completando tareas correctamente experimentará un incremento en su CSoR.

- **Establecido por el propio proveedor**

  Este sistema daría lugar a una especie de *subasta de ciclos* en el que un proveedor establecería su propio CSoR inicialmente y lo tendría que ir modificando según las necesidades del mercado. Así, si no recibe peticiones de render es posible que

haya establecido un precio muy alto y los clientes usen otros proveedores más económicos para sus proyectos. Si por el contrario recibe muchas más peticiones de las que puede satisfacer, no es mala idea subir el precio, ya que aunque las peticiones bajarán, saldrá beneficiado económicamente.

El mecanismo de pago se podría establecer usando PayPal [22]. Este sistema permite el intercambio sencillo y seguro de dinero en la web. Lo único que se necesita para transferir dinero a alguien es conocer su dirección de correo. Para incluir su funcionalidad en el sistema, sería necesario asociar cada cuenta de Yafrid con una cuenta Paypal. Además, los datos necesarios para levantar esta infraestructura ya son almacenados por Yafrid con lo que su implantación sería bastante asequible:

- Dirección de correo de los distintos usuarios.
- Tiempos detallados dedicados a cada proyecto.
- Tiempos dedicados por cada proveedor a cada proyecto.

Con estos datos, los clientes pagarían a Yafrid una cantidad por los proyectos renderizados. Sería el propio Yafrid el que pagaría a los proveedores según sus tarifas quedándose además con un tanto por ciento de cada transacción realizada.

Este sistema favorecería además que Yafrid contase con un volumen de proveedores constante que de otro modo se verían desmotivados a participar en el proyecto y ceder sus ciclos de CPU sin obtener nada a cambio.

Otra ventaja sería la posibilidad de intercambiar dinero en diferentes divisas lo que facilitaría la existencia de proveedores de distintas nacionalidades.

- **Modificación a medida del sistema.** El sistema se podría modificar y adaptar a las necesidades de un determinado proyecto a gran escala como la realización de un corto con software libre por ejemplo. En este caso, el coste dependería del tiempo invertido en modificar el sistema Yafrid.

- **Ofrecer servicio de render dedicado.** Se podría pensar en ofrecer los servicios que proporciona Yafrid de manera exclusiva a clientes determinados. El coste dependería de las necesidades de dichos clientes.

- **Venta del código fuente.** El código fuente completo del sistema Yafrid podría ser adquirido por una cantidad de entre 6.000 y 10.000 euros.

5.3. PUBLICACIONES

El proyecto Yafrid se ha presentado a varios congresos y jornadas en el ámbito de la computación gráfica. Esto ha servido tanto para ir dando a conocer Yafrid a potenciales usuarios e investigadores del área como para pulsar el interés de la comunidad sobre este tipo de proyectos. También han sido enriquecedores las mejoras y comentarios recibidos.

- Fernández-Sorribes J.A., González-Morcillo C. y Jiménez-Linares L. *Grid architecture for distributed rendering.* Ibero-American Symposium on Computer Graphics 2006.
- Fernández-Sorribes J.A. y González-Morcillo C. *Sistema GRID para el render de escenas 3D distribuido: YAFRID.* Blendiberia 2005.

Capítulo 6

CONCLUSIONES Y PROPUESTAS

6.1. CONSECUCIÓN DE LOS OBJETIVOS

Yafrid es el primer sistema de sus características y posee varias ventajas importantes que se pasan a describir a continuación.

Gracias a su **arquitectura multiplataforma** y a las tecnologías utilizadas para su desarrollo, el sistema funciona en distintos sistemas operativos y con independencia del hardware. Se han realizado pruebas con las distintas versiones del sistema operativo MS Windows y con diferentes distribuciones de GNU/Linux con resultados satisfactorios. Así, nos encontramos ante un sistema que no es cluster (solución clásica al problema del render). Yafrid es un grid en el que los proveedores de servicio pueden ser **heterogéneos** (en términos de hardware y software) y **geográficamente distribuidos**. El único requisito es poseer una conexión a Internet. Estas características lo alejan del cluster y lo acercan a las aplicaciones peer-to-peer.

Una de las ventajas de todas las aproximaciones distribuidas y de esta en concreto es su **escalabilidad**. El rendimiento del sistema que el usuario percibe depende en gran medida

del número de proveedores suscritos y de las características de los mismos de una forma global. Generalmente, cuantos más y mejores proveedores posea Yafrid, mejores resultados se obtendrán.

El sistema se ha desarrollado para utilizarse con los **motores de render** libres más usados (Yafray y el que posee Blender 3D). Sin embargo, y teniendo en cuenta que era un requisito indispensable, el diseño realizado permite añadir un nuevo motor fácilmente. Para ello, sólo es necesario implementar una clase en Python que lo represente y registrar el nuevo motor en el sistema.

Si un usuario quiere beneficiarse de los servicios ofrecidos por Yafrid, lo único que necesita es un **navegador web**. Por medio de una sencilla interfaz y desde donde quiera, un cliente puede subir trabajos al grid o consultar el estado de sus proyectos con sólo un par de clics de ratón. El hecho de no tener que instalar ningún tipo de software es muy interesante para hacer de Yafrid un producto apetecible para la comunidad potencial de usuarios.

La **gestión avanzada de grupos** que posee Yafrid permite a los usuarios controlar en todo momento qué proyectos quieren renderizar (en el caso de los proveedores) o quiénes renderizan sus proyectos (en el caso de los clientes).

El único software que un usuario ha de instalarse (y sólo en el caso de que quiera formar parte del grid como proveedor) es muy **sencillo de instalar y de usar**. En el caso de sistemas MS Windows, basta con ejecutar un instalador que irá guiando al usuario durante el proceso. En el caso de los sistemas GNU/Linux sólo hay que ejecutar un script de instalación para que el software esté listo para usar.

Este software, siempre con el consentimiento del usuario, se conectará al grid al iniciar la sesión con lo que el usuario proveedor no tendrá que estar pendiente de conectarlo.

El sistema permite tanto dividir animaciones en frames (granularidad gruesa) como dividir frames en fragmentos de imagen (granularidad fina). Para ambos modos de funcionamiento utiliza el mismo concepto clave, el de **unidad de trabajo**.

Con la aproximación de granularidad fina, se pueden realizar **optimizaciones locales** en cada frame. En trabajos futuros se puede realizar un estudio previo de la escena para realizar una **división inteligente** renderizando los distintos fragmentos con distintas calidades. Con esta aproximación y el uso de N máquinas en el grid, siendo X el tiempo que tarda una máquina en renderizar la escena completa, se podrían obtener tiempos inferiores incluso a X/N. El tiempo de render de un frame será menor que la suma de los tiempos de render de cada fragmento debido a que alguno de ellos tardarán menos porque su calidad será menor.

Yafrid es un proyecto de **software libre**, por lo que quien lo desee puede participar en

mejorarlo. Además, esto hace que el sistema pueda ser usado en trabajos de investigación o puesto en explotación para su uso. Su implantación real es menos costosa en términos económicos de lo que hubiera sido empleando software privativo.

El diseño realizado permitirá en un futuro que se añadan nuevas funcionalidades o se mejoren las existentes de una forma sencilla. En todo momento se ha tenido en cuenta la necesidad de una división intuitiva en capas, del uso de patrones de diseño, de hacer un código fuente claro y entendible (incluso se han incluido comentarios en inglés) y de realizar una adecuada fase de documentación.

6.2. PROPUESTAS Y LÍNEAS FUTURAS

Son numerosos los trabajos que se pueden realizar para complementar o extender la funcionalidad del sistema desarrollado en el ámbito de este proyecto. El diseño realizado y la documentación aportada hacen que, en general, cualquier modificación que se desee acometer no requiera un esfuerzo excesivo.

A continuación se describen algunas de estas propuestas.

6.2.1. Creación de proyectos desde Blender

Desarrollo de un pequeño plugin para los clientes de Yafrid que se pueda usar desde el propio menú de Blender. De este modo, un usuario con este plugin instalado podría crear un proyecto y enviarlo al grid para ser renderizado sin necesidad de acceder a la interfaz web del proyecto.

Blender posee una API de programación en Python, lenguaje que se ha utilizado para varias partes del sistema. A través de esta interfaz de programación se podría desarrollar un sencillo programa que pidiera al usuario la introducción de una clave y una contraseña. Con estos datos, el programa accedería al grid para llevar a cabo la autentificación. Si esta se lleva a cabo correctamente, se comprimirían los ficheros fuente y se enviarían al sistema vía SFTP desencadenando las acciones necesarias para que el proyecto sea procesado por el grid.

El desarrollo de esta utilidad se podría integrar en el desarrollo de Blender para que las próximas versiones del software lo incorporaran. LLevarlo a cabo no sería muy complicado ya que sólo sería necesario añadir un par de campos para el login y un nuevo botón en la sección de render para *renderizar con Yafrid*.

6.2.2. Análisis de la escena

Tal y como se desprende del análisis de los resultados empíricos realizado con anterioridad, aplicar ciertas técnicas ayudaría a conseguir un menor tiempo de render. Una propuesta destinada a mejorar el rendimiento del sistema consistiría en analizar la escena antes de planificarla. Como resultado de este paso previo se obtendría una estimación de cuánto tiempo tardaría cada parte de la escena en ser procesada.

De acuerdo con la información obtenida, el sistema podría usar distintos niveles de granularidad para dividir un frame en unidades de trabajo con el fin de incrementar la eficiencia. Aquellas zonas de la escena que la estimación haya determinado que son más costosas en tiempo, serán divididas en fragmentos más pequeños. De este modo, el tiempo de salida total disminuirá debido a que los tiempos que requieren cada una de las unidades de trabajo en ser completadas está más equilibrado.

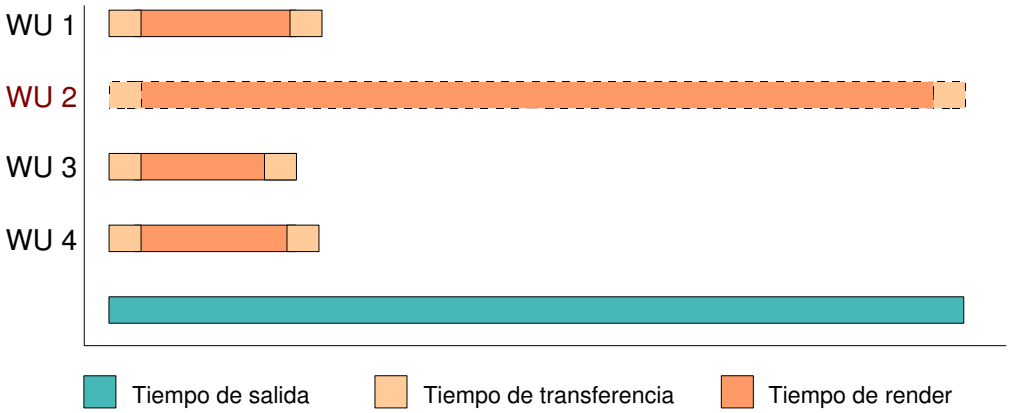

Figura 6.1: Escena dividida en cuatro unidades de trabajo.

En la Figura 6.1 se muestra el tiempo de salida del render de una escena que ha sido dividida en cuatro unidades de trabajo con el mismo área. Uno de estos fragmentos, la unidad de trabajo WU2, tarda mucho más que el resto lo que influye en el tiempo final.

En la figura 6.2 aparece la misma escena pero en esta ocasión se ha utilizado una granularidad variable para obtener las unidades de trabajo. El fragmento que en el ejemplo anterior tardaba mucho se ha dividido ahora en cuatro fragmentos con lo que se equilibran los tiempos de las distintas unidades mejorando el resultado final.

A partir de la información obtenida en la fase de análisis de la escena sería posible calcular cada una de las unidades de trabajo con una calidad distinta. Zonas que requieran poco detalle podrían ser renderizadas a una calidad menor que las contiguas. La imagen total

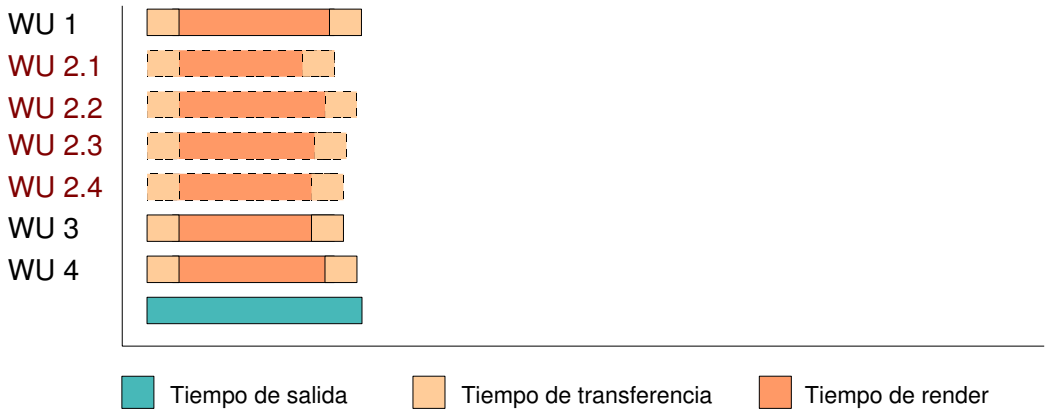

Figura 6.2: Escena dividida con granularidad variable.

no se verá afectada de forma perceptible gracias al proceso de interpolación lineal que se realiza al unir los fragmentos de imagen. De este modo, el tiempo de render global (el que se obtiene al sumar los tiempos de render de todas las máquinas) podría ser incluso menor que el tiempo en renderizar la misma escena con un sólo procesador.

6.2.3. Cálculo de la Irradiance Cache

Estudio de posibles optimizaciones sobre algunos motores de render en concreto con el fin de evitar cálculos duplicados.

Una de estas optimizaciones sería posible sobre Yafray. Este motor, para acelerar el render con iluminación global, hace uso de la llamada *irradiance cache*. Yafray realiza una primera pasada de render para determinar cuáles son las mejores áreas de una imagen para muestrear. En una superficie plana y grande, por ejemplo, se puede suponer que la iluminación cambiará con suavidad, por lo que hacen falta menos muestras. En una esquina, la iluminación cambiará más drásticamente, requiriéndose más muestras. La irradiance cache indica al motor de iluminación global cuáles son los mejores lugares para renderizar. Para más información sobre el proceso de render en Yafray, es conveniente consultar [37].

En Yafray, y en otros motores de render que utilicen esta técnica, sería más que recomendable calcular la irradiance cache solamente una vez en un paso previo, en lugar de que todos y cada uno de los proveedores calculen su propia irradiance cache. Este cálculo podría realizarlo un proveedor y enviar el resultado al resto de proveedores que vayan a renderizar la misma escena, que ya no tendrían que repetir este paso. Esto evitaría cálculos

redundantes, afectando positivamente al rendimiento global.

6.2.4. Utilización de varios servidores

Los sistemas con un solo Servidor adolecen de varios problemas. Uno de los más importantes es que la escalabilidad de un sistema como Yafrid queda comprometida teniendo un Servidor único.

Varias formas de utilizar varios servidores son:

- Reparto de roles.

  Un servidor se dedica a tareas de Identificador, otro a tareas de Distribuidor y otro mantiene la parte web.

- Replicación de servidores.

  Unos servidores almacenan los datos de los otros servidores. Así, si uno falla, la tarea del que ha dejado de funcionar puede ser realizada por los que quedan aún en pie. El proveedor posee una lista de servidores a los que conectarse. Si uno no está disponible pasa al siguiente y así hasta que encuentre a uno con el que poder conectarse.

- Redes de servidores.

  Los grupos de usuarios y proveedores se reparten entre los servidores de forma que cada servidor sólo atienda a las tareas relacionadas con un subconjunto de todos los usuarios del sistema.

6.2.5. Uso de protocolo peer-to-peer para el intercambio de ficheros

Tal y como se ha analizado en secciones anteriores, las necesidades del sistema con respecto al ancho de banda son muy importantes.

