Programación Concurrente
Trabajo de asignatura
Un juego de dominó distribuido

Escuela de Ingeniería y Arquitectura
Depto. de Informática e Ingeniería de Sistemas
Curso 11–12

1. Objetivos

Los objetivos de este trabajo de asignatura son los siguientes:

- Programar una aplicación distribuida con arquitectura cliente-servidor.
- Profundizar en el modelo de comunicación síncrona para procesos distribuidos.
- Asentar los conocimientos adquiridos para la programación de aplicaciones concurrentes y distribuidas en Ada.
- Trabajar en equipo.

2. Una aplicación distribuida para el juego del dominó

El objetivo concreto de este trabajo de asignatura es desarrollar una aplicación *software* para un juego de dominó en red. En primer lugar, en esta sección se resumen las reglas básicas que regulan el juego del dominó. Posteriormente, se describe la estructura de la aplicación que debe ser implementada.

2.1. Reglas del juego

Existen muchas versiones diferentes del juego del dominó. Dado que es un juego conocido, el objetivo de esta sección no es presentar en detalle todos los aspectos del juego, sino determinar el conjunto de reglas básicas que serán aplicadas en la implementación de la aplicación objeto de este trabajo.

Se propone una versión simplificada del juego. El objetivo de un jugador es ganar la partida conforme los criterios que se exponen a continuación. Las reglas de juego que deberán ser aplicadas son las siguientes:

- Los jugadores van a jugar individualmente (pese a que en las versiones más clásicas del juego, estos se organizan por parejas).
- Una vez que cuatro jugadores hayan expresado su interés en participar en una partida, las 28 fichas del dominó serán repartidas al azar (7 fichas a cada jugador).
- El último jugador que haya manifestado su interés en jugar será quien comience la partida (podrá colocar una ficha cualquiera, sin necesidad de que sea doble), continuando el turno por el jugador situado *a su derecha*.
- En su turno, cada jugador colocará una de sus piezas con la restricción de que dos piezas solo pueden colocarse juntas cuando los cuadrados adyacentes son del mismo valor.
- Si un jugador no puede colocar ninguna ficha en su turno, tendrá que pasar el turno al siguiente jugador.
- El final de la partida puede venir determinado por dos situaciones: (1) un jugador colocó la última de sus fichas y, por tanto, ganó la partida; o (2) todos los jugadores tienen aún fichas, pero ninguno puede colocar ninguna de ellas. En este último caso, ganará la partida el jugador cuyas fichas sumen menos puntos.

2.2. Diseño de alto nivel de la aplicación

El objetivo principal de este trabajo es la implementación de una **aplicación distribuida para el juego del dominó**. Esta aplicación será diseñada e implementada con una arquitectura cliente-servidor. Un despliegue concreto de la aplicación constará de un *servidor* y cuatro *clientes*. El servidor será el responsable de gestionar el desarrollo de una partida completa de dominó: determinar el inicio de la partida, repartir inicialmente las fichas entre los jugadores, mantener el estado actual de la partida (es decir, relación de fichas colocadas sobre el tapete), gestionar el turno de los jugadores, comprobar que un jugador pone una ficha de acuerdo a las reglas del juego y, finalmente, informar de la finalización de la partida y su correspondiente resultado. Por otro lado, cada cliente implementará la lógica de juego de un jugador de dominó. En otras palabras, un cliente

expresará su deseo de participar en una partida de dominó, recibirá las fichas con las que va a jugar y, cuando sea su turno, determinará qué ficha es la más adecuada colocar sobre el tapete con el objetivo final de ganar la partida. Por simplicidad, la aplicación a construir solo comenzará una partida cuando cuatro jugadores se hayan registrado.

3. Pautas para la implementación de la aplicación

En esta sección se describen las pautas que deben ser consideradas para el diseño y la implementación de los dos tipos de procesos que conforman la aplicación. La figura 1 muestra la estructura interna de los clientes y el servidor que deben ser desarrollados. En las dos siguientes secciones se describe en detalle cada uno de ellos. También se presenta el protocolo de comunicación establecido entre los procesos (flechas discontinuas en la figura), dado que este garantizará la correcta integración de los mismos.

