Plataforma basada en mensajes para el desarrollo de juegos para grandes redes

Matías Ignacio San Martín[1], Juan Gabriel Grande[1]

[1] Facultad de Ingeniería, Universidad de Buenos Aires,
Paseo Colón 850, C1063ACV, C.A.B.A, Argentina
juan.grande@gmail.com

Abstract. En este artículo proponemos una arquitectura para el desarrollo de juegos en red e implementamos la plataforma correspondiente basada en esta. La carga de procesamiento y comunicaciones se distribuye entre varios nodos, en donde cada uno se encarga de mantener una parte del estado global del juego. Estos nodos difunden las novedades a los jugadores a través de un protocolo de grupos. Se desarrolló una interfaz de programación para acceder a los servicios brindados por la plataforma que permite construir la aplicación sin necesidad de conocer los detalles de la arquitectura, simplificando significativamente la implementación del juego. Además, se desarrolló el prototipo de un juego para comprobar la validez de la plataforma y su correcto funcionamiento. El prototipo demostró que la arquitectura propuesta es una buena solución de compromiso entre el requerimiento de mantener una visión coherente del estado del juego y las necesidades de recursos, tales como la capacidad de procesamiento y comunicaciones; además se destaca la facilidad de desarrollo e integración que ofrece el uso de esta plataforma.

Keywords: sistemas distribuidos, juego, multijugador, middleware, mensajería de grupos

1 Introducción

Los dos modelos de arquitectura más utilizados actualmente para juegos en red son cliente-servidor y peer to peer [1].

En el modelo cliente-servidor se distinguen dos tipos de nodos, el cliente y el servidor. El servidor recibe las acciones de todos los jugadores, calcula el nuevo estado global, y lo difunde. Los juegos suelen tener una gran cantidad de interacciones entre los participantes y el estado global, por lo que requieren el uso de un servidor con gran capacidad de procesamiento y comunicaciones, pero tienen la ventaja de establecer muy pocas restricciones para los clientes. Una forma de disminuir los requerimientos de capacidad de procesamiento y comunicaciones impuestos al servidor es utilizar varios servidores (lo que se denomina red o pool de

servidores [1]), de este modo, cada uno de los servidores se encarga de un grupo de los clientes.

En una arquitectura basada en el modelo peer to peer (P2P), en el cual todos los nodos son del mismo tipo, cada jugador envía sus acciones a todos los demás, que son los encargados de calcular el nuevo estado del juego. El uso de este modelo de arquitectura para el desarrollo de juegos hace que no sea necesario depender de un equipo con gran capacidad de procesamiento, pero impone mayores requerimientos de procesamiento y comunicaciones en los participantes. Otra desventaja de usar el modelo P2P es que, por tratarse de un modelo totalmente distribuido, resulta complejo el mantenimiento de un estado consistente entre todos los participantes [2].

En [2] se propone una arquitectura mixta denominada peer to peer con árbitro central. Esta arquitectura consiste en agregarle a la arquitectura peer to peer un árbitro central que recibe las acciones de todos los jugadores. Cuando un jugador realiza una acción que genera una inconsistencia en el estado, el árbitro corrige esa inconsistencia e informa a todos los jugadores. De esta forma, el procesamiento en los jugadores se reduce, ya que no tienen que verificar las inconsistencias. Esta propuesta cuenta con la desventaja de que el juego puede permanecer inconsistente por ciertos períodos de tiempo y, además, requiere que el árbitro central mantenga una visión global del juego para poder hacer la verificación de consistencia.

En el presente trabajo proponemos el uso de un conjunto de nodos para mantener y asegurar el estado consistente del juego; estos nodos solo requieren procesar los mensajes correspondientes a la parte de estado global que le corresponde, reduciéndose así la capacidad requerida para estos, pero además asegurando que la consistencia del juego se mantiene en todo momento. Para la comunicación entre los nodos se utiliza un modelo P2P, lo que les permite mantenerse coherentes unos con otros. Para comprobar el funcionamiento de la plataforma propuesta tomaremos como ejemplo un juego específico. Se desarrolló un middleware que implementa una abstracción de la arquitectura, y un prototipo de aplicación (juego). El middleware es un Message-Oriented Middleware [3] asincrónico, que brinda servicios de mensajería punto a punto y de grupo.