En el sistema actual, el envío de ficheros fuente desde el servidor a los Proveedores y de los resultados desde éstos hacia el servidor se realiza mediante SFTP (Figura 6.3). Esto implica una gran sobrecarga del servidor ya que tiene que enviar el mismo fichero a cada uno de los Proveedores.

Una solución que obtendría un mejor rendimiento sería el uso de protocolos peer-to-peer para el envío de los ficheros fuente del servidor a los Proveedores. Para el envío de los resultados no se puede utilizar porque no existe compartición de ficheros sino envío de uno a

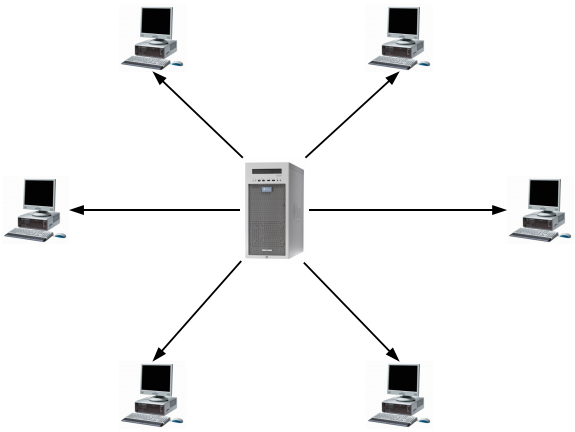

Figura 6.3: Envío de ficheros usado en Yafrid (SFTP).

uno. De entre las opciones disponibles, uno de los más adecuados es el protocolo **BitTorrent** (Figura 6.4).

BitTorrent [4] es un mecanismo para compartir ficheros de cualquier tamaño y de una forma muy eficiente. Una de sus principales ventajas es su escalabilidad. El aumento de usuarios de un fichero, lejos de perjudicar el rendimiento, lo incrementa, ya que cada nuevo usuario que interviene, además de demanda, añade suministro al sistema.

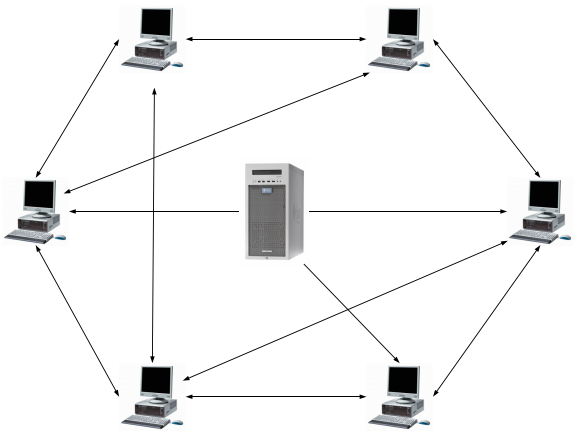

Figura 6.4: Compartición de ficheros con BitTorrent.

El primer paso para utilizar este protocolo consiste en crear un fichero **torrent** que describe el fichero a compartir, que denominaremos F. Este fichero torrent también define

la localización del **rastreador**[1], que es un servicio que controla el intercambio de ficheros.

De algún modo se hace accesible este torrent a usuarios que deseen compartir el fichero F y se comienza a subir (el único usuario que lo tiene en un primer momento recibe el nombre de *semilla inicial*).

Otros usuarios se hacen con el torrent y comienzan a bajar el fichero F. Estos usuarios, a la vez que se bajan partes de F, comparten las partes que ya han adquirido con otros usuarios que desean descargar ese mismo fichero. Así, la descarga se hace tanto más sencilla cuantas más fuentes haya disponibles.

Como el fichero está dividido en partes pequeñas, se necesita poco ancho de banda para llevar a cabo las transferencias. Una vez que un usuario completa la descarga del fichero, si desea continuar compartiéndolo con otros usuarios, se convierte en una nueva semilla.

El cliente oficial responde al mismo nombre que el protocolo y está escrito en python. Tiene la ventaja de ser muy ligero y de tener un interfaz por línea de comandos. A continuación se detalla cómo se podría integrar en Yafrid el uso de BitTorrent para el intercambio de los ficheros a renderizar usando este cliente.

- Cuando el servidor de Yafrid se inicia, además del servicio Identificador y el proceso Distribuidor, habría que iniciar el servicio rastreador del protocolo BitTorrent. Para ello, bastaría con indicar el puerto en el que *escucha* el servicio y el fichero donde se guardarán los logs. La sintaxis es la siguiente:

  $ bttrack –dfile yafrid-bittorrentlogfile.log –port 6969

- Cuando se lanza un proyecto Yafrid, se crea un nuevo fichero comprimido que contiene los ficheros fuente necesarios para realizar el render. El primer paso a seguir consiste en crear un fichero torrent de este fichero. Para ello se utiliza el comando *btmakemetafile* que utiliza a su vez la utilidad escrita en python *btmaketorrent.py* cuya sintaxis es la siguiente:

  $ btmakemetafile myfile tracker_announce_address [options ..]

  Donde *myfile* es el fichero a compartir y *tracker_announce_address* es la dirección en la que se instalará el servicio rastreador.

  La posible llamada en el caso del sistema Yafrid podría ser:

[1] En inglés, tracker. En las últimas versiones del protocolo no es imprescindible.

$ btmakemetafile /POOL/projectfile.zip http://www.yafrid.org:6969/announce

Esto generaría un fichero projectfile.zip.torrent que, entre otra información, tendría el valor hash del fichero que será usado para comprobar la integridad del mismo. El tamaño de este fichero depende del tamaño del fichero a compartir. Un fichero comprimido de alrededor de 11 MB generaría un fichero torrent de alrededor de 1KB.

Al principio del intercambio, el único nodo que tiene el fichero completo es el servidor de Yafrid con lo que constituye la semilla inicial. Para iniciar el intercambio el Servidor tendría que ejecutar el cliente de BitTorrent con la sintaxis que se verá en el siguiente punto.

- Cuando el Distribuidor envía a un Proveedor la petición de procesar una determinada unidad de trabajo, también se podría enviar el fichero torrent comprimido. Otra opción sería que la petición desencadenara que el Proveedor adquiriera vía SFTP el fichero torrent del servidor. Habría otra opción aún más sencilla consistente en incluir, en un campo de la petición, la url del fichero torrent para que el Proveedor hiciera referencia a ella directamente.

  Llegados a este punto, todo está listo para iniciar el intercambio. La sintaxis para iniciar el cliente de modo que se puedan compartir varios ficheros es la siguiente:

  $ btlaunchmany –url http://www.yafrid.org/projectfile.zip.torrent –saveas projectfile.zip

  Al principio, los Proveedores irán descargando las partes del fichero desde el servidor que será el único que tiene el fichero. Conforme vayan adquiriendo partes del mismo, los Proveedores compartirán unos con otros las partes que tengan, liberando de carga al servidor.

6.2.6. Seguridad

Las cuestiones acerca de la seguridad se han tenido en cuenta a lo largo del desarrollo del proyecto Yafrid haciendo de éste un sistema robusto. Sin embargo, es bastante probable que el análisis del funcionamiento del sistema en un ambiente real (y por lo tanto potencialmente hostil) haga que salgan a relucir cuestiones no contempladas. Mejoras en la seguridad, en la recuperación de errores, etc, . . . serían convenientes.

6.2.7. Utilización del proveedor

Aunque en el desarrollo del sistema no se ha abordado, sería interesante que el usuario pudiera establecer un tanto por ciento máximo de utilización de su CPU para tareas provenientes del grid.

Otra característica deseable sería que el sistema detectara cuándo no se está usando el ordenador y aprovechara estos tiempos libres para tareas del grid.

Bibliografía

[Bec00] K Beck. *Una explicación de la programación extrema. Aceptar el cambio*. Addison-Wesley, 2000.

[Boe88] B. W. Boehm. A spiral model of software development and enhancement. *IEEE Computer*, pages 61–72, May 1988.

[Bou70] W. J. Bouknight. A procedure for generation of three-dimensional half-toned computer graphics. *Communications of the ACM*, 1970.

[Bro02] M. C. Brown. Buil a Grid App with Python. 2002. `http://www.ibm.com/DeveloperWorks/`.

[Buy02] R. Buyya. Economic-based Distributed Resource Management and Scheduling for Grid Computing. April 2002.

[BV05] Rajkumar Buyya and Srikumar Venugopal. A Gentle Introduction to Grid Computing and Tecnologies. *CSI Communications*, 29(1):9–19, July 2005.

[CGR05] Coral Calero, Marcela Genero, and Francisco Ruiz. *Material docente de la asignatura de Bases de Datos*. 2005.

[DPH+03] D. DeMarle, S. Parker, M. Hartner, C. Gribble, and C. Hansen. Distributed interactive ray tracing for large volume visualization. *Proc. IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, pages 87–94, 2003.

[FK98] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufman, Elsevier, 1998.

[FKNT02] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. *The physiology of the grid: An open grid services architecture for distributed systems integration*. 2002.

[FKT02] Ian Foster, Carl Kesselman, and Steve Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputing Applications*, 15(3), 2002.

[FM03] Eduardo Fernández-Medina. *Material docente de la asignatura de Ingeniería del Software I*. 2003.

[Fos02] Ian Foster. What is the Grid? A Three Point Checklist. 2002. Web pages //www-fp.mcs.anl.gov/ foster/Articles/WhatIsTheGrid.pdf.

[FQKYS04] Zhe Fan, Feng Qiu, Arie Kaufman, and Suzanne Yoakum-Stover. Gpu cluster for high performance computing. *ACM / IEEE Supercomputing Conference*, November 2004.

[GAW04] I. J. Grimstead, N. J. Avis, and D. W. Walker. Automatic distribution of rendering workloads in a grid enabled collaborative visualization environment. *ACM/IEEE SC 2004 Conference (SC'04)*, pages 32–38, 2004.

[GHJV03] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Patrones de diseño. Elementos de software orientado a objetos reutilizable*. Pearson Educación, 2003.

[Gon05] Carlos González. *Material docente de la asignatura de Animación para la Comunicación*. 2005.

[GTGB84] C. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile. Modelling the interaction of light between diffuse surfaces. *Proceedings of SIGGRAPH*, 1984.

[Jen96] H. W. Jensen. *Global Illumination Using Photon Maps*. 1996.

[Jen01] H. W. Jensen. *Realistic Image Synthesis using Photon Mapping*. A.K.Peters, 2001.

[JRB99] I. Jacobson, J. Rumbaugh, and G. Booch. *The Unified Software Development Process*. Addison-Wesley, 1999.

[Kaj86] J. T. Kajiya. The rendering equation. *Computer Graphics*, 1986.

[Lar99] Craig Larman. *UML y patrones: introducción al análisis y diseño orientado a objetos*. Prentice Hall, 1999.

[LMV+04] W. Leister, S. Mazaher, J. I. Vestgården, B.Ø. Johansen, and B. Nordlund. Grid and related technologies. *Norwegian Computer Center Report*, June 2004.

[LSSH03] J. Ledlie, J. Shneidman, M. Seltzer, and J. Huth. Scooped, Again. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, USA, February 2003.

[MAB05] K. P. C. Madhavan, L. L. Arns, and G. R. Bertoline. A distributed rendering environment for teaching animation and scientific visualization. *IEEE Computer Graphics and Applications*, pages 32–38, 2005.

[Moy05] F. Moya. *Material docente de la asignatura de Arquitectura de Sistemas Distribuidos*. 2005.

[p2p02] Second IEEE international conference on peer-to-peer computing: Use of computers at the edge of networks (p2p, grid, clusters), September 2002.

[Pol05] Macario Polo. *Material docente de la asignatura de Ingeniería del Software II*. 2005.

[RS02] A. Reinefeld and F. Schintke. Concepts and Technologies for a Worldwide Grid Infrastructure. *Lecture Notes in Computer Science*, 2400:62–71, 2002.

[VG97] E. Veach and L. J. Guibas. Metropolis light transport. In *Proceedings of SIGGRAPH*, 1997.

[WH92] G. J. Ward and P. Heckbert. *Irradiance Gradients*. Third Eurographics Workshop on Rendering, 1992.

[Whi80] T. Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 33(6):343–349, June 1980.

[ZIK98] S. Zhukov, A. Iones, and G. Kronin. An ambient light illumination model. *Eurographics Rendering Workshop*, 1998.

Referencias web

[1] AdSense de Google. Anuncios de Google relevantes al contenido para desarrolladores web. https://www.google.com/adsense/.

[2] Apache. HTTP Server Project. http://httpd.apache.org/.

[3] Apache Tomcat. http://tomcat.apache.org/.

[4] BitTorrent. http://www.bittorrent.com/index.html.

[5] Blender. Open Source 3D graphics creation. //www.blender.org.

[6] Brazil Rendering System. http://www.splutterfish.com/.

[7] CAPTCHA Project. Completely Automated Public Turing test to tell Computers and Humans Apart. http://www.captcha.net/.

[8] Chaos Group. V-Ray, Aura and more. http://www.chaosgroup.com/.

[9] Comparativa entre distintos lenguajes de programación. http://www.ipd.uka.de/~prechelt/Biblio/jccpprtTR.pdf.

[10] Comparativa entre Ice y CORBA. http://www.zeroc.com/iceVsCorba.html.

[11] Escuela Superior de Informática de Ciudad Real. http://www.inf-cr.uclm.es/.

[12] Estadísticas de uso de PHP. http://www.php.net/usage.php.

[13] Final Render. http://www.finalrender.com/.

[14] Global Grid Forum. http://www.gridforum.org/.

[15] Grupo de Investigación Oreto. Sistemas Inteligentes Aplicados. http://oreto.inf-cr.uclm.es/.

[16] Las 10 tecnologías emergentes que cambiarán el mundo según el MIT. http://www.globalfuture.com/mit-trends2003.htm.

[17] Maxwell Render Engine. http://www.maxwellrender.com/.

[18] Mental Ray Renderer. http://www.mentalimages.com/index.html.

[19] NVIDIA Gelato. http://www.nvidia.com/page/gelato.html.

[20] OMG's CORBA Site. http://www.corba.org.

[21] Open Science Grid. http://www.opensciencegrid.org.

[22] PayPal. Sistema global de pagos en Internet (Página oficial en castellano). http://www.paypal.es/es.

[23] PHP: Hypertext Preprocessor. http://www.php.net.

[24] PIL: Python Imaging Library. http://www.pythonware.com/products/pil/.

[25] Pixar Animation Studios. http://www.pixar.com.

[26] Python Programming Language. http://www.python.org.

[27] Página oficial del proyecto Cygwin. http://www.cygwin.com/.

[28] RenderPlanet. The render farm service. http://renderplanet.com.

[29] Servicio de Supercomputación de Castilla-La Mancha. http://ssc.uclm.es/.

[30] SETI@Home. http://setiathome.berkeley.edu/.

[31] SETI@Home es el proyecto de computación distribuida con más éxito. http://www.grid.org/about/gc/seti.htm.

[32] Sun Microsystems. http://es.sun.com/.

[33] The Enabling Grids for E-sciencE project. http://public.eu-egee.org/.

[34] The Globus Alliance. http://www.globus.org/.

[35] Toxic. An Open Source Global Illumination Renderer. http://www.toxicengine.org/features.php.

[36] World Wide Web Consortium. http://www.w3c.es/.

[37] Yafray. Yet Another Free Raytracer. http://www.yafray.org.

[38] ZeroC. Internet Communications Engine. http://www.zeroc.com/.

ANEXO A

Diagramas

A.1. Diagramas de Casos de Uso UML

Los diagramas de casos de uso [Pol05] constituyen la herramienta que proporciona el estándar UML para representar la funcionalidad de un sistema. En ellos se representan principalmente las relaciones entre los actores y los casos de uso. Cada uno de los casos de uso representa un requisito funcional desde el punto de vista de los actores que intervienen.

