Middleware Para Sistema de Múltiples Servidores en Juegos Online

Hector Aquino Filho

Facultad de Ingeniería, Universidad de Buenos Aires, Argentina
haquino@fi.uba.ar

Cátedra: Sistemas Distribuidos I (75.74)
Profesora: María Feldgen

Resumen. Hoy en día existen diferentes arquitecturas y diseños para implementar sistemas de juegos multijugador en línea. Cada uno de ellos intenta brindar las funcionalidades necesarias para juegos de este estilo, permitiendo un mejor servicio orientado a los usuarios. Este trabajo analiza los diseños básicos y ofrece una nueva solución al problema, basada en un middleware orientado a mensajes, asincrónico y persistente, que utiliza el modelo de Publicador/Suscriptor. El objetivo es proveer una plataforma de bajo acople entre sus componentes y que abstraiga los elementos de la comunicación, simplificando la programación del sistema.

Keywords. sistemas distribuidos, middleware, Publicador/Suscriptor, grupos, juegos multijugador.

1 Introducción

Los sistemas de juegos en línea, no solo son grandes, sino que cada vez se hacen más complejos, ya que no solo tienen el problema de las comunicaciones, sino que se agregan dificultades de la multimedia, gráficos, inteligencia artificial, etc. Esto lleva a necesitar una solución que resuelva las comunicaciones del sistema y de ser posible parte de la interacción entre los componentes del sistema mismo [1] [2] [3].

Actualmente, existen varias arquitecturas utilizadas en los juegos online, siendo la mayoría de ellas basadas en el esquema de cliente-servidor [4] [5].

Para el diseño de mi solución a este tipo de problemas se tuvieron en cuenta los juegos en 2D y en especial un juego llamado The Mana World [6] para simplificar el manejo de las coordenadas del mapas. Este juego, implementado sobre un esquema, cliente-servidor, se basa en un mapa, denominado “Mundo”. Este Mundo, se encuentra en una partida del juego, la cual está formada por todos los jugadores con sus

objetos personales y los personajes controlados por el sistema, es decir, los datos necesarios para establecer el estado actual del juego. Cada jugador tiene la posibilidad de interactuar con el mundo utilizando sus personajes, ya sea moviéndose en él, atacando a otros jugadores y/o personajes controlados por el juego, utilizando objetos del mundo, interactuando con otros jugadores, etc. Todo esto crea una gran complejidad en el sistema del juego, la cual se complicada aún más con el manejo de las comunicaciones y sincronizaciones.

A partir de este juego se analizaron los requerimientos del sistema y para probarlos en escala, se desarrollo un pequeño juego prototipo, que verifica el correcto funcionamiento de la aplicación.

2 Requerimientos

Este tipo de sistemas tiene requerimientos claros:

- **Escalabilidad:** Se debe permitir que la cantidad de usuarios crezca.
- **Demora en las Comunicaciones:** Se debe reducir las demoras perceptibles por el usuario final.
- **Robustez:** No se puede dejar de funcionar correctamente en el eventual caso que una parte del mismo se desconecte de forma espontánea.
- **Consistencia:** En el sistema todos los usuarios deben ver la misma información.
- **Control de Acciones de Usuarios:** El sistema tiene que tener alguna forma de controlar las acciones a realizan por los usuarios de forma de evitar las alteraciones ilícitas de sus contenidos.
- **Adaptabilidad a las Distintas Arquitecturas de los Usuarios:** Debe soportar conexiones provenientes de diferentes sistemas operativos.
- **Programación Simple:** Un programador del juego solamente debe preocuparse de la lógica del juego. No debe en ningún momento tener en cuenta los sistemas de comunicación.
- **Persistencia de Datos:** Debe evitar la pérdida de datos por la caída de las conexiones de las computadoras de la red.

3 Soluciones

Principalmente existen tres tipos de arquitecturas para juegos online [1-3]:

- **Peer-To-Peer (Caso A):** Este tipo de sistemas se basan en que todas las computadoras son iguales dentro de la red [7]. En cada una de ellas puede crear una "partida" del juego, y el resto de los participantes se conectan a esta computadora. Luego esta máquina "servidor" puede eliminar la partida y unirse a la de otra.

Ejemplos de esta arquitectura se pueden ver en [4-5] [8-9].

