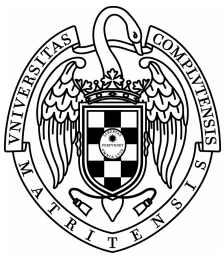


Sistemas Informáticos
Curso 2004-2005

Diseño e implementación en lenguaje Java de un gestor de recursos de hardware reconfigurable en un entorno de red

Aida Joaquín Acosta
José Luis Cruz Marcos
Daniel Lázaro Ángeles

Dirigido por:
Prof. Julio Septién del Castillo
Dpto. Arquitectura de Computadores y Automática

Facultad de Informática
Universidad Complutense de Madrid

SUMARIO

1. INTRODUCCIÓN 6

1.1. PLANTEAMIENTO DEL PROBLEMA 7

2. FUNCIONAMIENTO DEL SISTEMA 8

2.1 ¿QUÉ DEBE HACER EL SISTEMA? 8

2.2. SOLUCIÓN AL PROBLEMA 8

2.3. APLICACIONES EXISTENTES 9

2.4. SERVICIOS PROPORCIONADOS AL USUARIO HUMANO EN CADA APLICACIÓN 9

2.5. DIAGRAMA DE COMUNICACIONES 10

3. ESPECIFICACIÓN DE REQUISITOS 15

3.1 DISTINTAS FORMAS DE GESTIÓN DE HARDWARE DINÁMICAMENTE RECONFIGURABLE 15

3.2 GESTIÓN QUE NUESTRO SISTEMA REALIZA DE LOS DISPOSITIVOS DE HARDWARE DINÁMICAMENTE RECONFIGURABLES 18

3.3 ¿CÓMO HEMOS DEFINIDO Y DESARROLLADO NUESTRO SISTEMA? 19

4. DISEÑO DEL SISTEMA 20

4.1. MOTIVACIÓN DEL DISEÑO 20

4.2. DISEÑO DE LAS APLICACIONES 21

4.2.1. APLICACIÓN USUARIO 21

4.2.2. APLICACIÓN CLIENTE 23

4.2.3. APLICACIÓN SERVIDOR 26

4.3. PROTOCOLO DE COMUNICACIONES 28

4.3.1. TIPOS DE SOCKETS Y SU ESTABLECIMIENTO 28

4.3.2. TIPOS DE MENSAJES 29

4.4. CONTROL DE ERRORES 32

4.4.1. TIPOS DE ERRORES (según su magnitud) 32

4.4.2. EJEMPLOS DE SITUACIONES DE ERROR 34

4.4.3. OTROS ERRORES NO CONTEMPLADOS POR EL SISTEMA 35

4.4.4. PUNTOS DÉBILES DEL SISTEMA 35

5. IMPLEMENTACIÓN 37

5.1. APLICACIÓN USUARIO 37

5.2. APLICACIÓN SERVIDOR 38

5.3. APLICACIÓN CLIENTE 40

6. MANUAL DE USUARIO 42

6.1. REQUISITOS DEL SISTEMA 42

6.1.1. REQUISITOS HARDWARE 42

6.1.2. REQUISITOS SOFTWARE 42

6.2. INSTALACIÓN DE LAS APLICACIONES 42

6.2.1. SOBRE UNA PLATAFORMA Windows 42

6.2.2. SOBRE UNA PLATAFORMA Linux 46

6.3. CONFIGURACIÓN DEL SISTEMA 48

6.3.1. APLICACIÓN USUARIO 48

6.3.2. APLICACIÓN SERVIDOR 51

6.3.3. APLICACIÓN CLIENTE 52

6.4. USO DE LA APLICACIÓN USUARIO 54

6.5. USO DE LA APLICACIÓN SERVIDOR 61

6.6. USO DE LA APLICACIÓN CLIENTE 68

7. POSIBLES MEJORAS Y NUEVOS USOS 72

7.1. GESTIÓN DE FPGAs FÍSICAS (NO SIMULADAS) 72

7.2. MÚLTIPLES PARTICIONES POR TAREA 72

7.3. RECONFIGURACIÓN EN TIEMPO DE EJECUCIÓN DE RECURSOS 73

7.4. MEJORAS EN LA SEGURIDAD 74

7.5. PERFILES DE OPERADOR 75

7.6. RECUPERACIÓN DEL SISTEMA FRENTE A CAIDAS DEL SERVIDOR . 75

8. GLOSARIO DE TÉRMINOS 77

8.1. TÉRMINOLOGÍA BÁSICA 77

8.2. TÉRMINOLOGÍA AVANZADA 77

9. BIBLIOGRAFÍA 81

RESUMEN DEL PROYECTO

Los avances recientes en el hardware dinámica y parcialmente reconfigurable han desembocado en lo que se conoce como la multitarea hardware, que permite que unos dispositivos conocidos como FPGA's ejecuten diversas tareas hardware simultáneamente. Por otra parte, también recientemente, se ha establecido lo que se conoce como Grid Computing; resumidamente, sistemas de computación distribuidos con una alta interoperabilidad entre sus actores.

Nuestro sistema pretende reunir ambas ideas, construyendo un gestor distribuido de FPGA's que ofrece la capacidad computacional de estos dispositivos a unos usuarios remotos. El sistema es capaz, por lo tanto, de recibir peticiones de tareas hardware, de identificar los recursos capaces de ejecutar cada tarea y de asignarlos a los recursos de forma que se optimice el rendimiento global, y por último devolver el resultado fruto de los cálculos al usuario demandante de la tarea.

ABSTRACT

The recently technology advances in hardware partially and dynamically reconfigurable, have enabled multijob hardware, that is, simultaneous execution of many jobs in hardware devices as FPGAs (Filed-Programmable Gate Array). In addition, nowadays, Grid computing, as distributed computational systems which enables expansive and underutilized resource-sharing, is becoming more and more important.

Our aim is to combine both ideas, implementing a distributed management FPGA's system that offers the computational capabilities of these devices to remote users. The system is capable for receiving hardware job requests, identify the resources available for executing each job and assign them to the more efficient resource to optimize global performance. At last, return the result of the execution to the job owner.

LISTA DE PALABRAS CLAVE / KEYWORDS LIST

FPGA
GRID
GHADIR
Java
Gestor de Recursos Hardware
Protocolo de Red
Hardware Dinámicamente Reconfigurable

LICENCIA

Los autores de este proyecto, Aida Joaquín Acosta, José Luis Cruz Marcos y Daniel Lázaro Ángeles, autorizan mediante este texto a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

José Luis Cruz Marcos Aida Joaquín Acosta Daniel Lázaro Ángeles

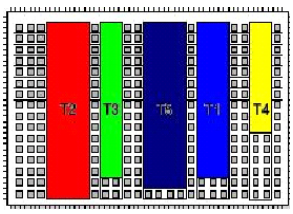
1. INTRODUCCIÓN

Dos tendencias de esta última década nos hacen plantearnos nuevos problemas y retos. Éstas son los continuos avances tecnológicos en hardware reconfigurable y por otro lado, el aprovechamiento de estos recursos caros y a menudo infrautilizados.

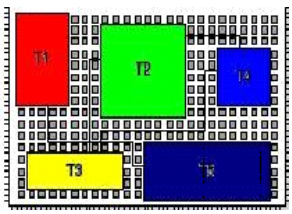
I. Los *continuos avances en la tecnología de hardware reconfigurable (HR)*, usado fundamentalmente para el prototipado rápido de circuitos integrados, han permitido, por un lado, la aparición de *hardware reconfigurable dinámicamente*, es decir, cuya funcionalidad puede modificarse en tiempo de ejecución. Algunas de estas FPGAs, incorporan la característica adicional de reconfiguración *parcial*, lo cual permite cambiar partes del dispositivo mientras que otras partes continúan trabajando, lo que es, una gestión dinámica y versátil. Por otro lado, *creciente densidad lógica*. Cuando en 1985 Xilinx lanzó al mercado la primera FPGA (Field-Programmable Gate Array), estaba contaba con cerca de 1,000 puertas lógicas, pero ya en el 2004 había alcanzado un tamaño 10,000 veces mayor y para el 2010, las previsiones son de un billón de transistores. Esta creciente densidad lógica de las FPGAs, permite dos cosas:

i. *Integración de sistemas en chips programables*, cuando la patente ha sido siempre de los Circuitos Integrados de Aplicaciones Específicas (ASICs). En comparación con los ASICs, las FPGAs proporcionaban mayor flexibilidad pero menor densidad-lógica. Con la reciente llegada de FPGAs de alta densidad, la integración de sistemas en lógica reconfigurable es ahora posible.

ii. *Multitarea Hardware*. Planificar, de forma dinámica, la ejecución de múltiples tareas HW sobre un mismo HR, permitiría, por ejemplo, que una parte de una FPGA dinámicamente reconfigurable ejecutase una o varias aplicaciones locales, mientras que otra queda disponible para otras tareas, por ejemplo, para su uso por parte de usuarios remotos. Una novedad que también han aportado las investigaciones más recientes en tecnología de FPGAs es la posibilidad de disponer de *múltiples contextos*, es decir, varias memorias de configuración cargadas en planos distintos, entre las cuales puede conmutarse muy rápidamente. Estas FPGAs son las que permiten los tiempos de reconfiguración dinámica mas reducidos.



Reconfiguración 1-D



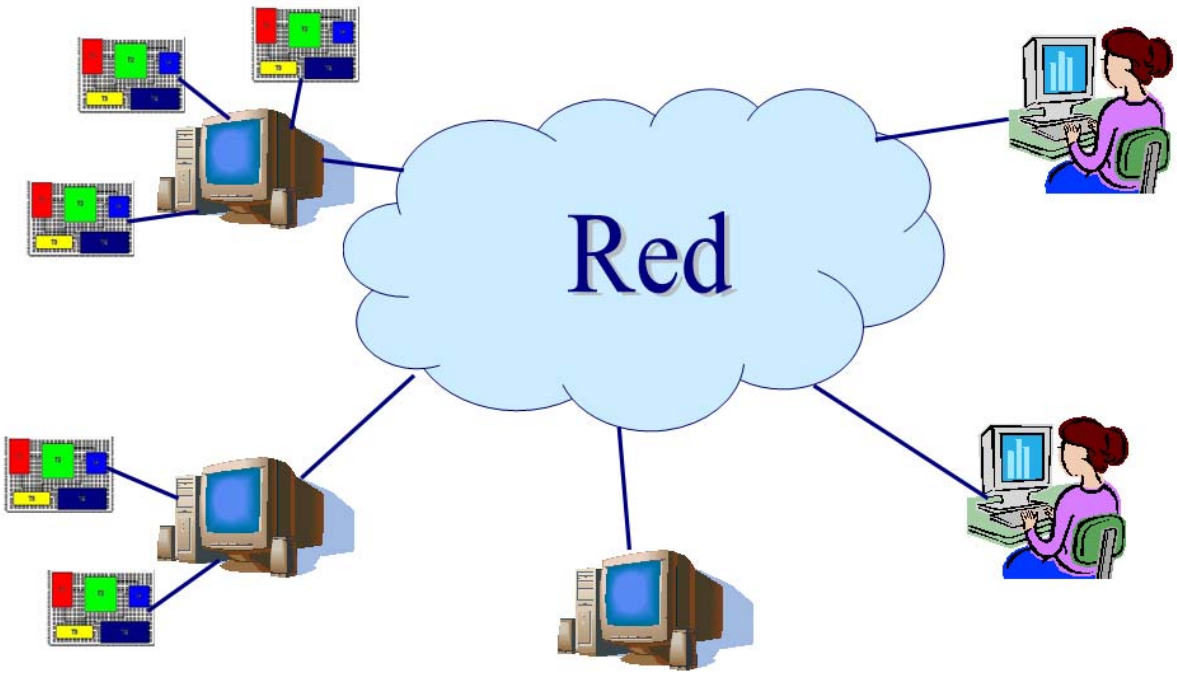
Reconfiguración 2-D

II. Por otro lado, los recursos reconfigurables distribuidos, como FPGAs (Field-Programmable Gate Array), *son caros y están infrautilizados*. Así pues, es importante permitir la compartición de esos recursos, al menos, en el seno de una misma organización. Además, cada vez resulta más habitual que en entornos universitarios y de investigación, en la mayoría de casos conectados entre sí por medio de redes locales de media o alta velocidad, se disponga de equipos en los que se han incluido tarjetas con hardware reconfigurable, tipo FPGA. En general, éste no es utilizado de manera continuada por sus usuarios habituales sino más bien de forma esporádica, para la realización de pruebas en momentos concretos, la implementación y depuración de prototipos, etc.

1.1. PLANTEAMIENTO DEL PROBLEMA

El *objetivo* de nuestro proyecto construir una aplicación, a modo se los sistemas Grid, los cuales permiten el intercambio de recursos software, el uso de redes, bases de datos y hasta el aprovechamiento de la potencia de los ordenadores de alto rendimiento, para aprovechar recursos hardware reconfigurables.

La siguiente figura da una idea bastante clara del problema al que nos enfrentamos.



2. FUNCIONAMIENTO DEL SISTEMA

2.1 ¿QUÉ DEBE HACER EL SISTEMA?

Como ya sabemos, el sistema que hemos decidido crear debe proporcionar una solución distribuida al problema de la ejecución de tareas en dispositivos de hardware dinámicamente reconfigurables (de tipo FPGA). Es decir, consiste en un entorno de gestión de este tipo de recursos en un contexto de red.

Más concretamente, el sistema debería aceptar tareas o lotes de tareas hardware de diversos usuarios que pueden estar geográficamente lejanos entre sí, y distribuirlas "automáticamente" a las FPGAs que posteriormente las ejecutarán. Finalmente el resultado de la ejecución debe ser enviado a los usuarios.

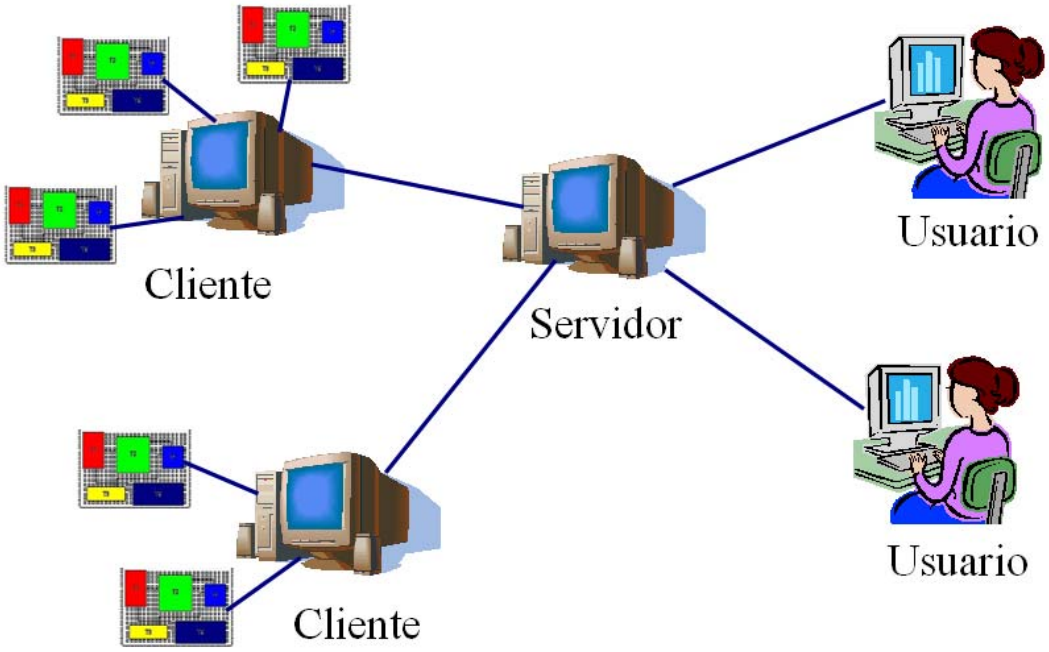
La parte del proceso correspondiente a la ejecución de las tareas en FPGAs cae fuera del alcance de nuestro proyecto. No obstante, **el entorno quedará preparado para poder ser unido muy fácilmente con FPGAs reales**.

Lo indicado anteriormente supone varias cosas. La primera es que el usuario propietario de la tarea y la FPGA que va a ejecutar no tienen por que encontrarse en el mismo host, si no que en general basta con que los dos se encuentren en un entorno de red (la comunicación se realiza a través de una arquitectura TCP/IP). La segunda es que el sistema aísla al usuario de tener que seleccionar el dispositivo HW concreto al que enviar la tarea, puesto que la aplicación es la que "automáticamente" distribuye las tareas. La tercera es que el sistema simula la ejecución de tareas en los dispositivos a los que han sido asignadas, puesto que cómo ya hemos indicado, la conexión de nuestro sistema con una FPGA real cae fuera del alcance de nuestro proyecto. El resultado devuelto a los usuarios es, por tanto, un resultado ficticio.

2.2. SOLUCIÓN AL PROBLEMA

La solución que planteamos al problema es la de definir tres tipos de equipos:

- Equipos con recursos, que llamaremos **Clientes**.
- Equipos con usuarios pero sin recursos, que llamaremos **Usuarios**.
- Equipo capaz de gestionar a los dos anteriores, que llamaremos **Servidor**.



2.3. APLICACIONES EXISTENTES

El entorno supondrá la existencia de tres tipos de sistemas, todos ellos con comunicación a través de una arquitectura TCP/IP: Un número indefinido de **usuarios** (cada uno de ellos ejecutando una instancia de la "**Aplicación Usuario**"), que tienen las aplicaciones HW que desean ejecutar (las configuraciones o mapas de bits que van a cargarse en las FPGAs), un **servidor** único (ejecutando la "**Aplicación Servidor**") que conoce el tipo y el estado de cada recurso HW (FPGA) disponible en el entorno, y un conjunto de **clientes** (cada uno de ellos ejecutando una instancia de la "**Aplicación Cliente**") que se comunican con éste, y que son los que ofrecen sus recursos (FPGAs) para que sean usados por los usuarios con acceso al entorno.

Cada una de las aplicaciones se identifica por la dirección IP de la máquina en la que se ejecuta, de forma que en un mismo host pueden ejecutarse simultáneamente las tres aplicaciones. Sin embargo, sólo una instancia de cada tipo de aplicación puede ejecutarse simultáneamente en cada host.

2.4. SERVICIOS PROPORCIONADOS AL USUARIO HUMANO EN CADA APLICACIÓN

En la Aplicación Usuario, el operador o usuario humano tiene la posibilidad de crear nuevas tareas HW, o abrir tareas existentes que previamente habían sido creadas y guardadas. Una vez dispone de las tareas, puede demandar la ejecución de una de ellas o de varias a la vez (lote de

tareas). En todo momento el usuario humano tendrá conocimiento del estado actual de las tareas solicitadas.

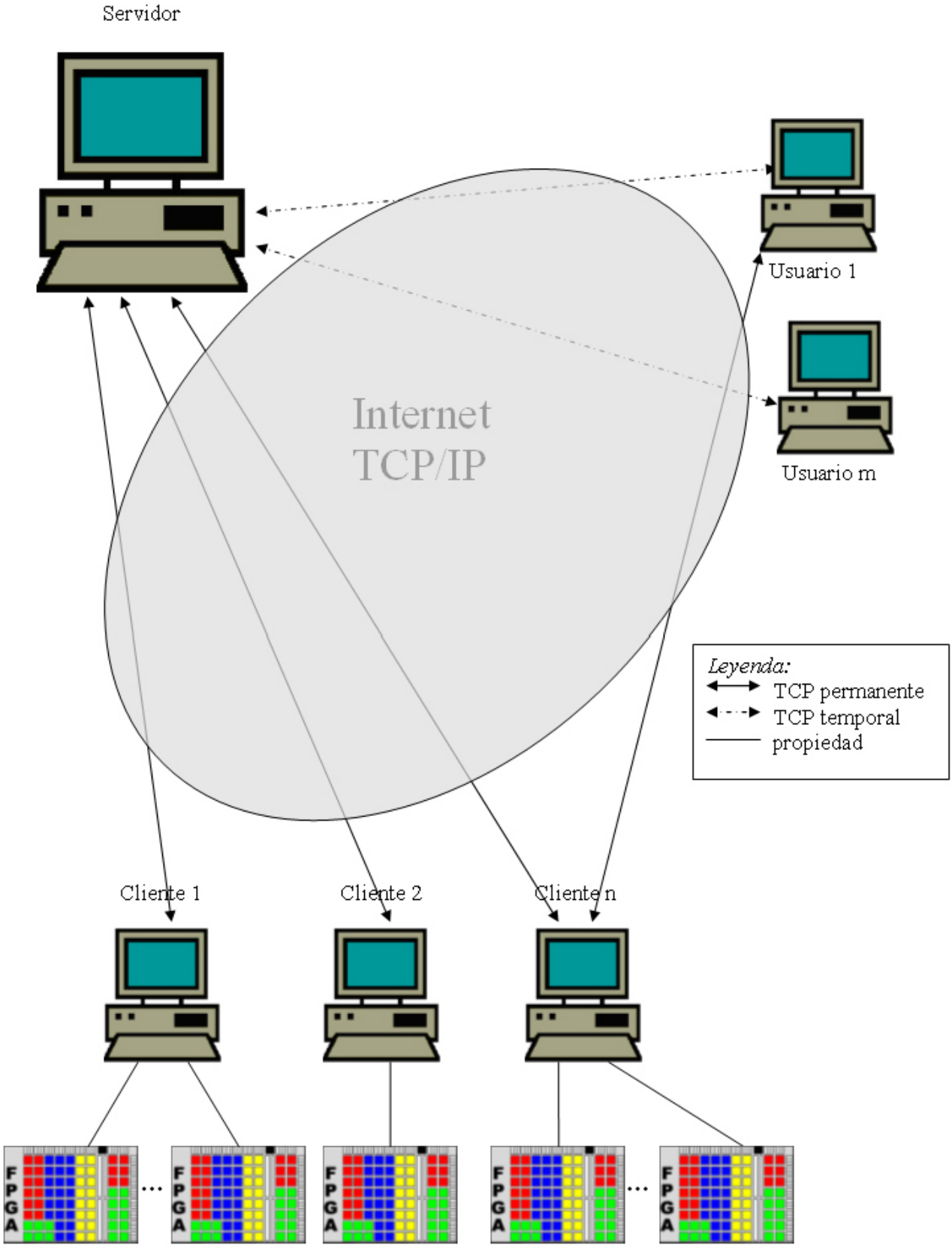
En la Aplicación Cliente, el usuario humano tiene la posibilidad de definir nuevos recursos HW (de tipo FPGA) o eliminar existentes. Además, podrá ver en tiempo real el estado de cada uno de sus recursos, es decir, las tareas que en él se están ejecutando.

La Aplicación Servidor es autónoma, de forma que el operador humano se limita a observar el estado global del sistema desde tres puntos de vista:

- punto de vista de los usuarios conectados, obteniendo para cada uno de ellos las tareas que se están ejecutando
- punto de vista de los clientes conectados, obteniendo para cada uno de ellos las tareas que tienen en ejecución
- punto de vista de las transacciones del sistema, obteniendo las transacciones que suceden en el sistema a modo de fichero de registro histórico o de log

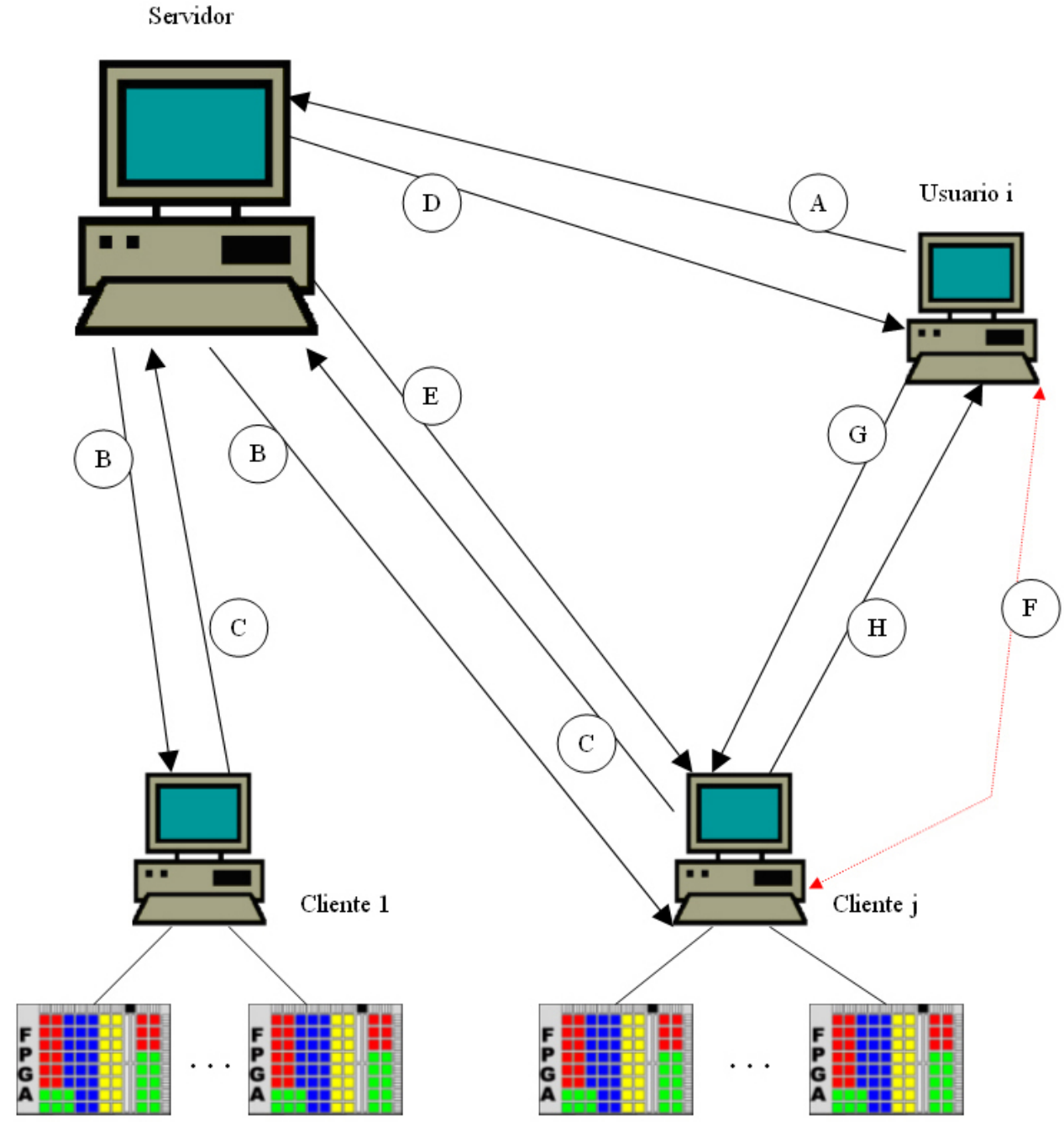
2.5. DIAGRAMA DE COMUNICACIONES

Mostramos a continuación el diagrama de comunicaciones de la aplicación atendiendo a los tipos de conexiones existentes entre las distintas aplicaciones: volátiles (temporales) o permanentes.