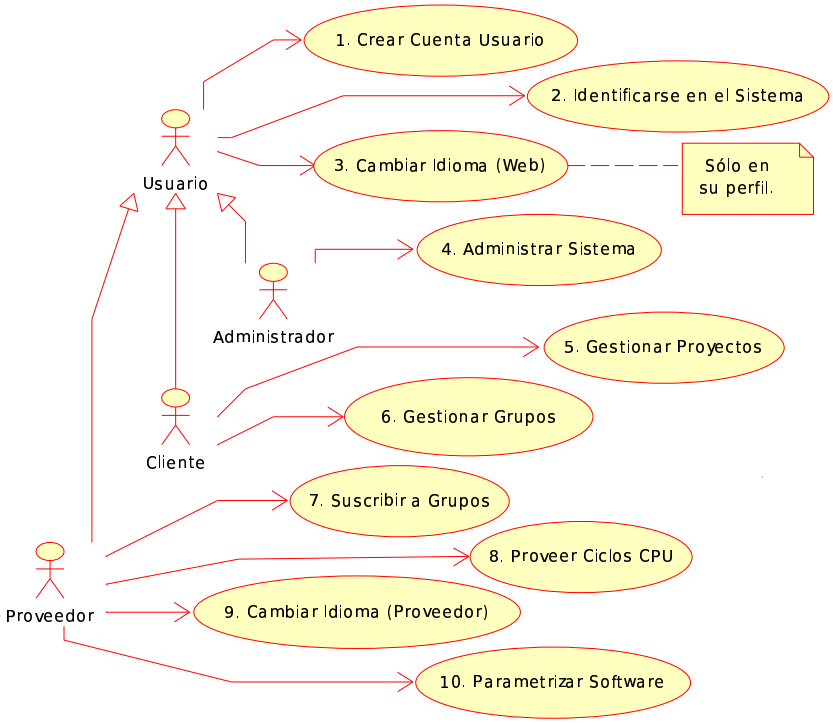

Figura A.1: Diagrama de casos de uso general de Yafrid.

Creación de Cuentas de Usuario

La creación de una cuenta de usuario (Figura A.2) es una funcionalidad general a la que puede acceder cualquier potencial usuario.

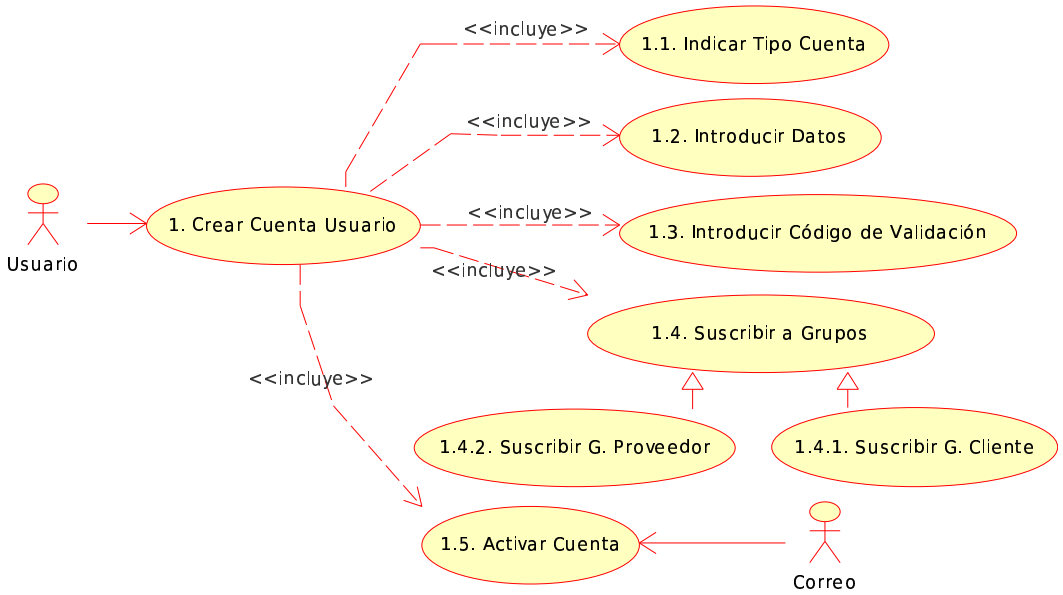

Figura A.2: Caso de Uso 1: *Crear Cuenta de Usuario*.

Proveedores de servicio

Un usuario proveedor puede acceder a algunas funcionalidades (Figura A.3) de las que carecen los otros tipos de Usuario. A través del software que se proporciona puede ceder sus ciclos de CPU de forma transparente.

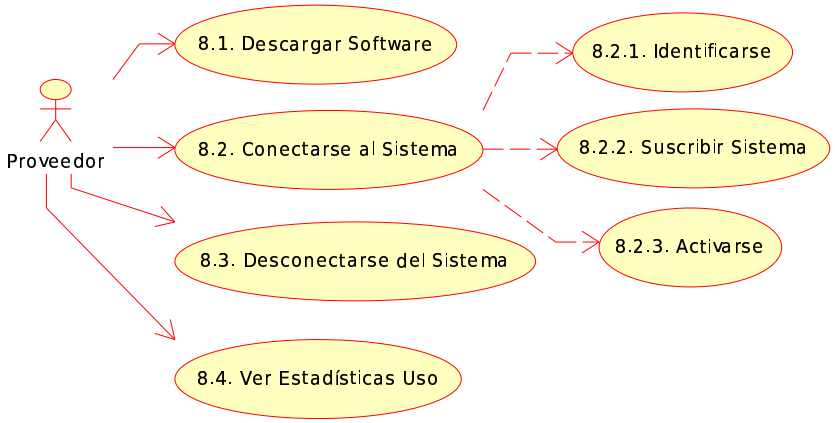

Figura A.3: Caso de Uso 8: *Proveer Ciclos de CPU*.

Gestión de Proyectos

La creación y gestión de proyectos (Figura A.4) constituye la funcionalidad principal de los clientes. En todo momento se puede cambiar el estado de un proyecto, monitorizar su progreso, conocer las estadísticas asociadas y previsualizar los resultados parciales.

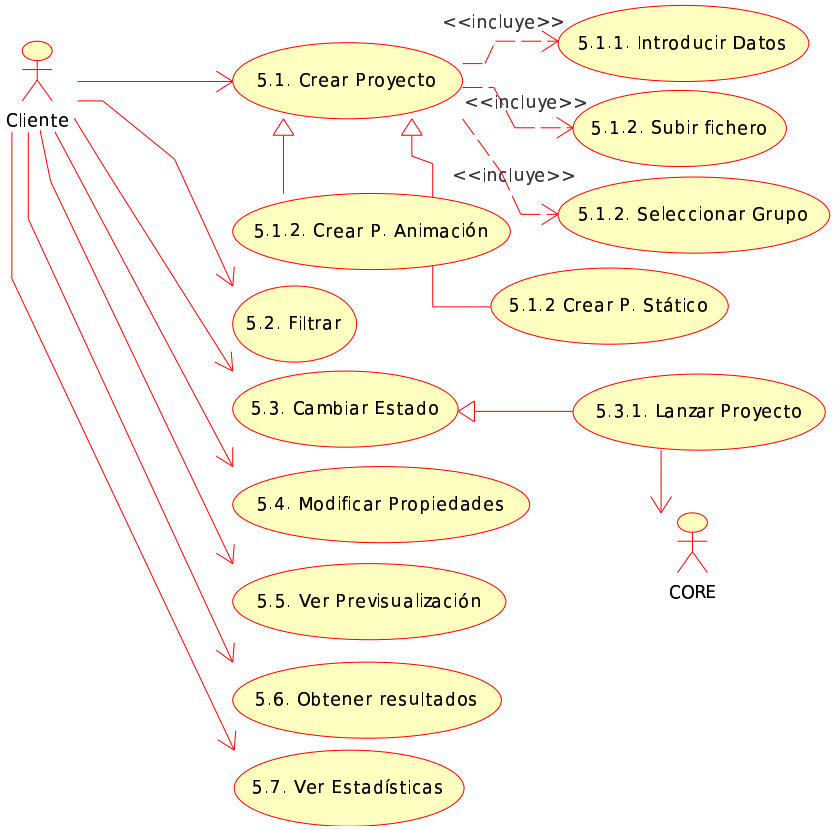

Figura A.4: Caso de Uso 5: *Gestionar Proyectos*.

Gestión de Grupos

La creación y gestión de grupos (Figura A.5) es una funcionalidad secundaria que permite a un cliente tener un mayor control sobre el procesamiento de sus proyectos en el sistema.

Administración del Sistema

El administrador tiene un control completo (Figura A.6) sobre la configuración del sistema, tanto de la parte web como de la parte no interactiva. Posee además acceso a la información sobre proyectos, grupos y usuarios que existen en el sistema.

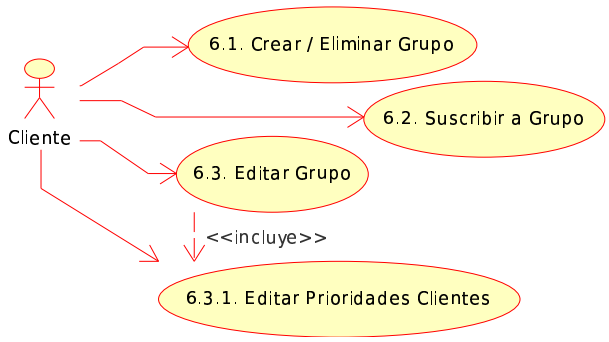

Figura A.5: Caso de Uso 6: *Gestionar Grupos*.

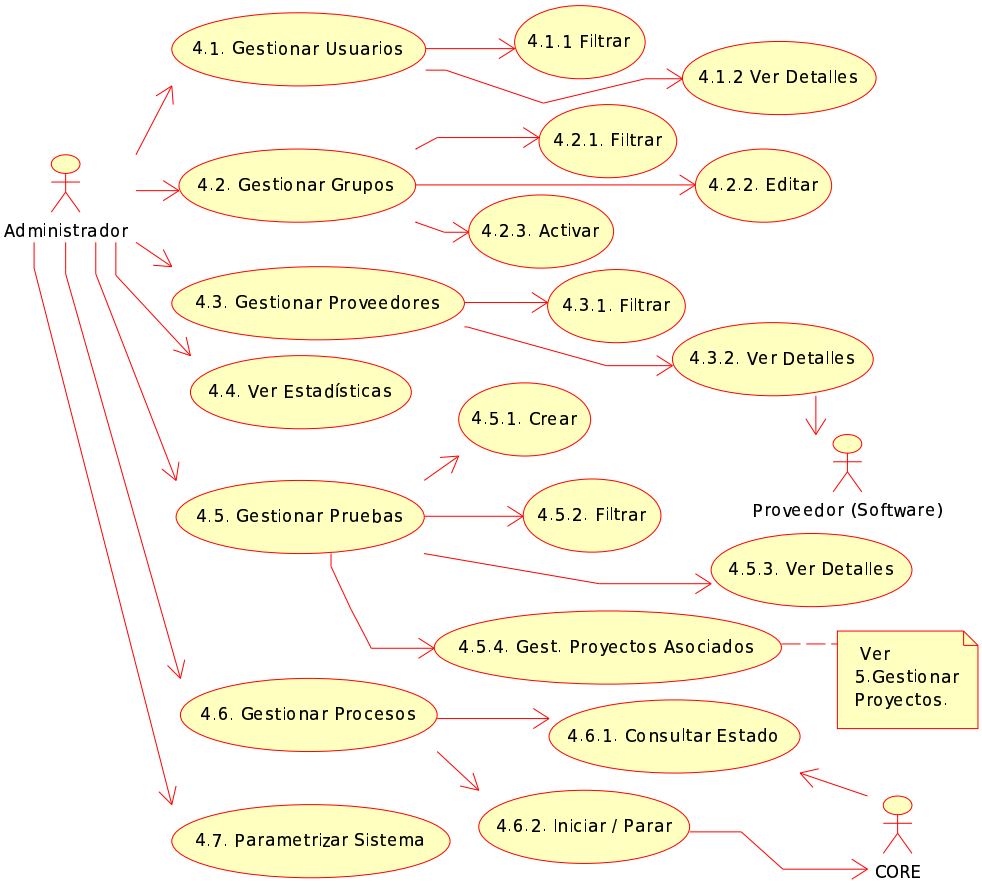

Figura A.6: Caso de Uso 4: *Administrar Sistema*.

A.2. Diagramas de Clases UML

A lo largo del documento se han utilizado numerosos diagramas de clases. En esta sección se incluyen algunos diagramas adicionales[1] que no han sido imprescindibles a la hora de analizar los distintos componentes y que, por ello, no han sido utilizados hasta ahora.

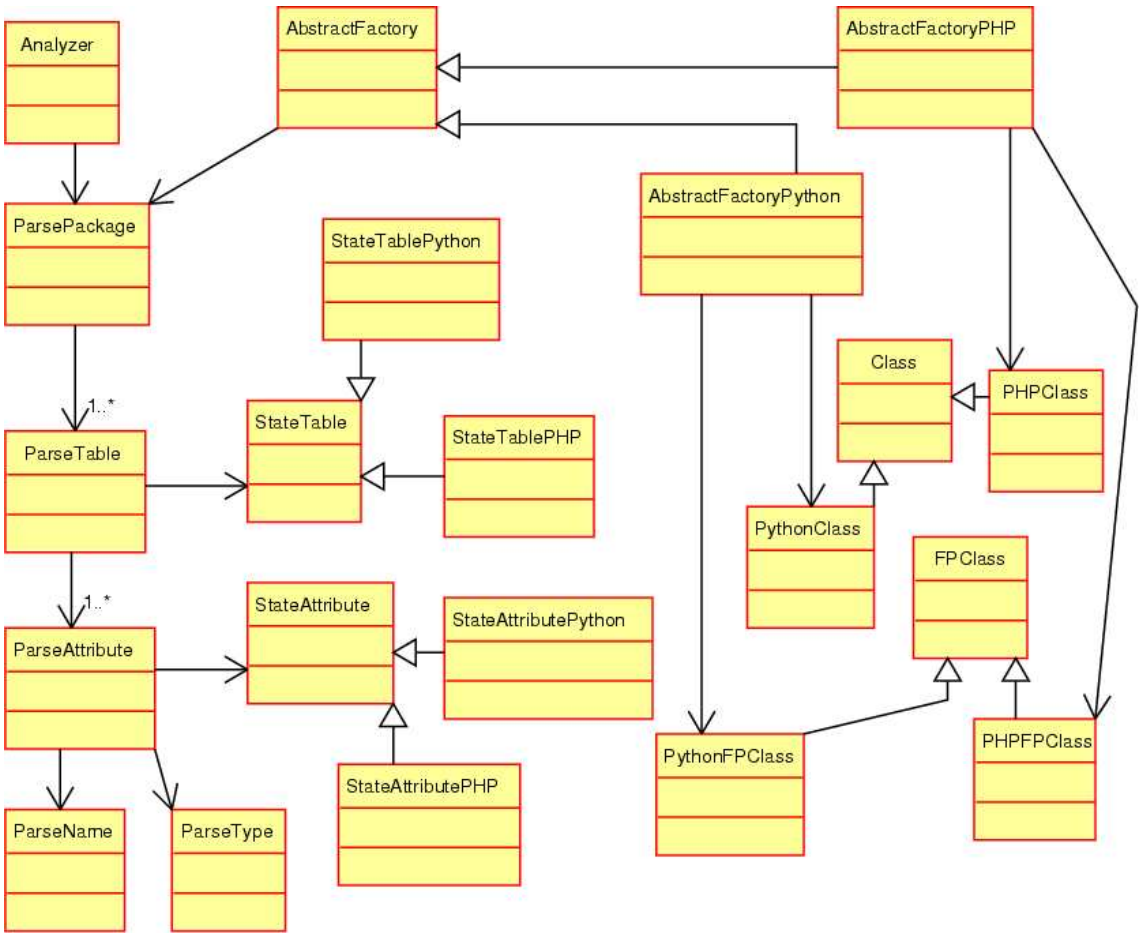

Figura A.7: Diagrama de clases de gendb. La clase Analyzer genera en un primer paso una estructura de árbol formada por instancias de las clases cuyos nombres comienzan por *Parse* (patrón *Intérprete*). Una vez creada la estructura, los objetos tabla y los objetos atributo serán decorados con instancias de las clases *Estado* correspondientes al lenguaje de salida. El proceso de generación de clases y *Fabricaciones Puras* está gestionado por *Fábricas Abstractas*.