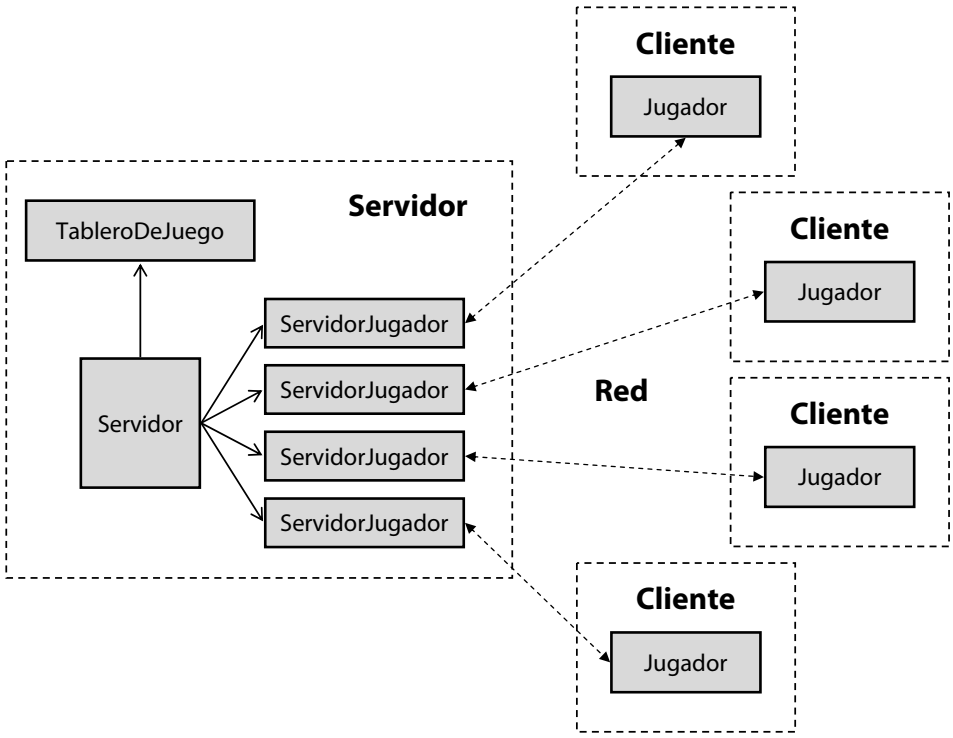

Figura 1: Diseño de la aplicación completa

3.1. Implementación del servidor

El servidor de la aplicación constará como mínimo de un programa Ada denominado `Servidor`, un tipo de tarea denominado `servidorJugador` y un tipo abstracto de datos

denominado `tableroDeJuego`. En los siguientes párrafos se explica la responsabilidad y el diseño de cada una de ellas.

El TAD `tableroDeJuego` es responsable de representar y gestionar el estado de una partida. A nivel de datos, internamente almacena el turno del siguiente jugador y las fichas colocadas sobre el tapete (qué fichas han sido colocadas y en qué orden). La parte pública del TAD, como mínimo, debe ofrecer procedimientos y funciones para: repartir fichas a un jugador al inicio de la partida, determinar si es el turno de un jugador, conocer el estado del tablero de juego (las fichas colocadas), colocar una nueva ficha (comprobando si es correcta la ficha que se desea colocar) y, finalmente, determinar si una partida ha finalizado y, si es así, qué jugador ha ganado la partida. Este diseño es básico y, por tanto, puede ser ampliado con todo aquello que se considere necesario para la adecuada representación de la parida y la gestión de su desarrollo.

El programa `servidor` implementa la aplicación servidor. Dentro de este programa se declarará una variable del TAD `tableroDeJuego` que represente la partida. Se solicitará al usuario que introduzca por teclado el puerto que usará el servidor (para facilitar aspectos de configuración). Este puerto será utilizado para crear un *socket* servidor a través del cual se aceptarán las peticiones de juego de los jugadores. Cada vez que un cliente remoto solicite participar en la partida se llevarán a cabo las siguientes acciones: creará un *socket* cliente para gestionar la conexión con el jugador remoto, se le asignará un identificador de jugador único (comprendido entre los valores 1 y 4 y asignado por orden de llegada) y se citará con una tarea del tipo `servidorJugador` que gestionará en el lado del servidor la comunicación con el jugador remoto. Esta cita tendrá como parámetros el *socket* cliente, el identificador del jugador y un puntero al tablero de juego gestionado por el servidor.