En la sección 2 describimos el juego que inspiró este trabajo. En la sección 3 proponemos una arquitectura que satisface los requerimientos del juego. En la sección 4 enumeramos cuales servicios debe brindar una interfaz de programación para una plataforma que implemente dicha arquitectura, y describimos la implementación del mismo. En la sección 5 se compara la arquitectura propuesta contra las arquitecturas cliente-servidor y peer to peer en términos de las comunicaciones necesarias en cada nodo. La sección 6 concluye este artículo.

2 Descripción del Juego

El juego elegido para la prueba de la plataforma propuesta es el Galcon [4]. Este juego contiene básicamente dos elementos, los planetas y las flotas.

Los planetas son las entidades encargadas de crear naves. Cada planeta tiene un tamaño definido y, proporcional a este tamaño, crea una cierta cantidad de naves a medida que pasa el tiempo. Un planeta puede pertenecer a un jugador, ya sea porque este lo conquistó o porque le fue asignado al ingresar al juego. Un planeta también puede estar libre (no pertenecer a ningún jugador), en cuyo caso no creará naves (aunque contendrá una cantidad de naves asignada al comenzar el juego). Para que un jugador conquiste un planeta, este debe hacer arribar (por lo menos) una nave más que la cantidad de naves que tenga el planeta. Solamente un jugador puede ser dueño de un planeta en un determinado momento.

Las flotas son entidades que representan un conjunto de naves de un jugador. Una flota es generada en un planeta (origen) como un porcentaje (sin superar al 100%) de la cantidad de naves que tiene este al momento en que se le indica que cree la flota. Las naves de una flota avanzan continuamente hacia un planeta (destino) determinado; el destino puede ser modificado por orden del usuario en cualquier momento previo al arribo de la flota (llegada de su última nave), en cuyo caso, las naves restantes de la flota cambiarán su rumbo hacia el nuevo destino.

Un juego en progreso tiene una cierta cantidad fija de planetas distribuidos en posiciones estáticas en todo el espacio. El tiempo que tarde una flota en llegar de un planeta a otro, será proporcional a la distancia entre los planetas. Al llegar una nave a un planeta, esta incrementará o decrementará en uno la cantidad de naves en el planeta, dependiendo de si el planeta pertenece o no al mismo jugador que la nave, respectivamente. El juego se termina cuando un jugador se adueña de todos los planetas; siendo declarado este, ganador.

3 Arquitectura Propuesta

En este trabajo llamaremos servidores a los nodos encargados de mantener el estado del juego, tanto para describir la arquitectura como la plataforma que la implementa, y llamaremos clientes a los nodos encargados de mostrar ese estado a los usuarios y generar nuevas acciones.

Nuestra propuesta es dividir el estado global del juego en M particiones disjuntas, y asignar cada una de ellas a un servidor. Cuando un cliente realiza una acción debe determinar la partición del estado que afecta y enviar dicha acción al servidor correspondiente. Cada acción realizada se envía a un único servidor. Este servidor actualiza su parte del estado y la difunde a todos los clientes mediante mensajes asincrónicos. Si la acción afecta a más de una partición, entonces el servidor que recibió la acción inicialmente debe difundirla a los demás que estén involucrados.

3.1 Utilización en el Juego

Para aplicar la arquitectura propuesta al juego descripto en la sección 2, definiremos que el estado global corresponde al universo del juego. Además, cada servidor se

encargará de mantener el estado de un planeta (es decir, se toma cada planeta como único elemento de una partición). Finalmente, cada jugador corresponde a un cliente.