[1] No se incorporan los diagramas UML de todas las clases del sistema ya que su número es muy grande. Sólo se muestran ciertas clases que por alguna razón puedan resultar interesantes.

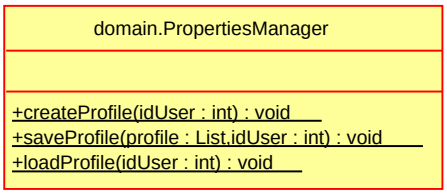

Figura A.8: Clase PropertiesManager que permite el acceso a los perfiles de los usuarios de Yafrid-WEB.

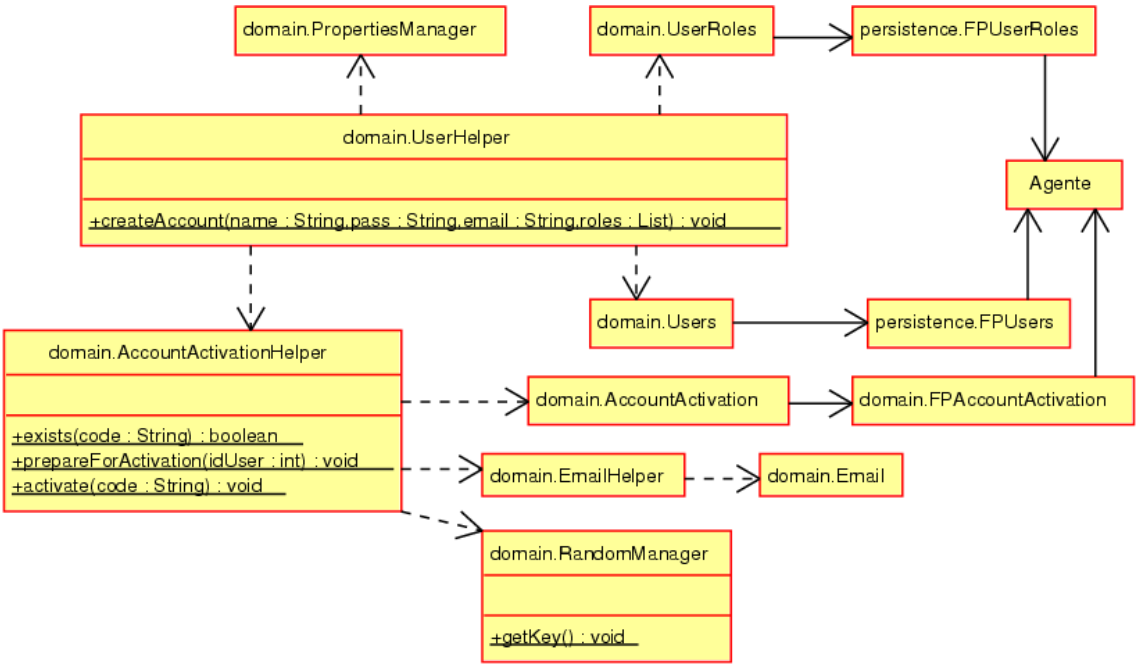

Figura A.9: Clase UserHelper que implementa las operaciones relacionadas con los usuarios que suponen una responsabilidad excesiva como para delegarla en la clase Users.

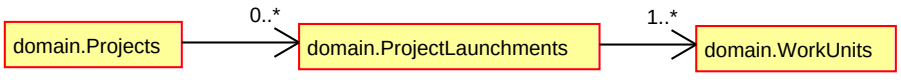

Figura A.10: Relación entre las clases de análisis Proyecto, Lanzamiento y Unidad de Trabajo. Mientras que un proyecto puede no tener lanzamientos asociados, todo lanzamiento tiene al menos una unidad de trabajo asociada.

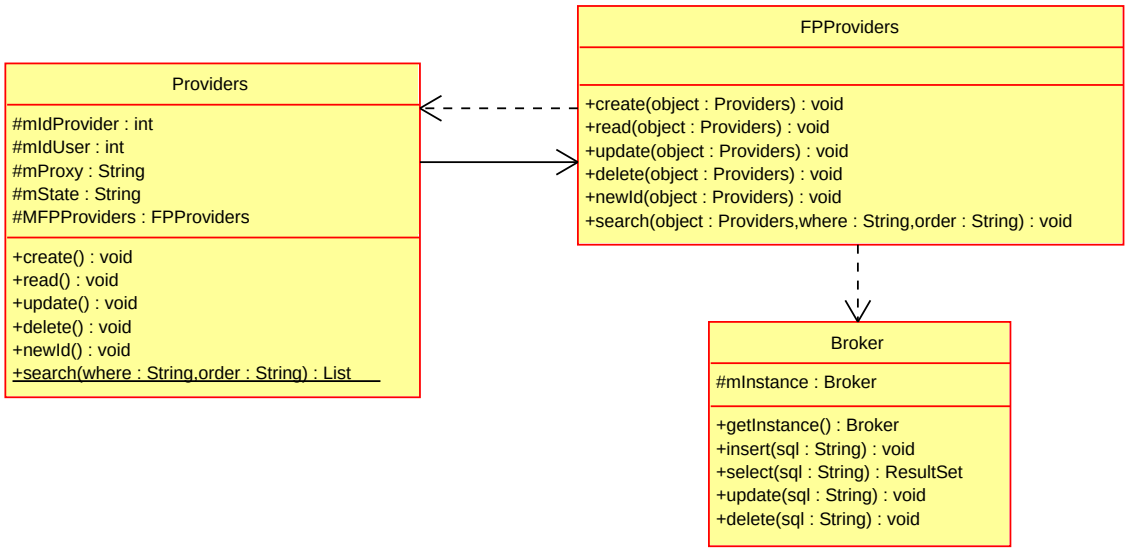

Figura A.11: Clase Providers junto con su Fabricación Pura que implementa su persistencia a través del Broker. La gran mayoría de clases 1C1T presentan este esquema.

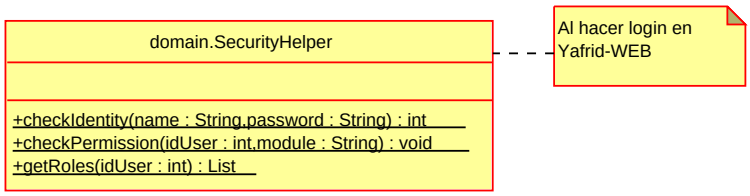

Figura A.12: Clase SecurityHelper que implementa la seguridad en Yafrid-WEB.

A.3. Diagramas relacionales

En esta sección se incluyen algunos diagramas que muestran las distintas tablas de la base de datos y las relaciones entre las mismas[2].

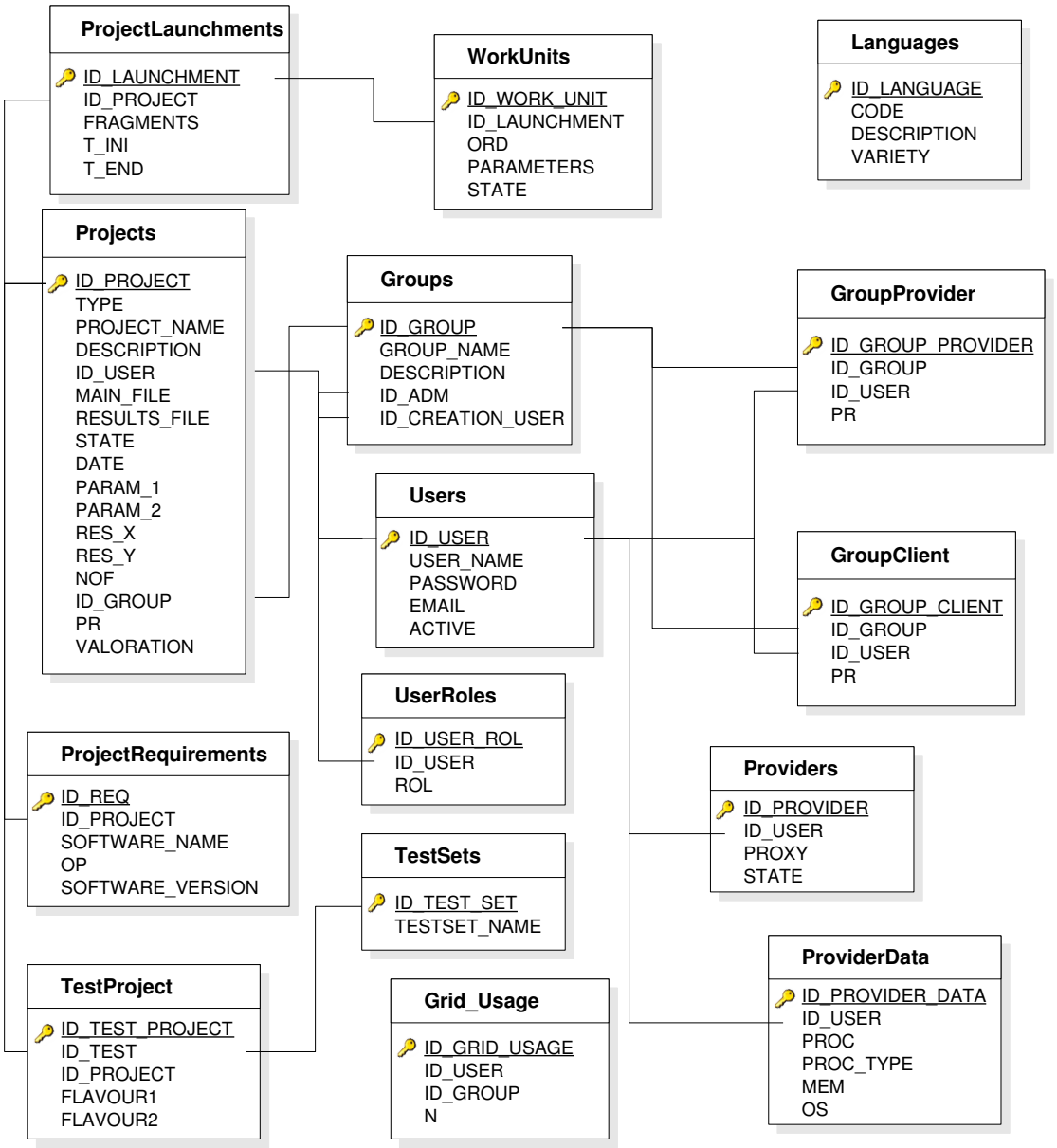

Figura A.13: Diagrama relacional de Yafrid.

[2]Igual que en el caso de las clases, tampoco se pretende en esta sección ser exhaustivo mostrando las tablas por lo que no aparecen todas sino las más importantes.

A.4. Diagrama de componentes y despliegue UML

Un diagrama de despliegue UML muestra los componentes software y hardware de un sistema y las relaciones que se establecen entre ellos en tiempo de ejecución. En el ejemplo se muestra de forma simplificada la implantación del sistema en su fase actual en un servidor del Grupo de Investigación Oreto [15] de la Escuela Superior de Informática de Ciudad Real [11].

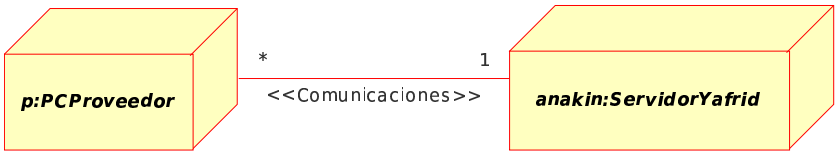

Figura A.14: Diagrama general de despliegue de Yafrid.

En cada uno de los nodos—el servidor y los distintos equipos proveedores— se despliegan una serie de componentes que han sido ampliamente estudiados a lo largo del presente documento.

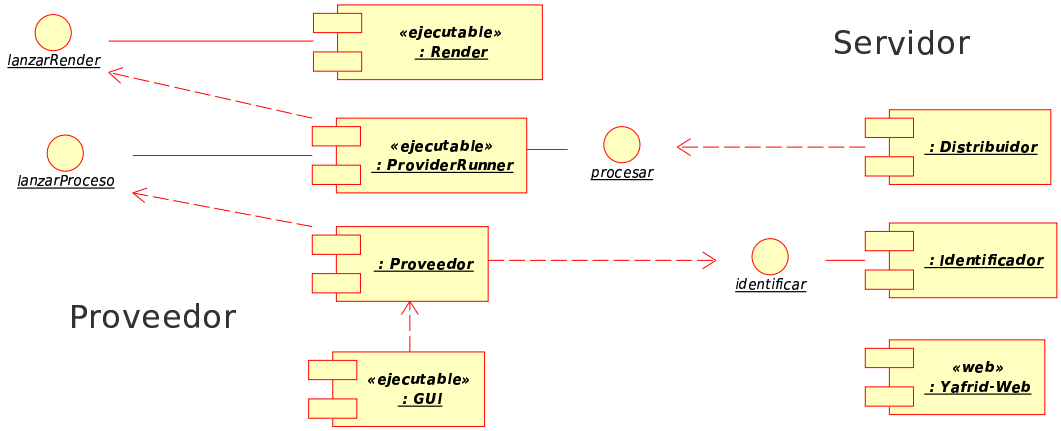

Figura A.15: Diagrama de despliegue con detalle de componentes.

ANEXO B

Manual de Usuario e Instalación Yafrid Provider 0.0.2

B.1. Creación de una cuenta de Proveedor

El primer paso para convertirse en un proveedor del sistema Yafrid consiste en la creación de una cuenta de usuario. Para ello, es necesario acceder a la web de Yafrid desde cualquier navegador. Actualmente, el sitio web de Yafrid se encuentra en la dirección siguiente:

http://anakin.inf-cr.uclm.es:4080/apps/yafrid/yafrid-service/

Desde esta página (Figura B.1) se podrá adquirir una nueva cuenta de usuario. Una vez que se posea la cuenta, será posible acceder al sistema introduciendo el nombre de usuario y la contraseña con los que se creó la cuenta. Cada una de estas acciones se puede llevar a cabo mediante los paneles de la parte derecha de la ventana.

Tipo de Cuenta

Suponiendo que es la primera vez que se accede al sistema, es necesario crear una nueva cuenta. Para ello, se hará clic en el enlace ***Create new account*** *(Crear cuenta nueva)*. Una vez hecho esto, se accede a la página que se muestra en la Figura B.2.

Para continuar el proceso de creación de la cuenta, se debe marcar la opción de ***Provider*** *(Proveedor)* y hacer clic sobre el botón ***Next*** *(Siguiente)*.

Este tipo de cuenta le proporcionará la posibilidad de ceder sus ciclos de CPU para llevar a cabo trabajos del grid. Desde el interfaz web podrá consultar sus datos de conexión y su estado y obtener estadísticas de su desempeño dentro del sistema.

Figura B.1: Pantalla principal del sitio web de Yafrid.

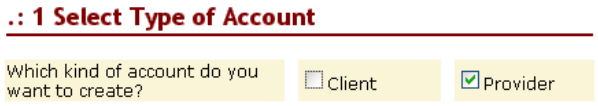

Figura B.2: Paso 1. Selección del tipo de cuenta.

Datos de Usuario

Esto da paso a una ventana en la que se deberán completar los datos de la cuenta del proveedor (Figura B.3).

Además, habrá que introducir el texto que aparece en una imagen generada aleatoriamente por motivos de seguridad. Si no ve esta imagen, debe asegurarse de que la configuración de su navegador permite visualizar imágenes e intentarlo de nuevo. Si no entiende bien la palabra, trate de reproducirla de todas formas. Aunque lo introduzca mal, tendrá una nueva oportunidad de introducir otra palabra.