Servidor
Usuario 1
Usuario m
Internet
TCP/IP
Leyenda:
TCP permanente
TCP temporal
propiedad
Cliente 1
Cliente 2
Cliente n
FPGA
FPGA
FPGA
FPGA
FPGA

Las conexiones permanentes son las que se mantienen siempre activas aunque no se produzca ningún intercambio de información entre las aplicaciones, y por consiguiente, consumen recursos durante todo el tiempo que están activas. Las conexiones volátiles, por el contrario, son creadas “ad hoc” cada vez que se quiere transmitir una información y destruidas inmediatamente después, optimizando por tanto el consumo de recursos.

Mostramos a continuación el diagrama de comunicaciones de la aplicación atendiendo al flujo de información (necesaria para transmitir una tarea HW) entre las diferentes aplicaciones suponiendo que ya existe una conexión entre un usuario y el servidor (volátil) y entre los clientes y el servidor (permanente):



Leyenda:

conexión TCP permanente

propiedad

A	Solicitud de tarea del usuario
B	Reenvío de la solicitud de tarea a todos los clientes conectados
C	Respuesta a la solicitud de tarea
D	Confirmación o rechazo de la solicitud
E	Asignación de tarea
F	Establecimiento de conexión (TCP permanente)
G	Envío del mapa de bits asociado a la tarea
H	Resultado de la ejecución de la tarea

Comentamos brevemente el diagrama anterior:

El usuario i solicita la ejecución de una tarea al servidor **(A)**. Esta tarea vendrá especificada por una serie de parámetros, como son el tipo de FPGA para la que está compilada, el tiempo que tardará en ejecutarse, el tiempo máximo de espera del usuario (time out), etc. El servidor entonces, reenvía esa solicitud a todos los clientes conectados en ese momento **(B)**. Dichos clientes evalúan la tarea solicitada y envían la respuesta al servidor **(C)**, indicando si cada uno de sus recursos HW es capaz de ejecutar esa tarea (es del tipo adecuado y cumple la restricción del time-out); y en caso afirmativo en qué condiciones (tiempo de espera hasta la ejecución y espacio desperdiciado en el recurso si la tarea se destinase a ese recurso). Posteriormente, el servidor evalúa las respuestas anteriores de los clientes y decide si la tarea puede ser ejecutada; y en caso de que pueda, a qué cliente y a qué recurso dentro de cada cliente asignarlo. Para notificar la decisión, envía al usuario la confirmación o el rechazo de la solicitud **(D)**. Si la solicitud se rechaza, finaliza la comunicación entre las aplicaciones. En caso de que se acepte, el servidor envía la asignación de tarea **(E)** al cliente en cuestión (cliente j), indicando además la dirección IP del usuario con la que el cliente debe establecer comunicación **(F)**. Una vez establecida la comunicación, el usuario i proporciona al cliente j el mapa de bits asociado a la tarea. Finalmente, cuando ha transcurrido el tiempo de ejecución de la tarea (simulación de la ejecución de la tarea en una FPGA real) el cliente envía al usuario el resultado (ficticio) de la ejecución.

Por supuesto, la aplicación es concurrente, lo que quiere decir que a partir del mensaje de confirmación o rechazo que el servidor envía al usuario, el servidor está disponible para atender a nuevas peticiones de otros usuarios (o del mismo).

3. ESPECIFICACIÓN DE REQUISITOS

En el apartado “Funcionamiento del sistema” se dio una visión global del objetivo del sistema y de su funcionamiento. Aquí especificaremos de forma mucho más refinada las características concretas del gestor de recursos, fundamentalmente de lo directamente relacionado con las FPGAs. Por otra parte, indicaremos la organización seguida por los integrantes del grupo de proyecto para alcanzar la solución al problema.

A groso modo, sabemos que la misión del sistema es distribuir las tareas hardware enviadas por el usuario para que se ejecuten (si pueden) en los dispositivos (recursos HW) adecuados. La cuestión que inmediatamente surge aquí y que desarrollaremos a lo largo de este capítulo es **cómo gestionar esos recursos HW**.

3.1 DISTINTAS FORMAS DE GESTIÓN DE HARDWARE DINÁMICAMENTE RECONFIGURABLE

Los diferentes tipos de gestión de este tipo de hardware, se basan fundamentalmente en dos características que pueden o no soportar las FPGAs: la reconfiguración dinámica y la reconfiguración parcial.

La *reconfiguración dinámica* consiste en permitir cargar una tarea en la FPGA sin necesidad de parar las tareas que estaban previamente en ejecución. La *reconfiguración parcial* (o multitarea hardware) consiste en subdividir la FPGA en particiones de modo que cada pueda ejecutar una tarea independientemente de las demás. En función de estas características, tenemos las siguientes formas de gestión:

1. Gestión estática
2. Gestión dinámica
    a. Una Dimensión
        i. Celdas de tamaño uniforme y fijo
        ii. Celdas de tamaño no uniforme y fijo
        iii. Celdas de tamaño variable
    b. Dos dimensiones
        i. Celdas de tamaño uniforme y fijo
        ii. Celdas de tamaño no uniforme y fijo
        iii. Celdas de tamaño variable

Pasamos ahora a definir brevemente cada uno de estos tipos de gestión.

1. GESTIÓN ESTÁTICA

Es la forma más fácil de gestión, puesto que no soporta ningún tipo de reconfiguración parcial (implementada en HW). Es decir, los CLBs que componen el espacio utilizable de la FPGA se tratan como un todo, de forma que la FPGA sólo puede estar ejecutando una tarea en cada instante.

Entre las ventajas de este tipo de gestión tenemos el abaratamiento del coste del hardware y la ausencia de fragmentación. Sin embargo, la principal desventaja es que la FPGA queda desaprovechada.

2. GESTIÓN DINÁMICA

Mediante la gestión dinámica podemos optimizar el aprovechamiento de estos dispositivos al permitir reconfiguración parcial.

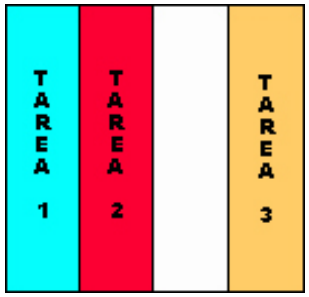
Los diferentes tipos de gestión surgen de los distintos tipos de particiones que podemos definir.

a. UNA DIMENSIÓN

Definimos particiones de una dimensión, es decir, la altura de las mismas coincide con la altura de la FPGA (en número de CLBs), mientras que la anchura viene en función del número de divisiones deseadas. En resumen, se divide la FPGA en un número finito de columnas.

i. Celdas de tamaño uniforme y fijo

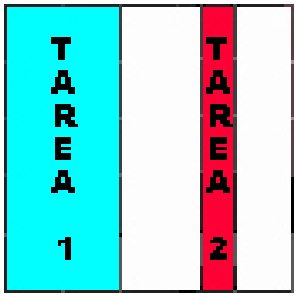
Consiste en definir N particiones del mismo tamaño y que ocupen todo el ancho de la FPGA. De esta manera podemos tener simultáneamente en ejecución N tareas, siempre y cuando cada una de ellas quepa en la partición (en número de CLBs y en forma).



Como ventaja tenemos la ausencia de fragmentación externa (fragmentación interna existe en todos los modelos de gestión con particiones fijas). La principal desventaja es la ausencia de flexibilidad, ya que sólo se admitirán tareas con formato de columna.

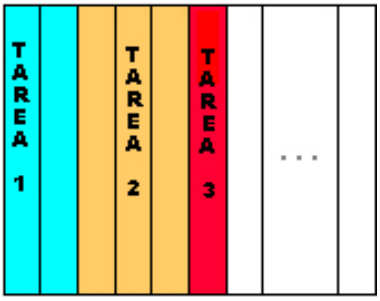
ii. Celdas de tamaño no uniforme y fijo

Este tipo de fragmentación pretende evitar la excesiva importancia que tiene el tamaño de la partición a la hora de aprovechar el recurso. Es similar a la anterior, salvo que en este caso no todas las particiones tienen el mismo tamaño, con lo que prácticamente se elimina la fragmentación interna.



iii. Celdas de tamaño variable

Con esta forma de gestión se puede seleccionar el tamaño de la partición en función de las necesidades de la tarea a ejecutar, utilizando exactamente el número necesario de columnas (cada una ocupando la anchura correspondiente a un CLB.

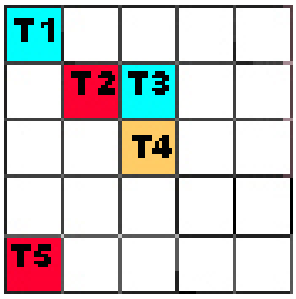


Evidentemente, esta es la mejor forma de gestión en una dimensión, pero es la más costosa de implementar. Desaparece la fragmentación interna, aunque seguimos teniendo el problema de la forma de las tareas (inherente a la gestión en una dimensión) y de la gran fragmentación externa.

b. DOS DIMENSIONES

Para solucionar el problema geométrico de las tareas, necesitamos usar FPGAs con una gestión en 2D. No obstante, las tareas no pueden tener cualquier forma geométrica, por el gran incremento que supondría en el coste del algoritmo de gestión, pero se pueden plantear simplificaciones que ayudan a rentabilizar el uso de los dispositivos HW (FPGAs).

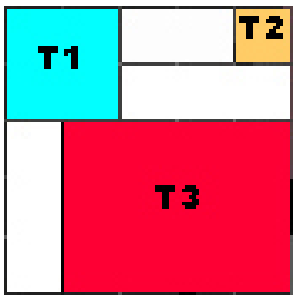
i. Celdas de tamaño uniforme y fijo



Se divide el área en N particiones del mismo tamaño (cuadrados) y que ocupen toda la matriz de celdas de la FPGA.

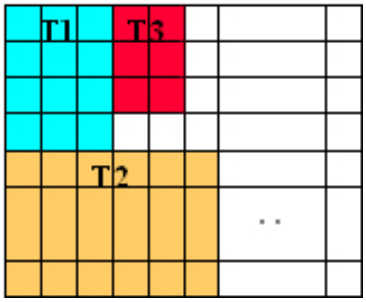
La ventaja fundamental de este tipo de gestión es la sencillez, mientras que la desventaja la encontramos en el desperdicio de espacio y en la fragmentación interna.

ii. Celdas de tamaño no uniforme y fijo



Similar a la anterior, salvo que en este caso no todas las particiones tienen el mismo tamaño ni la misma geometría, con lo que se reduce mucho la fragmentación interna. Por supuesto, el coste de este modelo es superior al anterior.

iii. Celdas de tamaño variable



El tamaño y la geometría de la partición se selecciona en función de las necesidades de la tarea a ejecutar. Es la mejor forma de gestión de todas las consideradas hasta ahora y también la más costosa.

3.2 GESTIÓN QUE NUESTRO SISTEMA REALIZA DE LOS DISPOSITIVOS DE HARDWARE DINÁMICAMENTE RECONFIGURABLES

Una vez conocidos todos los tipos de gestión existentes, ya podemos indicar que los tipos de gestión que nuestro sistema permite son: gestión estática y gestión dinámica en una dimensión con celdas de tamaño uniforme y fijo permitiendo dos o cuatro particiones.

Para implementar estos tipos de gestión, cada recurso dispone de una lista de particiones. La longitud de dicha lista podrá ser:

- 1, si el recurso se gestiona de forma estática. A este tipo de gestión la llamaremos "todo o nada" o "una partición"
- 2, si el recurso se gestiona de forma dinámica (en una dimensión) con dos particiones, cada una ocupando la mitad de la superficie útil de la FPGA. A este tipo de gestión la llamaremos "dos particiones"
- 4, si el recurso se gestiona de forma dinámica (en una dimensión) con cuatro particiones, cada una ocupando una cuarta parte de la superficie útil de la FPGA. A este tipo de gestión la llamaremos "cuatro particiones"

Hay que destacar que el tratamiento de la gestión estática como una lista de particiones con un solo elemento, permite que los tres modelos de gestión se traten de forma homogénea.

En nuestro sistema, el modelo de gestión va unido al recurso HW, de manera que a un recurso se le asigna el modelo de gestión que el usuario desee cuando se da de alta, y ya no se puede cambiar de modelo de gestión durante el tiempo durante el cual se mantenga vivo el recurso. Es decir, el sistema no permite "reasignación dinámica del modelo de gestión". No obstante, ya existe una persona del departamento ACYA trabajando para conseguir esta mejora y se espera que en un breve periodo de tiempo esté disponible. La ventaja que aporta la "reasignación dinámica del modelo de gestión" es que menos tareas serán rechazadas por no caber en una partición, ya que bastaría con fusionar particiones. También puede disminuirse el espacio desperdiciado, sin más que dividir particiones.

Por otra parte, cada partición tendrá asociada una cola de tareas software con las tareas que la misma tiene pendientes de ejecutar. Una vez el cliente ha solicitado la tarea y ésta ha sido aceptada y asignada a la partición de un recurso, dicha tarea se colocará en la última posición de la cola SW asociada a la partición asignada. Cuando la tarea llegue a la cabeza de la cola, ésta podrá pasar a ejecución, siempre y cuando disponga ya del fichero de mapa de bits y del fichero de datos asociados a la tarea.

La Aplicación Cliente no espera a que la tarea llegue a la cabeza de la cola para solicitar los dos ficheros anteriores, sino que tan pronto como la tarea se acepte, el cliente intenta establecer contacto con el usuario propietario de la tarea para que éste le transmita los ficheros. Se ha elegido esta opción para mejorar el tiempo de respuesta del sistema, ya que el envío de los ficheros de

mapa de bits y de datos es, con mucha diferencia, la transmisión más lenta que se produce entre las aplicaciones.

Finalmente, solo queda darse cuenta de que los dos ficheros se transmiten directamente del usuario (propietario) al cliente, sin pasar por el servidor (a diferencia de lo que ocurre con el resto del flujo de la comunicación), lo que permite minimizar la sobrecarga de la red, y traducirse en última instancia en una mejora en el rendimiento.

En el siguiente punto resumimos los pasos que hemos seguido para atacar el problema y por qué decidimos actuar así.

3.3 ¿CÓMO HEMOS DEFINIDO Y DESARROLLADO NUESTRO SISTEMA?

El modelo de proceso o patrón, que hemos seguido para diseñar el software, ha sido un **modelo evolutivo incremental**. Partimos de un núcleo que funciona y una vez evaluado, incrementamos el producto con otra versión operativa más completa. Creímos que era la mejor solución, ya que podría adaptarse con mayor facilidad a los cambios de requisitos; nuestro director de proyecto podría ver el núcleo funcionando e ir añadiendo más especificaciones y para el número de personas que constituyen el equipo, parecía razonable este planteamiento.

Como toda gestión de un proyecto comienza con la planificación para facilitar las estimaciones de recursos y planificaciones temporales, nuestro primer paso en la planificación fue identificar el *ámbito* del proyecto, es decir, el contexto, los objetivos de información, funciones y rendimientos. A continuación, fue necesario hacer una *estimación de recursos*, en cuanto aplicaciones necesarias, sistemas hardware y también gestión de personal, que al ser tres personas, no fue muy complicado el estimar disponibilidades y riesgos. La decisión de organización del equipo fue no tener un jefe permanente, sino nombrar uno en función de cada tarea. Las decisiones, problemas y enfoques se acordaban en grupo, es decir, una organización de tipo descentralizada y democrática, como define Mantei.

Y por último, restaba la descripción de los servicios y restricciones, esto es, los *requisitos de sistema*, que requirieron varias entrevistas con el director, en las que presentábamos una memoria con los requisitos que habíamos entendido en anteriores reuniones, y la corregíamos hasta dar con los objetivos deseados y así establecer la línea base desde la que comenzar a trabajar.

La planificación temporal y posibles riesgos fueron planteados en grupo pero gestionado por un solo miembro.

4. DISEÑO DEL SISTEMA

En el primer apartado, pasamos a comentar, brevemente, cuestiones generales del diseño de las tres aplicaciones, es decir, qué tipo de software queríamos conseguir, y cuáles han sido nuestros pasos para aproximarnos a él. En los siguientes apartados mostramos el diseño de las tres aplicaciones que definen el sistema.

4.1. MOTIVACIÓN DEL DISEÑO

Este proyecto fue planteado como una herramienta de investigación para el GHADIR; así que, nuestro objetivo era crear un sistema útil, eficiente y eficaz, que se ajustase al máximo a las especificaciones del grupo de trabajo. Aportamos todo el equipo nuestro conocimiento en Ingeniería del Software para crear un sistema *reutilizable*, asignando responsabilidades para mantener bajo acoplamiento entre módulos distintos y alta cohesión entre clases con funcionalidades relacionadas y pequeñas, *escalable*, y a la vez, *amigable*; es decir, construir un edificio sólido que permita elevarlo hasta convertirlo en rascacielos.

Con esta motivación, las tres aplicaciones que constituyen el sistema tienen ciertas características de diseño comunes, las cuales se enumeran a continuación:

i. El uso de *factorías abstractas*: con esto, pretendemos que el sistema sea independiente de cómo se crean, se componen y se representan sus productos. De esta forma, hemos agrupado, por un lado, las estructuras de datos típicas como colas, listas y tablas, en una familia que constituye una factoría de estructuras de datos y, por otro lado, una que recoge otro tipo de estructuras más complejas como el tipo de recursos o el de tareas que usa el sistema, diseñadas para ser usadas conjuntamente (una familia de componentes). Así, la configuración del sistema quedará determinada por la implementación que se elija de dichas familias.

ii. *Patrón Singleton*: se trata de garantizar que exista una única instancia de algunas clases. De esta forma, mantenemos un acceso a controlado a la única instancia y un refinamiento de operaciones y de la representación. Las factorías implementan este patrón.

iii. *Patrón Fachada*: el cual nos da un interfaz unificado y simplificada de alto nivel, que nos permite acceder a los servicios generales del subsistema (del que hace de fachada) de manera más sencilla. Además, minimizamos las comunicaciones y las dependencias entre los subsistemas, reduciendo así su complejidad. Cada aplicación tiene una fachada más general de cara a las otras aplicaciones y otras, para cada subsistema que la forma.

iv. *Patrón Peso Ligero*: compartimos objetos en vez de utilizar copias de éstos para todo, así, reducimos el coste en recursos y en integridad, para mantener la coherencia en todo el sistema. Las tareas y los recursos que se gestionan son un ejemplo.

v. *Patrón Mediador*: este patrón de comportamiento nos permite encapsular en un objeto la forma de interactuar de otros. Evitamos que los objetos se refieran unos a otros explícitamente y permite variar su interacción de forma independiente. Nosotros recogemos en un subsistema la construcción de mensajes con los que se comunican entre sí las tres aplicaciones del sistema. Este subsistema, en cada aplicación respectivamente, media entre los otros subsistemas para obtener la información necesaria para construir los mensajes del protocolo de comunicaciones.

vi. Para mantener la consistencia entre las interfaces y la funcionalidad (o modelo), utilizamos el patrón *Modelo-Vista-Controlador (MVC)*, simplificado, como mecanismo de propagación de cambios. De esta forma hacemos que el modelo sea independiente de la representación de la salida y del comportamiento de la entrada. Permitimos que haya varias vistas (o interfaces) para un mismo modelo. Concretamente, hacemos que el modelo (o implementación) solo referencie a la vista a través de un interfaz con la funcionalidad de actualizar.

4.2. DISEÑO DE LAS APLICACIONES

A continuación, pasamos a analizar el diseño de cada aplicación. En cada uno de los tres casos, presentamos una descomposición del modelo de la aplicación en subsistemas, sus interfaces y las dependencias entre ellos. Usaremos como notación el Lenguaje Unificado de Modelado (UML) por Booch, Rumbaugh, y Jacobson.

4.2.1. APLICACIÓN USUARIO

La aplicación debe permitir al usuario crear tareas y especificar sus restricciones temporales y de recursos. Pero, no olvidemos que esta aplicación forma parte de un sistema formado por otras dos aplicaciones, luego, a estas funcionalidades propias debemos añadir la correspondiente de las comunicaciones entre los módulos.
Ver figura 1.

La aplicación esta constituida por cuatro módulos fundamentales:

i. *Módulo de factorías*, el cual nos proporciona una fábrica de componentes y estructuras de datos.

ii. *Módulo de tareas*, que recoge la creación y gestión de tareas del usuario.

iii. *Módulo de protocolo*, que encapsula la estructura de los mensajes del protocolo de comunicaciones del sistema, construyéndolos o interpretándolos.

iv. *Módulo de gestión de comunicaciones,* controla y gestiona el envío de información entre aplicaciones con un hilo estable o TCP (en este caso, RecepcionClientes, con la aplicación Cliente).

Dentro de cada módulo existe una alta cohesión y entre ellos se ha buscado el mínimo acoplamiento.

El punto de acceso a toda la aplicación o *fachada*, es señalado como interfaz IUsuario. Ésta además, actúa como *mediador* entre las tareas que éste desea ejecutar (ITareas) y el protocolo de comunicación correspondiente (IProtocoloUsuario) de manera que, si se desea cambiar la estructura de los mensajes del protocolo sólo es necesaria la nueva implementación correspondiente del interfaz IProtocoloUsuario y el resto de clases se pueden mantener con las mínimas modificaciones, o incluso, sin ellas.

Diagrama UML:

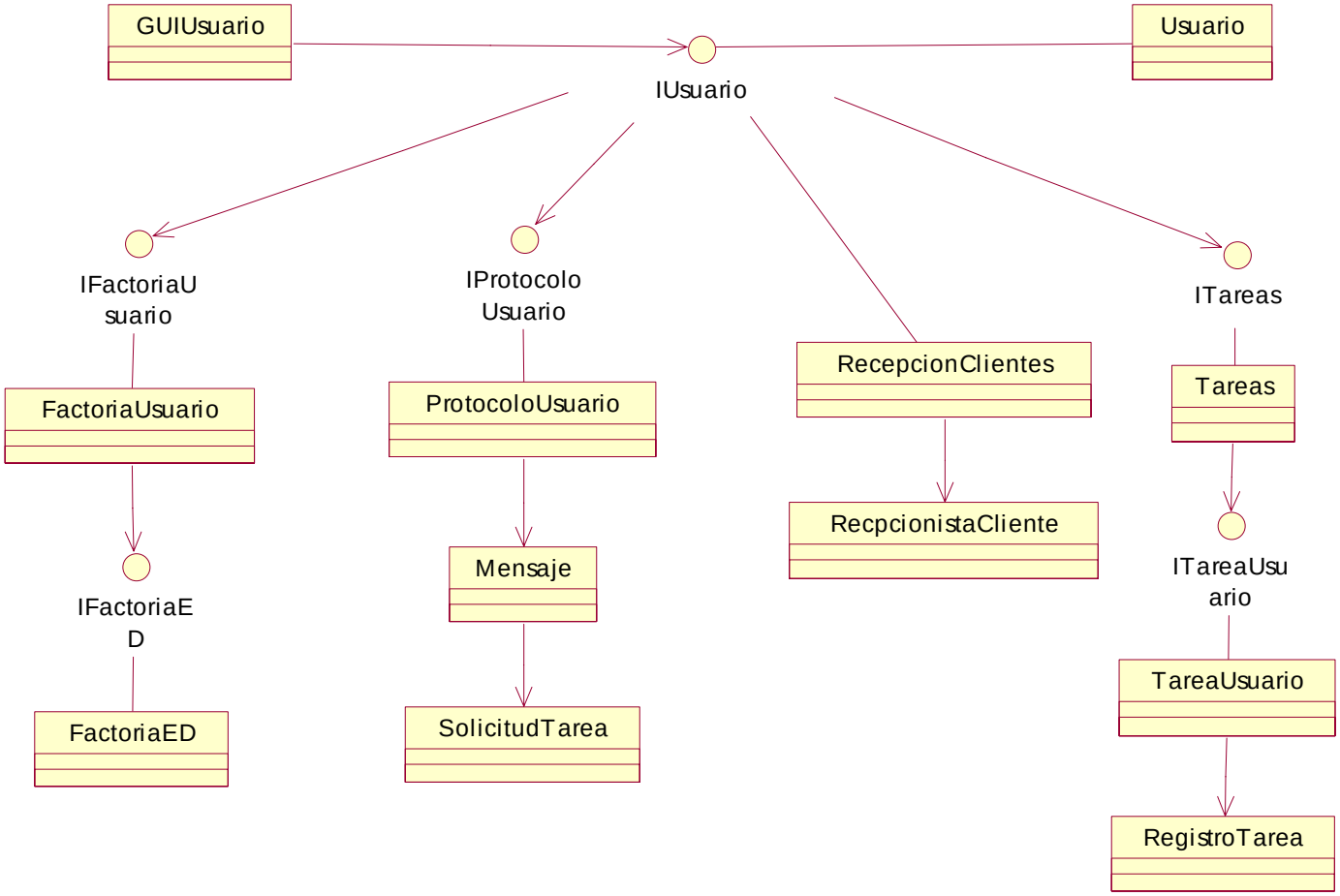


Figura 1. Diagrama de clases de la aplicación Usuario.