En cualquier momento del juego, un jugador puede decidir crear una flota, con origen en un planeta suyo, y con destino a cualquier otro planeta. En este caso, se envía un mensaje al planeta origen, informando la creación de la flota. El planeta calcula el tamaño de la misma, realiza algunas verificaciones de consistencia, y luego le informa el evento al resto de los jugadores. Luego, el origen le informa al destino la creación de la flota. Es el planeta destino quien se encargará de calcular e informar el arribo de cada una de las naves.

Una vez creada la flota, el jugador puede decidir cambiar su rumbo. En este caso, se enviará un mensaje al planeta destino actual, informándole cuál es el nuevo destino. El primero se encargará de calcular el estado actual de la flota, verificar la consistencia de la operación, informar al resto de los jugadores acerca del cambio, y finalmente informar al nuevo destino.

Por otro lado, a lo largo del juego, los planetas envían en forma periódica mensajes a todos los jugadores, informando la cantidad de naves que hay en ellos (asegurando que los jugadores mantienen una visión coherente del estado del juego). Además, cuando un planeta es conquistado por un jugador, o bien cuando su dueño abandona la partida, el mismo informa la novedad al resto de los jugadores.

3.2 Clases de Actividades Distribuidas

En la figura 1 analizamos la arquitectura anterior utilizando las siguientes clases de actividades [5]:

- el despacho consiste en asignar las tareas a los participantes
- la difusión consiste en hacer llegar las tareas a los participantes correspondientes
- la serialización consiste en ordenar las tareas para que puedan ser procesadas por un único participante
- la ejecución consiste en llevar adelante la tarea

Los rectángulos de línea punteada representan los distintos sitios, donde Jx son los jugadores y Px son los planetas.

Los jugadores son los encargados de, a partir de la interacción del usuario con el juego, generar las acciones, determinar el planeta que se debe notificar (despacho) y enviar el aviso a dicho planeta (difusión). El planeta recibe acciones de varios jugadores simultáneamente, por lo tanto debe serializarlas antes de ejecutarlas. Finalmente, se deberán serializar y difundir las respuestas generadas por los planetas. Por simplicidad, solo se incluyen en la figura las interacciones disparadas por la acción de un jugador.

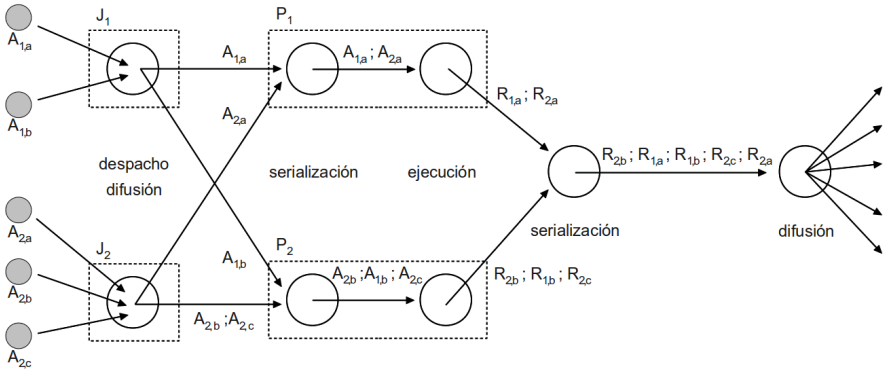

Fig. 1. En esta figura se muestran las clases de actividades distribuidas involucradas en el juego.

3.3 Directorio

La inicialización de una partida puede descomponerse en dos fases. La primera fase comienza con la creación de un universo. Durante esta fase, se incorporan planetas al universo. Cada universo creado tiene un tiempo de vencimiento (dado en minutos) y una cantidad máxima y mínima de planetas. Una vez transcurrido (desde la creación del universo) el tiempo de vencimiento y, si la cantidad de planetas inscriptos alcanza al mínimo, entonces el mismo es puesto a disposición para albergar una partida. En caso contrario, toda la información del universo es destruida.

La segunda fase comienza con la creación de una partida asociada a uno de los universos disponibles, y definiendo una cantidad mínima de participantes para la misma. A partir de allí, pueden inscribirse jugadores. Una vez alcanzada la cantidad mínima de participantes, comienza el juego.

