Patrones de Diseño para Mundos Virtuales Orientados a Objetos (MOO's)

Fernando Martínez, Alejandro Miguez, Alejandro Fernandez, Alicia Díaz

Lifia, Facultad de Informática, UNLP
La Plata, Argentina
[yagui, parker, casco, alicia]@sol.info.unlp.edu.ar

Abstract: Los patrones de diseño describen problemas que ocurren repetidamente y presentan una capa de solución para dicho problema, de manera tal que pueden ser usados en varias ocasiones en diferentes contextos y aplicaciones. Los patrones permiten amplio reuso de arquitecturas de software y mejoran la comunicación dentro y a través de los equipos de desarrollo de software en la medida que proveen un vocabulario compartido y conciso. Explícitamente capturan el conocimiento que los diseñadores utilizan implícitamente. La descripción de los patrones provee un framework para asentar intercambios y alternativas de diseño. Un lenguaje de patrones es un conjunto parcialmente ordenado de patrones relacionados que trabajan juntos en el contexto de un cierto dominio de aplicación. Este paper presenta un lenguaje de patrones para el diseño de MOO's (mundos virtuales orientados a objetos), ejemplificándolos sobre LambdaMoo.

Keywords: realidad virtual, mundo virtual, orientación a objetos, objetos virtuales, patrones de diseño, MOO.

1. Introducción

La investigación en realidad virtual, trabajo cooperativo y sistemas de información, ha evolucionado recientemente hacia una nueva área: mundos virtuales. Un mundo virtual es un espacio virtual basado en redes, donde las personas pueden interactuar entre sí en tiempo real. Estas pueden navegar el espacio virtual e interactuar con objetos o items encontrados en el camino. El espacio virtual es generalmente dividido en espacios virtuales más pequeños (habitaciones). Cada mundo virtual tiene un propósito general y cada habitación dentro del mismo, un propósito específico.

En estos días, hay muchos tipos de aplicaciones de mundos virtuales: para reuniones de trabajo, para educación a distancia, para el comercio electrónico, para entretenimiento, etc.

Una nueva alternativa para implementar mundos virtuales en estos momentos es la tecnología de los MOO [1] (Multiuser Object Oriented Dungeons). Los MOO son realidades virtuales orientadas a objetos. La mayoría son basados en texto y se asemejan mucho al mundo real, y pueden ser usados también como sistemas de información o ambientes de simulación.

Hay muchos ejemplos de MOOs: LambdaMOO [7], de propósito general; TecfaMOO [8], comunidad de aprendizaje; Diversity University Main Campus [10], primer MOO diseñado específicamente para la simulación de aulas; IPL-MOO [9] (Internet Public Library MOO), está siendo desarrollado como una biblioteca pública para la comunidad de Internet.

Los MOO se construyen a partir de objetos. Estos actúan en respuesta a algún pedido o en forma autónoma. En cualquier caso la respuesta se traduce a la ejecución de un verbo. Cada objeto define el conjunto de verbos que es capaz de ejecutar. Los verbos pueden hacer uso de los atributos

que posee el objeto donde este está definido. Los atributos pueden ser testeados, devueltos, cambiados y pasados como parámetros a funciones u otros verbos. Los verbos pueden hacer pedidos a otros objetos.

Cuando un objeto recibe un pedido (en forma de invocación de verbo o mensaje), este ejecuta el verbo correspondiente. Si no define un verbo para este pedido, busca uno adecuado en su jerarquía, comenzando desde la superclase inmediata. Si no se encuentra ningún verbo para ese pedido, entonces el objeto no lo puede completarlo y no hace nada.

Existen dos tipos de objetos que son de especial interés: habitaciones y personajes.

Una habitación es un espacio que es habitado por objetos de propósito general, personajes y otras habitaciones. El MOO está definido como un conjunto de habitaciones conectadas. Las habitaciones están conectadas por medio de entradas y salidas, las cuales permiten a los usuarios y objetos navegarlas.

En una habitación particular, los personajes y objetos pueden interactuar directamente con la habitación y con los objetos contenidos en ella.

Los personajes representan los ciudadanos (usuarios) del MOO. Estos recogen objetos, son dueños de objetos, y se mueven dentro del MOO. Los personajes activan a los objetos para llevar a cabo una actividad, así como también se comunican con otros personajes.

Para construir MOOs, es necesario diseñar:

- las habitaciones, sus conexiones y estructura;
- los objetos dentro de la habitación, sus propiedades y comportamiento;
- los personajes y los posibles grupos de personajes y sus capacidades, y
- cómo todos estos trabajan en cooperación.

Dado todos los aspectos que deben ser tenidos en cuenta, el diseño de un mundo virtual no es una tarea trivial. La funcionalidad y la distribución espacial deben pensarse bien, para construir un mundo virtual utilizable. Las técnicas de diseño formales son necesarias para llevar a cabo nuestra misión.