4.2.2. APLICACIÓN CLIENTE

La aplicación Cliente es la que se ejecutará en la máquina que ofrece al servidor sus recursos. Por tanto, por un lado, debe gestionar los recursos del cliente, con altas, bajas y modificaciones, además de la asignación de tareas a cada uno de ellos. Por otro lado, proporcionar una implementación para el protocolo de comunicaciones.
Ver figura 2.

La aplicación Cliente se estructura en cuatro módulos fundamentales:

i. *Módulo de factorías*, el cual, al igual que en el caso anterior, nos proporciona una fábrica de componentes y estructuras de datos.

ii. *Módulo de recursos*, que gestiona las FPGAs que formarán parte del sistema. A su vez, cada FPGA o recurso, gestiona sus particiones y cada partición, gestiona las tareas que se

ejecutan en ella. Delegando, quedan bien definidas las responsabilidades de cada subsistema de éste módulo. Accedemos a él a través de la fachada (interfaz) IRecursos.

iii. *Módulo de protocolo*, que como en la aplicación Usuario, configura e interpreta los mensajes del protocolo de comunicaciones. En este caso, el cliente tiene más variedad de tipos de mensajes para, además de gestionar tareas, gestionar recursos y mantener coherencia con el servidor. El acceso al módulo se realiza a través de la interfaz IProtocoloCliente.

iv. *Módulo de gestión de comunicaciones*, esta vez formado por dos subsistemas: el encargado de gestionar la comunicación cliente-servidor y el de la comunicación cliente-usuario. Ambos hilos de comunicaciones son estables (TCP).

La fachada de la aplicación es ICliente. Actúa como mediador entre el módulo de protocolo, que construye los mensajes, y el módulo de recursos, ya que los datos que necesita de los recursos, particiones o tareas no le llegan directamente de los recursos, sino desde una clase intermedia.

Al igual que en la aplicación Usuario, podemos realizar cualquier otra implementación del protocolo si se desea cambiar el diálogo entre las partes y sin necesidad de modificar la gestión interna de los recursos del cliente accesible a través del interfaz ICliente.

También, se puede realizar una extensión para que, un usuario conectado directamente a un cliente (sin mediación del servidor) pueda ejecutar una tarea.

Diagrama UML:

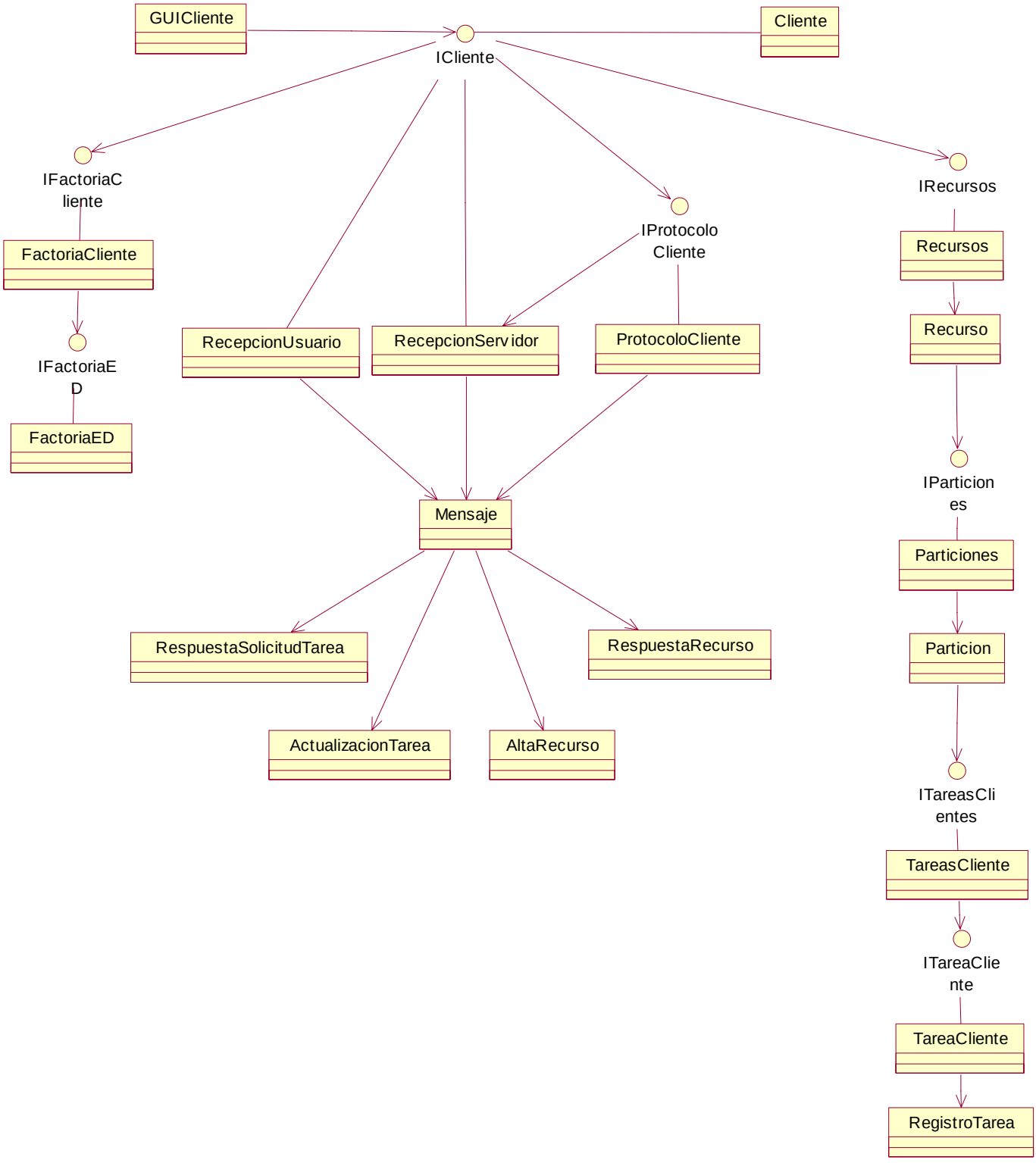


Figura 2. Diagrama de clases de la aplicación Cliente.

4.2.3. APLICACIÓN SERVIDOR

La aplicación Servidor debe llevar cuenta de:

1. Clientes que ofrezcan sus servicios.
2. Usuarios que soliciten esos recursos.

Por su parte, cada cliente y cada usuario es responsable de mostrar la información correspondiente a sus recursos, tareas, etc., de manera que el servidor no necesita conocer qué tipo concreto de recurso está tratando, sino que lo ve de forma genérica o polimórfica.
Además, el servidor posee una unidad independiente para el establecimiento del criterio de selección de recursos y otra para el protocolo de comunicación.
Ver figura 3.

La aplicación, se divide en los siguientes módulos:

i. *Módulo de factorías*, el cual, al igual que en el resto de las aplicaciones, este módulo nos proporciona una fábrica de componentes y estructuras de datos.

ii. *Módulo de clientes*, gestiona los clientes que forman parte del sistema con sus respectivos recursos.

iii. *Módulo de usuarios*, gestiona las peticiones de tareas que realizan los usuarios cuando se conectan al sistema.

iv. *Módulo de protocolo*, como en lo casos anteriores, este módulo se encarga de encapsular la estructura común de los mensajes del protocolo de comunicaciones.

v. *Módulo de gestión de comunicaciones*, formado por dos subsistemas independientes (dos hilos) que gestionan el envío de mensajes entre servidor-cliente y servidor-usuario.

vi. *Módulo Criterio de Selección*, éste módulo es exclusivo de la aplicación servidor. Su función es seleccionar el recurso y partición al que enviar una tarea, que la ejecute de la forma más eficiente posible. Lo que queramos decir con “eficiente”, queda determinado por el algoritmo concreto que define el criterio de selección de un recurso u otro. Éste módulo podría haberse simplificado en una función, algo compleja; pero tomamos esta decisión de diseño para permitir ampliaciones y pruebas de nuevas algoritmos más fácilmente.

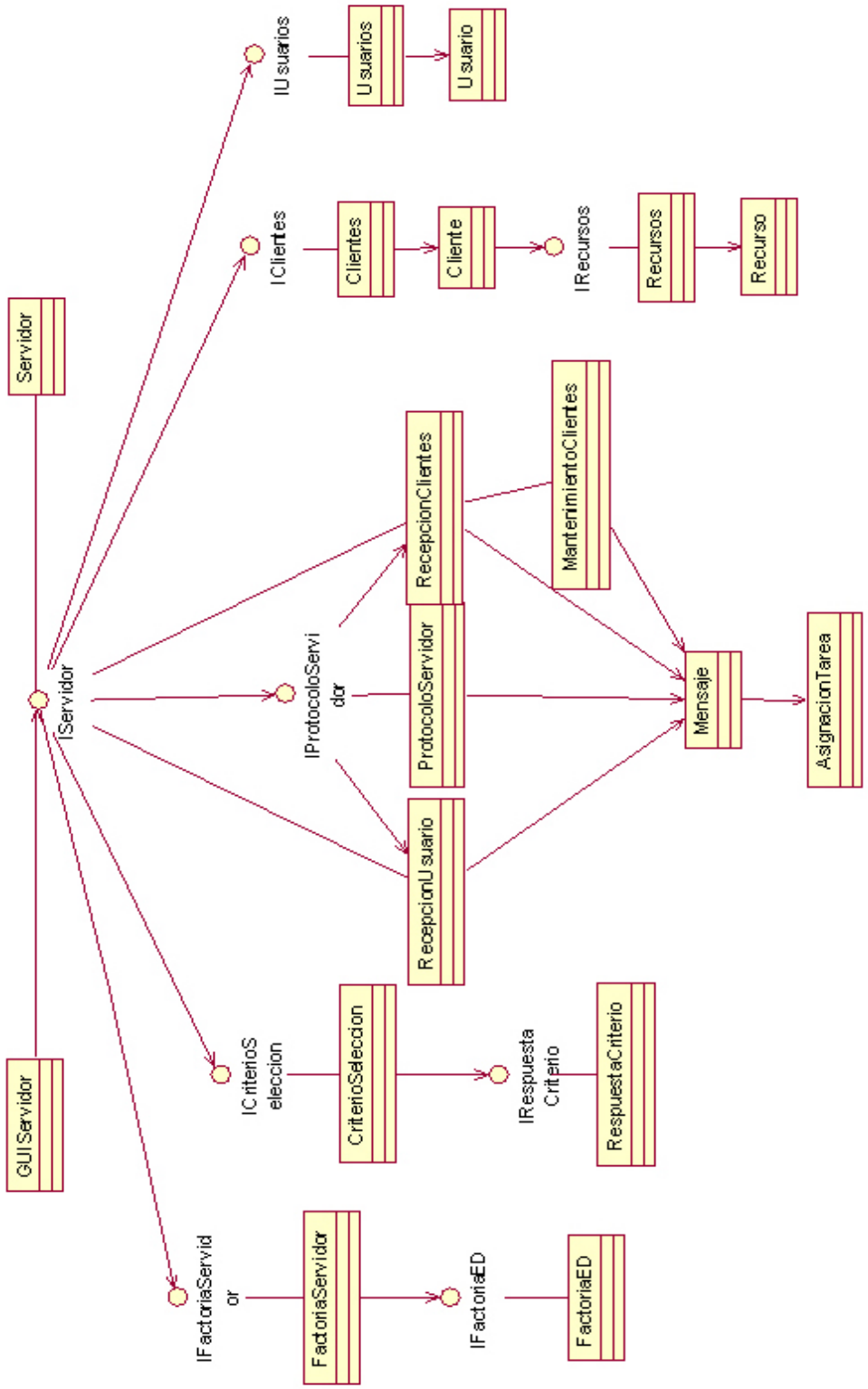


Figura 3. Diagrama de clases de la aplicación Servidor.

4.3. PROTOCOLO DE COMUNICACIONES

En esta sección se discuten los sockets establecidos por el sistema, su naturaleza, su propósito y el tipo de mensajes que se inyectan en ellos. Desde el punto de vista del diseño todas estas cuestiones quedan encapsuladas en las clases Protocolo mencionadas en el apartado Diseño, lo que favorece la escalabilidad del sistema. Por último queremos destacar el concepto de protocolo como uno de los pilares del paradigma Grid Computing, con el que se pretende conseguir la mayor interoperabilidad posible entre los actores del sistema.

4.3.1. TIPOS DE SOCKETS Y SU ESTABLECIMIENTO

Nos referimos en esta sección al apartado de funcionamiento, donde se presentaban las conexiones de la aplicación y su naturaleza. Resumidamente estas eran: un socket TCP “volátil” (es decir, que no existe durante toda la vida del sistema) entre cada usuario y el servidor; dos sockets TCP “fijos” (es decir, que permanecen durante toda la vida del cliente) entre cada cliente y el servidor, uno de ellos para labores de mantenimiento y otro principal; y un socket TCP “volátil” entre cliente y usuario cuando el sistema lo requiera, como veremos más adelante.

El **socket entre el usuario y el servidor** se establece cada vez que el usuario quiere hacer una comunicación al servidor, en general, por dos motivos: el usuario quiere darse de alta o de baja en el sistema, el usuario quiere solicitar que el sistema ejecute una tarea. Esta conexión se mantiene hasta que el servidor responda a la petición del usuario; y, en el caso de la ejecución de una tarea, cuando el sistema asigne la tarea a un cliente o cuando compruebe que no es posible asignarla a ninguno de los clientes disponibles. En todas estas conexiones el rol del usuario es el del cliente de un socket cliente-servidor.

El **socket principal entre el cliente y el servidor** se utiliza principalmente para responder a las encuestas del servidor con los parámetros sobre si es posible ejecutar una tarea concreta y en qué condiciones. El rol de la aplicación cliente es el de un servidor en un socket cliente-servidor, ya que es la aplicación cliente la que se bloquea en lectura del socket para atender las peticiones del servidor. Este socket se establece al conectar el cliente al sistema, y permanece, si no hay errores fatales, hasta que el cliente se desconecte.

El **socket secundario entre el cliente y el servidor** se utiliza las solicitudes y mensajes de mantenimiento que el cliente tenga que realizar al servidor. Esto quiere decir que en este socket es la aplicación servidor la que se bloquea en lectura. Los mensajes típicos que enviará el cliente serán tales como avisos sobre el cambio de estado de una tarea o avisos sobre la configuración de los recursos del sistema (al darse de alta) . Este socket se establece al conectar el cliente al sistema, y permanece, si no hay errores fatales, hasta que el cliente se desconecte.

El **socket entre el cliente y el usuario** se establece cuando el cliente recibe la asignación de la tarea del usuario. Mediante este socket se transmite el fichero .bit para la ejecución de la tarea y se comunica el resultado al usuario de la misma cuando esta termine. Una vez se ha comunicado el resultado el socket se destruye.

4.3.2. TIPOS DE MENSAJES

Todos los mensajes que circulan por la aplicación son de tipo Mensaje. Como se vio en el diseño, este tipo tiene tres atributos:

- La identificación del remitente (por su dirección IP).
- Un código de tipo entero con el tipo de mensaje que se envía.
- Un atributo del tipo Object (el genérico para las clases Java) que contiene el mensaje en si, la información necesaria, o null si no es necesario nada más (como en el caso de algunos ACK).

Los códigos de los mensajes están declarados en los interfaces asociados a cada protocolo (IProtocoloCliente, IProtocoloUsuario e IProtocoloServidor). Esta es su declaración:

```
//------------------------------------------------------------------------
// TIPOS DE MENSAJE QUE PUEDE GESTIONAR EL PROTOCOLO DEL USUARIO
//------------------------------------------------------------------------

/**
 * El usuario quiere que el servidor le dé de alta en su base de datos de
 * usuarios (es decir, le reconozca como usuario permitido)
 */
public final static int SOLICITUD_ALTA = 0;

/**
 * Confirmación o rechazo de un mensaje de tipo MENSAJE_ALTA
 */
public final static int RESPUESTA_ALTA = 1;

/**
 * El usuario quiere darse de baja de la base de datos de usuarios del
 * servidor
 */
public final static int SOLICITUD_BAJA = 2;

/**
 * Confirmación o rechazo de un mensaje de tipo MENSAJE_ALTA
 */
public final static int RESPUESTA_BAJA = 3;

/**
```

```
 * El usuario solicita al servidor la ejecución de una tarea
 */
public final static int SOLICITUD_TAREA = 4;

/**
 * Confirmación o rechazo de un mensaje de tipo SOLICITUD_TAREA
 */
public final static int RESPUESTA_TAREA = 5;

/**
 * Mensaje de datos, conteniendo parte del mapa de bits
 */
public final static int DATOS_BIT = 6;

/**
 * Mensaje de datos, conteniendo la longitud del último trozo del mapa de
 * bits
 */
public final static int FIN_DATOS_BIT = 7;

/**
 * Mensaje de datos, resultado de la ejecución del mapa de bits
 */
public final static int RESULTADO_EJECUCION = 8;

/**
 * Mensaje de datos, conteniendo el nombre de la tarea que contiene el
 * mapa de bits
 */
public final static int NOMBRE_TAREA_DATOS_BIT = 9;

/**
 * Mensaje de datos, conteniendo parte del fichero de datos asociado al
 * mapa de bits
 */
public final static int DATOS_DATABIT = 10;

/**
 * Mensaje de datos, conteniendo la longitud del último trozo del fichero
 * de datos asociado al mapa de bits
 */
public final static int FIN_DATOS_DATABIT = 11;

/**
 * Mensaje de datos, conteniendo parte del fichero de resultado
 */
public final static int DATOS_RESULTADO = 12;

/**
 * Mensaje de datos, conteniendo la longitud del último trozo del fichero
```

```
 * resultado de la ejecución
 */
public final static int FIN_DATOS_RESULTADO = 13;



//---------------------------------------------------------------------
// TIPOS DE MENSAJE QE PUEDE GESTIONAR EL PROTOCOLO DEL CLIENTE
//---------------------------------------------------------------------

/**
 * El cliente solicita al servidor darse de alta
 */
public final static int SOLICITUD_ALTA_CLIENTE = 1000;

/**
 * El cliente solicita al servidor darse de alta
 */
public final static int RESPUESTA_ALTA_CLIENTE = 1001;

/**
 * El cliente solicita al servidor darse de baja
 */
public final static int SOLICITUD_BAJA_CLIENTE = 1002;

/**
 * El cliente solicita al servidor darse de baja
 */
public final static int RESPUESTA_BAJA_CLIENTE = 1003;

/**
 * El servidor solicita al cliente la ejecución de una tarea
 */
public final static int SOLICITUD_TAREA = 1004;

/**
 * Respuesta a una solicitud de tarea (encuesta del servidor)
 */
public final static int RESPUESTA_SOLICITUD = 1005;

/**
 * Asignación de una tarea a un recurso en concreto
 */
public final static int ASIGNACION_TAREA = 1006;

/**
 * El cliente envía al servidor una actualización de estado de una tarea
 */
public final static int ACTUALIZACION_TAREA = 1007;
```

4.4. CONTROL DE ERRORES

En todo programa se producen errores. Y en la ejecución de un programa pueden aparecer situaciones en las que su comportamiento no es el habitual y el que cabe esperar.

Para tratar con estas situaciones indeseables, el sistema posee una serie de medidas de protección, entre las que se incluyen:

- **Prevención**: medidas para evitar el daño. Estas medidas preventivas incluyen desde el bloqueo de acciones realizables por el usuario hasta la notificación a los mismos de las consecuencias que ciertas operaciones pueden ocasionar en un futuro.
- **Detección**: medidas para detectar cuándo se ha dañado, alterado o puesto en riesgo el funcionamiento del sistema. El objetivo principal de las medidas de detección es determinar de qué manera se ha puesto en peligro el sistema, y específicamente, quién o qué fue el causante del daño.
- **Reacción**: medidas que permiten recuperar una situación de normalidad una vez el daño ya se ha producido.

Nos ocuparemos principalmente de las medidas de prevención perceptibles por el usuario humano y de las medidas de detección y reacción, puesto que son a las que los usuarios humanos tendrán que enfrentarse en caso de producirse.

Básicamente, el ciclo de actuación conjunto de las medidas de Prevención, Detección y reacción es el siguiente:

- identificar las situaciones de riesgo y sus causantes (prevención y detección)
- realizar una evaluación cualitativa de tales situaciones (detección)
- informar al usuario del tipo de error, magnitud y descripción del mismo (detección)
- intentar paliar los efectos de tales situaciones (reacción)

4.4.1. TIPOS DE ERRORES (según su magnitud)

Los errores (o situaciones de error) propios de nuestro sistema los hemos clasificado en tres grandes grupos, de menor a mayor peligrosidad: avisos, errores y errores fatales. Los avisos y errores suelen ser provocados por los usuarios del sistema de manera no intencionada, mientras que los errores fatales son provocados por personas malintencionadas o por circunstancias totalmente ajenas al sistema y a los usuarios que lo utilizan.

- **Avisos**

Los avisos o “warnings” no llegan a ser situaciones de error estrictamente hablando. Sin embargo, estas situaciones desembocarán en error a menos que se haga algo para evitarlo. Así que, es el usuario el que tiene la última palabra en estas situaciones, y por consiguiente la responsabilidad de poder terminar con la situación peligrosa.

Los avisos se identifican por el símbolo:

- **Errores**

Los errores (o errores ordinarios) son situaciones de error recuperables. Esto significa que la aplicación se mantiene en pie y no sufre ningún daño estructural. Lo que sí sucede es que la acción que desencadenó el error no se llega a completar. En caso de que la ejecución de dicha acción sea imprescindible para el usuario de la aplicación, éste debería encontrar la causa del error (a partir de la información que el sistema le da del mismo) y subsanarlo, para así poder ejecutar esa acción correctamente.

Generalmente, estos errores se deben a la imposibilidad de establecer comunicación entre las aplicaciones o a problemas encontrados en el análisis de archivos de configuración.

Estas situaciones vienen identificadas por el símbolo:

- **Errores fatales**

Los errores fatales son situaciones de error irrecuperables, lo que significa que la aplicación en la que se produce este error finaliza bruscamente, y lo que es más importante, puede provocar la "caída" (finalización abrupta) del resto de aplicaciones con las que está conectada, causando un "efecto dominó".

Una vez se ha producido un error fatal, ya no existe solución posible al mismo. Además, la ocurrencia de estos errores no garantiza en forma alguna el estado en el que quedará la aplicación ni los datos que se estaban manejando en el momento de la caída. La mejor forma de enfrentarse a estos errores es volver a lanzar todas las aplicaciones que directa o indirectamente se cerraron a causa del error fatal, volviendo además a efectuar las operaciones que estaban en curso en el momento del error.

Como ya dijimos, estas situaciones pueden estar provocadas por agentes o eventos externos al sistema, como la desconexión de la red. Y también pueden estar provocadas por personas malintencionadas, como las que desobedeciendo las recomendaciones que la aplicación proporciona frente a una situación de aviso, conducen a la aplicación a una situación de error fatal.

Las situaciones de error fatal vienen identificadas por el mismo símbolo que los errores ordinarios

y se distinguen de aquellos porque los errores fatales provocan la finalización incondicional de la aplicación.

4.4.2. EJEMPLOS DE SITUACIONES DE ERROR

Por supuesto, estas situaciones de error son diferentes según la aplicación considerada. A continuación mostramos una tabla con las principales situaciones de error en cada una de las aplicaciones.