Para cumplir con estos requerimientos, es necesario desarrollar un componente que llamaremos directorio, y que debe brindar, al menos, los siguientes servicios:

1. Registrar un nuevo universo.
2. Consultar los universos registrados.
3. Incorporar un planeta a un universo.
4. Consultar los universos disponibles para albergar partidas.
5. Registrar una nueva partida.
6. Consultar las partidas disponibles.
7. Incorporarse a una partida.

En base a la definición recién realizada, podemos listar las características que limitarán su implementación:

- Solo es utilizado durante la inicialización de las partidas.
- Debe estar en una ubicación bien conocida, de forma tal que cualquier persona con intención de incorporarse a una partida pueda encontrarlo.
- Se debe garantizar exclusión mutua entre las operaciones.
- Las operaciones son del tipo request-reply.

Hemos tomado la decisión de utilizar Remote Procedure Call (RPC) [6] para implementar este componente ya que cumple con las especificaciones dadas y, además, facilita ampliamente su desarrollo. RPC es un protocolo que permite invocar procedimientos en equipos remotos de forma muy similar a como se invoca un procedimiento en un lenguaje de programación.

4. Plataforma

4.1 Requerimientos de la Aplicación

Para definir los servicios de mensajería que debe brindar el middleware, adoptamos una estrategia de tipo top-down. En base a las necesidades de la aplicación, identificaremos cuáles son los servicios que debe proveer el middleware.

El primer servicio de mensajería corresponde a los mensajes unicast enviados desde un jugador a un planeta. Estos mensajes corresponden a las órdenes generadas en los jugadores para modificar el comportamiento de un planeta (por ejemplo, crear o modificar el rumbo de una flota). Dado que estas acciones deben procesarse en el orden en que los jugadores las generan, es imperativo que el protocolo utilizado para enviar los mensajes garantice orden causal [5]. Existe orden causal cuando todos los mensajes enviados entre dos participantes preservan el orden.

El segundo servicio de mensajería corresponde a los mensajes multicast (i.e. los destinados a dos o más receptores) enviados desde un planeta hacia todos los jugadores. En este caso, no alcanza con mantener el orden causal con cada uno de los jugadores por separado, tal como se muestra en la figura 2. El jugador J1 es dueño del planeta P1, y el jugador J2 es dueño del planeta P2. Ambos están atacando el planeta del otro. Si no se impone ninguna restricción al orden en que los mensajes de los planetas son entregados a los jugadores, puede suceder que los dos jugadores se dan por perdedores (como se representa en la figura), cuando en realidad ganó el jugador J2.

Es necesario que el protocolo utilizado garantice el orden total [5] de los mensajes, de forma tal que todos los jugadores tengan una vista consistente del juego. Existe orden total cuando dos mensajes cualquiera recibidos por dos participantes cualquiera son entregados en el mismo orden a ambos.

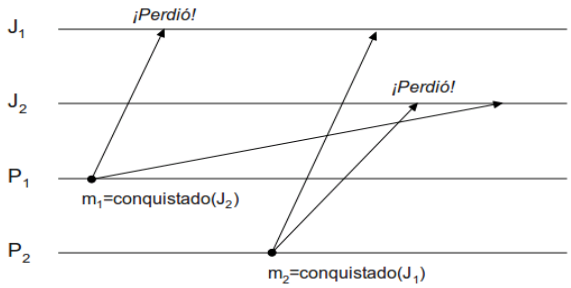

Fig. 2. En esta figura se representa una partida con dos jugadores y dos planetas.

El último servicio de mensajería corresponde a los mensajes enviados entre planetas. Tal como se muestra en la figura 3, en este caso tampoco basta con garantizar el orden causal de los mensajes. El planeta P1 informa la creación de la flota, primero a los jugadores a través del canal multicast (m2), y luego al planeta P2 a través de un canal punto a punto (m3). A continuación, P2 envía un mensaje multicast informando el arribo de la flota (m4). Dado que el canal multicast garantiza únicamente orden total, el mensaje m2 puede retrasarse, y ser entregado a los jugadores luego de m4.