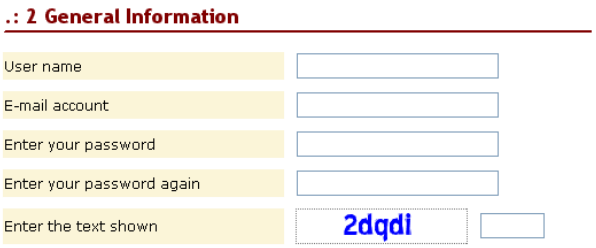

Figura B.3: Paso 2. Introducción de datos de usuario.

Suscripción a Grupos de Usuarios

Además de los datos de creación de la cuenta, se ofrece la suscripción a los distintos grupos del sistema (Figura B.3). En la segunda columna aparece el orden del 0 al 10 en el que se dará servicio a los distintos grupos. Así, usted procesará los trabajos de un grupo con prioridad 10 antes que los de un grupo con prioridad 5. Esta configuración podrá modificarse una vez creada la cuenta.

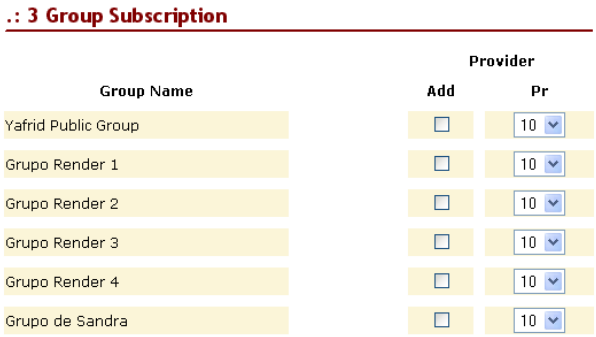

Figura B.4: Paso 3. Suscripción a grupos.

Una vez completados los datos se procederá a crear la cuenta haciendo clic en el botón ***Create Account*** *(Crear Cuenta)*. Esto tendrá como resultado que se recibirá un correo electrónico en la cuenta de correo que se usó para dar de alta la cuenta.

Activación de la Cuenta

Para que la cuenta sea completamente funcional habrá que activarla haciendo clic en el enlace que incorpora el correo recibido.

Una vez hecho esto, la cuenta estará activa y lista para ser usada. Para entrar en el sistema habrá que introducir el nombre de usuario y la contraseña en la página principal.

B.2. Descarga del software necesario

Con la cuenta creada en el apartado anterior ya es posible acceder a los contenidos destinados a los proveedores del sistema Yafrid.

Una vez dentro del sistema, en la sección de software (Figura B.5) se encontrarán las versiones de **Yafrid Provider 0.0.2** para las distintas plataformas, así como algunos enlaces a otros programas que pueden ser necesarios como **Blender** o **Yafray**.

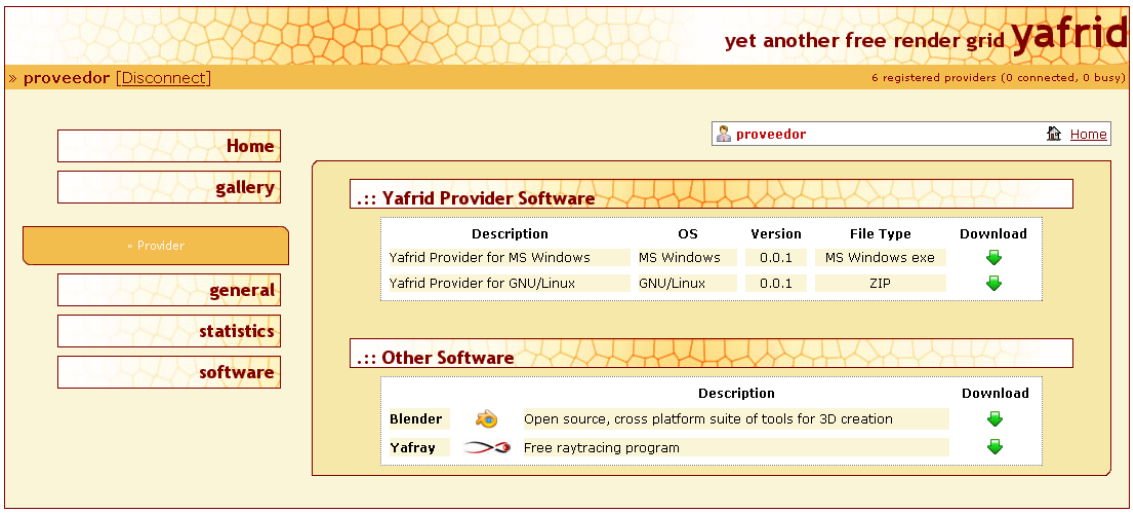

Figura B.5: Pantalla de descarga de software.

B.3. Instalación de Yafrid Provider 0.0.2

En las siguientes secciones se detallará la instalación de Yafrid Provider 0.0.2 tanto en sistemas MS Windows como sistemas GNU/Linux.

B.3.1. Instalación en MS Windows

Haciendo clic en el enlace que corresponde a la versión para MS Windows de Yafrid Provider 0.0.2 se iniciará automáticamente la descarga del archivo **setup.exe**. Una vez

descargado, bastará con ejecutarlo para que aparezca el asistente para la instalación del software.

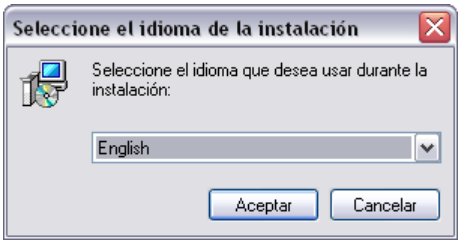

Figura B.6: Elección del idioma del proceso de instalación.

Lo primero que habrá que decidir es el idioma en el que se desarrollará el proceso de instalación (Figura B.6). Tras este primer paso se irán sucediendo una serie de pantallas en las que se podrán decidir algunos detalles de la instalación como el propio directorio de instalación o los iconos a crear. Todas estas opciones son las habituales en este tipo de asistentes a las que cualquier usuario medio de MS Windows estará habituado. Las ventanas que se irán sucediendo son las que aparecen en la Figura B.7.

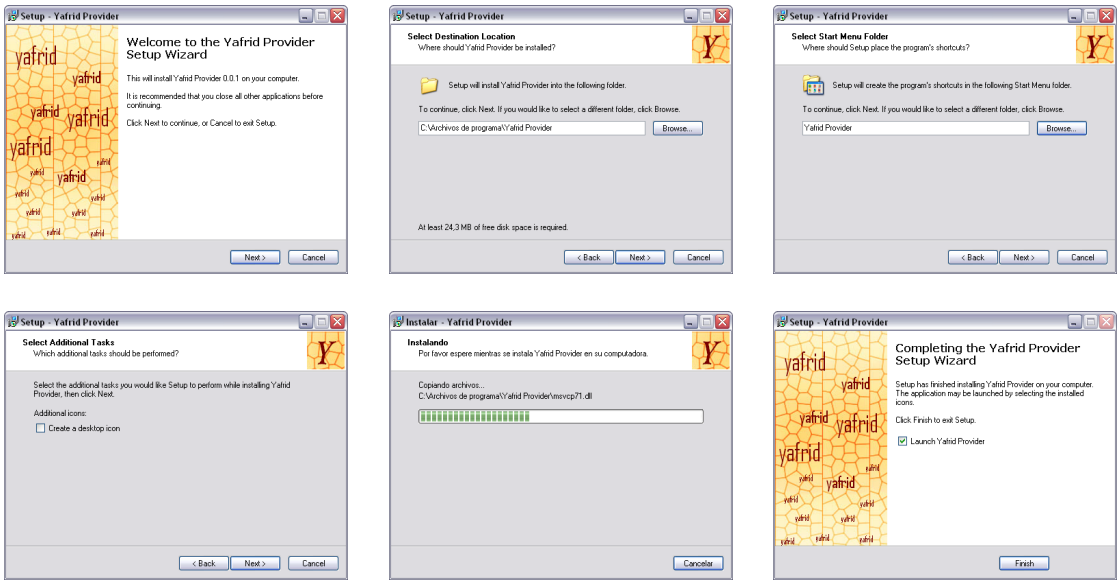

Figura B.7: Instalación de Yafrid Provider 0.0.2 en MS Windows.

Como resultado, y si no ha ocurrido ningún problema, Yafrid Provider 0.0.2 se habrá instalado correctamente en el sistema.

Iniciar al arrancar el sistema

Por defecto, el asistente crea un acceso directo en el Menú Inicio. Esto hace que, cada vez que se arranque el sistema, se inicie Yafrid Provider. Para evitarlo sólo hay que eliminar dicho acceso directo.

B.3.2. Instalación en GNU/Linux

B.3.2.1. Antes de instalar

Paquetes necesarios. Para poder hacer uso de Yafrid Provider 0.0.2 en un sistema GNU/Linux, hay algunos paquetes que es necesario tener instalados. Estos paquetes se detallan a continuación.

- **python2.4.** Lenguaje de programación en el que está desarrollado Yafrid Provider 0.0.2.
- **python2.4-tk.** La interfaz gráfica.de Yafrid Provider 0.0.2 ha sido realizada con Tkinter.
- **python2.4-imaging.** Contiene la librería gráfica PIL, imprescindible para el tratamiento de las imágenes.

El segundo de ellos, *python2.4-tk*, sólo será necesario en el caso de usar el interfaz gráfico de Yafrid Provider. Si se usa solamente por línea de comandos no será necesaria su instalación.

En caso de que algunos de estos paquetes no estén presentes en su sistema, la forma de instalarlos es sencilla en sistemas basados en **Debian**. Para ello se utilizan los llamadas gestores de paquetes como apt-get, Synaptic o Adept. Con el primero de ellos bastaría con hacer lo siguiente:

apt-get update
apt-get install python2.4 python2.4-tk python2.4-imaging

Sobre el ejecutable de python. Una forma sencilla de comprobar que el sistema cuenta con la versión adecuada de python es introducir en una consola lo siguiente:

python -V

Esto nos devolverá la versión de python que se está usando (la adecuada será la **2.4**).

Es común tener instaladas varias versiones de python en el sistema por lo que se debe indicar a Yafrid Provider que use la 2.4. Para ello, y si en la prueba anterior hemos obtenido una versión distinta, habrá que hacer lo siguiente o algo similar (dependiendo del lugar donde se encuentre instalado python).

ln -f -s /usr/bin/python2.4 /usr/bin/python

B.3.2.2. Instalación

Haciendo clic en el enlace que corresponde a la versión para GNU/Linux de Yafrid Provider se iniciará automáticamente la descarga del archivo **provider.zip**. Una vez descargado dicho fichero habrá que abrir una consola y seguir los siguientes pasos:

1. Descomprimir el fichero zip en un directorio.

   mkdir provider-0.0.2
   cd provider-0.0.2
   unzip provider.zip

2. Ejecutar el script que iniciará el proceso de instalación. Para ello, habrá que hacer lo siguiente desde el directorio *provider-0.0.2*:

   ./yafrid_provider.sh

   Si se han seguido los pasos descritos hasta ahora, el script no debería fallar; si en algún momento lo hace, habría que volver a los puntos anteriores.

B.3.2.3. Iniciar al arrancar el sistema

Para configurar Yafrid Provider 0.0.2 para que se inicie al arrancar el sistema habrá que hacer lo siguiente:

cd provider-0.0.2
. **./yafrid_at_init.sh add**

Del mismo modo, para eliminar esta característica y que Yafrid Provider 0.0.2 no se inicie al arrancar, basta introducir los siguientes comandos:

cd provider-0.0.2
. **./yafrid_at_init.sh remove**

B.4. Puesta en marcha de Yafrid Provider 0.0.2

B.4.1. General

Una vez que Yafrid Provider está correctamente instalado, el funcionamiento es sencillo. Básicamente se nos presenta la ventana que aparece en la Figura B.8 (puede haber diferencias de aspecto en los diferentes sistemas y algunas particularidades más importantes que se detallarán a continuación).

Figura B.8: Interfaz de Yafrid Provider 0.0.2.

En principio no aparecerá ningún *usuario* ni *contraseña*. Una vez conectado correctamente al sistema, se almacenará el último login correcto con lo que no será necesario introducir de nuevo los datos de conexión.

En el interfaz se encuentran los siguientes botones y menús.

- ***Connect*** *(Conectar)*. Activo cuando Yafrid Provider está desconectado.
- ***Disconnect*** *(Desconectar)*. Activo cuando Yafrid Provider está conectado.
- ***Hide*** *(Ocultar)*. Oculta el interfaz en la barra de tareas. (Sólo en sistemas MS Windows).
- ***Quit*** *(Salir)*. Sale de la aplicación.
- ***Help*** → ***About*** *(Ayuda → Acerca de)*. Información básica sobre Yafrid Provider.
- ***File*** → ***Update Configuration*** *(Archivo → Actualizar Configuración)*. Actualiza la información de conexión de Yafrid introduciendo la ruta donde está ubicado el servidor.

http://anakin.inf-cr.uclm.es:4080/apps/yafrid/yafrid-service/software

- ***File*** → ***Exit*** *(Archivo → Salir)*. Sale de la aplicación.

Una vez se hace clic sobre el botón ***Connect*** *(Conectar)*, Yafrid Provider comienza el proceso de conexión.

El proceso tiene varias etapas de las que se va informando a medida que se superan (Figura B.9).

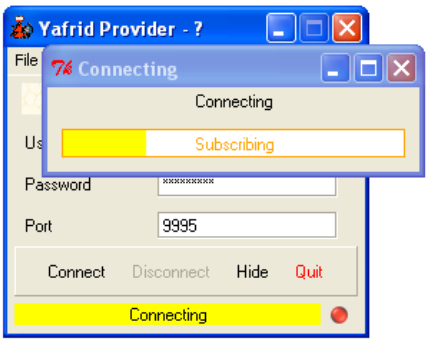

Figura B.9: Interfaz de Yafrid Provider 0.0.2.

Estas etapas son las siguientes:

- Getting controller. Obtener el Controlador mediante el que se conectará al grid.
- Checking workunits. Verificar las unidades que no son ya necesarias.
- Subscribing. Suscripción.
- Identifying. Identificación.
- Launching. Se lanza el proceso que esperará que le envíen trabajos.
- Activating. Activación del proveedor.
- Saving Configuration. Se almacena la configuración de conexión.

Al final del proceso y si todo ha salido como se esperaba se mostrará la pantalla de la Figura B.9.

Figura B.10: Interfaz de Yafrid Provider 0.0.2.

B.4.2. Ejecución en MS Windows (Particularidades)

Para ejecutar, simplemente hay que hacer doble clic en uno de los iconos creados por el asistente con lo que se lanzará la aplicación.

Al arrancar el sistema, si no se ha eliminado el acceso directo del Menú Inicio, Yafrid Provider hace un máximo de tres intentos de conectarse a Yafrid. Si consigue conectarse, se oculta el interfaz y se convierte en un icono de la barra de tareas. Si no lo consigue, el programa termina y deja un log con los errores encontrados.

Si no se ha editado el fichero yafrid-provider.conf añadiendo un login (usuario y contraseña) válido o no se ha hecho esto automáticamente tras una conexión correcta previa, el inicio automático fallará.

B.4.3. Ejecución en GNU/Linux (Particularidades)

B.4.3.1. Modo gráfico

Para iniciar la aplicación habrá que ejecutar el mismo script que se usó para instalar la aplicación. De este modo, para lanzar Yafrid Provider, basta con hacer lo siguiente.

cd provider ./yafrid_provider.sh

La ejecución del script sin ningún parámetro tendrá como resultado el inicio de la aplicación en modo gráfico. Su funcionamiento será el mismo descrito en el Apartado B.4.1 con la salvedad de que el botón 'Hide' no aparecerá, así como tampoco su comportamiento asociado.

B.4.3.2. Modo línea de comandos

El script anterior puede aceptar un parámetro que en caso de existir hace que la aplicación no muestre interfaz gráfico.