Situaciones de error en la Aplicación Usuario

Tipo	**Ejemplos**
Aviso	- Cierre de la aplicación sin haber efectuado la desconexión del servidor - Intento de eliminar una tarea solicitada y aceptada pero no finalizada - Intento de desconexión del servidor cuando hay tareas no finalizadas - Warning producido en el análisis de algún fichero de configuración
Error	- Imposibilidad de establecer conexión con el servidor - Creación, apertura, lectura, escritura o cierre de un archivo de mapa de bits, de datos asociado o de resultado - Definición de una tarea con parámetros incorrectos - Intento de solicitud de tarea o de desconexión al estar falsamente conectado (la Aplicación Usuario cree estar conectada al servidor sin estarlo realmente) - Error producido en el análisis de algún fichero de configuración
Error fatal	- Fallo o desconexión en la red - Caída del servidor cuando se estaba conectado a él - Intento de escucha por el puerto asignado a los clientes, cuando está siendo ocupado por otra aplicación - Caída de cliente con el que se estaba conectado

Situaciones de error en la Aplicación Cliente

Tipo	**Ejemplos**
Aviso	- Cierre de la aplicación sin haber efectuado la desconexión del servidor - Intento de desconexión del servidor cuando hay tareas en ejecución - Eliminación de un recurso - Warning producido en el análisis de algún fichero de configuración
Error	- Imposibilidad de establecer conexión con el servidor - Creación, apertura, lectura, escritura o cierre de un archivo de mapa de bits, de datos asociado o de resultado

	- Intento de desconexión al estar falsamente conectado - Intento de eliminar recurso estando conectado - Error producido en el análisis de algún fichero de configuración
Error fatal	- Fallo o desconexión en la red - Caída del servidor cuando se estaba conectado a él - Caída de usuario con el que se estaba conectado

Situaciones de error en la Aplicación Servidor

Tipo	**Ejemplos**
Aviso	- Cierre de la aplicación cuando hay usuarios y/o clientes conectados
Error	
Error fatal	- Fallo o desconexión en la red - Fallo de comunicación con un cliente (al intentar establecer comunicación, enviar o recibir mensaje o finalizar comunicación) - Fallo de comunicación con un usuario (al intentar establecer comunicación, enviar o recibir mensaje o finalizar comunicación) - Intento de escucha por el puerto asignado a los usuarios, cuando están siendo ocupados por otra aplicación - Intento de escucha por el puerto asignado a los clientes, cuando están siendo ocupados por otra aplicación

4.4.3. OTROS ERRORES NO CONTEMPLADOS POR EL SISTEMA

Aparte de las situaciones consideradas anteriormente, existen muchas otras situaciones que nuestro sistema es incapaz de controlar, como fenómenos naturales, incidentes catastróficos que puedan causar un daño severo del hardware, cortes de luz, deterioros en el sistema operativo sobre el que se monta el sistema, etc.

4.4.4. PUNTOS DÉBILES DEL SISTEMA

A partir de los ejemplos de errores mostrados es las tablas, podemos deducir que el punto débil del sistema es la Aplicación Servidor. Es decir, en general, un error fatal acaecido en una instancia de la Aplicación Cliente o de la Aplicación Usuario provoca solamente la finalización de la misma, mientras que si éste sucede en la Aplicación Servidor, provocará un efecto dominó y la caída en cadena de todas las demás aplicaciones con las que se encuentre conectado en el momento del fallo.

Observamos que en la tabla de situaciones de error asociada a la Aplicación Servidor no aparece ningún ejemplo de error ordinario, lo cuál no significa que no exista ningún error de este

tipo. Lo que sucede es que este tipo de errores, a diferencia de lo que ocurre en las otras aplicaciones, no son subsanables por el operador humano que interactúa con la Aplicación Servidor, puesto que como ya sabemos, la Aplicación Servidor es autónoma y el operador humano de la misma se limita a observarla. Algunos errores ordinarios que pueden aparecer en la Aplicación Servidor son:

- Error producido en el análisis de algún fichero de configuración
- Alta de un usuario o cliente que ya se encontraba dado de alta
- Baja de un usuario o cliente que no se encontraba dado de alta

5. IMPLEMENTACIÓN

5.1. APLICACIÓN USUARIO

i. *Módulo de Protocolo*: su clase principal es *ProtocoloUsuario*, con su respectivo interfaz *IProtocoloUsuario*. Aquí, se definen, como constantes, los tipos de mensajes válidos para la comunicación con el servidor y con los clientes. Realiza la función de construcción de los mensajes como alta/baja de usuario y solicitud de ejecución de una tarea. Cuando el usuario establecer una comunicación (invocando algún método de la clase Usuario), ésta delega en el protocolo para que, con los parámetros necesarios que le pase la clase Usuario, pueda construir el mensaje y resolver el problema. Además de crear el mensaje, el Protocolo envía y recibe la respuesta, efectuando por tanto funciones de comunicación, como veremos a continuación.

ii. *Módulo de gestión de comunicaciones*: Está formado por dos subsistemas: el encargado de las comunicaciones Usuario-Cliente y el encargado de las comunicaciones Usuario-Servidor. Sus clases principales son *RecepcionistaClientes* y *ProtocoloUsuario* respectivamente. La primera clase establece una comunicación ex proceso (síncrona) para el envío de los ficheros de mapa de bits y de datos asociados de usuario a cliente y el fichero de resultado de cliente a usuario. La segunda es la que se encarga de enviar peticiones al servidor y recibir la respuesta de las mismas. Por lo tanto, este módulo encapsula toda la parte lógica de las comunicaciones, es decir, la creación y el manejo de los *sockets* para cada diálogo.

La interacción con las otra clases ocurre del siguiente modo: la clase Usuario, que implementa el interfaz o fachada IUsuario, actúa como mediador al proporcionar a la clase Protocolo los atributos de este módulo que necesite. Así, una vez que el Protocolo constituye un mensaje, lo envía y espera a recibir la respuesta.

iii. *Módulo de tareas*: El acceso a este módulo se realiza a través del interfaz o fachada *ITareas*. Ofrece las funciones típicas de añadir una tarea, buscar una tarea y eliminar una tarea. Por lo tanto, cuenta con una lista de objetos de tipo *TareaUsuario*.

iv. *Módulo de factorías abstractas*: su función es la de fábrica de componentes. El interfaz fachada es *IFactoriaUsuario*. Ésta nos proporciona, usando el patrón “Singleton”, una sola instancia de cada fachada comentada en los módulos anteriores, es decir:

a. *IProtocoloUsuario* del módulo protocolo;
b. *ITareas* del módulo de tareas.
c. *IFactoriaED* que es otra factoría que proporciona estructuras de datos (EDs). Para nuestro sistema solo hemos especificado estructuras del tipo de interfaces *List* y *Map*, definidas en Java.

Con respecto a la interfaz gráfica, ésta ha sido desarrollada de una forma muy modular y aplicando el patrón "Modelo-Vista-Controlador" (MVC) de forma simplificada adaptándose a nuestra necesidades.

La Interfaz Gráfica de Usuario (GUI) tiene como clase principal *FramePrincipalGUI* y es el frame o ventana principal sobre el que se establecen todos los demás componentes.

Por último, la clase *AplicacionUsuario* es el origen de toda la aplicación, y creará un Usuario con los siguientes elementos:

i. *Lógica*, que recibe la factoría abstracta en el momento de ser creada;
ii. *Interfaz grafica*, que se encargará de visualizar todo lo que se considere relevante;

5.2. APLICACIÓN SERVIDOR

i. *Módulo de Protocolo*: su clase principal es *ProtocoloServidor*, con su respectivo interfaz *IProtocoloServidor*. Aquí, se definen, como constantes, los tipos de mensajes válidos para la comunicación con clientes y usuarios. Realiza la función de intérprete de los mensajes recibidos como alta/baja de usuario, alta/baja de cliente, solicitud de ejecución de una tarea (por parte de un usuario) y de mensajes de actualización del estado de una tarea (por parte del cliente). Cuando a la clase Servidor le llega un mensaje, éste se delega en el protocolo para que, con los parámetros que le pase además el servidor, que sabe que necesitará, pueda desenmarañar el mensaje y resolver el problema. Si lo necesita, informará al Servidor de lo que tiene que hacer. También se pondrá en contacto con las otras aplicaciones, construyendo los mensajes necesarios. Utiliza la clase *Mensaje*, cuyo contenido es otro objeto que, dependiendo del fin de mensaje será uno u otro tipo. Así, tratamos los mensajes en todo el sistema de forma polimórfica.

ii. *Módulo de gestión de comunicaciones*: Está formado por dos subsistemas: el encargado de las comunicaciones Servidor-Cliente y el de Servidor-Usuario. Sus clases principales son *RecepcionClientes* y *RecepcionUsuarios* respectivamente. Ambas clases se limitan a esperar el posible envío de mensajes por parte de clientes o usuarios (los cuales reenvían directamente al Servidor), y a enviar mensajes ya construidos. Están constituidas como hilos y cada una escucha por puertos diferentes. Por lo tanto, este módulo encapsula toda la parte lógica de las comunicaciones, es decir, la creación y el manejo de los *sockets* para cada diálogo.

La interacción con las otra clases ocurre del siguiente modo: la clase servidor, que implementa el interfaz o fachada IServidor, actúa como mediador al proporcionar a la clase Protocolo las clases de este módulo que necesite. Así, una vez que el Protocolo constituye un mensaje, delega en la clase correspondiente para que lo envíe.

iii. *Módulo de clientes*: El acceso a este módulo se realiza a través del interfaz o fachada *IClientes*. Ofrece las funciones típicas de dar de alta/baja a un cliente, búsqueda de un

cliente y devolver como salida un cliente seleccionado. Por lo tanto, cuenta con una lista de objetos del tipo *Cliente*. A su vez, cada cliente tiene un atributo del tipo *Recursos*, que una lista de objetos del tipo *Recurso*, y que realiza también las funciones de alta/baja de recurso y de búsquedas.

iv. *Módulo de usuarios*: Es el análogo al módulo anterior. Su punto de acceso es el interfaz *IUsuarios*. Gestiona el alta/baja de usuarios al sistema y realiza las correspondientes búsquedas. De forma similar a los clientes, cada usuario, tendrá una lista de tareas asociadas, cada una alguno de los estados que hemos definido: enviada (el servidor ha recibido la solicitud), aceptada (ya tiene un cliente, recurso y partición asignada para su ejecución), en ejecución o finalizada.

v. *Módulo Criterio de Selección*: Este módulo está constituido por una sola clase: *CriterioSeleccion*. Pretende ser una biblioteca de algoritmos que nos seleccionen el cliente, recurso y partición que puedan ejecutar una tarea de la forma más eficiente. Nosotros consideramos inicialmente dos posibles criterios:

a. *Mínimo tiempo*: selecciona la opción en la que se ejecutará más rápido la tarea.
b. *Mínimo espacio desperdiciado*: selecciona la opción que menos desaproveche el espacio del recurso.

Los parámetros de cada recurso que utiliza para tomar la decisión, son del tipo *RespuestaSolicitudTarea* que recogerán el tiempo de espera para ejecución de la tarea y el porcentaje de recurso desperdiciado, según el caso.

vi. *Módulo de factorías abstractas*: su función es la de fábrica de componentes. El interfaz fachada es *IFactoriaServidor*. Ésta nos proporciona, usando el patrón Singleton, una sola instancia de cada fachada comentada en los módulos anteriores, es decir:

a. *IProtocoloServidor* del módulo protocolo;
b. *IClientes* del módulo de los clientes.
c. *IUsuarios* del módulo de los usuarios.
d. *IFactoriaED* que es otra factoría que proporciona estructuras de datos. Para nuestro sistema solo hemos especificado estructuras del tipo de interfaces *List* y *Map*, definidas en Java.

Con respecto a la interfaz gráfica, ésta ha sido desarrollada de una forma muy modular y aplicando el patrón *Modelo-Vista-Controlador* (MVC) de forma simplificada adaptándose a nuestra necesidades. A continuación detallamos las características más generales:

La *Interfaz Gráfica de Usuario* (GUI) es la clase principal, es el panel básico sobre el que establecen los demás. Ésta clase es el origen primero de toda la aplicación: creará un Servidor con los siguientes parámetros:

i. *IFactoríaServidor*, que recibe la GUI en el momento de ser creada con una implementación determinada;

ii. *Interfaz grafica del usuario*, la que se encargará de mostrar los datos del módulo de usuarios;
iii. *Interfaz gráfica de clientes*, mostrará los datos de los clientes, sus recursos y tareas;
iv. *Interfaz gráfica de fichero de log*, que permitirá mostrar toda la información detallada sobre los procesos que ocurren en la aplicación.

El sistema está diseñado de tal forma que permite que exista una *factoría de interfaces gráficas*, de forma que la GUI principal sabe qué tipo de interfaz es pero no conoce su implementación o vista.

Cada interfaz que muestra la información de cada módulo, es decir, usuarios, clientes y log, implementan un interfaz siguiendo el patrón MVC. Este interfaz es *IActualizaGUI* y solo tiene un método: *actualiza* que recibe un código, que indica qué cambio se ha producido en la lógica, y el objeto modificado que debe sustituir para mostrar la actualización.

5.3. APLICACIÓN CLIENTE

En la aplicación se implementaron dos paquetes: aplicacioncliente con la lógica de la aplicación e interfazcliente con la interfaz gráfica. Dentro del primero distinguimos los siguientes módulos:

i. *Módulo de recurso.* Controla la simulación de cada recurso. Cabe destacar el interfaz IRecurso, que modela un recurso y el interfaz IParticiones, que modela un conjunto de particiones en el que está dividido el recurso.
ii. *Módulo de comunicaciones.* Controla el establecimiento y destrucción de sockets, así como los mensajes que se inyectan o reciben de los mismos. Debe atender, por una parte los dos sockets con el servidor (uno principal y otro de mantenimiento) y además un socket con cada usuario que sea necesario. Para ello debe tener hilos activos para cada socket, de manera que son necesarias clases que extiendan la clase Java Thread para estos deberes. Se han creado por lo tanto tres tipos de clases Recepcion diferentes: RecepcionServidor, RecepcionServidorMan y RecepcionUsuario.

Como las demás aplicaciones, esta cuenta con una factoría abstracta de datos que le proporciona las implementaciones necesarias de estructuras de datos.

Con respecto al paquete interfazcliente, que contiene la interfaz gráfica, destacamos la clase principal, GUIPrincipalCliente que contiene todos los elementos de la interfaz gráfica. Los cambios se comunican a esta interfaz siguiendo el patrón Observer, con un interfaz propio, IGUIObservadora, que implementan los objetos que quieran comunicar cambios a la interfaz

gráfica (por ejemplo las colas de tareas de las particiones, para comunicar los cambios en los estados de las tareas que va ejecutando).

6. MANUAL DE USUARIO

6.1. REQUISITOS DEL SISTEMA

Consideramos como requisitos las plataformas hardware en las que se ha probado la aplicación, además de las versiones del JRE Entorno de ejecución de Java) que existían en dichas plataformas. Se desconoce lo que ocurriría en otros casos.

6.1.1. REQUISITOS HARDWARE

AplicacionUsuario1.0, *AplicacionServidor1.0* y *AplicaciónCliente1.0* se soporta sobre Intel Pentium II, Intel Pentium III e Intel Pentium 4.

Se recomienda disponer de un mínimo de 20 MBytes de disco duro para su instalación, así como 256 MBytes de RAM.

6.1.2. REQUISITOS SOFTWARE

Se recomienda disponer de la versión 1.4.2 de Java 2 JRE. Cualquier actualización es válida (1.4.2_01, 1.4.2_03, 1.4.2_05, etc.).

A la hora de ejecutar las aplicaciones, se observará un deterioro importante del rendimiento si las tres se ejecutan sobre el mismo equipo.

6.2. INSTALACIÓN DE LAS APLICACIONES

6.2.1. SOBRE UNA PLATAFORMA Windows

A continuación, guiaremos el proceso de instalación del gestor de recursos de hardware reconfigurable sobre un sistema operativo Windows 2000, Windows 2003 y Windows XP. La instalación incluye la Aplicación Cliente, la Aplicación Usuario y la Aplicación Servidor.

1) Comprobar el tamaño de los archivos descargados (opcional)

Es recomendable comprobar que los archivos *AplicacionUsuario1.0.exe*, *AplicacionCliente1.0.exe* y *AplicacionServidor1.0.exe* se han descargado completamente y no están corruptos.

2) Comprobar que se dispone de las herramientas necesarias

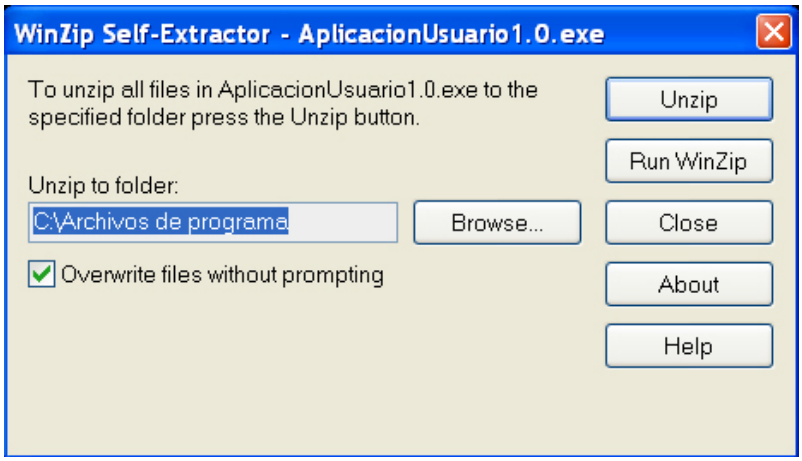
Para instalar la Aplicación Usuario es necesario disponer de la herramienta de compresión y descompresión **WinZip®**. Ésta está disponible a través de la web www.winzip.com.

A su vez se necesita tener instalado el Entorno de Ejecución de Java (Java Runtime Environment o JRE) disponible a través de la web http://java.sun.com/.

3) Instalar, ejecutar y desinstalar la Aplicación Usuario

3.1) Ejecutar el instalador de la Aplicación Usuario

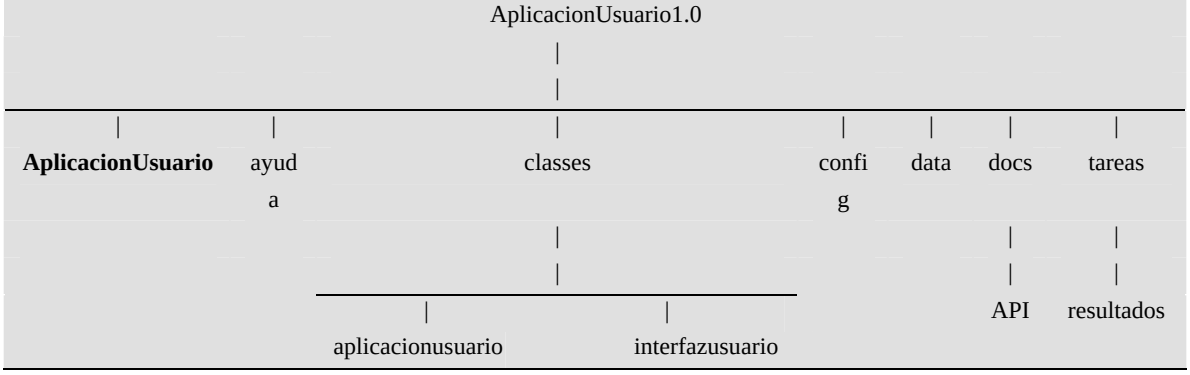
El archivo *AplicacionUsuario1.0.exe* es el instalador de la Aplicación Usuario. Debemos hacer doble clic sobre él, apareciendo entonces el siguiente cuadro de diálogo:



A continuación, tenemos que elegir la carpeta donde se desea instalar la aplicación. Esta será por defecto *C:\Archivos de programa.* (Se recomienda mantener este directorio de destino para usuarios no familiarizados con el entorno Windows). Una vez hayamos elegido la carpeta, debemos pulsar el botón **Unzip**.

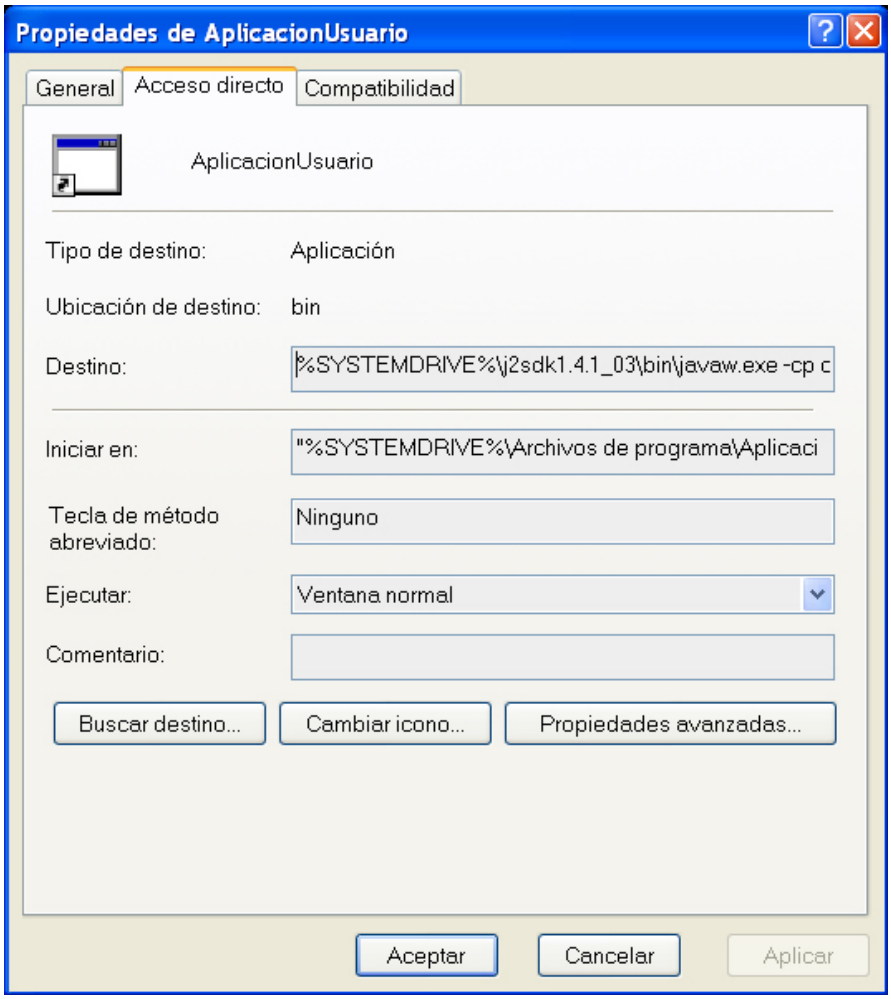
3.2) Configurar el acceso directo

Situándonos en el directorio de instalación elegido en el punto anterior, comprobamos que se ha creado la siguiente estructura arborescente de archivos y directorios:



donde el nodo del árbol destacado en negrita representa un acceso directo y el resto de nodos representan directorios

Hacemos clic con el botón derecho en el acceso directo *AplicacionUsuario1.0* y accedemos al elemento **propiedades** del menú desplegable. Nos aparecerá el siguiente diálogo:



En el campo **destino** indicaremos la ruta correcta al directorio donde se encuentra el Entorno de Ejecución de Java (dentro del cual deberíamos encontrar el comando *javaw.exe*). Por defecto el directorio es: *C:\j2sdk1.4.1_03\bin*.

En caso de que no se haya elegido la carpeta de instalación por defecto, también modificaremos el campo **Iniciar en** para que apunte al directorio “AplicacionUsuario1.0” dentro de la carpeta de instalación elegida.

Finalmente, pulsando el botón **Cambiar icono…** podremos modificar el icono por defecto asociado al acceso directo. En el directorio *data/* nos aparece un icono con el logotipo de GHADIR llamado *iconoGhadir.ico* que podemos utilizar.