Para resolver este problema, es necesario garantizar que todos los mensajes enviados por los planetas se entreguen respetando el orden lógico [5]. Un mensaje m1 precede a otro m2 en orden lógico si y solo si m1 fue enviado antes que m2 por el mismo participante, o si m1 es entregado al emisor de m2 antes de que envíe m2, o si existe m3 tal que m1 precede a m3 y m3 precede a m2.

Siendo así, m2 precede a m4 por propiedad transitiva, dado que el mensaje m2 precede a m3 (fue enviado por el mismo emisor), y m3 precede a m4 (fue entregado a P2 antes de que P2 enviara m4).

4.2 Implementación

El servicio para envío de mensajes de un cliente a un servidor puede implementarse en forma trivial utilizando TCP/IP como capa de transporte, dado que garantiza la entrega de los mensajes en orden causal.

Otro servicio de mensajería es el utilizado por los servidores para difundir su estado a todos los clientes. Este servicio corresponde a un multicast con orden total, cuya implementación no es trivial. Para implementar este servicio optamos por un algoritmo basado en privilegios [7]. Este algoritmo se basa en permitir que un servidor envíe mensajes multicast solo cuando tiene el privilegio para hacerlo. Este privilegio se pasa entre los servidores haciendo circular un token (i.e: un mensaje de pequeño tamaño). Si se numeran los mensajes enviados, entonces queda determinado el orden total de los mismos.

En [7] se describen las limitaciones de los algoritmos basados en privilegios. En particular, nos interesa analizar dos de ellas:

1. Es necesario que los emisores se conozcan entre sí para poder pasarse el token. En nuestra implementación, los servidores se conocen antes de comenzar el juego, por lo tanto el conjunto de emisores es bien conocido desde el comienzo.
2. Un participante podría apropiarse del token y enviar mensajes en forma indiscriminada, quitándole a otros el privilegio para hacerlo. Para evitar esta situación, nuestra implementación impone una cantidad máxima de mensajes que un participante puede enviar por turno.

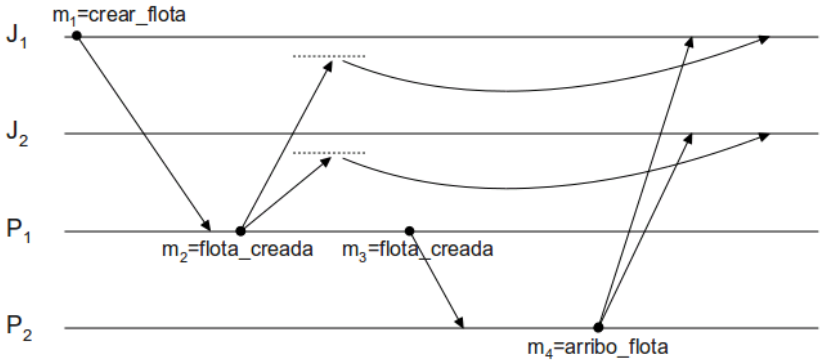

Fig. 3. El jugador J1 crea una flota (m1) desde el planeta P1 hacia el planeta P2.

Por último, debemos garantizar el orden lógico de los mensajes enviados por los servidores. Con muy pocas modificaciones, el esquema de privilegios recién descripto nos permite implementar esta condición. Un servidor puede enviar mensajes a otros servidores solo cuando tiene el token en su poder. Además, los mensajes enviados a otro servidor deben estar numerados, siguiendo una secuencia independiente para cada destino. Esta información aumenta el tamaño del token, pero si la cantidad de servidores se mantiene relativamente baja, no presenta grandes problemas.

Además de los servicios de mensajería ya explicados, es necesario proveer a la aplicación de un servicio administrativo para la creación y el mantenimiento del grupo de mensajería. Este servicio le brinda a la aplicación la posibilidad de crear un nuevo grupo, indicando inicialmente qué hosts funcionarán como servidores. Una vez que el grupo está creado, pueden agregarse y quitarse dinámicamente clientes.