Hay dos valores posibles para este parámetro y su significado se muestra a continuación.

./yafrid_provider.sh start

Yafrid Provider trata de conectarse a Yafrid. Lo intenta en tres ocasiones como máximo.

./yafrid_provider.sh stop

Finaliza la ejecución de Yafrid Provider si éste estaba funcionando previamente.

Éste es también el modo en el que se lanza Yafrid Provider si se configuró la aplicación para iniciarse al arrancar el sistema.

Si no se ha editado el fichero yafrid-provider.conf añadiendo un login (usuario y contraseña) válido o no se ha hecho esto automáticamente tras una conexión correcta previa, la aplicación fallará.

B.5. Solución de Problemas

B.5.1. Errores al intentar conectar Yafrid Provider a Yafrid

PROVIDER cannot be set up.

- **Explicación.**

  Bien el proveedor ya está iniciado, o bien existe una aplicación que está establecida en el mismo puerto (por defecto el 9995).

  El proveedor no está conectado a internet o su configuración de red impide la conexión.

- **Solución.**

  Parar la instancia de Yafrid Provider existente en el sistema o cambiar el puerto en el que se inicia Yafrid Provider según el caso.

DISTRIBUTOR is unreachable.

- **Explicación.**

  El archivo de configuración tiene un valor incorrecto en los parámetros de configuración del servidor (clave DISTRIBUTORIP). Otra razón puede ser que el equipo en el que se está ejecutando Yafrid Provider no tenga una dirección IP válida (o que se trate de un equipo sin conexión a internet).

- **Solución.**

  Si la causa es la primera de las dos expuestas, la solución pasa por cambiar el valor de la clave/claves afectadas por el correcto o bien configurarlo automáticamente mediante *File → Update Configuration*.

Incorrect username or password.

- **Explicación.**

  Los parámetros de conexión no son correctos.

- **Solución.**

  Usar los parámetros correctos. Comprobar si tiene activo BLOQ. MAYÚS.

B.5.2. Errores al instalar Yafrid Provider en GNU/Linux

Paquete no instalado.

- **Explicación.**

  Al intentar importar algún módulo, la aplicación falla. En ese caso se muestra una traza similar a la siguiente.

```
Traceback (most recent call last):
File "Interfaz.py", line 1, in ?
from Tkinter import *
ImportError: No module named Tkinter
```

  En la traza mostrada, el import falla debido a que *python2.4-tk* no está instalado. Lo mismo ocurriría si no estuviera el paquete *python2.4-imaging*. En ese caso la traza mostraría un error al importar el paquete PIL.

- **Solución.**

  Instalar los paquetes necesarios (ver Sección sobre la Instalación en GNU/Linux).

dpkg: command not found

- **Explicación.**

  La aplicación dpkg no está instalada en el sistema.

- **Solución.** Este programa se usa únicamente para comprobar los paquetes instalados. No es imprescindible para el funcionamiento de Yafrid Provider. Su instalación es sencilla mediante *apt-get*.

ANEXO C

Manual de Usuario
Cliente Yafrid

Este manual está dirigido a aquellos usuarios que accedan al sistema Yafrid como clientes. Es decir, que su cometido consista en enviar trabajos al sistema para ser procesados.

C.1. Gestión de Proyectos

El cliente puede dar de alta proyectos de tres tipos:

- Yafray.
- Blender Animation.
- Blender Static.

Para crear cualquiera de ellos, el fichero a subir será un archivo comprimido en formato zip que contenga los ficheros necesarios. En el caso de proyectos de Yafray, el xml que define la escena vendrá acompañado de los ficheros de texturas, etc, ...En el caso de los proyectos Blender, el .blend tiene que contener todas las texturas y demás ficheros necesarios empaquetados.

En la parte inferior de la pantalla, aparecerán los proyectos que el usuario haya dado de alta. Para cada uno de ellos se mostrará su tipo, su nombre y descripción y el estado en el que se encuentran.

Accediendo a las propiedades de alguno de ellos se pasará a la pantalla de previsualización en la que se podrá ir viendo en tiempo real cómo el proyecto es procesado.

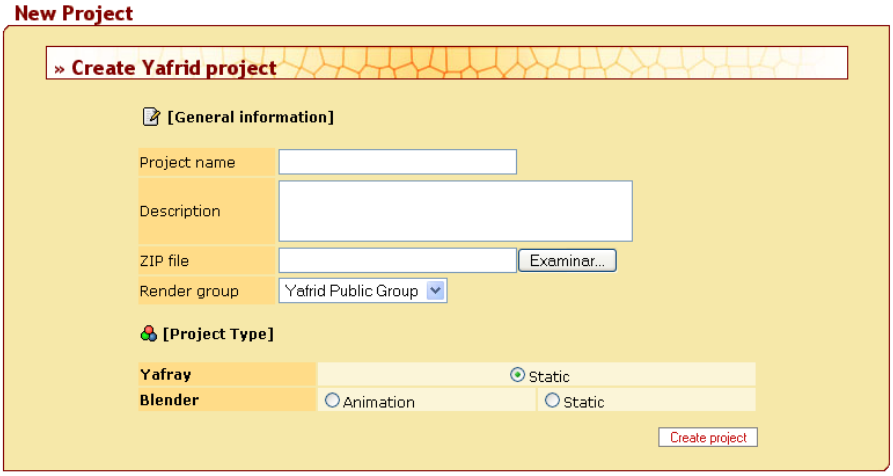

Figura C.1: Detalle de la creación de proyectos.

En el caso de las animaciones aparecerá una barra de progreso indicando cuánto falta para que el proyecto finalice. En el caso de los proyectos de render estáticos, aparecerá además la previsualización de la imagen compuesta por los fragmentos que hayan sido realizados hasta el momento.

Existe una pantalla que permite determinar los parámetros con los que se realizará el render (Figura C.5). Hay parámetros de dos tipos:

- Básicos. Corresponden con las dimensiones de la imagen a generar o, en en caso de las animaciones, con los frames de inicio y fin.

- Avanzados. Permiten determinar el tamaño de la unidad de trabajo y el ancho de la banda de interpolación. En función del motor de render permite variar unos parámetros u otros. Un ejemplo se muestra en la Figura C.5 en la que se pueden variar parámetros como el nivel de *oversampling* de Blender.

C.2. Estadísticas

En todo momento, un cliente puede acceder a las estadísticas concernientes a los proyectos que ha creado. Las estadísticas tendrán el mismo aspecto que las que aparecen en el resto de la aplicación. Los valores aparecerán acumulados, detallados por proveedor, por proyecto, etc.

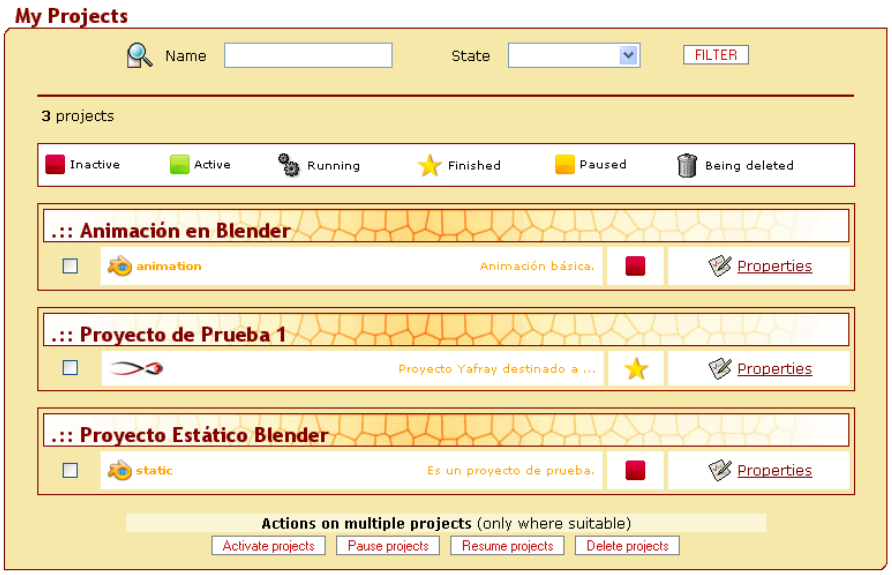

Figura C.2: Detalle del listado de proyectos.

C.3. Grupos de Usuarios

La creación y mantenimiento de grupos es una tarea propia del rol de cliente. Sirve para tener un control total acerca de quiénes renderizan los proyectos de quién. Los grupos que cree un usuario estarán administrados por él mismo por defecto aunque esto se puede cambiar accediendo al detalle del grupo.

La prioridad con la que un cliente se suscribe a un grupo está determinada por el sistema y puede variar en función del desempeño de éste en el sistema.

Figura C.3: Pantalla de previsualización de resultados.

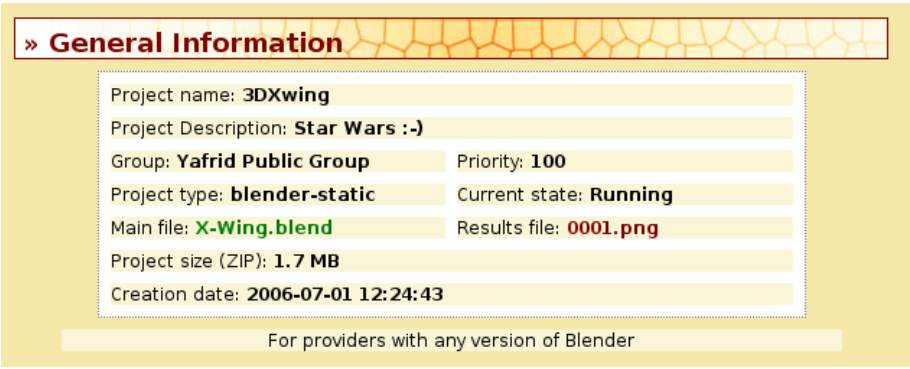

Figura C.4: Pantalla de información general del proyecto.

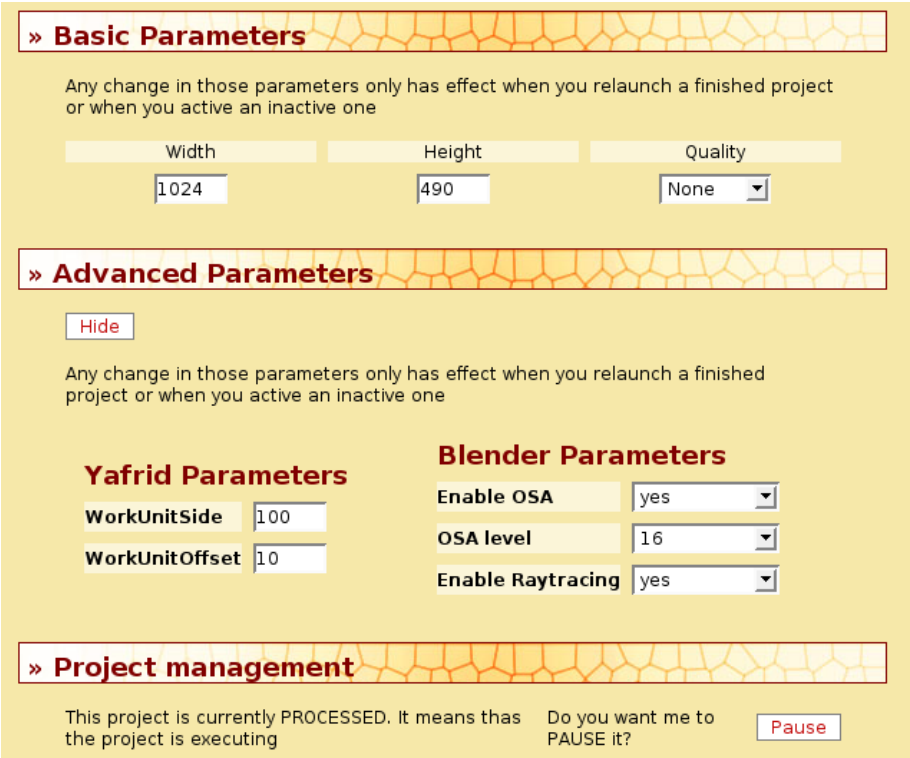

Figura C.5: Pantalla de parámetros de lanzamiento (Blender).

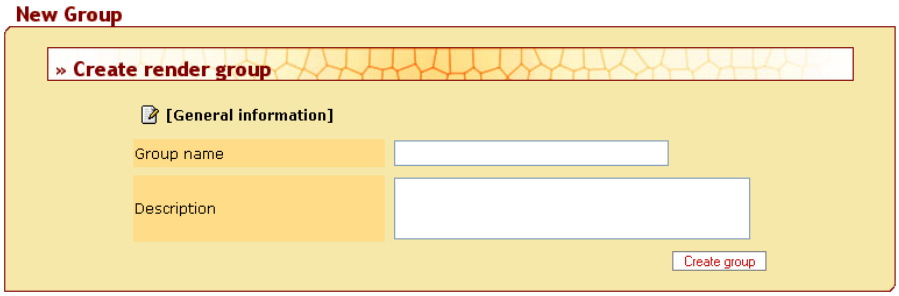

Figura C.6: Detalle de la creación de grupos.

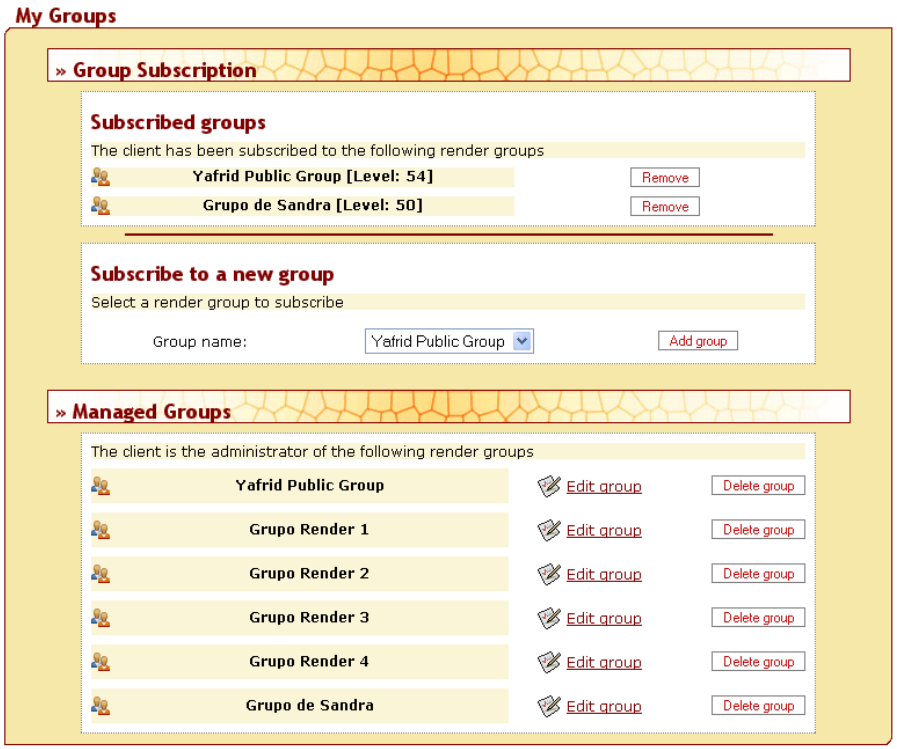

Figura C.7: Detalle de la suscripción a grupos.

ANEXO D

Manual de Usuario
Administrador Yafrid

Este documento está dirigido a aquellos usuarios que tengan privilegios de administrador en el sistema Yafrid. Proporciona una guía rápida para las distintas funcionalidades que proporciona el sistema.

D.1. Gestión de Usuarios

La pantalla de Gestión de Usuarios es una pantalla que permite conocer los usuarios que se encuentra registrados en el sistema (Figura D.1).