- **Cliente-Servidor Centralizado (Caso B):** Esta arquitectura es la más utilizada en los juegos aunque sea la más difícil de implementar, debido a que al estar el control completo en un único equipo, se genera grandes problemas para coordinar todas las partes del sistema. Se basa en un servidor central que contiene el Mundo y todos los jugadores se conectan a él [10].

Ejemplos de esta arquitectura se pueden ver en [3].

- **Middleware Orientado A Mensajes (Caso C):** Esta arquitectura se basa en el pasaje de mensajes y está orientado a los servicios. Un middleware, es una capa entre el sistema operativo y la aplicación que abstrae las diferencias entre sistemas, ofreciendo una interfaz única a la aplicación [10]. La interfaz que provee puede ser vista como un peer-to-peer, cliente/servidor u otra arquitectura, mientras que por dentro funciona de forma totalmente diferente [7]. En la misma, todos los participantes se conectan al middleware que se encarga de coordinar la partida y mantener el estado del juego.[11]

Ejemplos de esta arquitectura se pueden ver en [1-2].

4 Análisis de Arquitecturas por Requerimiento

Ahora analizo cómo cumple con los requerimientos cada tipo de arquitectura:

- **Escalabilidad:** En los casos A y B, la escalabilidad es muy reducida. En especial en el cliente-servidor dónde siendo un único servidor, puede formarse un cuello de botella cuando aumentan las interacciones, reduciendo la operatividad del sistema. Luego, en el caso C, la escalabilidad se puede lograr muy fácil por estar distribuidas las operaciones, las cuales se busca que tengan poca carga.
- **Demora en las Comunicaciones:** En el caso A, las demoras del sistema son totalmente dependiente de la computadora que ofrezca la partida, si la computadora es lenta, el juego se verá lento. En el caso B, existe mucho retardo, ya que depende de la distancia entre las computadoras y el servidor, que para el caso de este juego puede ser demasiado grande. En el caso C, existen demoras adicionales debido a que existe una sincronización entre las partes del sistema por el intercambio de información de control.
- **Robustez:** En el caso A y B, una caída de la computadora del jugador que ofrece la partida, o el servidor central en el caso B, significa que el juego dejará de existir. En el caso C, si se cae un jugador, el resto no se ven afectados, pero además si se cae una parte del sistema, solo esa parte del mundo deja de funcionar, no todo el juego.
- **Consistencia:** En el caso A, el juego solo sería consistente dentro de una misma partida. En el caso B, cliente-servidor, dado que toda la información la contiene un solo servidor la consistencia de la información está asegurada. En el caso C, el middleware garantiza la persistencia y consistencia.
- **Control de Acciones de Usuarios:** En el caso A, no hay control. Los casos B y C, el usuario no tiene información de control interno.

- **Adaptabilidad a las Distintas Arquitecturas de los Usuarios:** El caso A es dependiente de la arquitectura del sistema operativo. Los casos B y C, la adaptabilidad es alta, ya que los mismos están diseñados para este requerimiento.
- **Reducción al Mínimo de las preocupaciones del Programador:** En el caso A y B, el programador resuelve las comunicaciones y la sincronización. En el caso C, el middleware abstrae las comunicaciones y la sincronización ya que tiene control sobre los sistemas que interactúan entre sí.
- **Persistencia de Datos:** En el caso A, no hay persistencia. En el caso B, la persistencia es fácil de lograr, ya que al tener un servidor único, este hace la persistencia necesaria sin perjudicar a los usuarios. En el caso C, la persistencia de datos es parte del middleware, por lo que está asegurada.