5. Comparación con las Arquitecturas Cliente/Servidor y Peer-To-Peer

Nuestra propuesta, frente a la arquitectura basada en el modelo cliente-servidor, ofrece la gran ventaja que, por distribuir el trabajo entre varios "servidores", cada servidor requiere menor capacidad de procesamiento y comunicaciones. Si bien son necesarias comunicaciones entre los múltiples servidores para mantener la consistencia, estos se encuentran en una red controlada, por lo que las comunicaciones serían locales y los retardos despreciables respecto a los de Internet.

Comparando la propuesta con una arquitectura peer to peer, se nota que la primera, al no imponer mayores restricciones sobre las necesidades de procesamiento y comunicaciones en los clientes, ofrece la posibilidad a mayor cantidad de dispositivos de acceder a los servicios (por ejemplo, celulares, netbooks, etc.). Además, simplifica en gran medida la implementación del sistema ya que es mucho más simple la estructura necesaria para asegurar la consistencia.

Como ejemplo práctico de la simplicidad y utilidad de la plataforma e interfaz de servicios propuestas, se puede mencionar la experiencia en el desarrollo del prototipo. A partir de esta interfaz se implementaron y probaron las funcionalidades de aplicación asociadas a los jugadores y a los planetas por separado. El middleware también fue implementado sin tener en cuenta más que la interfaz requerida, y fue probado con objetos dummies (que no implementan la funcionalidad real) sobre él. La integración de los tres componentes (aplicación jugador, aplicación planeta y middleware) no llevó más de 20 horas de trabajo.

6. Conclusión

En este artículo se presentó una plataforma que distribuye la funcionalidad de una aplicación entre múltiples nodos de una red. Cada nodo se encargaría de mantener parte del estado de la aplicación y de procesar los eventos pertinentes al mismo. Para facilitar el desarrollo de la aplicación sobre la plataforma, se analizaron las funcionalidades que debería ofrecer una capa de middleware y se implementó dicha capa y su correspondiente interfaz.

Como prueba de uso de la plataforma, se desarrolló un prototipo basado en el juego Galcon [4]. Con este prototipo no solo se pudo probar la factibilidad de la plataforma propuesta, sino que también se comprobó su simplicidad arquitectural y las ventajas del uso de una capa de middleware al momento del desarrollo de una aplicación distribuida.

Toda la documentación asociada al proyecto y prototipo se puede encontrar en [8]. El código fuente (distribuido bajo licencia MIT [9]) del proyecto se puede encontrar en [10].

References

1. Smed J., Kaukoranta T., Hakonen H: Aspects of Networking in Multiplayer Computer Games. The Electronic Library, Vol. 20, No. 2, pp. 87–97 (2002)
2. Pellegrino J. D., Dovrolis C.: Bandwidth requirement and state consistency in three multiplayer game architectures. Proceedings of the 2nd workshop on Network and system support for games, pp. 52-59 (2003)
3. W. Emmerich: Software engineering and middleware: a roadmap. Proceedings of the Conference on the Future of Software Engineering, Limerick, Ireland (2000)
4. Galcon, http://www.galcon.com/classic/index.html
5. Veríssimo P., Rodrigues L.: Distributed Systems for System Architects. Kluwer Academic Publishers (2001)
6. Tanenbaum A. S., van Steen M.: Distributed Systems: Principles and Paradigms. Prentice Hall (2002)
7. Défago X., Schiper A., Urbán P.: Total Order Broadcast and Multicast Algorithms: Taxonomy and Survey. ACM Computing Surveys, Vol. 36, No. 4, December 2004, pp. 372-421 (2004)
8. Documentación del proyecto, http://distribuidos.wikidot.com/
9. Licencia MIT, http://www.opensource.org/licenses/mit-license.php
10. Código fuente del proyecto, http://code.google.com/p/fiuba-distribuidos1-109/