3.3) Ejecutar la Aplicación Usuario

En este momento, el computador está listo para ejecutar la Aplicación Usuario. Simplemente tenemos que hacer doble clic en el acceso directo *AplicacionUsuario1.0.*

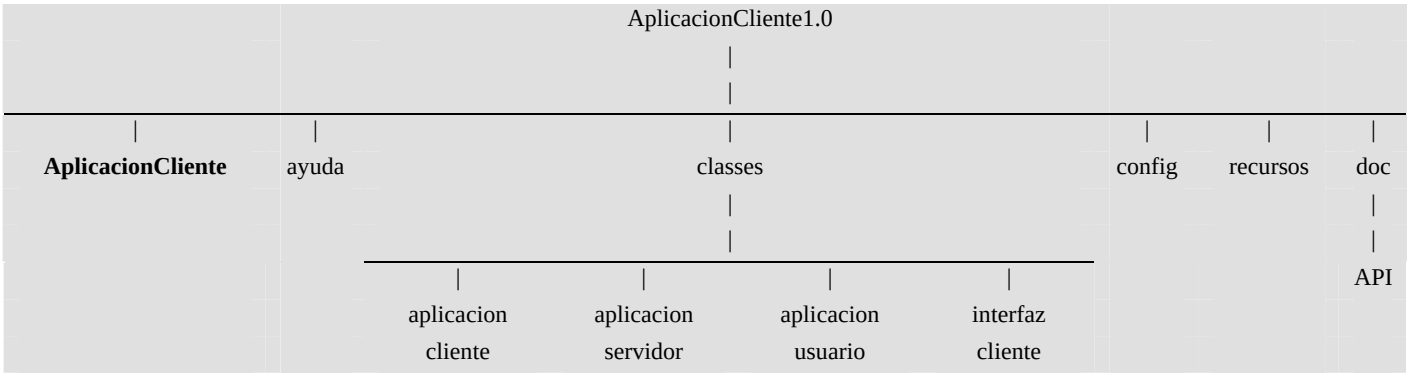
3.4) Desinstalar la Aplicación Usuario

Para desinstalar la aplicación, únicamente tenemos que borrar recursivamente el directorio *AplicacionUsuario1.0.*

4) Instalar, ejecutar y desinstalar la Aplicación Cliente

(De forma análoga a la Aplicación Usuario)

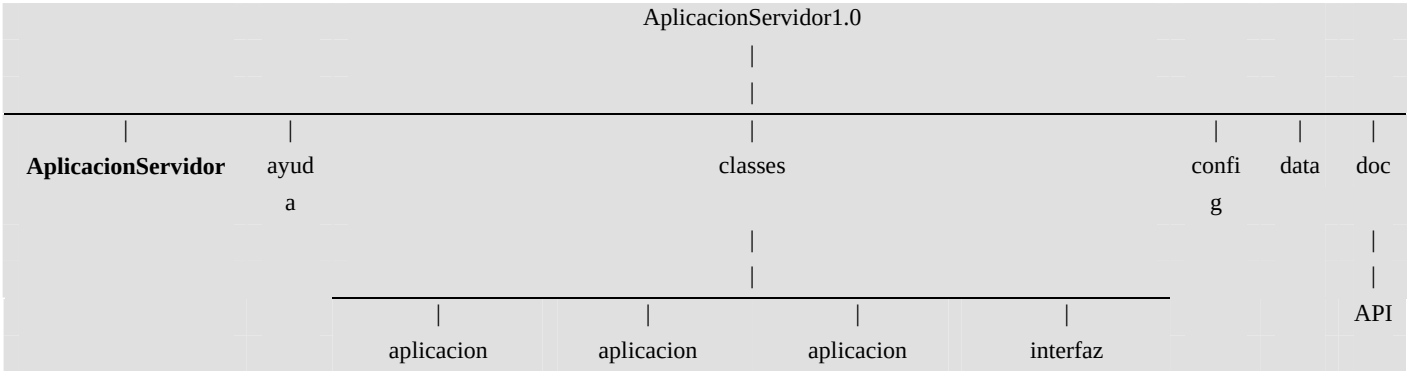
En este caso, situándonos en el directorio de instalación elegido, se debe haber creado la siguiente estructura arborescente de archivos y directorios:



5) Instalar, ejecutar y desinstalar la Aplicación Servidor

(De forma análoga a la Aplicación Usuario)

En este caso, situándonos en el directorio de instalación elegido, se debe haber creado la siguiente estructura arborescente de archivos y directorios:



cliente	servidor	usuario	servidor

6.2.2. SOBRE UNA PLATAFORMA Linux

A continuación, guiaremos el proceso de instalación del gestor de recursos de hardware reconfigurable sobre un sistema operativo Linux RedHat 8.0 y Linux RedHat 9.0. La instalación incluye la Aplicación Cliente, la Aplicación Usuario y la Aplicación Servidor.

1) Comprobar el tamaño de los archivos descargados (opcional)

Es recomendable comprobar que los archivos *AplicacionUsuario1.0.tar.gz*, *AplicacionCliente1.0.tar.gz* y *AplicacionServidor1.0.tar.gz* se han descargado completamente y no están corruptos.

2) Comprobar que se dispone de las herramientas necesarias

Para instalar la Aplicación Usuario es necesario tener instalado el Entorno de Ejecución de Java (Java Runtime Environment o JRE) disponible a través de la web http://java.sun.com/.

3) Instalar, ejecutar y desinstalar la Aplicación Usuario

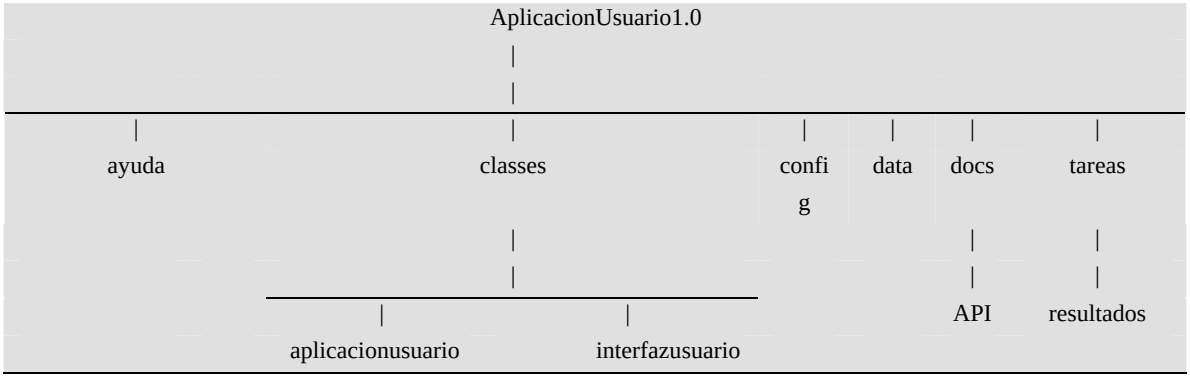
3.1) Instalar la Aplicación Usuario

Abrimos una ventana de terminal y nos situamos en el directorio de instalación deseado (por ejemplo, /*home/daniel/SistemasInformaticos*). Una vez allí, descomprimimos el archivo *AplicacionUsuario1.0.tar.gz*:

```
daniel:/home/daniel/SistemasInformaticos>tar –zxvf
      ../descargas/AplicacionUsuario.tar.gz
```

donde /*home/daniel/descargas* es el directorio donde se ha descargado el archivo *AplicacionUsuario1.0.tar.gz*

Situándonos en el directorio de instalación elegido, comprobamos que se ha creado la siguiente estructura arborescente de directorios:



3.2) Ejecutar la Aplicación Usuario

En este momento, el computador está listo para ejecutar la Aplicación Usuario. Simplemente tenemos que invocar al "lanzador" de Java desde el directorio *AplicacionUsuario1.0*, de la siguiente forma:

```
daniel:/home/daniel/SistemasInformaticos/AplicacionUsuario1.0>/usr/
      java/j2re1.4.2_05/bin/java -cp classes
      aplicacionusuario.AplicacionUsuario
```

donde */usr/java/j2re1.4.2_05* es el directorio donde se encuentra instalado el Entorno de Ejecución de Java.

3.3) Desinstalar la Aplicación Usuario

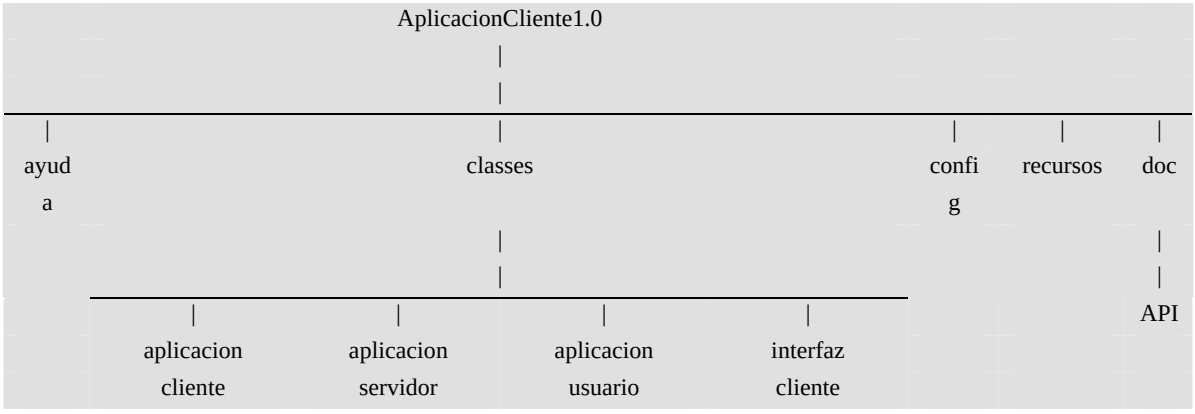
Para desinstalar la aplicación, únicamente tenemos que borrar recursivamente el directorio *AplicacionUsuario1.0*:

```
daniel:/home/daniel/SistemasInformaticos>rm –r AplicacionUsuario1.0
```

4) Instalar, ejecutar y desinstalar la Aplicación Cliente

(De forma análoga a la Aplicación Usuario)

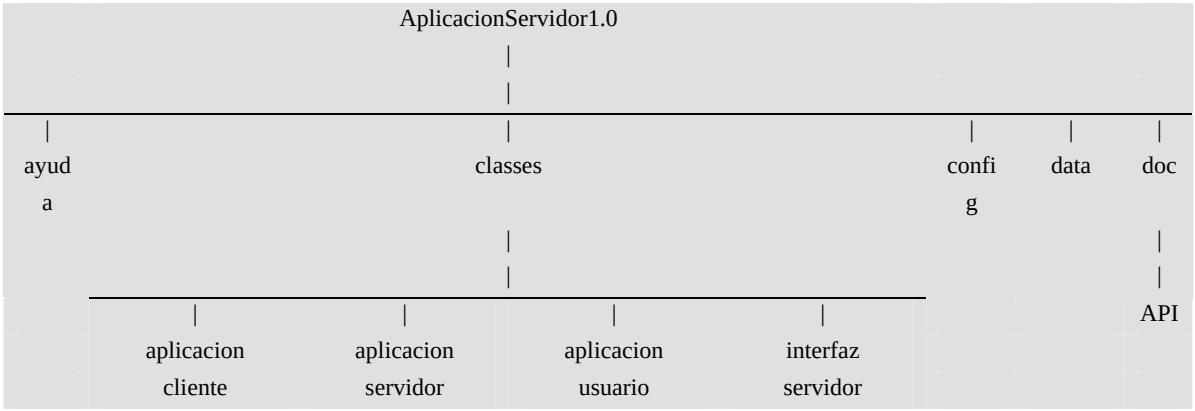
En este caso, situándonos en el directorio de instalación elegido, se debe haber creado la siguiente estructura arborescente de archivos y directorios:



5) Instalar, ejecutar y desinstalar la Aplicación Servidor

(De forma análoga a la Aplicación Usuario)

En este caso, situándonos en el directorio de instalación elegido, se debe haber creado la siguiente estructura arborescente de archivos y directorios:



6.3. CONFIGURACIÓN DEL SISTEMA

Para configurar el sistema, el usuario del mismo tiene a disposición una serie de archivos de configuración. Estos archivos los encontramos en el directorio “config” de cada una de las tres aplicaciones, es decir, en el directorio “config” de la Aplicación Usuario, en el directorio “config” de la Aplicación Cliente y en el directorio “config” de la Aplicación Servidor.

Los archivos de configuración son documentos XML y cada uno de ellos lleva asociado una DTD para poder validarlo. Se recomienda, por tanto, modificar tales archivos utilizando para ello un editor de XML con validación (por ejemplo XMLSpy). No obstante, esto no es obligatorio, puesto que el sistema también realiza validación al analizar los documentos, e informa de si el documento no está bien formado o no es válido.

Comentaremos a continuación las características de configuración de cada aplicación:

6.3.1. APLICACIÓN USUARIO

En esta aplicación encontramos dos ficheros XML de configuración con sus correspondientes DTDs:

- “opcionesConexion.xml” (junto con “opcionesConexion.dtd”), que permite configurar los parámetros de conexión de la Aplicación Usuario;
- “tiposFPGAs.xml” (junto con “tiposFPGAs.dtd”), que permite configurar los tipos de FPGAs concretos con los que el usuario de la Aplicación Usuario podrá tratar.

En el archivo de opciones de conexión, lo primero que podemos especificar es la dirección IP de la máquina en la que se ejecuta la aplicación servidor. En caso de que la Aplicación Usuario y la Aplicación Servidor compartan el mismo host, el contenido de este elemento puede ser vacío (aunque si se prefiere, se puede indicar la dirección del host local).

A continuación, debemos especificar los puertos asociados a esta aplicación. El primer puerto es el puerto del servidor (puerto remoto) a través del cuál los usuarios deben conectarse al servidor. El segundo puerto es el puerto del usuario (puerto local) a través del cuál los clientes deben conectarse con el usuario. Por supuesto, los números de puerto anteriores deben ser superiores a 1023, ya que los mil veinticuatro primeros están reservados.

Finalmente, se debe indicar el tamaño (en bytes) del buffer de lectura de los ficheros de mapa de bits y de datos asociados. Es decir, estos ficheros se enviaran al cliente en trozos de tamaño especificado. Este tamaño debe estar comprendido entre 1 y 1024.

Hay que destacar que el puerto remoto tiene su homólogo en el archivo de configuración de parámetros de conexión correspondiente de la Aplicación Servidor. Es decir, la modificación de este parámetro en las aplicaciones Usuario debería ir acompañada de la modificación del mismo parámetro en la Aplicación Servidor (además de en el resto de aplicaciones Usuario), y viceversa. El puerto local, sin embargo, no tiene homólogo en el correspondiente archivo de configuración de la Aplicación Cliente, es decir, sólo se configura en la Aplicación Usuario, siendo el sistema el que se encarga de poner esta información en conocimiento del cliente.

En el archivo de tipos de FPGAs se debe especificar una lista (no vacía) de fabricantes de estos dispositivos. Para cada fabricante se necesita a su vez una lista (no vacía) de los modelos de los que se dispone. Estos modelos vendrán especificados además de por el nombre, por las dimensiones de la matriz de CLBs que los componen, o lo que es lo mismo, por su altura y su anchura (medida en número de CLBs).

Este archivo tiene uno semejante en la Aplicación Cliente. Se recomienda que los archivos de configuración de tipos de FPGAs de las aplicaciones Usuario sean un “subconjunto de la unión” de los archivos de configuración de las aplicaciones Cliente, puesto que el sistema no podrá gestionar tareas HW para las que no se dispone de dispositivo HW. Sin embargo, cumplir la recomendación anterior no garantiza que el sistema pueda gestionar la tarea, puesto que es posible que ningún cliente tenga definido un recurso con tales características. Es decir, el fichero de configuración semejante en la Aplicación Cliente indica los tipos de dispositivos HW que puede definir un cliente, lo cual no significa que el cliente tenga definido un dispositivo de cada tipo.

EJEMPLOS DE FICHEROS

- opcionesConexion.xml

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE opcionesConexion SYSTEM "opcionesConexion.dtd">

<opcionesConexion>

      <!-- La direccion IP del servidor es la del host donde se ejecuta
este usuario -->
```

```
	<direccionServidor></direccionServidor>

	<!-- El servidor escucha a los usuarios en el puerto 4441 -->
	<puertoUsuarios>4441</puertoUsuarios>

	<!—Este usuario escucha a los clientes en el puerto 7766 -->
	<puertoClientes>7766</puertoClientes>

	<!-- Tamano del buffer de lectura de los ficheros de mapa de bits y
de datos asociados: 256 bytes -->
	<tamanoBuffer>256</tamanoBuffer>

</opcionesConexion>
```

- opcionesConexion.dtd

```
<?xml version='1.0' encoding='utf-8'?>

<!ELEMENT opcionesConexion (direccionServidor, puertoUsuarios,
puertoClientes, tamanoBuffer)>

<!ELEMENT direccionServidor (#PCDATA)>

<!ELEMENT puertoUsuarios (#PCDATA)>

<!ELEMENT puertoClientes (#PCDATA)>

<!ELEMENT tamanoBuffer (#PCDATA)>
```

- tiposFPGAs.XML

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE tiposFPGAs SYSTEM "tiposFPGAs.dtd">

<tiposFPGAs>

	<!-- ######################### -->
	<!-- Fabricante XILINX Virtex 2.5 V -->
	<!-- ######################### -->
	<fabricante nombre="XILINX Virtex 2.5 V">

		<!-- Modelo XCV50 -->
		<modelo nombre="XCV50">
			<altura>16</altura>
			<anchura>24</anchura>
		</modelo>
```

```
            <!-- Modelo XCV100 -->
            <modelo nombre="XCV100">
                  <altura>20</altura>
                  <anchura>30</anchura>
            </modelo>
      </fabricante>


      <!-- ########################### -->
      <!-- Fabricante XILINX Virtex-E 1.8 V -->
      <!-- ########################### -->
      <fabricante nombre="XILINX Virtex-E 1.8 V">

            <!-- Modelo XCV50E -->
            <modelo nombre="XCV50E">
                  <altura>16</altura>
                  <anchura>24</anchura>
            </modelo>

      </fabricante>

</tiposFPGAs>
```

- tiposFPGAs.dtd

```
<?xml version='1.0' encoding='utf-8'?>

<!ELEMENT tiposFPGAs (fabricante)+>
<!ELEMENT fabricante (modelo)+>
<!ATTLIST fabricante
      nombre CDATA #REQUIRED>
<!ELEMENT modelo (altura, anchura)>
<!ATTLIST modelo
      nombre CDATA #REQUIRED>
<!ELEMENT altura (#PCDATA)>
<!ELEMENT anchura (#PCDATA)>
```

6.3.2. APLICACIÓN SERVIDOR

Aquí encontramos sólo un fichero XML de configuración con sus correspondiente DTD:

- “opcionesConexion.xml”, que junto con “opcionesConexion.dtd” permite configurar los parámetros de conexión de las aplicaciones Usuario y Cliente.

Sólo dos parámetros se deben especificar aquí. El primero es el puerto del servidor (puerto local) a través del cuál los usuarios deben conectarse al servidor, como ya se indicó al considerar la Aplicación Usuario. El segundo es el puerto del servidor (puerto local) a través del cuál los clientes deben conectarse al servidor (como se verá al considerar la Aplicación Cliente).

EJEMPLOS DE FICHEROS

- opcionesConexion.xml

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE opcionesConexion SYSTEM "opcionesConexion.dtd">

<opcionesConexion>

      <!-- El servidor escucha a los usuarios en el puerto 4441 -->
      <puertoUsuarios>4441</puertoUsuarios>

      <!-- El servidor escucha a los clientes en el puerto 4441 -->
      <puertoClientes>5557</puertoClientes>

</opcionesConexion>
```

- opcionesConexion.dtd

```
<?xml version='1.0' encoding='utf-8'?>

<!ELEMENT opcionesConexion (puertoUsuarios, puertoClientes)>
<!ELEMENT puertoUsuarios (#PCDATA)>
<!ELEMENT puertoClientes (#PCDATA)>
```

6.3.3. APLICACIÓN CLIENTE

Encontramos dos ficheros XML de configuración con sus correspondientes DTDs:

- "opcionesConexion.xml" (junto con "opcionesConexion.dtd"), que permite configurar los parámetros de conexión de la Aplicación Cliente;
- "tiposFPGAs.xml" (junto con "tiposFPGAs.dtd"), que permite configurar los tipos de FPGAs concretos con los que el usuario de la Aplicación Usuario podrá tratar.

El archivo de opciones de conexión es prácticamente idéntico al descrito para la Aplicación Usuario. Existen dos diferencias: sólo se debe especificar un puerto y el tamaño del buffer no tiene por qué coincidir con el del usuario.

El puerto a especificar será el puerto del servidor (puerto remoto) a través del cuál los clientes se conectarán al servidor (mayor que 1023).

La segunda diferencia se debe a que el buffer, en este caso, es el buffer de lectura del fichero resultado de la ejecución. Es decir, estos ficheros se enviaran al usuario en trozos de tamaño especificado, y por tanto este parámetro es independiente del tamaño de los trozos del mapa de bits que se reciben del usuario. En cualquier caso, el tamaño de este buffer de lectura también debe encontrarse entre 1 y 1024 bytes.

EJEMPLOS DE FICHEROS

- opcionesConexion.xml

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE opcionesConexion SYSTEM "opcionesConexion.dtd">

<opcionesConexion>

	<!-- La direccion IP del servidor es la del host donde se ejecuta este usuario -->
	<direccionServidor></direccionServidor>

	<!-- El servidor escucha a los clientes en el puerto 5557 -->
	<puertoClientes>5557</puertoClientes>

	<!-- Tamano del buffer de lectura de los ficheros resultado: 128 bytes -->
	<tamanoBuffer>128</tamanoBuffer>

</opcionesConexion>
```

- opcionesConexion.dtd

```
<?xml version='1.0' encoding='utf-8'?>

<!ELEMENT opcionesConexion (direccionServidor, puertoClientes, tamanoBuffer)>
<!ELEMENT direccionServidor (#PCDATA)>
<!ELEMENT puertoClientes (#PCDATA)>
<!ELEMENT tamanoBuffer (#PCDATA)>
```

- tiposFPGAs.XML

```
<?xml version='1.0' encoding='utf-8'?>
```

```
<!DOCTYPE tiposFPGAs SYSTEM "tiposFPGAs.dtd">

<tiposFPGAs>

      <!-- ########################## -->
      <!-- Fabricante XILINX Virtex-E 1.8 V -->
      <!-- ########################## -->
      <fabricante nombre="XILINX Virtex-E 1.8 V">

            <!-- Modelo XCV50E -->
            <modelo nombre="XCV50E">
                  <altura>16</altura>
                  <anchura>24</anchura>
            </modelo>

      </fabricante>

</tiposFPGAs>
```

- tiposFPGAs.dtd

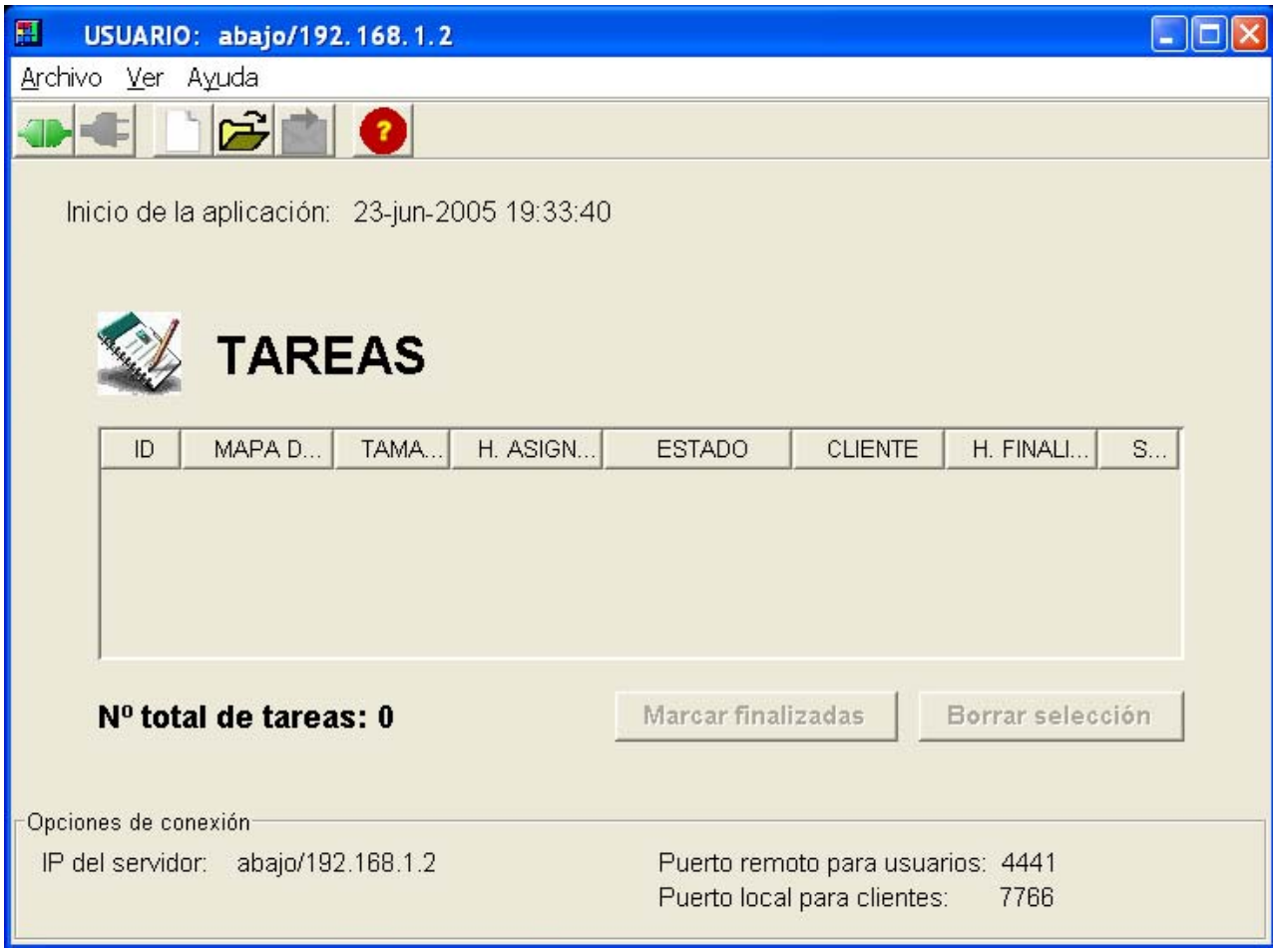
```
<?xml version='1.0' encoding='utf-8'?>

<!ELEMENT tiposFPGAs (fabricante)+>
<!ELEMENT fabricante (modelo)+>
<!ATTLIST fabricante
      nombre CDATA #REQUIRED>
<!ELEMENT modelo (altura, anchura)>
<!ATTLIST modelo
      nombre CDATA #REQUIRED>
<!ELEMENT altura (#PCDATA)>
<!ELEMENT anchura (#PCDATA)>
```

6.4. USO DE LA APLICACIÓN USUARIO

1. Cómo entrar y salir de la aplicación

Para entrar en la aplicación es necesario haberla instalado previamente (tal y cómo se indica en el apartado “instalación de las aplicaciones”). Una vez instalada simplemente tenemos que hacer doble clic en el acceso directo *AplicacionUsuario* (si ejecutamos la aplicación sobre Windows). Aparecerá entonces la siguiente pantalla de carga durante aproximadamente cuatro segundos:

Inmediatamente después de desaparecer la imagen anterior, podremos apreciar el aspecto de la Aplicación Usuario, que será semejante al siguiente:



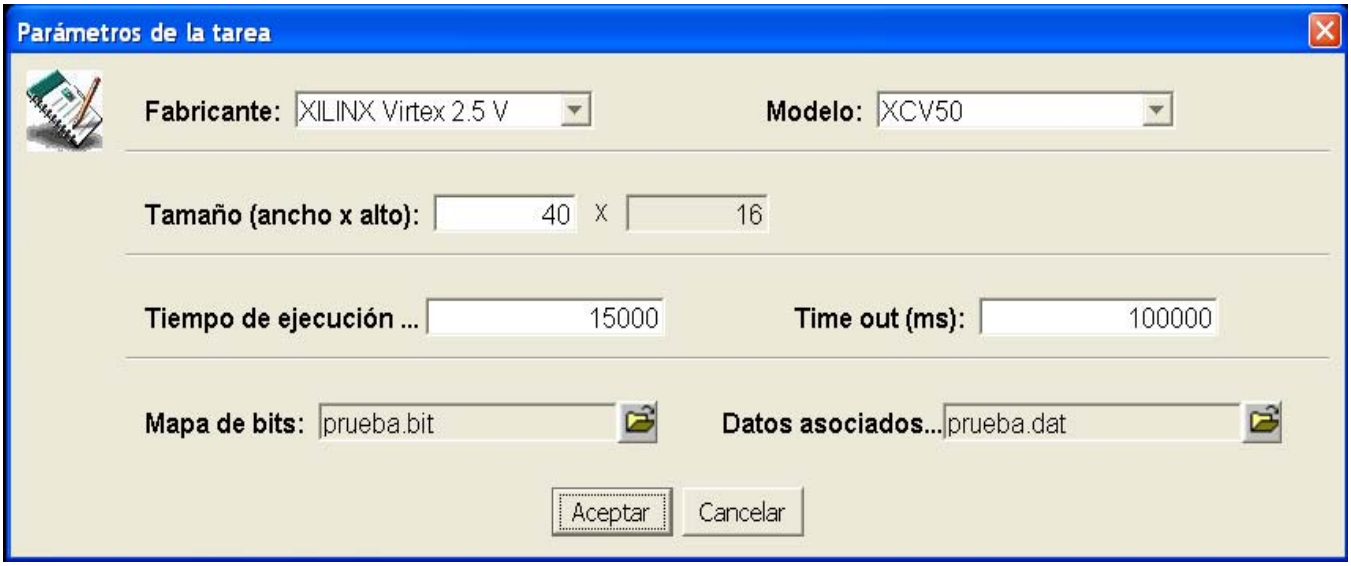
Para abandonar la aplicación, pulsamos el elemento **Salir** del menú **Archivo** o hacemos clic sobre el aspa roja de la esquina superior derecha de la ventana.