5 Solución Elegida. ¿Cómo Implementarla?

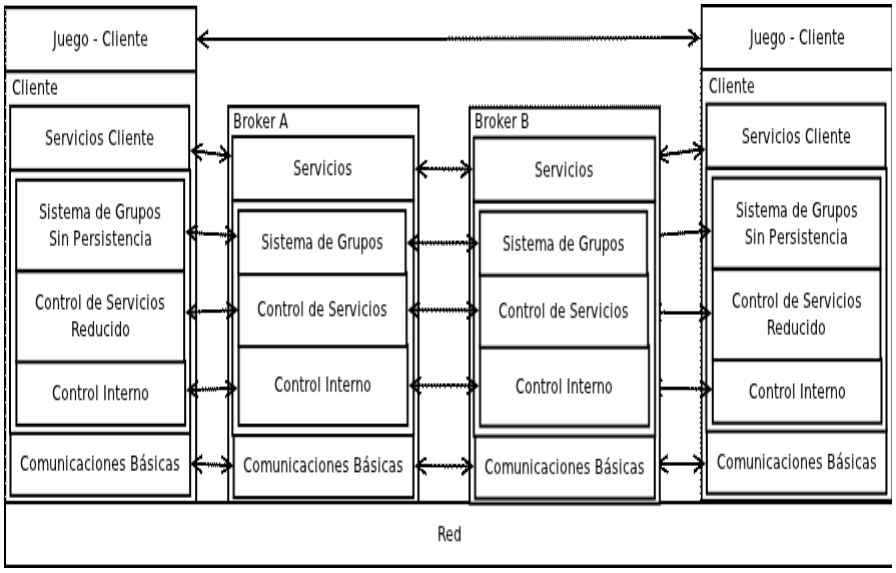

Fig. 1. Capas del Middleware

Por todo lo visto, la mejor opción es la arquitectura basada en un middleware orientado a mensajes. La misma utiliza el esquema de Publicador/Suscriptor, el cual se basa en eventos publicados por participantes, a los cuales otros se suscriben para poder leerlos [7]. Este esquema es utilizado para desacoplar las partes del sistema, ya que el publicador no conoce a los suscriptores y viceversa. Esto a su vez permite escalar fácilmente el sistema. En este esquema existen grupos, en los cuales no existe un destino u origen determinado y cada participante se adhiere a uno o más grupos y luego indica qué tipo de información desea recibir (tema). Es decir un participante puede publicar y suscribirse a información.

5.1 El sistema

Este sistema se basa en un "mundo" del juego, divido en N partes, y cada parte en un broker. Un broker es una entidad encargada de mediar la comunicación entre dos sistemas, que pueden o no ser distintos. La misma permite el desacople entre el receptor y el transmisor de las comunicación, de forma de dividir la carga del sistema. Las comunicaciones entre los mismos se hacen de forma directa, ya que el sistema sabe en todo momento cuántos brokers hay y qué parte del mundo tiene cada uno. O usa comunicación multicast en el caso de conectarse un nuevo broker, avisando al sistema cual es la parte del mundo que contiene. El sistema entrega a cada broker un valor para identificarse unívocamente, llamado identificador.

La interfaz del sistema, también llamada API, Application Programming Interface (Interfaz de Programación de Aplicación) presenta los elementos para controlar a todos los servicios del mismo. Estos elementos son llamados "handlers". Existe una clase que es creada e inicializada al iniciar el sistema que es la que devuelve los handlers para la capa de servicios.

Un detalle muy importante a tener en cuenta, es que debido a que se lo intenta utilizar con un juego ya implementado se debe utilizar la interfaz que posee, en este caso es la interfaz provista por la biblioteca ENet [11], producida por Lee Salzman. La razón para trabajar con esta interfaz, es reducir el trabajo de integración del middleware con el juego, aumentando al mismo tiempo sus capacidades de trabajo. Para incluir las funciones del middleware en la nueva implementación de la interfaz se agregaron las funciones necesarias sin modificar la API. Inicialmente esta biblioteca brinda la posibilidad de enviar y recibir mensajes por UDP de forma confiable, es decir brinda las características de TCP como fragmentación, secuencia en los mensajes, etc. Con la interfaz del middleware propuesto se toman todas las capacidades iníciales y se le agregan nuevas funciones para poder distribuir la plataforma. Además se utiliza TCP o UDP en las comunicaciones según sea el tipo de mensaje a enviar. Ambos tipos de comunicaciones son procesados en forma paralela.

5.2 Capa de Servicios

La capa de servicios brinda los handlers para el manejo de los datos de control y los handlers para el manejo de la persistencia de los diferentes tipos de datos (datos de usuarios, de brokers, de mapas y de grupos).

Existen tres tipos de handlers de control, uno para mensajes de control internos, uno para mensajes provenientes de usuarios normales del juego y uno para el sistema de mensajería. La razón que existan handlers separados para los mensajes de control y mensajes de usuario, es que los de control no pueden esperar a que el sistema procese primero a todos los mensajes de los usuarios.