Las tareas del tipo `servidorJugador` gestionan en el lado del servidor la actividad de un jugador. Como mínimo, cada tarea definirá las siguientes variables: el correspondiente *socket* cliente, un canal síncrono de lectura y escritura (*stream*) para comunicarse con el cliente remoto al que representa en el lado del servidor, la identidad del jugador al que representa y un puntero al tablero de juego. Por otro lado, la tarea tendrá ofrecerá como mínimo una cita de inicialización, responsable de inicializar las variables anteriores, prestando especial atención al establecimiento del canal de comunicación. A continuación se describe en alto nivel el algoritmo de las tareas:

Una vez la partida haya finalizado, los *sockets* serán cerrados de forma adecuada.

3.2. Implementación del cliente

La aplicación cliente implementa un jugador remoto. Desde el punto de vista de diseño, será un programa Ada llamado `jugador`. El programa contará como mínimo con un procedimiento denominado `siguienteJugada` responsable de determinar cuál será la siguiente ficha a colocar cada vez que sea el turno del jugador. Este último procedimiento

Algoritmo 1 Algoritmo en alto nivel de las tareas `servidorJugador`

Cita de inicialización con el programa servidor
Envío al jugador de su identificador en la partida
Envío al jugador de sus fichas iniciales
mientras que ¬(*final de partida*) **hacer**
  Notificación al jugador de que es su turno de juego
  Envío al jugador del estado de la partida
  Recepción del jugador de la jugada a ejecutar
  Ejecución de la jugada sobre el tablero
  Envío al jugador del resultado de la jugada
  Paso del el turno
fmq

encapsulará la *inteligencia de juego* de cada jugador concreto implementado. En su versión más simple, un jugador podría decidir colocar siempre una ficha cualquiera al azar de entre fichas que sea posible colocar.

A continuación se describe en detalle el algoritmo de alto nivel del programa `jugador`. Inicialmente, deberá pedirle al usuario que introduzca por teclado la dirección IP y el puerto del servidor del juego (para facilitar la configuración del cliente). Estos datos serán utilizados para crear un *socket* que será posteriormente utilizado para obtener el canal de comunicación con la tarea *servidorJugador* que representa al jugador cliente en el lado del servidor. El algoritmo de alto nivel de este programa es el siguiente:

Algoritmo 2 Algoritmo en alto nivel del programa `jugador`

Petición de IP y puerto del servidor
Creación de un socket
Recepción de su identificador en la partida
Recepción de sus fichas iniciales
mientras que ¬ (*fin de partida*) **hacer**
  Recepción de la notificación de turno de juego
  Recepción del estado de la partida
  siguienteJugada(. . .)
  Envío al servidor de la jugada a ejecutar
  Recepción del resultado de la jugada
  Actualización de sus fichas restantes
fmq
Cierre del socket

Como se puede comprobar, como parte de este algoritmo es invocado el procedimiento `siguienteJugada` para determinar cuál será la siguiente ficha que colocará el jugador. Los parámetros de entrada de este procedimiento como mínimo serán las fichas que aún tiene el jugador pendientes de colocar y el estado actual de la partida (qué fichas ya han sido colocadas por los jugadores y en qué orden han sido colocadas). Basándose en esta

información de entrada y otro tipo de información que localmente pueda gestionar o recopilar una implementación concreta del jugador, el procedimiento determinará como resultado qué ficha es recomendable que coloque el jugador con el objetivo final de ganar la partida. La implementación de este procedimiento puede ser muy simple (tirar una ficha al azar entre las posibles) o muy compleja, utilizando estrategias que tengan en cuenta lo que pasó en la partida y lo que se prevea que ocurra en función de la ficha se juegue. No es objetivo de este trabajo desarrollar estrategias de juego especialmente complejas, pero sí deseable que la implementación de jugadores competitivos.

Cuando la partida haya finalizado, las conexiones con el servidor serán cerradas de forma adecuada.

3.3. Protocolo de comunicación para el desarrollo del juego

En esta sección se describe en detalle el protocolo de comunicación entre el servidor y sus clientes. El nivel de descripción corresponde con cada mensaje concreto que es intercambiado entre ambos tipos de procesos. Desde un punto de vista de implementación en Ada, recordad que cada mensaje enviado a través de un canal de comunicación (*stream*) va a ser una cadena de caracteres (`string`). Por tanto, cuando se habla de mensaje se está haciendo referencia a la cadena de caracteres que contiene la información del mensaje intercambiado.

3.3.1. Envío del identificador que el servidor asigna a un cliente en una partida

El mensaje enviado por el servidor se ajustará al siguiente formato: `"IDENTIDAD=valor"`, donde *valor* será sustituido por el identificador numérico concreto (entre 1 y 4) asignado al cliente.