2. Creación de una tarea

Existen dos formas de crear una tarea, las cuales se detallarán en los siguientes subapartados. La primera de ellas es interactivamente, es decir, desde la Aplicación Usuario. La segunda de ellas es desde un archivo de texto.

2.1 Creación manual (interactiva)

Para crear una nueva tarea de esta forma, accedemos al elemento **Crear tarea** del menú **Archivo**, o pulsamos el botón  de la barra de herramientas. El programa pedirá al usuario (de forma interactiva) la tarea a crear, a través del siguiente cuadro de diálogo:



Los parámetros de la tarea que debemos especificar son:

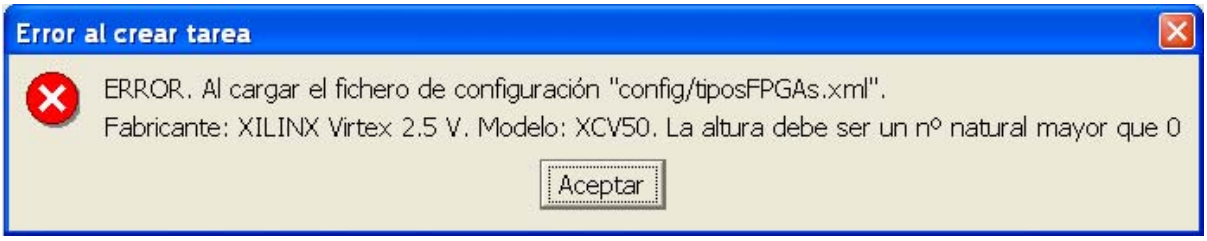
- el fabricante y el modelo de la FPGA para la que está compilado el mapa de bits asociado a la tarea
- la anchura, medida en número de CLBs
- el tiempo de ejecución y el time-out (tiempo máximo de espera del usuario), medidos ambos en milisegundos
- la ruta al fichero de mapa de bits y al fichero de datos asociados dentro del host en el que se ejecuta la Aplicación Usuario

Varios detalles sobre el diálogo anterior debemos destacar:

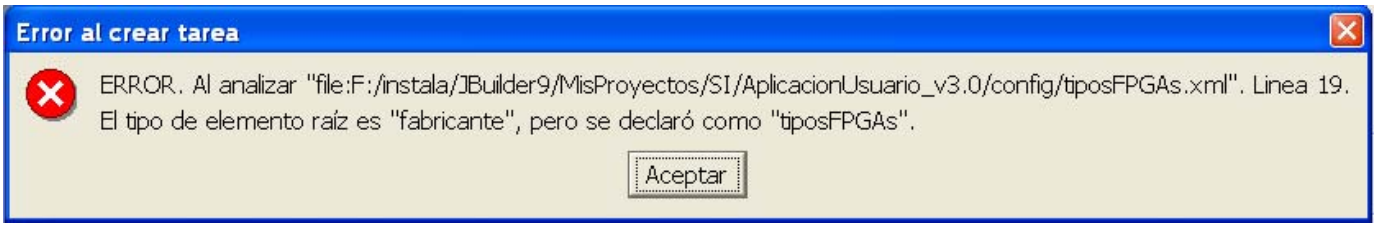
a) Todos los parámetros anteriores son obligatorios.

b) Las opciones para elegir el fabricante y el modelo son las que vienen especificadas en el archivo de configuración *tiposFPGAs.xml*. En caso de que el documento XML

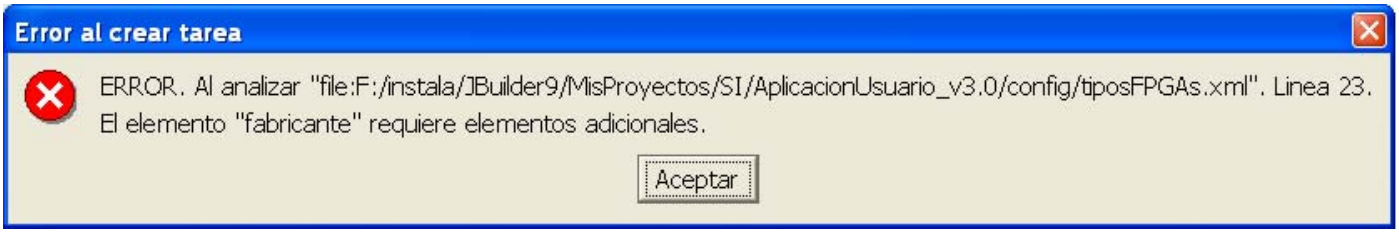
anterior no esté bien formado, no sea válido (no cumpla la DTD especificada en *tiposFPGAs.dtd*), o los parámetros que allí se especifiquen no sean factibles, se producirá un error. Por ejemplo, si el parámetro *altura* de un modelo de FPGA tomara un valor negativo (parámetro no factible), aparecería el siguiente error ordinario:



En caso de que el documento no estuviera bien formado, por ejemplo porque faltase el elemento raíz <*tiposFPGAs*>:



Finalmente, si el documento no fuera válido, por ejemplo porque la lista de modelos asociada a un fabricante fuera vacía:



c) La anchura de la tarea es configurable, pero la altura no, puesto que esta es inherente al modelo de la misma. La anchura debe ser un número natural mayor que 0 y menor o igual que la anchura del modelo de FPGA.

d) El tiempo de ejecución y el time-out deben ser números naturales mayores que 0.

e) Deben especificarse rutas válidas a los ficheros de mapa de bits y de datos asociados.

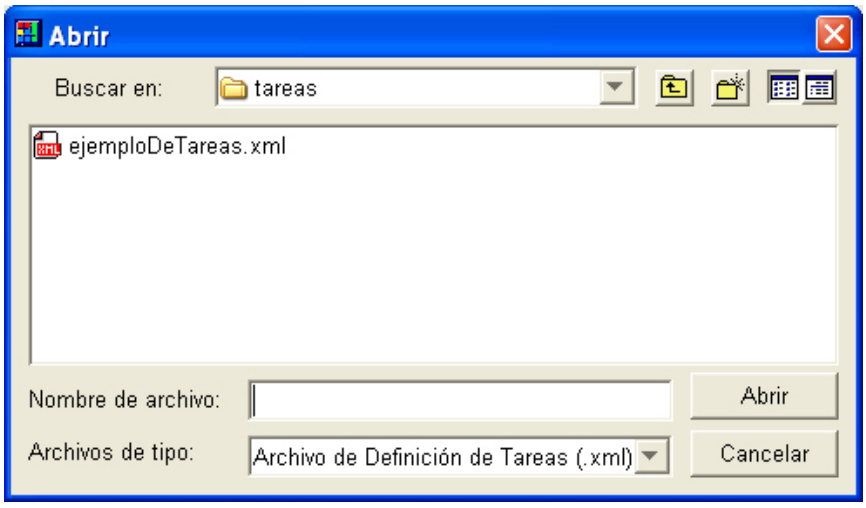
2.2 Creación desde archivo

La creación manual está aconsejada para usuarios poco familiarizados con la aplicación construida, o para los que quieran hacer pequeñas pruebas (en número de tareas). Para el resto de los usuarios, la creación manual resulta un tanto tediosa, de forma que se incluye esta otra.

La forma de utilizarla será a través de un archivo XML. Dicho archivo consistirá básicamente en una lista de tareas y para cada tarea habrá que especificar los mismos parámetros que en el caso interactivo, es decir, el fabricante, el modelo, la anchura, el tiempo de ejecución y el

time-out, y la ruta al fichero de mapa de bits y de datos asociados. A diferencia del caso interactivo, en éste se permiten crear varias tareas a la vez, es decir, lo que llamaremos **"lotes de tareas"**. Para más detalles sobre la creación de un fichero de este estilo, consultar el archivo *ejemploDeTareas.xml* (junto con *ejemploDeTareas.dtd*), que se encuentra en el directorio *tareas*.

Para cargar una tarea de archivo, accedemos al elemento **Cargar tarea** del menú **Archivo**, o pulsamos el botón  de la barra de herramientas. El programa pedirá al usuario la tarea a cargar, a través de una ventana de selección de archivos similar a la siguiente:



En cualquier caso, da igual que tipo de creación empleemos, si la tarea ha sido creada (y cargada) correctamente, debe aparecer una nueva entrada en la tabla de tareas como la siguiente:

donde ID es el identificador de la tarea (interno a la aplicación),

MAPA DE BITS el fichero de mapa de bits,

TAMAÑO (KB) es el tamaño del fichero anterior (en KiloBytes),

H. ASIGNACIÓN es la hora a la que la tarea se asigna a algún cliente para ser ejecutada (en blanco mientras la tarea no se solicite o se solicite pero no se acepte),

ESTADO es el estado actual de la tarea,

CLIENTE es el cliente asignado para ejecutar la tarea (en blanco mientras la tarea no se solicite o se solicite pero no se acepte),

H. FINALIZACIÓN es la hora en la que la tarea termina de recibir el resultado de la ejecución de la tarea (en blanco mientras la tarea no finalice),

SEL es la casilla de verificación que permite seleccionar cada tarea (fila de la tabla de tareas).

3. Eliminación de una tarea

Para eliminar una tarea cargada, debemos marcar la casilla de verificación (SEL) correspondiente a la tarea en la tabla de tareas. Posteriormente, pulsamos el botón **Borrar selección**.

Si queremos marcar de una vez todas las tareas que han sido solicitadas, aceptadas y han finalizado, pulsamos el botón **Marcar finalizadas**. Las tareas que han sido solicitadas y aceptadas pero no han finalizado no se pueden eliminar.

4. Almacenamiento y recuperación de una tarea

En este punto se detallan las acciones que permiten guardar una tarea en archivo y recuperarla de nuevo a partir de ese archivo.

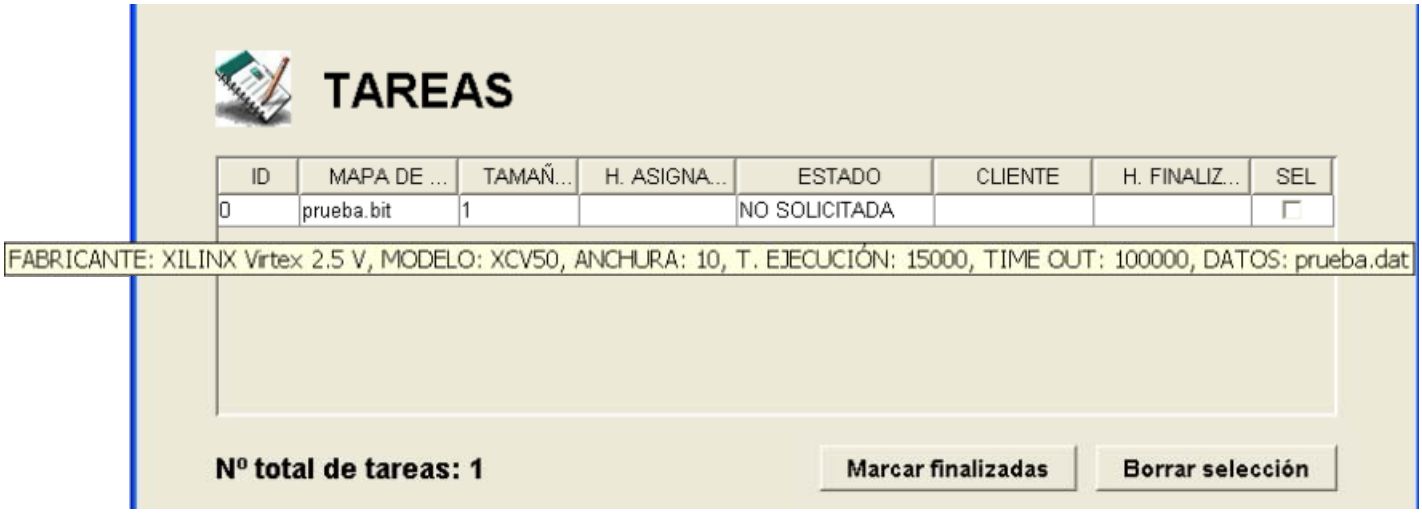
La recuperación de una tarea que previamente había sido guardada en archivo ya se ha indicado en el subapartado “Creación desde archivo” del punto anterior. Por otra parte, desde la Aplicación Usuario, no tenemos forma de salvar una tarea en archivo, así que la **única forma de almacenar una tarea es crear el archivo XML de forma externa al programa**.

5. Modificación de una tarea

Una vez una tarea ha sido creada de forma interactiva, ésta no se puede modificar. Si ha sido creada desde archivo, la modificación consistirá en alterar dicho archivo, siempre que la tarea no haya sido cargada en el programa. Por tanto, **solamente podrán modificarse las tareas creadas desde archivo mientras no se carguen en el programa**. Una vez cargada, lo más que se puede hacer es borrarla y crear otra nueva.

6. Visualización de una tarea

La visualización de una tarea permitirá al usuario ver los parámetros de la tarea, puesto que ninguno de ellos se detallan en una columna de la tabla de tareas. Para visualizar una tarea no tenemos más que posicionar el ratón y moverlo sobre la fila de la tabla de tareas deseada. Nos aparecerá algo como lo siguiente:



7. Conexión y desconexión del servidor

Para conectarnos al servidor, seleccionamos el elemento **Conectar** del menú **Archivo** o hacemos clic sobre el botón de la barra de herramientas.

Para desconectarnos del servidor tenemos que estar previamente conectados y seleccionar el elemento **Desconectar** del menú **Archivo** o hacer clic sobre el botón de la barra de herramientas.

8. Tratamiento de una tarea

En este apartado se detalla la acción más importante, y para el cuál se ha creado toda la estructura de la aplicación. Dicha acción es la de **solicitar la ejecución de una tarea a servidor**.

Para solicitar una tarea debemos estar previamente conectados y tener alguna tarea en estado "NO SOLICITADA". Entonces, seleccionamos el elemento **Enviar tarea** del menú **Archivo** o hacemos clic sobre el botón de la barra de herramientas. La ventana de envío de tareas tiene el siguiente aspecto:

donde TAREAS DISPONIBLES son las tareas que no han sido solicitadas

TAREAS A ENVIAR son las tareas disponibles seleccionadas para enviar

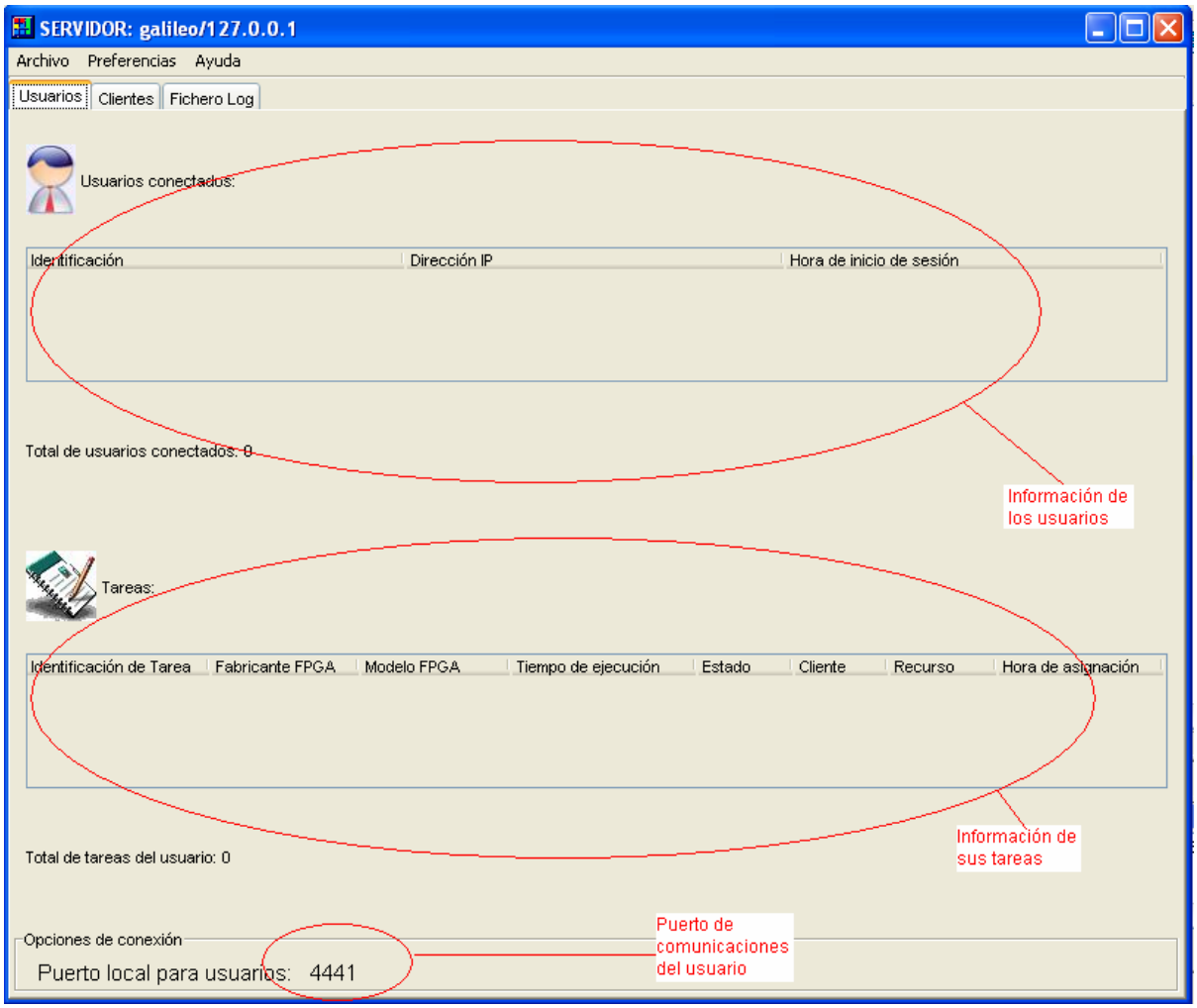
6.5. USO DE LA APLICACIÓN SERVIDOR

1. Cómo entrar y salir de la aplicación

Para entrar en la aplicación es necesario haberla instalado previamente (tal y cómo se indica en el apartado "instalación de las aplicaciones"). Una vez instalada simplemente tenemos que hacer doble clic en el acceso directo *AplicacionServidor* (si ejecutamos la aplicación sobre Windows). Aparecerá entonces la siguiente pantalla de carga durante aproximadamente cuatro segundos:

Una vez iniciada la aplicación, podremos ver la interfaz gráfica del servidor. Está constituida por tres paneles que muestran la información de usuarios, clientes y el fichero de log que se va generando conforme surgen nuevos sucesos.

1. El **panel Usuarios** muestra la información correspondiente a los usuarios conectados al sistema y las tareas que desean ejecutar. Ver figura:

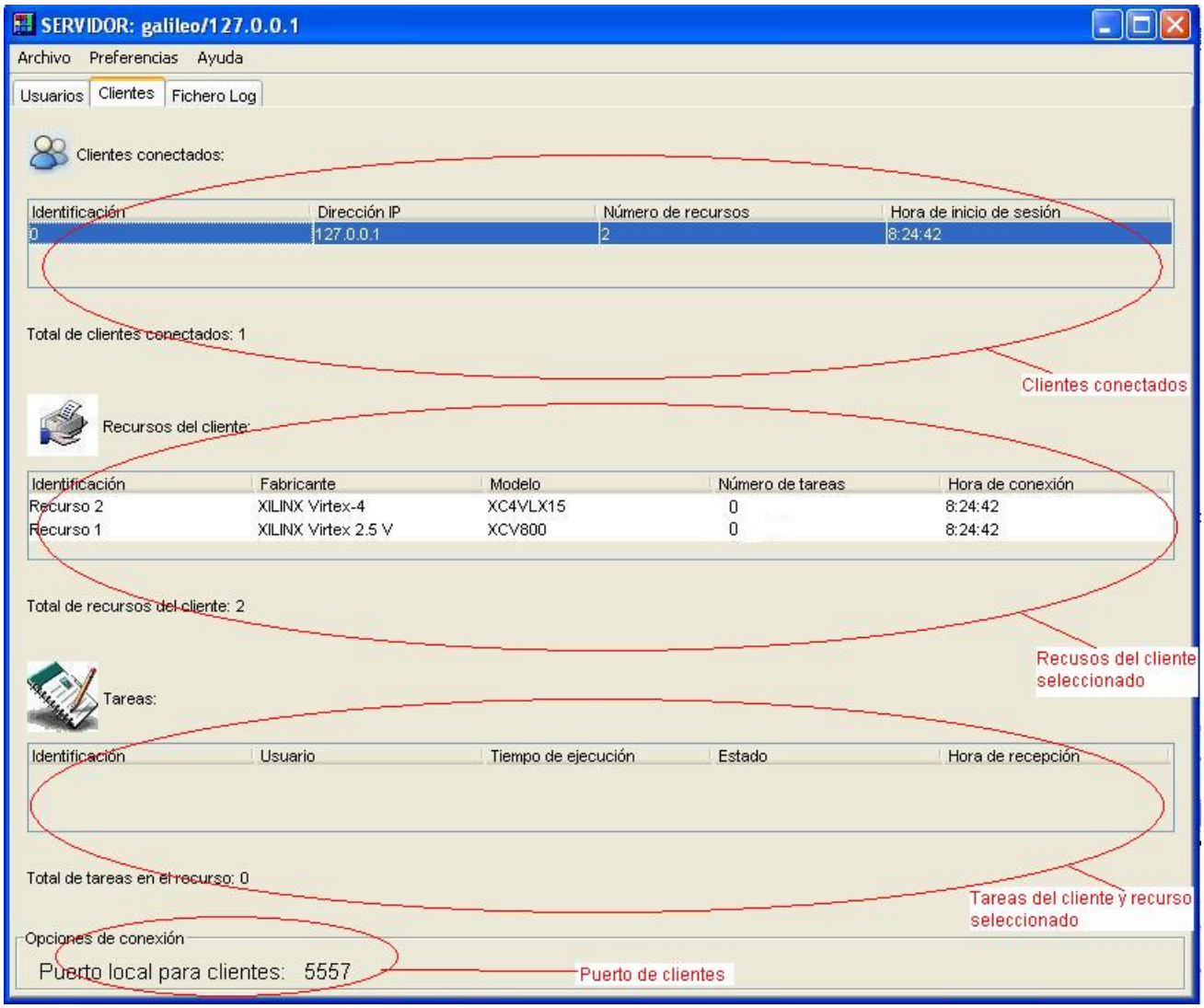


Como vemos, en la primera tabla, se muestran los datos de los usuarios conectados al sistema: un código identificativo, su dirección IP y la hora de inicio de sesión.