Los handlers de persistencia permiten altas, bajas y modificaciones de los datos. Además, permiten consultar sobre datos particulares, como por ejemplo, la cantidad de usuarios conectados en un momento determinado. Un detalle importante es que del

lado del cliente, la aplicación también tiene handlers, pero con funcionalidades reducidas.

Las comunicaciones dentro de esta capa se basan en la actualización de datos, ya que la información de todos los mapas, jugadores, objetos, etc. está replicada en el resto de los brokers para poder aumentar la disponibilidad de la información en el caso de una caída de alguno de ellos y para reducir la demora en la recepción de información usando el broker más cercano.

5.3 Capa de Control

La capa de control consta de tres partes:

- Manejo de control interno (utiliza comunicación multicast para los mensajes).
- Manejar los mensajes de los clientes (utiliza comunicación usando TCP).
- Manejo de la mensajería (utiliza el concepto de Publicador/Suscriptor).

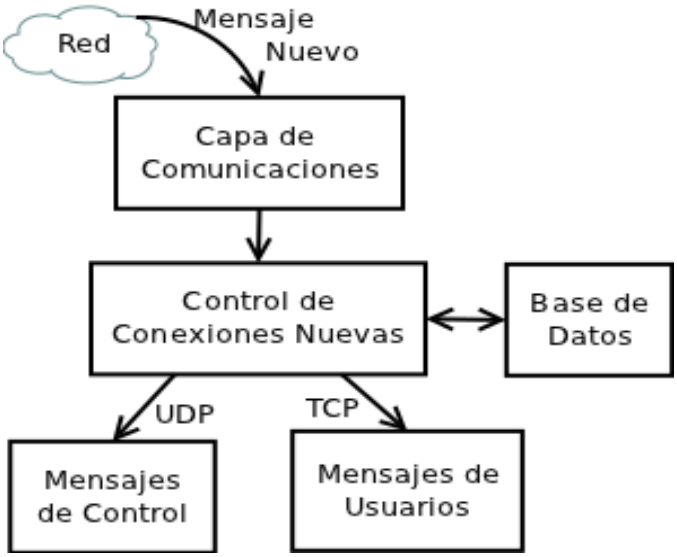

Fig. 2. Mensajes Nuevos en el sistema

5.4 Subcapa de Mensajería

El sistema de mensajería, está basado en identificadores de grupos e identificadores de usuarios. Cada usuario del sistema tiene acceso a un handler, provisto por la API que le brinda un servicio de mensajería. Este servicio, permite que cada usuario del sistema, ya sea un cliente o un broker pueda enviar mensajes de texto plano a cual-

quier otro usuario o grupo. Cada uno de estos usuarios puede crear, suscribirse o desuscribirse de un grupo cualquiera.

El esquema de Publicador/Suscriptor permite que un mensaje que se envía al grupo, sea distribuido a todos los interesados que pertenecen a ese grupo solamente. Además este sistema es de vital importancia para comunicar mensajes a todos los jugadores de parte de los administradores del juego, que es una actividad corriente en el juego analizado.

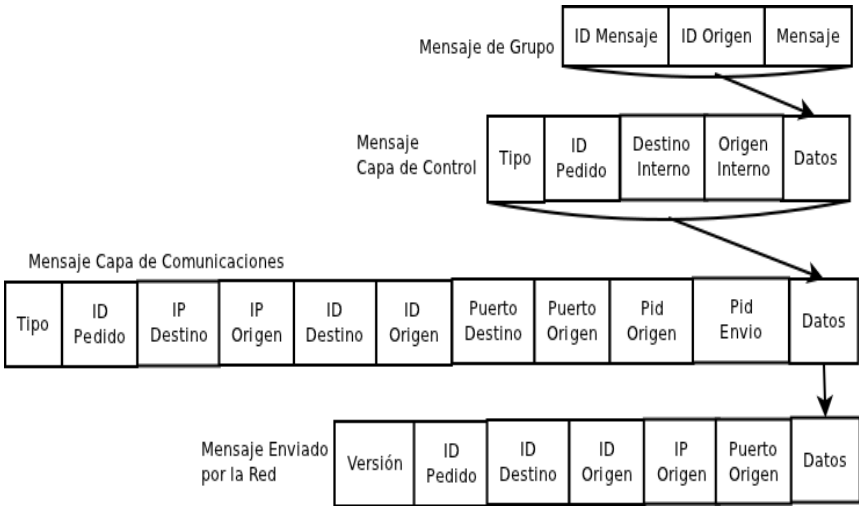

Fig. 3. Encapsulamiento de Mensajes

Por defecto, cada jugador se suscribe automáticamente al grupo de jugadores global y al del broker al cual está conectado actualmente, esto permite que si un broker se va a cerrar por algún motivo les avise a sus jugadores.