Las técnicas de modelización y diseño deben tratar con la arquitectura, navegación espacial, ambientes virtuales habitados por objetos y usuarios. Los modelos de diseño deben tener en cuenta la dinámica de los mundos virtuales.

El objetivo del paper es describir problemas que ocurren repetidamente en el diseño de mundos virtuales, y presentar una solución para dicho problema, de manera tal que puedan ser usados en varias ocasiones en diferentes contextos. Esto permitirá tener un mecanismo de reuso de diseño muy importante así como también proveerá a los diseñadores de mundos virtuales un vocabulario compartido y conciso.

Este paper está organizado de la siguiente forma: la sección 2 presenta una visión de los patrones de diseño y los lenguajes de patrones. La sección 3, muestra un catálogo de patrones de diseño para mundos virtuales. Finalmente, la sección 4 expone las conclusiones y trabajos futuros.

2. Patrones de Diseño y Lenguajes de Patrones

Los patrones de diseño [6] están siendo usados incrementalmente en el diseño de software, pero hasta donde sabemos, siguen sin ser explorados en el campo de las realidades virtuales.

Uno de los tipos más importantes de reuso, es el reuso de diseño, y los patrones de diseño son un buen medio para asentar experiencia en dicho campo.

Los lenguajes de patrones también están siendo construidos para dar forma a organizaciones complejas y a sus procesos de desarrollo.

El movimiento de los patrones comenzó en el área del diseño en arquitectura hace aproximadamente ya 20 años con el trabajo de Christopher Alexander [5]. Recientemente, la comunidad de orientación a objetos ha empezado a enfocar mayormente en dos áreas diferentes: los patrones de diseño y los lenguajes de patrones. Los patrones de diseño son lo suficientemente

generales pata resolver problemas recurrentes de diseño en diferentes dominios. Por ejemplo, la definición de una dependencia de “uno a muchos” entre objetos, de modo tal que cuando un objeto cambia su estado, todos sus dependientes sean notificados y actualizados automáticamente (el patrón de diseño Observer). Otro patrón de diseño aparece cuando definimos familias de algoritmos, encapsulando cada uno y haciéndolos intercambiables (el patrón de diseño Strategy). Los patrones son actualmente considerados los bloques más básicos de construcción del trabajo diario de los diseñadores de software.

En algunos dominios como por ejemplo, comunicaciones o sistemas de tiempo real, los problemas de diseño pueden ser más específicos y entonces surgen patrones más especializados. Estos son usualmente la base de un lenguaje de patrones: un conjunto parcialmente ordenado de patrones relacionados que trabajan juntos en un contexto de un cierto domino de aplicación. Los lenguajes de patrones también están siendo construidos para dar forma a organizaciones complejas y a sus procesos de desarrollo. Estos últimos son llamados patrones organizacionales. El objetivo de nuestro trabajo es establecer la base para un lenguaje de patrones para aplicaciones de realidad virtual.

Los patrones más abstractos, por ejemplo, aquellos que abarcan soluciones más abstractas que pueden ser implementadas de diferentes formas, así como los patrones organizacionales, usualmente aparecen en el formato “Alexandrino”. Este contiene: nombre, contexto, problema, solución y patrones relacionados [5].

Los patrones en un lenguaje de patrones, están comúnmente relacionados entre sí. Las relaciones expresan que algunos patrones usualmente aparecen juntos, que algunos patrones son mutuamente complementarios o excluyentes.

La siguiente sección presenta un catálogo que complementa el diseño orientado a objetos de mundos virtuales. Los patrones de este catálogo comprenden la mayoría de los problemas/aspectos de diseño que competen a la construcción de mundos virtuales. Los patrones Area y Gates tratan con el diseño estructural de mundos virtuales. Locomotion y Transport tratan con la navegación dentro de los mundos virtuales. Collector muestra el comportamiento comúnmente encontrado en objetos virtuales.

3. Catálogo de Patrones para Mundos Virtuales (MOOs):

Nombre: Area

También conocido como:

Space unit, Container.

Propósito:

Construir objetos capaces de contener/almacenar otros objetos. Representan límites espaciales para los demás objetos. Las áreas son la unidad de espacio más simple.

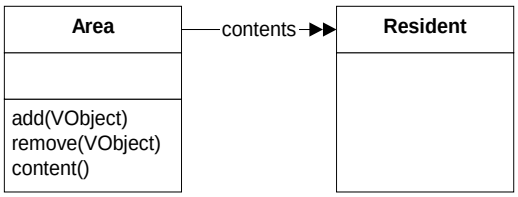
Motivación:

Usted está construyendo un museo virtual. Ya tiene creadas pinturas y estatuas para que sus visitantes disfruten. Ahora es tiempo de organizarlos en diferentes áreas de acuerdo a su edad y origen, para hacer las visitas al museo más simples y placenteras. Estas áreas deben ser objetos capaces de contener a otros: pinturas y estatuas y eventualmente personajes que vengan a visitarlo. Deben ser disjuntas, lo que hará imposible que un objeto esté en dos lugares al mismo tiempo. Cuando un visitante entre en un área percibirá todos (y exclusivamente) aquellos objetos dentro de ella. Las áreas actúan como límite espacial.