3.3.2. Envío de las fichas inicialmente repartidas a un jugador

Cada ficha concreta se representará con el siguiente formato: `"|valor1|valor2|"`, donde *valor1* y *valor2* son los valores concretos de cada extremo de la ficha (las barras verticales actúan como delimitadores de la información de una ficha). Obviamente, *valor1* y *valor2* deberán tener un valor numérico comprendido entre 0 y 6. El mensaje enviado por el servidor al cliente con sus fichas iniciales será una secuencia de fichas, de acuerdo al formato anterior. Por ejemplo, el mensaje `"|2|3||5|2||3|0|"` representa que las fichas `|2|3|`, `|5|2|` y `|3|0|` han sido repartidas al jugador (recordad que se le repartirán siete fichas a cada jugador, aunque en el ejemplo por simplicidad solo se han representado como parte del mensaje tres fichas).

3.3.3. El servidor notifica a un jugador que es su turno

El mensaje enviado desde el servidor al cliente para notificarle que es su turno de juego se ajustará al siguiente formato: `"ES_TU_TURNO"`.

3.3.4. Un jugador interroga al servidor para conocer el estado de la partida (el tablero)

Cuando un cliente quiera conocer el estado de la partida, es decir, qué fichas han sido colocadas sobre el tapete y en qué orden, enviará al servidor un mensaje con el siguiente formato: `"ESTADO_PARTIDA"`. La respuesta por parte del servidor será una cadena de texto formada por la secuencia de fichas que representan el estado de la partida. Por ejemplo, la respuesta `"|2|1||1|1||1|4||4|4|"` representa que hay cuatro fichas colocadas, concretamente, las fichas `|2|1|`, `|1|1|`, `|1|4|` y `|4|4|`. Por tanto, el jugador deberá tratar de poner una ficha que tenga como valor un 2 (por la izquierda) o un 4 (por la derecha).

3.3.5. Un jugador envía al servidor su jugada

Una vez el jugador conozca el estado de la partida, decidirá qué ficha va a colocar. Para ello enviará al servidor un mensaje con el siguiente formato:
`"COLOCAR_FICHA=|valor1|valor2|,extremo"`, donde *valor1* y *valor2* representan los valores de la ficha y *extremo* significa si se desea colocar por la derecha (valor de *extremo* `"DERECHO"`) o por la izquierda (valor de *extremo* `"IZQUIERDO"`) respecto la secuencia de fichas que representa el estado de la partida. El servidor analizará si es legal colocar la ficha indicada, en función de la situación actual de la partida, y devolverá el mensaje `"CORRECTO"` si es legal o `"ERROR"` si no lo es.

En el caso de que el jugador no pueda colocar ninguna ficha, enviará el mensaje `"PASO"`, a lo que el servidor contestará con el mensaje `"CORRECTO"`.

3.3.6. Un jugador desea comprobar si es final de partida

El mensaje enviado por parte del cliente al servidor para conocer si es final de partida se ajusta al siguiente formato: `"ES_FINAL_PARTIDA`. El servidor responderá con el mensaje `"SI"` o `"NO"` en función del estado de la misma.

4. Instrucciones para el desarrollo y la entrega del trabajo

4.1. Organización inicial del trabajo

El trabajo tiene una naturaleza *no presencial*. Los alumnos se organizarán en equipos de dos, a través de sorteo realizado en la sesión de prácticas del día 24-5-2012. Cada pareja deberá realizar una implementación en Ada del servidor y los clientes, conforme a las pautas de implementación descritas en las secciones previas.

4.2. Ficheros a entregar como resultado

Cuando se finalice el trabajo, se deberá entregar un fichero comprimido en formato ZIP con el siguiente contenido:

1. Un fichero de texto denominado *autores.txt* que contendrá el NIP, los apellidos y el nombre de los dos autores del trabajo.

   A continuación, se incluirá en dicho fichero un resumen de las principales dificultades encontradas durante la realización del trabajo.

2. Los ficheros correspondientes a los ficheros fuente Ada, incluyendo «`servidor.adb`», «`jugador.adb`», «`tableroDeJuego.ads`», «`tableroDeJuego.adb`» y todos aquellos ficheros que sean necesarios para compilar los programas `jugador` y `servidor`.

4.3. Procedimiento y fechas de entrega de la práctica

Para la entrega del fichero ZIP con el contenido descrito anteriormente, se mandará un correo electrónico al profesor de prácticas (`latre@unizar.es`). La fecha límite para enviar el trabajo es el jueves 7 de junio de 2012.