El handler de esta capa provee al programador la capacidad de suscribirse y desuscribirse a grupos, enviar mensajes a grupos o usuarios únicos, ver la cantidad de mensajes pendientes y ver los integrantes del grupo. Para ello, el handler intercambia los mensajes necesarios para poder realizar estas acciones.

5.5 Subcapa de Control de Servicios

Esta subcapa trabaja con los pedidos que operan sobre datos de usuarios, mapas, brokers y opciones. Cada uno de estos subtipos de mensajes, puede ser redireccionando a un destino separado para su procesamiento. Los mensajes corresponden a altas, bajas y modificaciones de datos de usuarios, mapas, brokers y opciones. Y a través del handler, el programador tiene el poder de cambiar la configuración de los procesos encargados de manejar cada tipo de mensaje. En la figura 3, se puede ver el formato de cada mensaje que se trasmite. El tipo representa a que sección del área de control está dirigido (Interno, Usuarios o Mensajería), luego el destino y origen interno, simbolizan a cuál proceso específico está dirigido este mensaje.

Esta capa trabaja de forma directa con el área de persistencia, ya que al realizarse una modificación en los datos, estos son distribuidos al resto de los brokers. Aunque debo notar que la frecuencia de distribución depende del tipo de información a enviar, ya que la información que cambia mucho, como la de los personajes de los jugadores, no son enviados de forma inmediata, sino que es agrupada o directamente se envía la última modificación.

5.6 Subcapa de Control Interno

Esta subcapa se encarga de configurar el comportamiento del sistema con dos tipos de mensajes de control, uno encargado del control del middleware en sí, al cual el programador no tiene acceso y otro encargado de los mensajes particulares del juego, como por ejemplo los registros de usuarios nuevos. Estos mensajes particulares son configurados a través del handler de la capa. El handler envía al proceso correspondiente de la capa, el nuevo destino de los mensajes específicos del juego. Los mensajes configurables son: pedido de identificador nuevo, inicio de conexión, fin de conexión, cambio de mapa por parte de un usuario y actualización de la información de los usuarios. Los mensajes que no son configurables son los encargados de la administración del middleware.

5.7 Capa de Comunicaciones

La última capa es la capa de comunicaciones del sistema, la cual dado un identificador y un mensaje, se encarga de transmitirlo a través de la red a su destino. Los mensajes que usa se pueden ver en la figura 3. Los identificadores nuevos son agregados de forma automática.

Esta capa es asincrónica, los pedidos y las respuestas son procesados por procesos distintos. Sin embargo el pedido de la respuesta es bloqueante para el proceso que la solicitó.

La recepción se divide en TCP y UDP, cada una en un proceso distinto, ya que además de usar protocolos distintos, los de TCP se mantienen abiertos permanentemente. Esto permite un ahorro de recursos y tiempo, ya que abrir las conexiones es muy costoso, pero no lo es tanto mantenerlas. Luego de recibir el mensaje, este se persiste y posteriormente se envía al proceso que lo pide. Excepto que el identificador o IP hayan sido prohibidos, en cuyo caso, los mensajes serían borrados automáticamente.

La recepción de mensajes UDP puede ser por unicast o por multicast. En el caso que se utilice una capa que no implemente comunicación multicast, se puede alterar el subproceso de envío, para que envíe un mensaje de unicast por cada destino deseado.