En esta pantalla y en las demás que son básicamente de gestión se permite realizar un filtrado de los elementos mostrados. Se puede usar el carácter comodín *asterisco* (*) que es equivalente a cualquier carácter. De este modo, si introducimos en el campo de filtrado el texto *Pro**, se mostrarán todos los elementos cuyo nombre comience por *Pro*.

De cada usuario se muestra el papel o papeles que desempeña en el sistema así como su estado. Los usuarios cuyas cuentas hayan sido activadas tendrán un icono distinto a aquellos que no están activados.

Haciendo clic en Propiedades se accede a los datos de cada usuario.

D.2. Gestión de Grupos

En esta pantalla se puede acceder a un listado en el que aparecen los distintos grupos del sistema junto con alguna información sobre los usuarios que lo componen (Figura D.2).

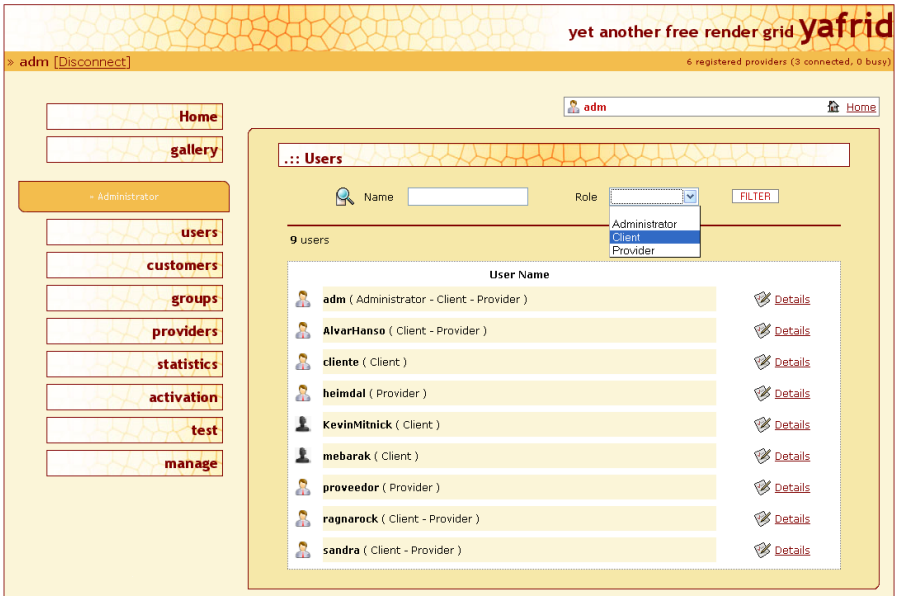

Figura D.1: Pantalla de Gestión de Usuarios.

Haciendo clic en las Propiedades de cualquiera de ellos, se accede a la pantalla de edición (Figura D.3) en la que se pueden modificar algunas de las características del grupo. También se puede editar la prioridad que los usuarios tienen en ese grupo.

D.3. Gestión de Proveedores

En esta sección, el administrador puede consultar el estado de los proveedores del sistema.

En la Figura D.4 se muestra la pantalla de *Proveedores* en la que se muestran los distintos proveedores del sistema. La información que se presenta para cada proveedor es el estado codificado por un color (Cuadro D.1) y los motores de render que soporta.

Para acceder a la información general y de conexión de cada proveedor basta con hacer clic en el enlace de *Propiedades* (Figura D.5).

Color	Significado
Rojo	El Proveedor no está conectado
Verde	El Proveedor está conectado y libre
Amarillo	El Proveedor está conectado y ocupado.

Cuadro D.1: Código de colores para el estado de los proveedores.

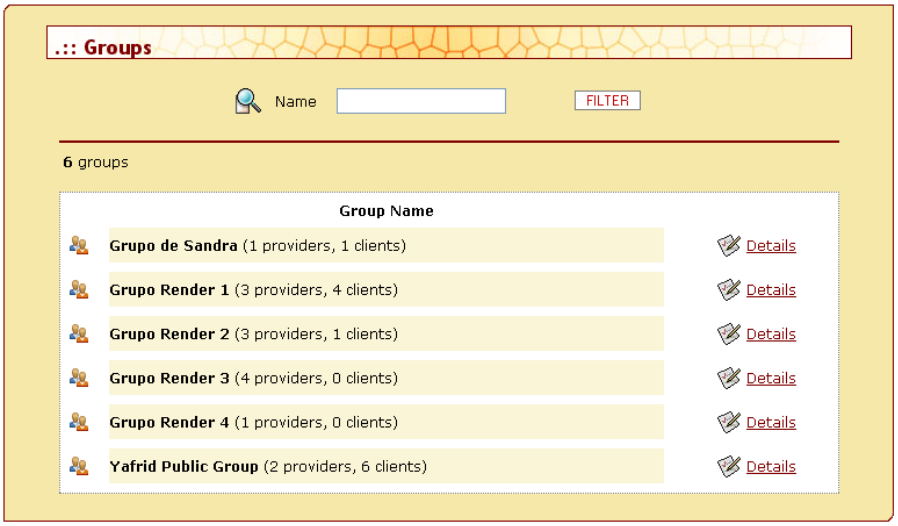

Figura D.2: Detalle de la pantalla de grupos.

D.4. Estadísticas

El funcionamiento del sistema tendrá como efecto colateral la generación de un gran volumen de estadísticas.

En esta sección se muestran estadísticas de los proyectos, de los conjuntos de pruebas y de los proveedores. Estos datos van desde tiempos de render parciales, completos o acumulados hasta el número de unidades realizadas o perdidas por un determinado proveedor.

D.5. Activación

Si la variable de configuración ACTIVATION_REQUIRED está activada, cada vez que se dé de alta un grupo, se enviará un correo electrónico a la cuenta del sistema. Estos grupos no serán visibles para los usuarios hasta que no hayan sido activados.

En esta pantalla aparecerán los distintos grupos en este estado junto con su descripción y el usuario que pidió su creación (Figura D.6). Dos acciones son posibles:

- Denegar. El grupo será borrado del sistema.
- Activar. El grupo se activará y podrá ser utilizado por los distintos usuarios del sistema.

En ambos casos, un correo electrónico se enviará al usuario que dio de alta el grupo.

Figura D.3: Detalle de la pantalla de edición de grupo.

D.6. Pruebas

El módulo de pruebas ofrece a los administradores del sistema un servicio destinado principalmente a automatizar la creación de múltiples proyectos.

Mediante el interfaz web (Figura D.7) se pueden generar, a partir de un solo fichero comprimido, múltiples proyectos. Al conjunto de proyectos creados se le denomina **conjunto de pruebas**.

Los proyectos generados presentarán diferencias en el tamaño de la unidades de trabajo en las que se dividen o en la calidad de la imagen resultante. Con la configuración inicial que incluye tres grados de calidad y cinco tamaños de unidad de trabajo, se generarían quince proyectos en cada conjunto de pruebas.

En la parte inferior de la pantalla (Figura D.8) se muestran los conjuntos de prueba creados por el administrador. También es posible consultar el estado de los proyectos pertenecientes al conjunto de pruebas.

Esta aplicación es interesante a la hora de comparar resultados y sacar conclusiones acerca de los mejores parámetros de división para cada tipo de escena.

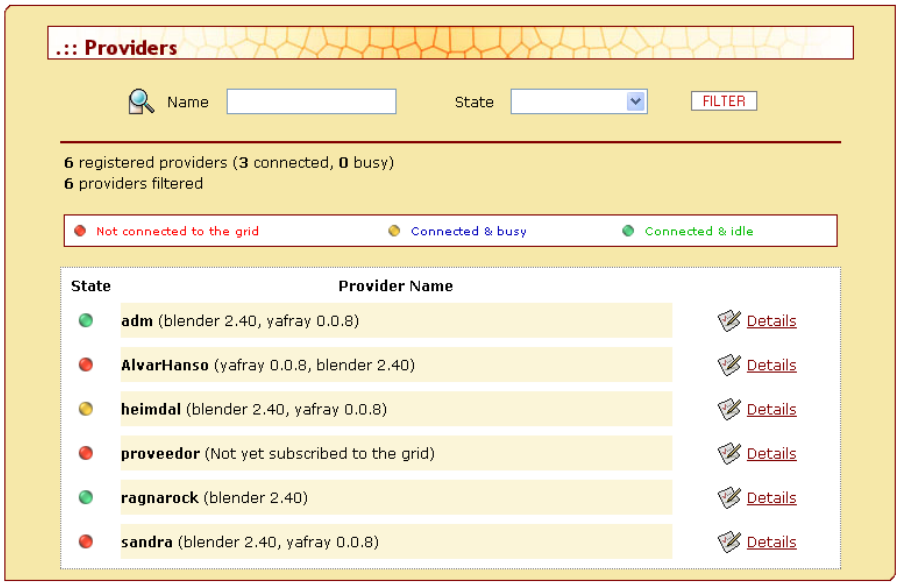

Figura D.4: Detalle de la pantalla de proveedores.

D.7. Gestión de Procesos

Desde el interfaz web, el administrador del sistema puede gestionar el funcionamiento de los procesos que componen Yafrid-CORE, es decir, la parte no interactiva del sistema.

En la pantalla de *Gestión de Procesos* (Figura D.9) se puede consultar el estado de los dos procesos existentes, el Identificador y el Distribuidor. Para ello, existe un script denominado YAFRID_checkServer cuya ruta se puede parametrizar. Junto al nombre de los procesos aparece su descripción, y su estado. Un proceso apagado se mostrará con el color rojo y uno activo en color verde.

Para gestionar los procesos Identificador y Distribuidor que forman Yafrid-CORE existe una pequeña utilidad llamada YAFRID_manageServer. Esta utilidad puede ser llamada directamente o a través del interfaz web con idéntico comportamiento.

- En GNU/Linux es un fichero de script llamado YAFRID_manageServer.sh.
- En sistemas MS Windows consiste en un fichero de proceso por lotes denominado YAFRID_manageServer.bat.

Ambos tienen la misma sintaxis:

./YAFRID_manageServer.[bat|sh] [acción] [alcance]

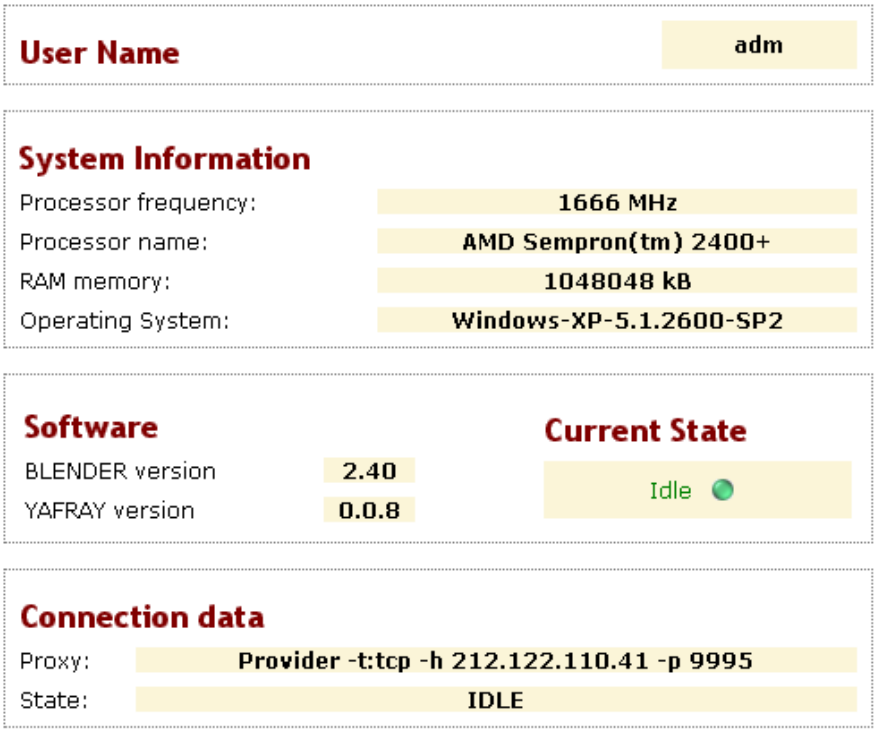

Figura D.5: Datos de conexión de un Proveedor.

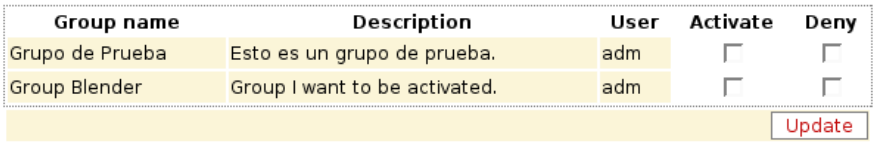

Figura D.6: Detalle de la pantalla de activación.

El argumento *acción* indica la acción a realizar sobre los procesos. Los valores válidos son **start** y **stop** que realizan las acciones obvias.

El argumento **alcance** indica qué procesos se verán afectados por la acción. Hay tres valores válidos: **I** (Identificador), **D** (Distribuidor) y **A** (ambos).

Así, una llamada posible sería:

./YAFRID_manageServer.sh start A

que se haría en sistemas GNU/Linux y tendría como resultado que ambos procesos se iniciaran.

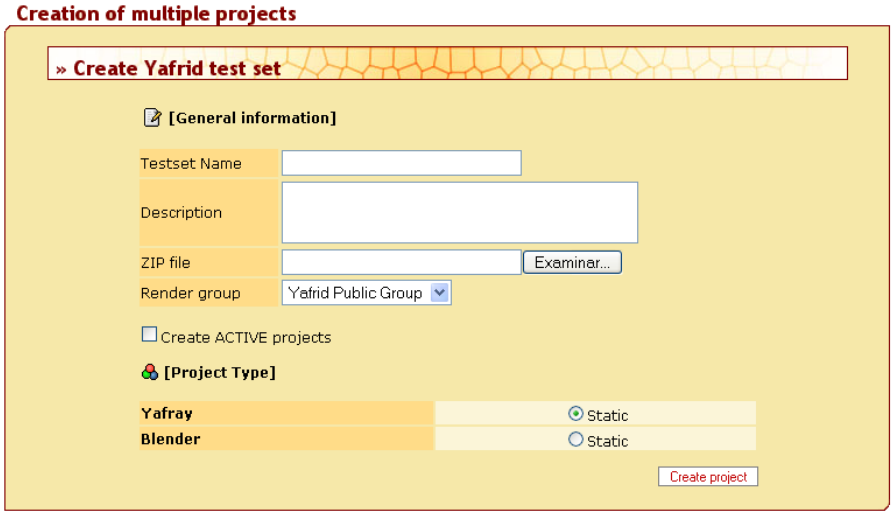

Figura D.7: Detalle de la pantalla de creación de conjuntos de pruebas (I).

Figura D.8: Detalle de la pantalla de creación de pruebas (II).

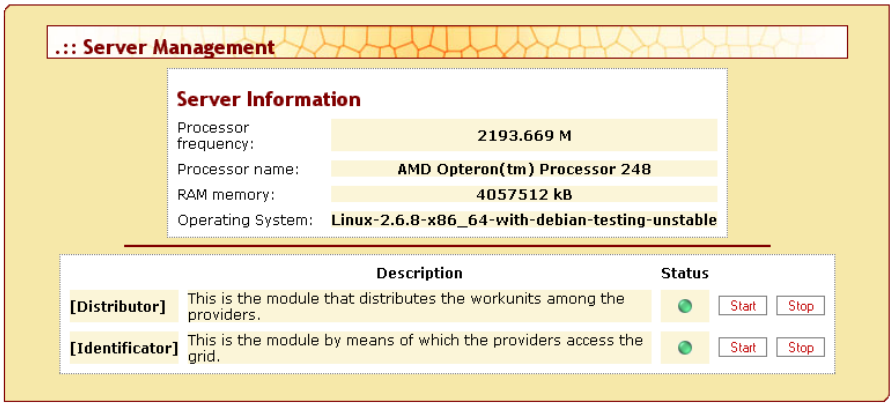

Figura D.9: Detalle de la pantalla de gestión de procesos.