En la tabla siguiente, aparece la información de las tareas del usuario que se ha seleccionado en la tabla anterior: código identificativo, el fabricante de la FPGA para el que la tarea está compilada, el modelo de FPGA concreto, el la duración de la ejecución de la tarea, el estado (aceptada si ya tiene), el cliente y recurso que ejecutarán la tarea y la hora en que se le asignó dicho recurso.

Por último, se muestra el puerto por el que la aplicación servidor escucha peticiones de usuarios. Por defecto, se establece el número de puerto 4441.

2. En la siguiente pestaña, encontramos el **panel Clientes**. Aquí se mostrará toda la información útil relacionada con los clientes que ofrecen sus recursos al sistema, información de los propios recursos y de las tareas que ejecutan. Ver figura:



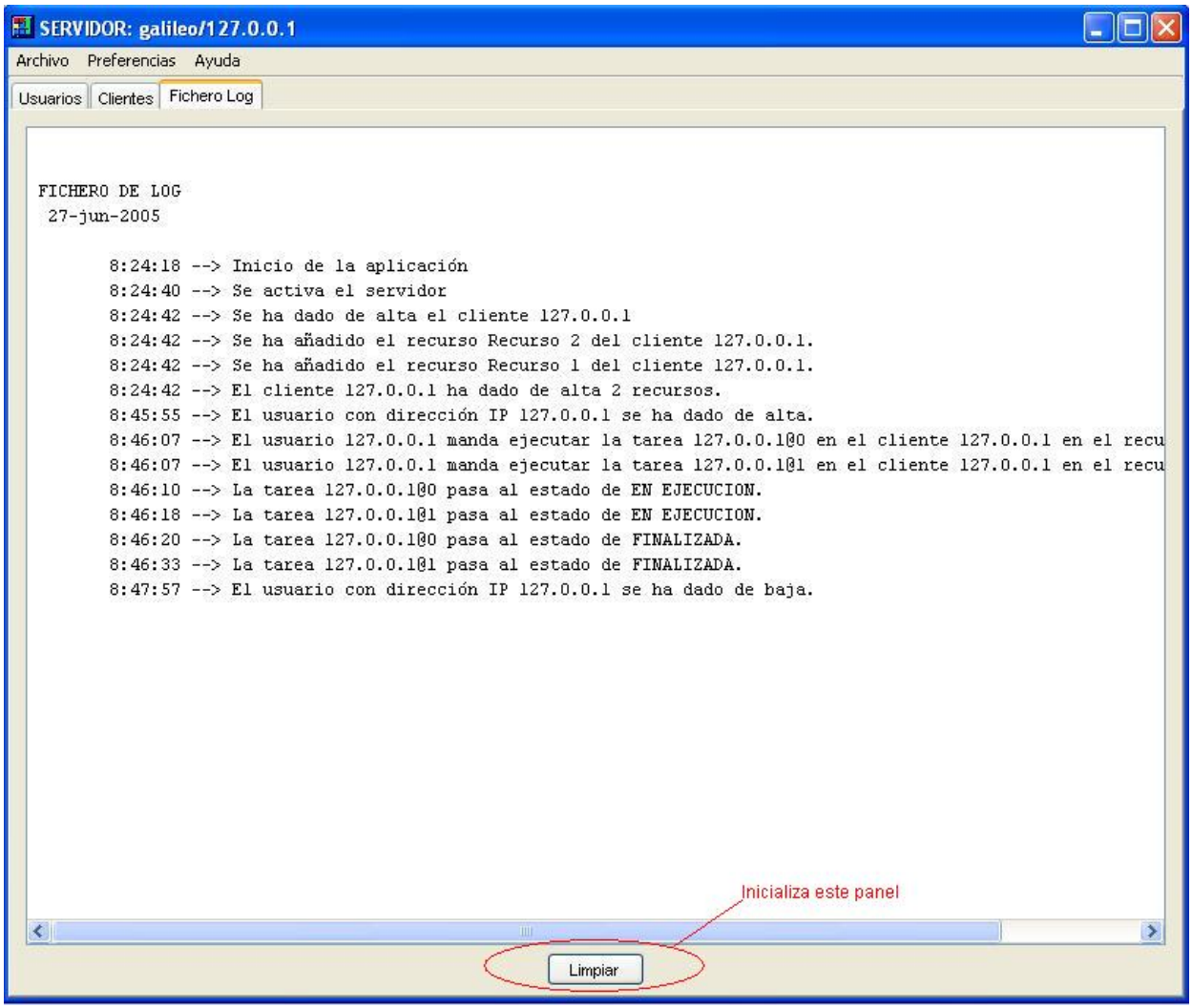
La primera tabla muestra los datos de los clientes conectados como: código de identificación, su dirección IP, el número de recursos que poseen y la hora de inicio de sesión.

En la siguiente, mostramos los recursos del cliente seleccionado en la tabla anterior: Identificación del recurso, su fabricante y modelo, el número de tareas que tiene en ejecución y so hora de conexión al sistema.

A continuación, se muestran las tareas asociadas a un cliente y recurso concreto: identificación de la tarea, usuario al que pertenece, el tiempo de ejecución y estado y la hora de recepción de la misma.

Por último, se muestra el puerto por el que la aplicación escucha a los clientes. Por defecto, establecemos el puerto número 5557.

3. En la última pestaña, encontramos el **panel del fichero log**. Todos los acontecimientos significativos que tiene lugar, de forma transparente en la aplicación, y los que suceden muy rápidamente como para ser apreciados, se muestran en este panel. Ver figura:



El fichero log de la aplicación se guarda automáticamente en un archivo, pero se puede consultar seleccionando este panel. En él se muestran todos los sucesos importantes: hora de inicio de sesión, clientes, recursos y usuarios que se conectan, que se dan de baja, tareas que se solicitan, ejecutan y finalizan, etc.,…

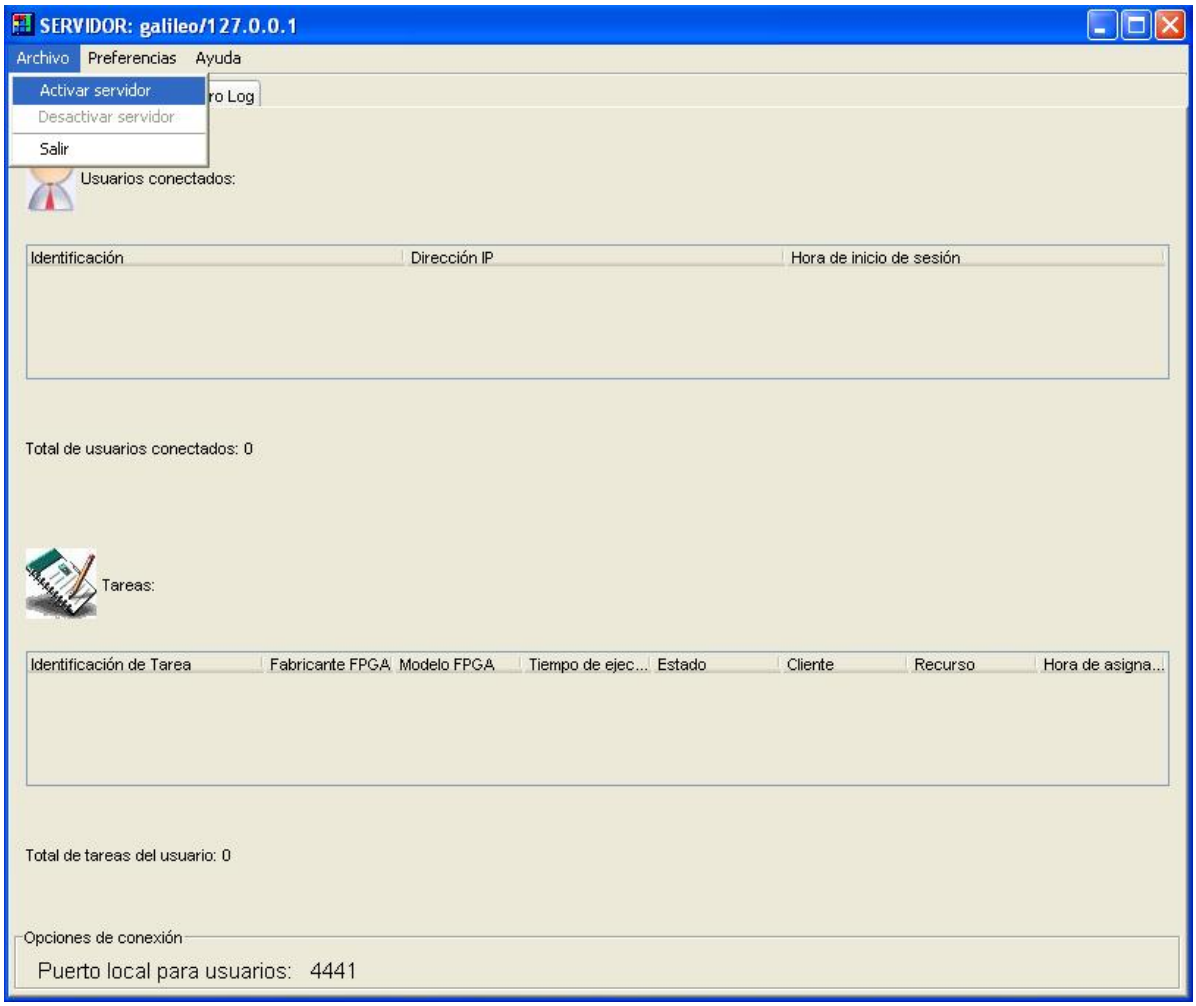
El botón “Limpiar”, inicializa el contenido del panel, aunque toda la información siga estando almacenada en el archivo.

En este ejemplo, se ve como un cliente se da de alta y también da de alta a dos recursos suyos. Además, un usuario se inicia en el sistema y solicita la ejecución de dos tareas, de las cuales se irá modificando su estado (de solicitada a ejecución y posteriormente a finalizada).

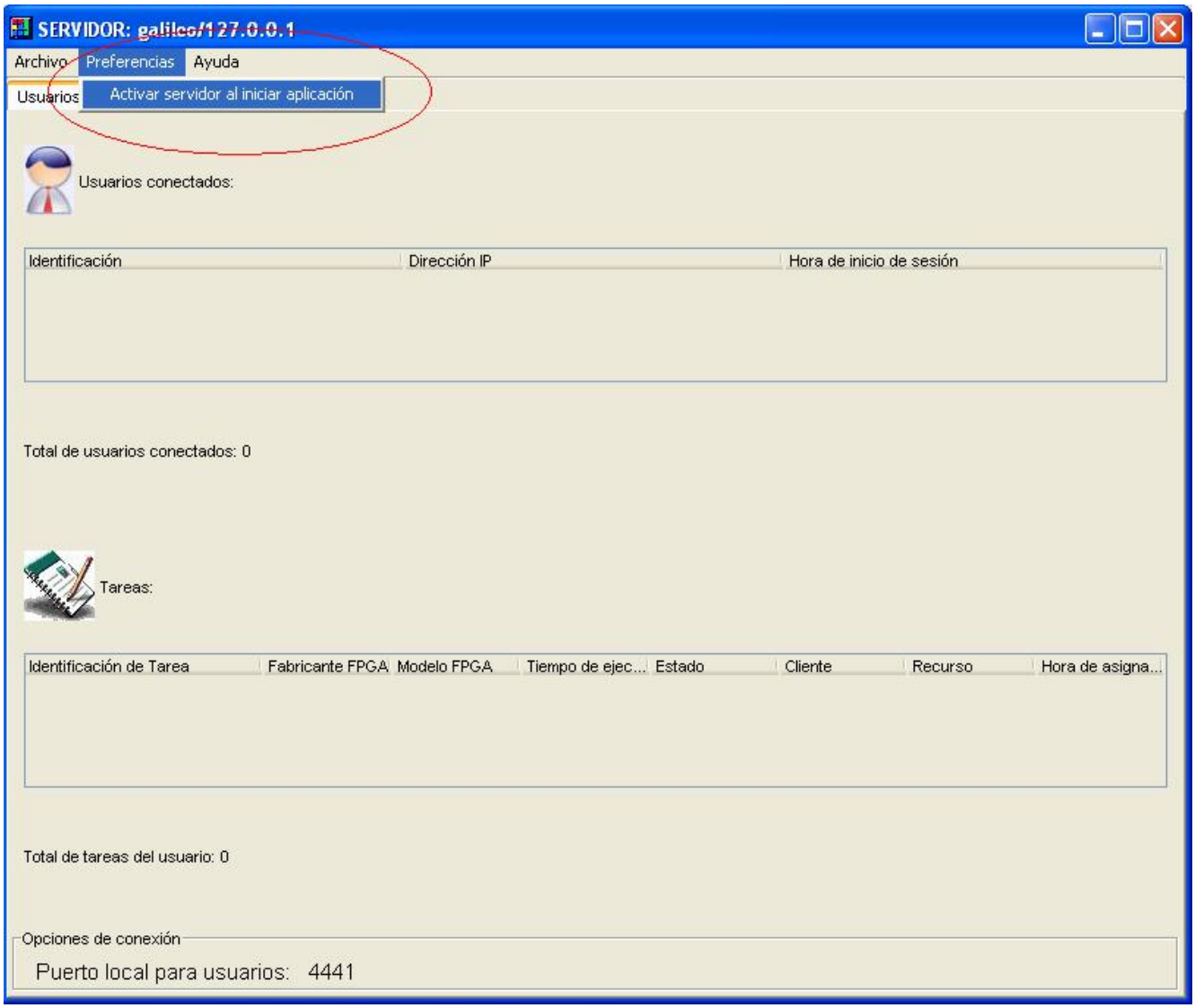
2. Activación del Servidor

El primer paso es conectar la aplicación servidor al sistema. Para ello, debemos seleccionar, en el menú Servidor, la opción Activar Servidor. Ver figura.

.



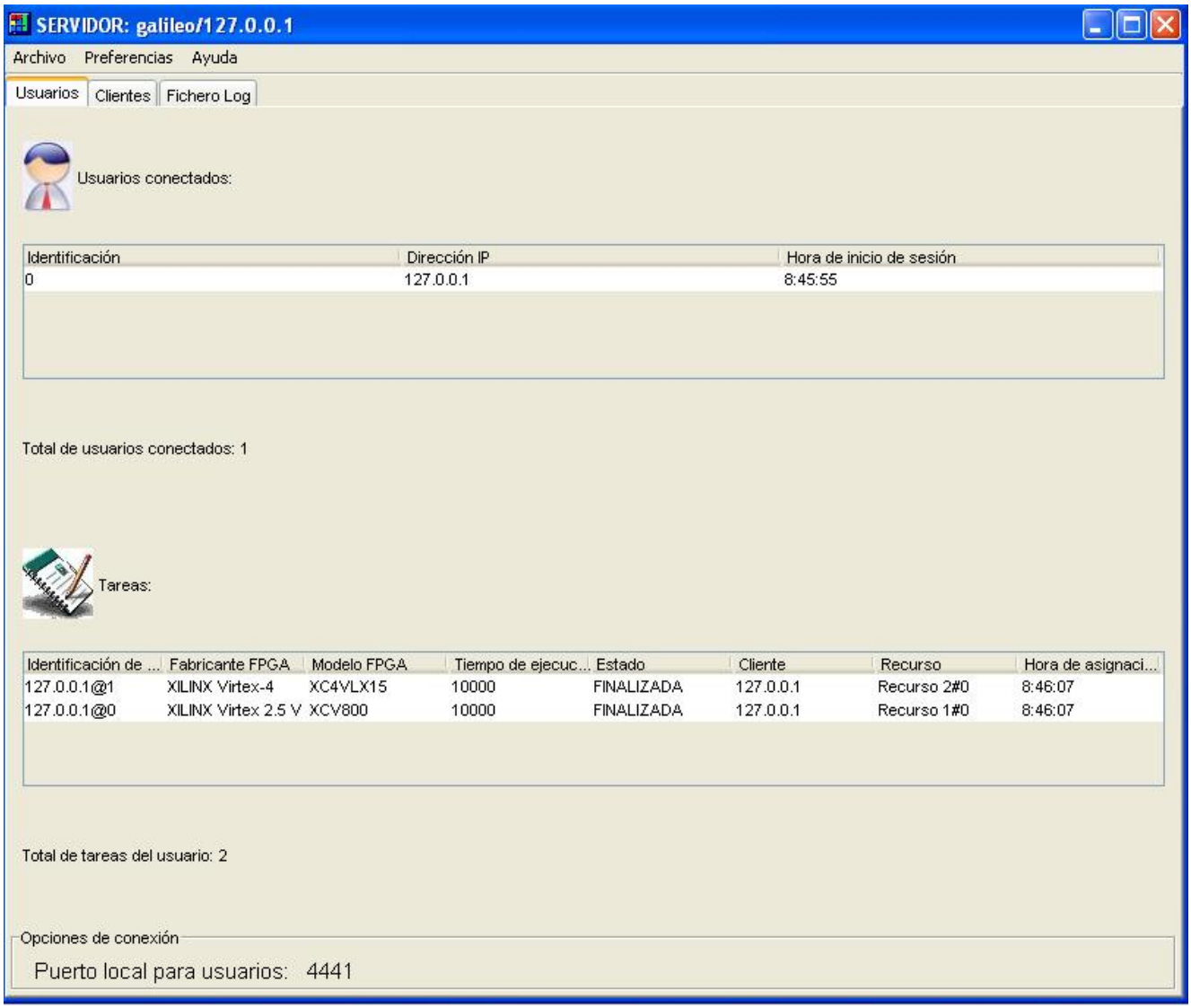
También, podemos configurar el sistema para que cada vez que se abra la aplicación, se conecte automáticamente. Para ello, en el menú Preferencias, seleccionamos la opción Activar Servidor al iniciar sesión. Ver figura.



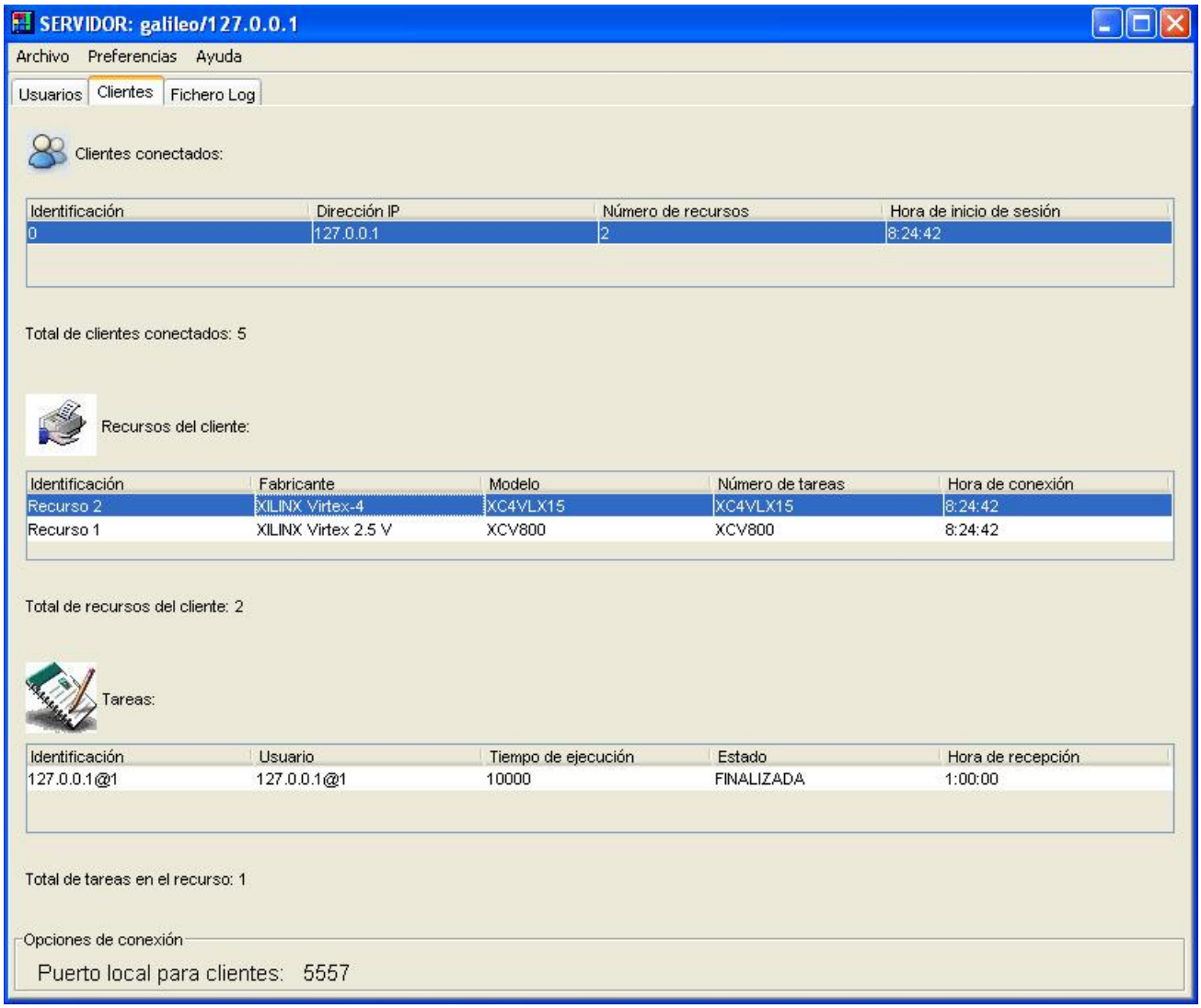
3. Visualización del envío y ejecución de una tarea del usuario

¿Qué ocurre cuando un usuario solicita la ejecución de una tarea o lote de tarea?

En el panel de usuarios se mostrará:



Mientras que en el de clientes veremos algo similar a la siguiente figura:



Al seleccionar el cliente y uno de sus recursos, vemos que la tarea 1 se ha ejecutado en el recurso 2.

De la misma forma, en todo momento se puede consultar el panel de log y obtener, por escrito, un resumen de todo lo que está sucediendo.