5.8 Proceso de Conexión

La conexión al sistema, depende de si se trata de un elemento conocido o desconocido del sistema. En el caso que sea desconocido, el sistema se inicializa con un iden-

tificador provisorio. Al recibir este identificador, el sistema se comunica con la administración del sistema principal, es decir se comunica con el grupo de brokers que contiene al mundo, indicándole que se conecta por primera vez. Esta comunicación se realiza por comunicación multicast. Este mensaje es recibido por todos los brokers, pero solo es procesado por uno de ellos que fue designado para la asignación de identificadores. Este decide un identificador nuevo único y se lo envía al que lo pidió. Una vez que este elemento obtiene su identificador, inicializa sus handlers y subsistemas. Si el elemento a conectarse ya fuera conocido por el sistema principal, directamente se conecta normalmente con su identificador. En el caso de que el elemento a conectarse sea un broker nuevo, este además debe avisar los mapas que tiene disponibles.

6 Conclusión

Utilizando la arquitectura elegida se desarrolló un prototipo del sistema en un middleware basado en pasaje de mensajes sobre un modelo Publicador/Suscriptor orientado a eventos.

La implementación constó en la realización de toda la plataforma descripta en el presente trabajo (con todas sus capas), además de un simulador del juego The Mana World. Este simulador se encargó de llamar las funciones y métodos de la biblioteca ENet que el juego original utilizaría, pero que están provistas por el middleware descripto. Todo esto permitió verificar las hipótesis presentadas en este trabajo y ver efectos colaterales no tenidos en cuenta con relación al juego.

La implementación se realizó y testeo utilizando los sistemas operativos Debian (Wheezy), Ubuntu (10.04) y Slax (6.1.2).

Se verificó que el sistema es escalable, ya que se podían agregar varios clientes y brokers y el sistema los configuraba automáticamente. Esta automatización necesita una sincronización, lo cual redunda en mayores demoras en el juego a medida que la cantidad de jugadores aumenta considerablemente.

Además, el juego es tolerante a caídas de brokers, ya que solo una parte del mundo quedaría inaccesible. Sin embargo, como la información está replicada en diferentes brokers, otro broker puede cumplir con las funciones del equipo caído por el tiempo que sea necesario.

Además, se permite el monitoreo de las acciones de cada jugador. El sistema de monitorea permite almacenar las acciones que se realizan para que puedan ser verificadas por otro proceso.

Las pruebas fueron satisfactorias y los resultados fueron los esperados con la plataforma. Esta plataforma inicial mostró algunos factores que aumentan la demora en el procesamiento que deberían rediseñarse en una versión posterior. En las pruebas se verificó que el aumento en las demoras se debe a la sincronización entre los jugadores o participantes y los brokers, en especial cuando una acción del jugador debe ser procesada por todos los brokers, aunque cabe destacar, que hace que el sistema sea más resistente a los fallos aunque a costa que sea más lento el juego. Lo mismo se verificó aumentando la frecuencia con la cual se persisten los datos. Son temas que requieren un estudio más detallado para reducir su impacto sobre el juego.

Referencias

1. Assiotis, M., Tzanov, V.: Distributed Architecture for MMORPG, Massachusetts Institute of Technology
2. Krishna Balan, R., Ebling, M., Castro, P., Misra, A.: Matrix: Adaptive Middleware for Distributed Multiplayer Games
3. Hsu, C., Ling, L., Li, Q., Kuo, J.: On the Design of Multiplayer Online Video Game Systems
4. J. Jardine and D. Zappala, "A hybrid architecture for massively multiplayer online games," *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games - NetGames '08*, p. 60, 2008.
5. A. Bharambe, J. Pang, and S. Seshan, "Colyseus: A Distributed Architecture for Online Multiplayer Games," *USENIX Association - NSDI '06: 3rd Symposium on Networked Systems Design & Implementation 155*, pp. 155-168, 2006.
6. The Mana World (http://themanaworld.org/)
7. Veríssimo, P., Rodrigues, L.: Distributed Systems for System Architects. Kluwer Academic Publishers, 2001.
8. S. Karunasekera, S. Douglas, E. Tanin, and A. Harwood, "P2P Middleware for Massively Multi-player Online Games."
9. T. Hampel, T. Bopp, and R. Hinn, "A peer-to-peer architecture for massive multiplayer online games," *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games - NetGames '06*, p. 48-es, 2006.
10. Tanembaum, A., van Steen, M.: Distributed Systems: Principles and Paradigms. Prentice-Hall, 2002.
11. Biblioteca ENet (http://enet.bespin.org/)