ANEXO E

Manual de Instalación y Configuración Yafrid Server 0.0.2

Este documento está dirigido a los usuarios que deseen instalar Yafrid Server 0.0.2 en sus sistemas. Esta guía le servirá para instalar y configurar los dos módulos de los que se compone el sistema, **Yafrid-WEB** y **Yafrid-CORE**.

NOTA: En caso de no desear proceder a la instalación de Yafrid Server 0.0.2 en local, es posible el acceso al sistema completamente funcional en la dirección siguiente:

http://anakin.inf-cr.uclm.es:4080/apps/yafrid/yafrid-service/

E.1. Instalación de Yafrid-WEB

La instalación de Yafrid-WEB en sistemas GNU/Linux y MS Windows es bastante similar. A continuación se explica paso a paso la instalación de Yafrid-WEB en una distribución basada en Debian y en el sistema operativo de Microsoft.

E.1.1. Sistemas GNU/Linux

Instalación de Paquetes Necesarios

Para el funcionamiento de Yafrid se requieren algunos paquetes básicos que se encuentran en los repositorios de las distintas distribuciones de GNU/Linux.

Mediante cualquier software de gestión de paquetes como apt-get, Adept o Synaptic instale los siguientes paquetes:

- libapache2-mod-php5
- php5-common
- php5-mysql
- php5-gd
- mysql-server
- blender[1]
- sendmail[2]

PHP.ini

Una vez instalados los paquetes necesarios, hay que realizar algunos cambios en la configuración de PHP que requerirán editar el fichero **/etc/php5/apache2/php.ini**. Este fichero está dividido en secciones y contiene variables que determinan el comportamiento de PHP.

En primer lugar es necesario habilitar la extensión de *MySQL* en PHP en la sección correspondiente a las extensiones dinámicas **(Dynamic Extensions)** del fichero PHP.ini. En la misma sección, también hay que habilitar la extensión para la librería de gráficos *GD*. Para ello, basta con descomentar o añadir las siguientes líneas:

extension=mysql.so
extension=gd.so

También es conveniente aumentar el tamaño de los ficheros a subir. Para ello, en la sección **File Uploads**, habrá que editar la constante *upload_max_filesize* de este modo:

upload_max_filesize=6M

Si se desea habilitar el envío de correos electrónicos (sólo necesario si la constante de configuración de Yafrid ACTIVATION_REQUIRED está activa) habrá que editar la sección adecuada añadiendo algo similar a lo siguiente:

[1] No es imprescindible. Sólo es necesario para crear proyectos Yafrid de Blender
[2] Servidor de correo. Sólo es necesario si la variable ACTIVATION_REQUIRED está activada.

[mail function]
sendmail_path = /usr/sbin/sendmail -t
sendmail_from = me@myserver.com

Para que los cambios realizados tengan efecto, es necesario reiniciar Apache:

/etc/init.d/apache2 [restart | stop | start]

Creación de la Base de Datos

Por defecto, Yafrid utiliza *root* como usuario para acceder a la base de datos y como contraseña, la cadena vacía, que es la configuración por defecto de MySQL. Para utilizar un usuario de base de datos distinto, basta con modificar el fichero de configuración de Yafrid (Apartado E.1.3).

Para crear la base de datos que usarán tanto Yafrid-WEB como Yafrid-CORE con una configuración inicial, introduzca desde un terminal lo siguiente:

mysql -u root < /Fuentes/SQL/yafrid-database.sql

Si ha creado un usuario de MySQL que desea utilizar para Yafrid, habrá que introducir:

mysql -u USERNAME -p < /Fuentes/SQL/yafrid-database.sql

con lo que se le pedirá que introduzca la contraseña de acceso para el usuario de nombre *USERNAME*.

Creación de Yafrid-WEB

Una vez hecho esto, copie el directorio **/Fuentes/Yafrid-WEB/** del CD adjunto en el directorio **/var/www/**. Para acceder al sistema web, abra un navegador e introduzca *http://localhost/Yafrid-WEB/* con lo que aparecerá la página de bienvenida de Yafrid (Figura E.1) si el proceso se ha realizado correctamente.

E.1.2. Sistemas MS Windows

La instalación de Yafrid-WEB en sistemas MS Windows es similar al proceso que se ha seguido en el punto anterior y en muchos de los pasos se le remitirá a apartados anteriores.

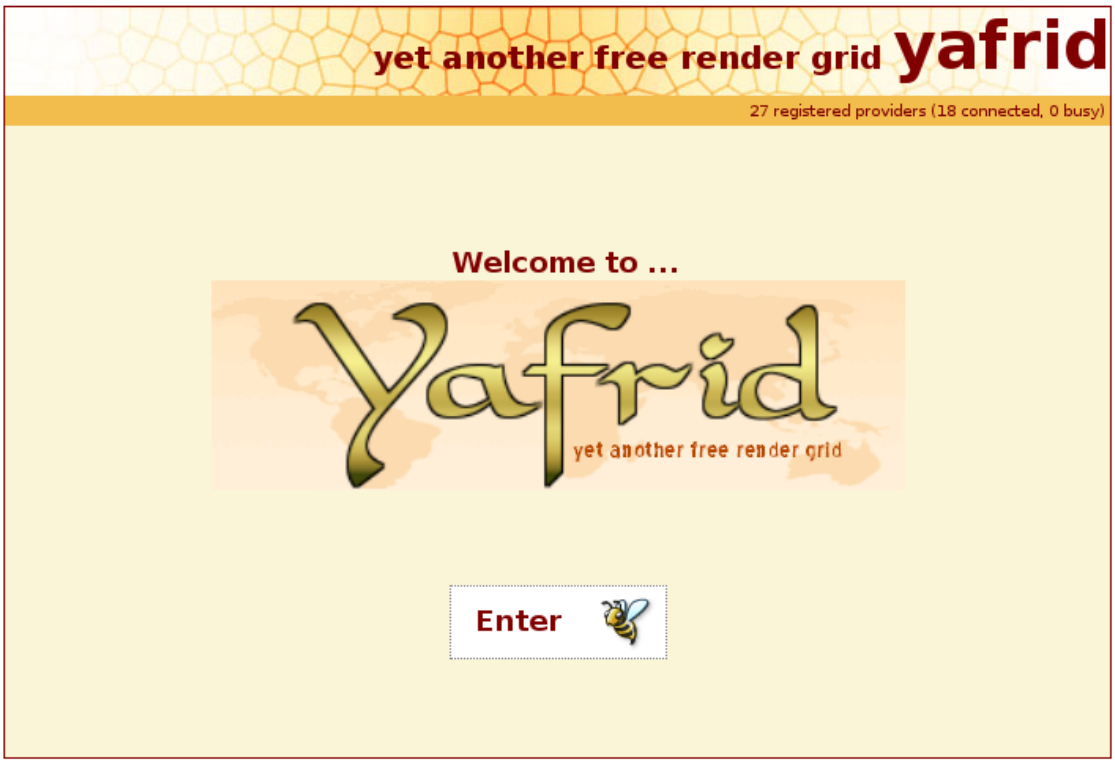

Figura E.1: Pantalla de bienvenida del sitio web de Yafrid.

Por un lado será necesario instalar una solución WAMP, es decir, el servidor web Apache, el servidor de bases de datos MySQL y PHP. Aunque pueden instalarse por separado, existen numerosos paquetes que incluyen todo lo necesario y se distribuyen bajo licencia GPL.

En el CD adjunto se ha incluido el ejecutable **wamp5_1.6.4a.exe** en el directorio */Distribución/Otros/*. Un asistente le guiará durante el proceso en todo momento.

También será necesaria la instalación de **Blender** para que sea posible crear proyecto Yafrid de este motor. Un ejecutable para MS Windows de la última versión de Blender hasta la fecha también se incluye en el directorio */Distribución/Otros/*

En el caso de que se quiera utilizar el mecanismo de activación mediante el correo electrónico será necesario contar con un servidor de correo. Para su configuración, consulte el Apartado E.1.1.

Finalmente, habrá que realizar los pasos de **creación de la base de datos** (Apartado E.1.1) y la **copia de Yafrid-WEB** al directorio base del servidor web (dependerá del directorio de instalación de la solución WAMP).

E.1.3. Configuración

Una vez instalado con éxito Yafrid-WEB, es necesario configurarlo adecuadamente. Para ello, sólo hay que editar el fichero de configuración **YAFRID-Web.ini** que encontrará en el directorio */Yafrid-WEB/domain/ini/*. No será necesario modificar todas las variables. Para la mayoría, la configuración por defecto será la adecuada. Sólo habrá que modificar aquellas que hacen referencia a rutas a directorios o ficheros.

A continuación se muestra un listado completo de las constantes que contiene este fichero organizadas por secciones.

- [GENERAL]
  - **YAFRID_ROOT.** Contiene el directorio base de Yafrid-WEB, por ejemplo */var/www/yafrid*.
  - **YAFRID_URL.** Es la dirección web que aparece en el navegador, por ejemplo *http://localhost/Yafrid-WEB*.
  - **ACTIVATION_REQUIRED.** Indica si es necesario activar vía correo electrónico las cuentas de usuario y los grupos después de crearlos o no.
  - **SUB.** Cadena de sustitución. Parámetro interno del sistema que tiene relación con la internalización de los mensajes de usuario.
  - **PASS_MIN.** Tamaño mínimo en caracteres de una contraseña válida.
  - **LOGIN_MIN.** Tamaño mínimo en caracteres de un nombre de usuario válido.
  - **INITIAL_CLI_PR.** Prioridad inicial con la que se crearán las cuentas de los clientes.
- [EMAIL]
  - **FROM_NAME.** Parámetro para el envío de correos.
  - **FROM_ADDRESS.** Parámetro para el envío de correos.
  - **STYLE.** Cadena en notación CSS que determina el aspecto del texto de los mensajes de correo.
- [MYSQL]
  - **HOST.** Host de MySQL. Por defecto *localhost*.

- **DB.** Base de datos de MySQL. Por defecto *yafrid*.
- **USER.** Usuario de MySQL. Por defecto *root*.
- **PASSWORD.** Contraseña de MySQL. Por defecto la cadena vacía.

- [PATH]
  - **CHECKSERVER.** Ubicación del script YAFRID_checkServer.sh.
  - **MANAGESERVER.** Ubicación del script YAFRID_manageServer.sh.
  - **BLENDER.** Ubicación del ejecutable de Blender 3D.
  - **USER_PROJECTS.** Ubicación del directorio de proyectos.
- [SOFTWARE]
  - **BLENDER.** URL de descarga de Blender 3D.
  - **YAFRAY.** URL de descarga de Yafray.
- [DEFAULT]
  - **LANGUAGE.** Lenguaje por defecto de la aplicación.
  - **PREVIEW_SIZE.** Tamaño por defecto de la previsualización de los resultados de los proyectos.
  - **REFRESH_FREQUENCY.** Tasa de refresco de la previsualización de los resultados de los proyectos.
- [PROJECT]
  - **SIZE.** Tamaño de la unidad de trabajo por defecto.
  - **OFFSET.** Ancho de la banda de interpolación por defecto.

E.2. Instalación de Yafrid-CORE

En este apartado se detalla la instalación del componente Yafrid-CORE que junto a Yafrid-WEB forman el Servidor de Yafrid.

E.2.1. Sistemas GNU/Linux

Para que funcione Yafrid-CORE, existen varios paquetes que son indispensables. Mediante un gestor de paquetes de su elección (los más conocidos son Adept, Synaptic y apt-get) instale lo siguiente:

- python2.4
- python2.4-imaging
- python2.4-mysqldb

También es necesario disponer de un servidor SFTP con una cuenta que será usada por el conjunto de los Proveedores y cuyos datos de conexión serán definidos en el fichero de configuración de éstos últimos.

Una vez instalados los paquetes necesarios, descomprimir el fichero **YafridServer.zip** que se encuentra en el directorio */Distribución/* del CD adjunto en el sistema de archivos local.

E.2.2. Sistemas MS Windows

El módulo Yafrid-CORE tiene dependencias con otros productos software por lo que, en un primer paso, habrá que preparar el sistema para que Yafrid-CORE funcione.

Es necesario instalar los programas y librerías que se muestran a continuación para lo que bastará con usar los ejecutables que se indican (y que se encuentran en el CD adjunto en el directorio */Distribución/Otros/*).

- **Python 2.4.1.** *python-2.4.1.msi*.
- **Python 2.4 pywin32 extensions.** *pywin32-204.win32-py2.4.exe*.
- **Python 2.4 PIL-1.1.5.** *PIL-1.1.5.win32-py2.4.exe*.
- **Python 2.4 MySQL-python.** *MySQL-python.exe-1.2.0.win32-py2.4.exe*.
- **Ice 2.1.2 para Visual Studio .NET 2003.** *Ice-2.1.2-VC71.msi*.

Una vez instalados, será necesario configurar las variables de entorno añadiendo lo siguiente:

PATH: C:\Ice-2.1.2\bin; C:\python24
PYTHONPATH: C:\Ice-2.1.2\bin; C:\Ice-2.1.2\python

Por último bastará con extraer el contenido de **YafridServer.zip** del directorio */Distribución/* a una ubicación en el sistema de archivos local. Como en el caso anterior también es necesario un servidor para realizar la transferencia de ficheros entre Servidor y Proveedores.

E.2.3. Configuración

Una vez instalado el módulo Yafrid-CORE, es necesario configurarlo editando el fichero **yafrid-distributor.conf** que se encuentra en el directorio */Yafrid-CORE/*. No todas las variables del fichero han de ser modificadas ya que en la mayoría de casos, el valor por defecto es válido para cualquier instalación.

A continuación se muestra un listado de las constantes del fichero de configuración.

- **POOL.** Directorio donde las unidades de trabajo serán almacenadas para su adquisición por parte de los Proveedores.
- **USER_PROJECTS.** Directorio raíz a partir del cual se creará el árbol de directorios con los proyectos de los clientes.
- **TMAX.** Tiempo máximo que se espera a que una unidad de trabajo sea procesada antes de consultar sobre su estado (en segundos).
- **TSLEEP.** Tiempo que marca la frecuencia de las iteraciones del Distribuidor (en segundos).
- **MYSQL_HOST.** Máquina para acceder al servidor MySQL.
- **MYSQL_USER.** Usuario de MySQL.
- **MYSQL_PASSWD.** Contraseña para conectar con el servidor MySQL.
- **MYSQL_DB.** Base de datos MySQL en la que se encuentran las tablas del sistema Yafrid.
- **WORK_ORDER.** Orden en el que serán procesadas las unidades de trabajo Los dos valores posibles son *ALEA* y *ORDERED*. El primero hace que las unidades se seleccionen aleatoriamente y el segundo que estén ordenadas en función de su posición en la imagen final.

- **TIMEOUT.** Parámetro interno del sistema. Tiempo en milisegundos tras el que caducarán los paquetes TCP/IP enviados a los Proveedores.

ANEXO F

CD Adjunto

- **Fuentes.** Ficheros fuente desarrollados para implementar Yafrid.
  - Yafrid-WEB. Proyecto *PHPEclipse* con los fuentes php de la parte web del sistema.
  - Yafrid-CORE. Proyecto *Eclipse-PyDev* con los fuentes de la parte batch del sistema.
  - Yafrid-Provider. Proyecto *Eclipse-PyDev* con los fuentes del Proveedor de Yafrid.
  - SQL. Estructura y contenido inicial de la base de datos de Yafrid.
- **Documentación.**
  - Fichero PDF a color del presente documento.
  - Ficheros fuente de LaTeX usados para la realización de este documento.
  - Diagramas UML desarrollados.
  - Imágenes incluidas en este documento y fuentes OpenOffice.
  - Otros documentos.
- **Distribución.**
  - Distribuciones de software para GNU/Linux y MS Windows.
  - Otros ficheros. Software adicional usado en el desarrollo.
- **Ejemplos.**
  - Ejemplos de imágenes obtenidas con el sistema.
  - Ficheros de proyectos de prueba (Blender y Yafray) listos para ser enviados al sistema.