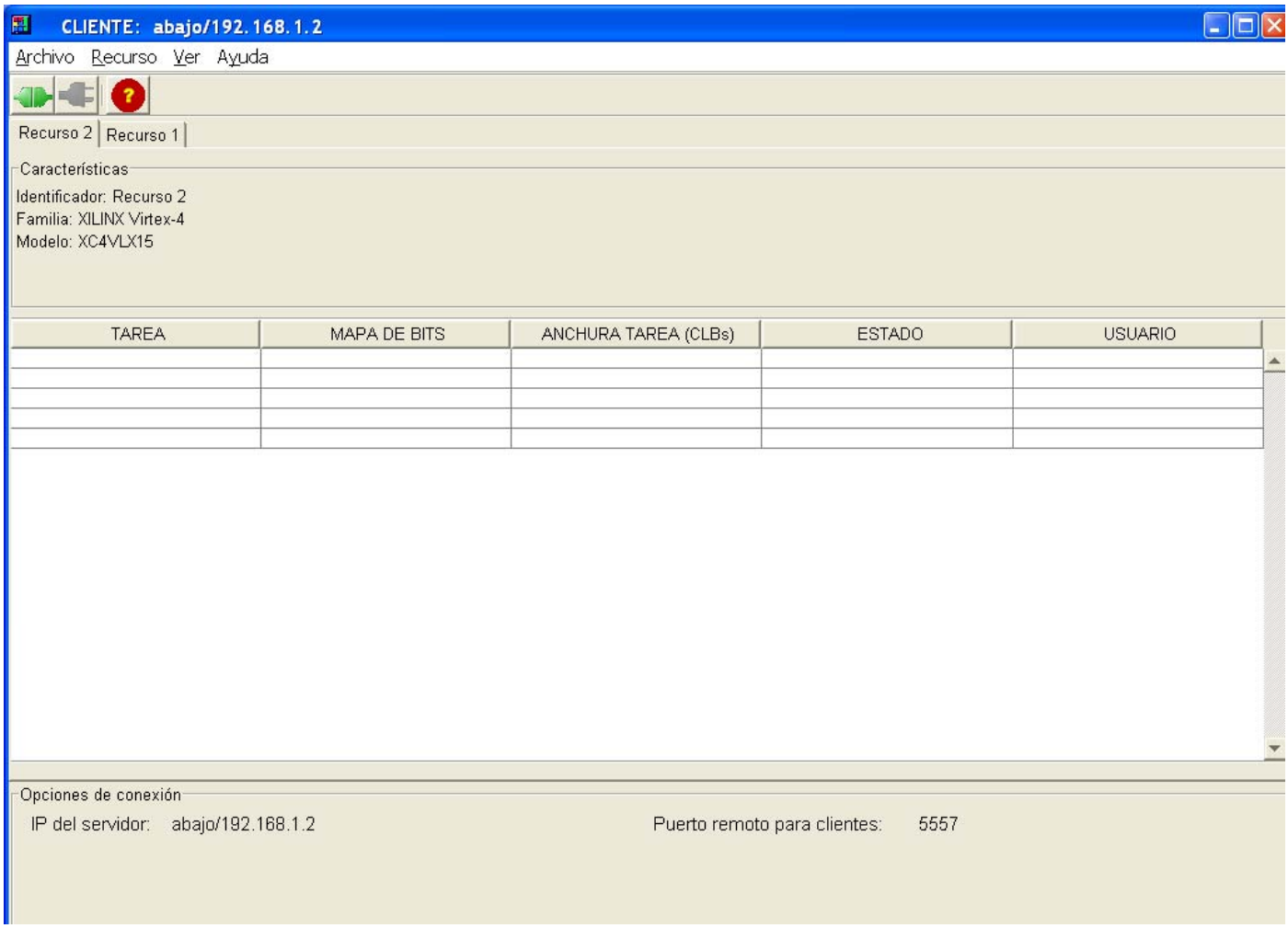
6.6. USO DE LA APLICACIÓN CLIENTE

1. Ejecución de la aplicación cliente.

Para poder ejecutar la aplicación cliente es necesario haberla instalado con éxito siguiendo los pasos de este manual. Una vez ejecutemos la aplicación (haciendo doble clic sobre el acceso directo *AplicacionCliente* si estamos en Windows) veremos la pantalla de presentación siguiente:

En seguida veremos la pantalla principal de la aplicación, desde donde podremos:

a) conectar y desconectar con el servidor del sistema
b) dar de alta y de baja nuestros recursos
c) monitorizar las tareas que realicen nuestros recursos
d) consultar la ayuda de la aplicación
e) salir de la aplicación



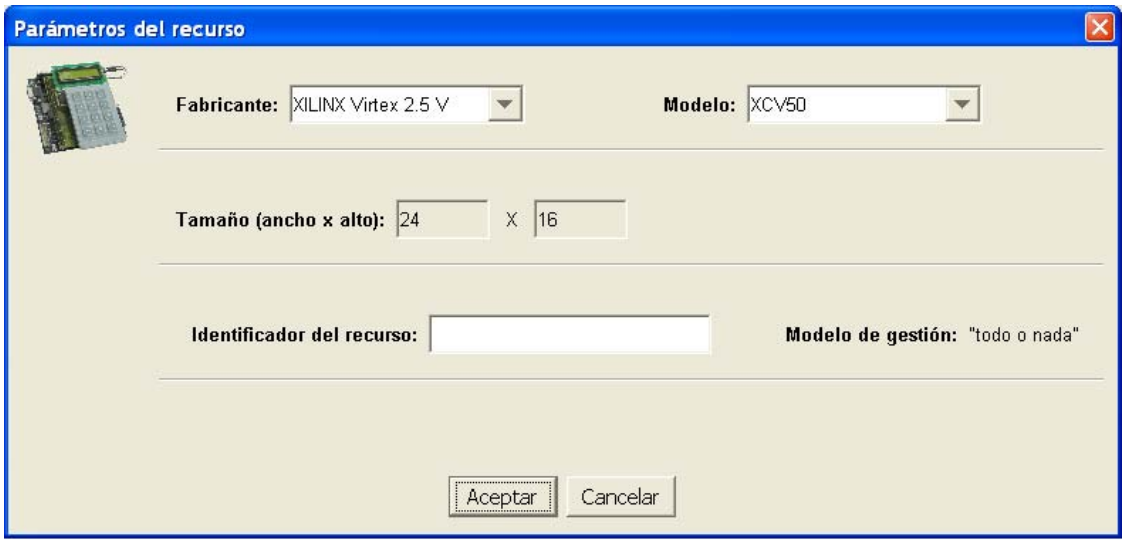
a) conectar y desconectar con el servidor del sistema

Para conectarnos al servidor, seleccionamos el elemento **Conectar** del menú **Archivo** o hacemos clic sobre el botón  de la barra de herramientas.

Para desconectarnos del servidor tenemos que estar previamente conectados y seleccionar el elemento **Desconectar** del menú **Archivo** o hacer clic sobre el botón  de la barra de herramientas.

b) dar de alta o de baja nuestros recursos

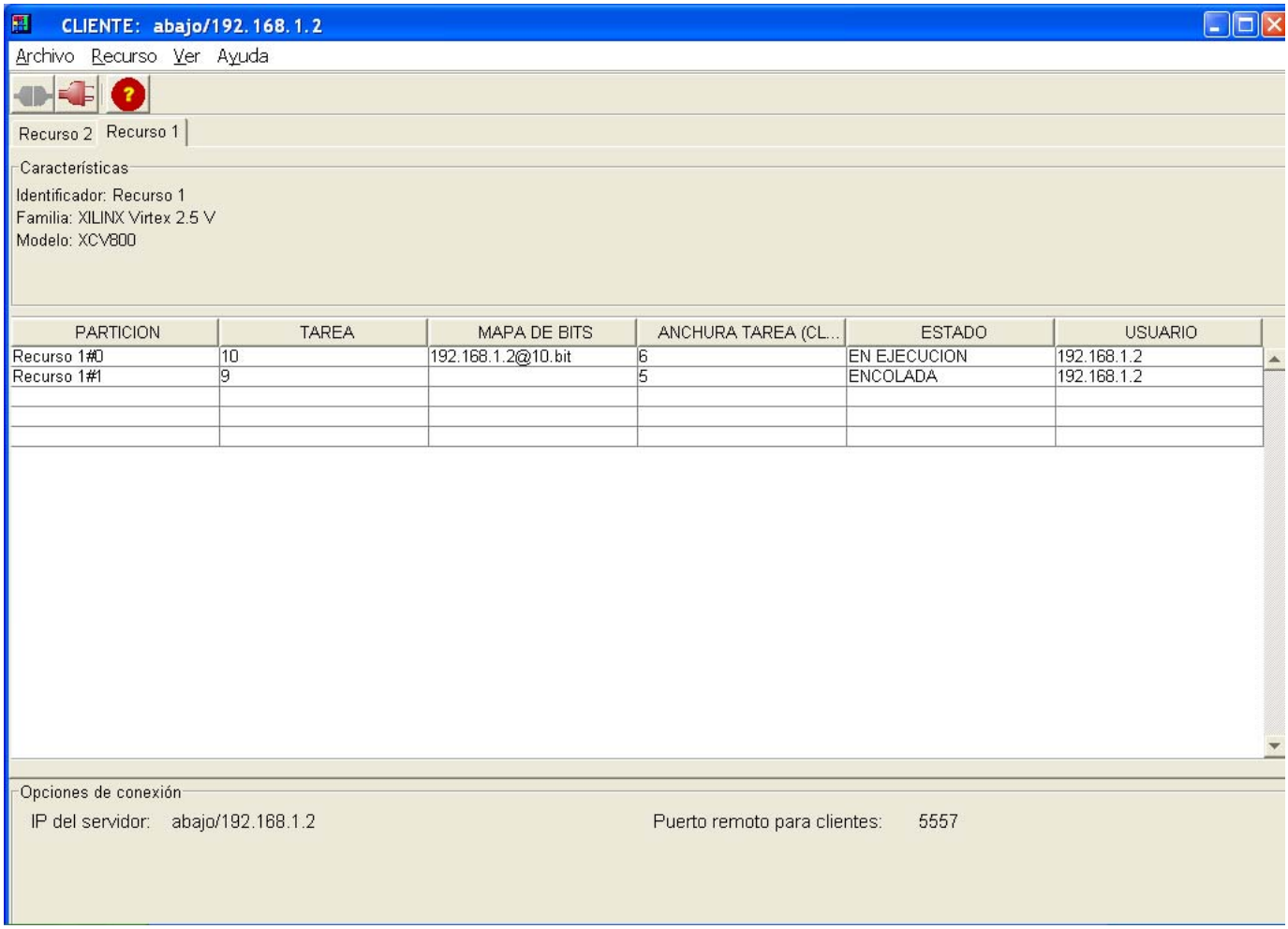
Para dar de alta un recurso elegimos la opción Crear Recurso del menú Recurso. Rellenamos los datos de nuestro recurso en el cuadro de diálogo que aparece a continuación (ver figura).



Para dar de baja un recurso en primer lugar seleccionamos la pestaña del recurso que deseemos y a continuación elegimos la opción Eliminar Recurso del menú Recurso.

c) monitorizar las tareas que realicen nuestros recursos

Todas las actividades de nuestros recursos aparecerán en la tabla de la pestaña apropiada de cada recurso, tal y como se muestra en la siguiente captura de pantalla.



7. POSIBLES MEJORAS Y NUEVOS USOS

En esta sección se incluyen las ampliaciones y las mejoras que se pueden realizar sobre el proyecto de forma sencilla y se explica en líneas generales cómo se podrían llevar a cabo, describiendo qué debería modificarse y cómo, así como los nuevos módulos que serían necesarios. Puesto que durante todo el curso se ha hecho hincapié en generar código reutilizable, cualquier ampliación será fácil de integrar en el proyecto completo.

Al principio de cada sección se identifica el objetivo de la mejora en un párrafo en cursiva. En los siguientes se discute la solución propuesta y el resultado esperado; así como las consecuencias del cambio y posibles modificaciones adicionales.

7.1. GESTIÓN DE FPGAs FÍSICAS (NO SIMULADAS)

La extensión más inmediata del proyecto consiste en su aplicación real sobre FPGA´s físicas. En realidad es necesario muy poco trabajo más, y no se ha incluido en la versión final del proyecto por caer fuera de los objetivos propuestos para este curso. Puesto que el sistema atiende peticiones de usuarios para ejecutar tareas sobre las FPGA´s, podría no simular la ejecución y resultados de las tareas, sino comunicarse mediante un interfaz con un gestor de FPGA´s, lanzar las ejecuciones pedidas y comunicar los resultados reales.

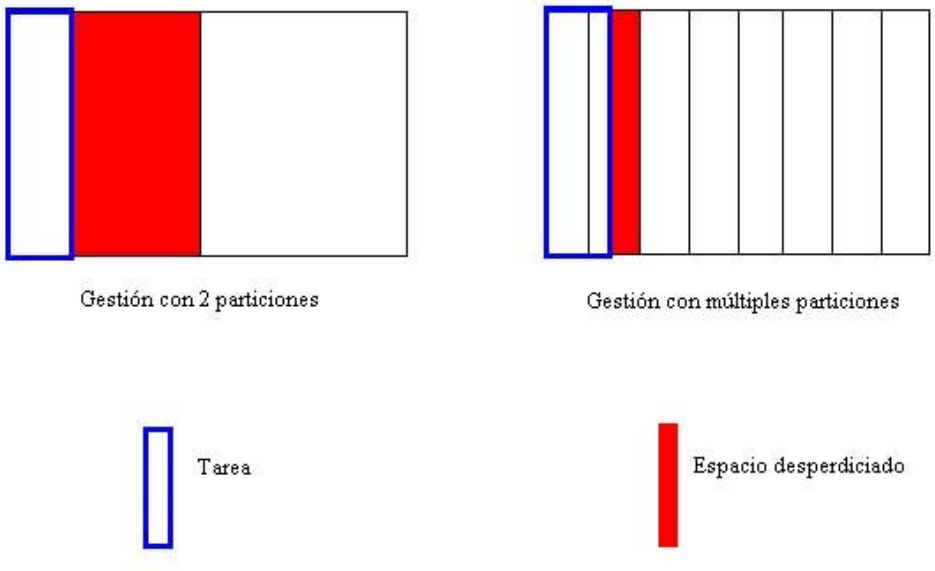
En efecto, bastaría con eliminar del proyecto la ejecución virtual de tareas y conectar con las FPGA´s físicas el sistema. Esto último necesita de un software que el sistema vería como un interfaz Java con unas pocas funciones sencillas, como cargar una tarea en la FPGA, lanzar su ejecución y recibir su resultado. La clase Particion seria la responsable natural de utilizar este interfaz para cumplir, ahora sí realmente, las instrucciones de la clase Recurso.

Por lo que se refiere a la dar de alta recursos en el sistema, el sistema puede atender las FPGA´s reales, aunque el operador debe dar de alta manualmente los recursos que conecte al cliente.

7.2. MÚLTIPLES PARTICIONES POR TAREA

La gestión de los recursos se realiza por un número reducido de particiones grandes. Sin embargo la investigación, por ejemplo del grupo GHaDir de la Universidad Complutense, y

algunos prototipos de FPGA´s industriales prevén una gestión basada en numerosas particiones pequeñas, de forma que se puedan asignar varias a una sola tarea, reduciendo la fragmentación interna (el espacio que se desperdicia al asignar una tarea pequeña a una partición grande).



Aunque a primera vista parezca que el cambio no es sustancial, y que basta con configurar el sistema de forma que permita, digamos 1024 particiones por recurso; no es posible conservar la clase Particion tal y como está diseñada. Esto es así porque cada clase Particion lleva asociada una cola de tareas y las ejecuta virtualmente. Pero lo que se quiere es desligar la cola de tareas de un espacio físico determinado en la FPGA. Por ello es necesario limitar a una sola cola de tareas las que ofrece cada recurso. Esto obliga a disponer de un nuevo módulo autónomo para la ejecución virtual de tareas, y a guardar un registro de cuántos de estos hilos están activos dentro de la clase Recurso.

Disponiendo de recursos que se gestionan con tanto detalle aparecen nuevos problemas, también objeto de los estudios anteriormente mencionados, como la fragmentación o la rotación del uso de áreas para refrescar los transistores; que aunque no son relevantes en nuestro sistema simulador, puede ser conveniente representar también.

7.3. RECONFIGURACIÓN EN TIEMPO DE EJECUCIÓN DE RECURSOS

Nuestro sistema solo permite determinar el modo de gestión de un recurso al darlo de alta. Sin embargo, no es difícil permitir que se determine en un momento arbitrario, quizá mientras está

ejecutando tareas. De hecho algunos productos industriales permiten incluso la reconfiguración con la FPGA ejecutando tareas, sin que estas ejecuciones deban interrumpirse.

Para llevar a cabo esta modificación, no es necesario añadir módulos nuevos; solo dotar de una funcionalidad de reconfiguración a los existentes. Para ello es necesario un mecanismo que bloquee las colas de tareas de un recurso durante la reconfiguración y que informe apropiadamente al servidor de los cambios realizados en su modo de gestión. Durante este proceso deberá mostrar una imagen coherente, por ejemplo, respondiendo a las encuestas con las consideraciones necesarias, como el tiempo extra de reconfiguración que se añade al tiempo de espera total.

La reconfiguración dinámica de recursos no solo permite a un operador humano mejorar la forma en que el sistema se gestiona, sino que dota al sistema mismo de una nueva y poderosa herramienta para optimizar su funcionamiento. El encargado de realizar estas reconfiguraciones más idóneo es el Servidor, ya que dispone de todas las peticiones de los usuarios del sistema para estudiarlas, hacer previsiones estadísticas respecto a los trabajos futuros y reconfigurar los recursos del sistema en consecuencia. Las sofisticaciones deseables en este punto son numerosas y de gran complejidad, aunque todas utilizaran criterios diversos basados en el tráfico presente, y quizá alguna otra consideración marginal (como la velocidad de reconfiguración de cada modelo de FPGA).

7.4. MEJORAS EN LA SEGURIDAD

Las posibles mejoras en aspectos de seguridad son muy variadas según el entorno en el que pretendamos instalar el sistema. El escenario que consideramos ahora es el de unos usuarios indeterminados que acceden al servicio desde cualquier lugar de Internet, y una red de clientes localizados e identificados que puede crecer, centralizada por un servidor. Los aspectos referidos a la protección de cada aplicación frente a operadores inapropiados se estudia más adelante, en la sección Perfiles de Usuarios.

En lo referente a los usuarios, el sistema debe poder asignar una clave privada para cada nuevo usuario que lo identifique unívocamente, y que no pueda ser replicada; para detectar y expulsar los usuarios que hagan un uso inapropiado del sistema. Respecto a los clientes, el mismo sistema de contraseña es útil. Esta sofisticación en las comunicaciones requeriría un protocolo de comunicaciones más cauto, sin embargo queremos hacer notar que todos estos cambios relativos a las comunicaciones bastaría con realizarlos en las clases protocolo; una vez más beneficiándonos del cuidadoso diseño del sistema.

Por último, también sería conveniente establecer un filtro para las tareas en el servidor, que detectara aquéllas que por su naturaleza pudieran resultar perniciosas para el servicio, informara a los usuarios, y tomara medidas contra los que insistieran en perturbar el sistema (por ejemplo con ráfagas de tareas). Aunque en general sería bastante difícil determinar cuándo un usuario está haciendo peticiones nocivas.

7.5. PERFILES DE OPERADOR

En este apartado se estudian las posibilidades de diferenciar entre operadores de cada aplicación mediante una clave personal, y de proporcionar diferentes privilegios a cada uno.

La aplicación que más partido puede sacar de esta mejora es la del servidor. Así tendría como mínimo un administrador con privilegios totales, y por ejemplo otros que solo tengan acceso a la información y no puedan modificar nada del sistema, y otros que puedan modificar cuestiones del sistema de forma limitada. La definición de estos perfiles, así como la base de datos de las contraseñas debería estar almacenada en el servidor, y convenientemente protegida mediante un cifrado del archivo.

También se pueden incluir perfiles en la aplicación cliente, análogamente, un administrador con privilegios totales; y por ejemplo, otro que solo permita consultar la información que proporciona la aplicación mediante su interfaz gráfica.

7.6. RECUPERACIÓN DEL SISTEMA FRENTE A CAIDAS DEL SERVIDOR

Aunque toda red centralizada es especialmente vulnerable a las caídas del servidor, creemos que hay algunas posibilidades para minimizar el impacto que tendría sobre el servicio.

La propuesta consiste en establecer un cliente (que se ejecute en una máquina distinta del servidor) como servidor secundario. Este cliente debe ser estable, y su dirección conocida para todos los actores del sistema, es decir, los demás clientes y todos los usuarios. Puesto que este cliente designado servidor secundario tiene un socket TCP fijo con el servidor, detectará rápidamente una caída del servidor, y lanzará un servidor alternativo en su misma máquina. Hasta que una intervención humana subsane el problema en el servidor, el servidor alternativo atenderá las nuevas peticiones de los usuarios, que deberán darse de alta nuevamente. Sin embargo, las tareas que estuvieran en el sistema no se perderán, ya que cada cliente es responsable absoluto de las tareas que le han sido asignadas.

8. GLOSARIO DE TÉRMINOS

8.1. TÉRMINOLOGÍA BÁSICA

En este apartado detallaremos los términos que han ido apareciendo a lo largo de este documento referidos a un uso básico del sistema. Fundamentalmente distinguiremos los nombres dados a las diferentes aplicaciones y actores que intervienen en el sistema.

Aplicación Usuario

Aplicación que permite definir tareas y lotes de tareas al usuario (operador humano) y solicitar su ejecución.

Aplicación Cliente

Aplicación que ejecuta las tareas y lotes de tareas del usuario.

Aplicación Servidor

Aplicación autónoma (sin intervención humana) que distribuye automáticamente las tareas solicitadas por los usuarios a los recursos de los clientes.

Cliente

Equipo en el que se ejecuta la Aplicación Cliente.

Servidor

Equipo en el que se ejecuta la Aplicación Servidor.

Usuario

Equipo en el que se ejecuta la Aplicación Usuario.

Además, utilizamos este término para referirnos al operador humano que controla cualquiera de las tres aplicaciones (Usuario, Cliente, Servidor) que constituyen el sistema.

8.2. TÉRMINOLOGÍA AVANZADA

En este apartado detallaremos los términos que han ido apareciendo a lo largo de este documento referidos a un uso avanzado del sistema. Fundamentalmente estará destinado a

programadores que en un futuro decidan basarse en nuestro proyecto para desarrollar otra aplicación.

Fragmentación

La fragmentación (en el contexto de las FPGAs) la podemos considerar de una manera simplificada como un desaprovechamiento en el uso de la superficie útil (matriz de CLBs) de una FPGA.

Existen dos tipos de fragmentación:

- Fragmentación interna: Fenómeno por el cual se malgasta espacio interno de una partición cuando el bloque de datos cargado (en nuestro caso, la tarea) es más pequeño que la partición.
  Se produce siempre que se trabaje con particiones de tamaño fijo.

- Fragmentación externa: Fenómeno por el cual se malgasta el espacio externo a una partición (es decir, el espacio entre particiones) debido al gran número de huecos pequeños que quedan entre las mismas.

Protocolo de red

Se le llama protocolo de red o protocolo de comunicación al conjunto de reglas que controlan la secuencia de mensajes que ocurren durante una comunicación entre entidades que forman una red. En el contexto de nuestro proyecto, las entidades de las cuales se habla son programas de computadora.

Los protocolos de red establecen aspectos tales como:

- Las secuencias posibles de mensajes que pueden arribar durante el proceso de la comunicación.
- La sintaxis de los mensajes intercambiados.
- Estrategias para corregir los casos de error.
- Estrategias para asegurar la seguridad (autenticación, encriptación).

Socket (en Java)

El interfaz de sockets es un interfaz normalizado de comunicación utilizado inicialmente en el protocolo TCP/IP.

Por otra parte, es el extremo de un enlace de comunicaciones bidireccional entre dos procesos. El socket consta de una dirección IP y un número de puerto que identifica el servicio.

La misión de un socket es mandar / recibir un flujo de datos entre aplicaciones que pueden estar o no en la misma máquina.

Existen dos tipos de sockets: TCP y UDP

- TCP sockets (En Java implementados en la clase Socket): Ofrecen una comunicación fiable y libre de errores, garantizando que los mensajes llegarán ordenados, sin duplicados y sin pérdidas. Antes de comenzar la transmisión necesitan una fase previa de establecimiento de la conexión.

- UDP sockets (En Java implementados en la clase DatagramSocket): Ofrecen un servicio no fiable de comunicación. Los paquetes pueden llegar duplicados, desordenados, o pueden perderse sin llegar a su destino. La comunicación es mucho más rápida que en los TCP Sockets, y no necesitan una fase de establecimiento de conexión.

TCP/IP

Conjunto básico de protocolos de comunicación de redes, popularizado por Internet, que permiten la transmisión de información en redes de computadoras. El nombre TCP/IP proviene de dos protocolos importantes de la familia, el *Transmission Control Protocol (TCP)* y el *Internet Protocol* (IP).

Por tanto, TCP/IP es un conjunto de protocolos de comunicaciones que definen cómo se pueden comunicar entre sí ordenadores y otros dispositivos de distinto tipo.

XML

XML es un metalenguaje (un lenguaje que sirve par especificar otros lenguajes) que es una simplificación de SGML.

XML se utiliza para describir documentos estructurados, de manera que los datos (información) del documento vienen estructurados por las marcas (elementos) del mismo. Estas marcas son identificadores encerrados entre los símbolos < y >. Por tanto, en XML se cumple la ecuación

Documento = contenidos (datos) + marcas

Conceptos relacionados con XML:

- Documento bien formado: Son los documentos que cumplen las reglas básicas de XML y que pueden ser procesados por un programa.

Las reglas básicas son:

- Debe haber exactamente un elemento raíz. Además, la primera marca del documento es la marca del elemento raíz
- Cada elemento debe tener una marca de inicio y una marca de fin, excepto los elementos vacíos que pueden indicarse con una marca especial
- El nombre de la marca de inicio debe coincidir con el de su marca de fin
- Los elementos deben estar adecuadamente anidados
- No puede aparecer un atributo más de una vez en un mismo elemento
- El valor de los atributos debe ir entre comillas dobles

- Documento válido: Documento XML bien formado que además sigue las reglas especificadas en una Definición de tipo de Documento (DTD) o en un esquema de documento.

9. BIBLIOGRAFÍA

- **Grid:**

IAN FOSTER, CARL KESSELMAN Y STEVEN TUECKE , “The Anatomy of the Grid”.

NLANR .(2000), "Beginner's Guide to Network-Distributed Resource Usage".

- **FPGAs:**

H. KALTE, M.PORRMANN, U.RÜCKERT (2004), "System-on-Programmable-Chip Approach Enabling Online Fine-Grained 1D-Placement".

TAREK EL-GHAZAWI, KRIS GAJ, NIK ALEXANDRIDIS, BRIAN SCHOTT, "Efective Use of Networked Reconfigurable Resources".

Web de Xilinx
http://www.xilinx.com/

- **Programación en Java:**

SANCHEZ ALLENDE J., HUECAS FERNANDEZ-TORIBIO G., FERNÁNDEZ MANJÓN B. y MORENO DÍAZ P. (2001), “Java 2. Iniciación y Referencia”, McGraw-Hill.

“Programación Java™. Guía del usuario” (2001), Sun Mycrosystems.

Herramientas básicas de Java
http://java.sun.com/j2se/1.4.2/docs/tooldocs/tools.html#basic

Tutorial de SAX (Simple API for XML)
http://java.sun.com/xml/jaxp/dist/1.1/docs/tutorial/sax/index.html

Foro Sun Mycrosystems
http://forum.java.sun.com/index.jspa

Foro javaworld.com
http://www.javaworld.com/javaforums

Foro javaalmanac.com
http://javaalmanac.com/

Foro JavaDesktop
www.javadesktop.org/forums/

- **Redes de computadores:**

STALLINGS W. (2000), “Comunicación y Redes de Computadores”, Prentice Hall.

Wikipedia
http://es.wikipedia.org/wiki/Portada

Axarnet
http://fmc.axarnet.es/redes/tema_06.htm

Universitat Jaume I
http://www3.uji.es/~vcholvi/teaching/java/cap.11.1/sockets.html

- **Modelos de proceso:**

PRESSMAN R. S. (2002), “Ingeniería del Software. Un enfoque práctico”, McGraw-Hill.

- **Programación en HTML:**

ÁLVAREZ GARCÍA A. (1999), “HTML. Creación de Páginas Web”, Anaya Multimedia.