En particular, en los MOO, el espacio es discreto, lo cual significa que no hay distinción de la ubicación de dos objetos dentro de un mismo área. Éste modelo no soporta la ubicación relativa

de los objetos con referencia al espacio que habitan. Todas las pinturas, estatuas y personajes dentro de un área están en el "mismo" lugar. Las áreas son las unidades de espacio.

Solución



Area: Las áreas conocen a todos los objetos que contienen. Implementan el protocolo para agregar/remover objetos.

Resident: es el objeto contenido en el área. Inicialmente, puede ser instancia de cualquier clase.

Consecuencias:

1. Las habitaciones están definidas como estructuras disjuntas para contener objetos. Esta restricción contribuye a asegurar la localización única de los objetos.
2. Para asegurar unicidad de ubicación, Resident debe implementar Locomotion (ver patrón Locomotion).

Implementación:

1. *Navegación dentro del área*: como fue previamente mencionado, nuestro modelo considera al espacio discreto con las áreas como unidad. De allí que no es posible moverse (o cambiar de ubicación) dentro de un área. Áreas "más grandes" pueden ser simuladas combinando muchas áreas juntas. Un corredor, por ejemplo, puede ser simulado conectando dos o más áreas en una línea. Con alguna ayuda del ambiente de programación del mundo virtual, las áreas en un corredor pueden ser "decoradas" para que parezcan un todo. Sin embargo la percepción de límites de un objeto en un corredor estará definida por el área simple que este habita.
2. *Áreas dentro de otras áreas:* como fue descripto por el diagrama de clases en la sección Solución de este patrón, un área puede contener cualquier objeto virtual. De hecho un área puede contener otra área. Un bolsillo llevando una billetera con monedas es un ejemplo. El bolsillo es un área conteniendo la billetera y quizás otros objetos. La billetera por sí misma es también un área que contiene por lo menos monedas.

Ejemplo:

En el LambdaMOO, existe un museo virtual. Este museo clasifica las componentes de los MOO, y separa cada una en un Area distinta.

Un aspecto importante a la hora de definir un espacio complejo (áreas anidadas, muchas áreas interconectadas que se quieren ver como un todo, etc.) es proveer al usuario con un mapa del lugar para que a éste se le haga más fácil la navegación dentro del mismo. En el caso del *LambdaMOO museum*, este presenta un mapa en el cual se marcan tanto las áreas que incluye como también las comunicaciones que tienen cada una (puertas).

En el museo, se utiliza como protocolo de contenido el mensaje ***#tell_contents().*** El protocolo para agregar y remover objetos en cualquier área en LambdaMOO es ***#drop*** <***objeto***> y ***#pick*** <***objeto***> respectivamente.

Usos conocidos:

Habitaciones, bolsas, bolsillos.

Patrones relacionados:

Locomotion: Locomotion define el mecanismo básico para mover objetos entre áreas. Ambos patrones cooperan para asegurar que los objetos habiten un solo lugar a la vez (unicidad de ubicación).

Gate: Gates son usados para conectar Areas.

Nombre: Gate

También conocido como:

Entrance, Exit.

Propósito:

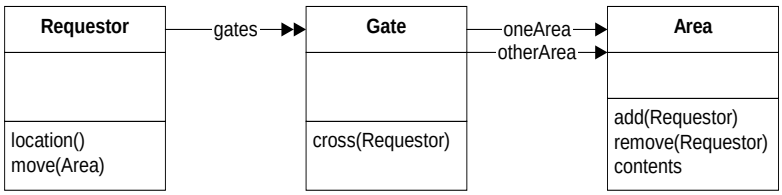
Comunicar áreas. Sugiere caminos de navegación y refuerza la distribución espacial.

Motivación

Usted está construyendo un mundo virtual, definiendo habitaciones y habitándolas con personajes y objetos. En este contexto, los objetos y personajes solamente pueden interactuar con otros en la misma habitación, lo que limita sus posibilidades. Un mecanismo para la navegación entre habitaciones es necesario para explotar totalmente las facilidades del mundo virtual y sacar provecho de los objetos ubicados en otra parte. Cualquier mecanismo de navegación que usted elija debe asegurar la consistencia del mundo (tomando en cuenta las propiedades del ambiente, distribución espacial, etc.) y permitir reforzar las restricciones como el control de acceso.

Una puerta es un tipo particular de objeto que le permite a los personajes y objetos navegar entre las habitaciones. Al mismo tiempo, las puertas sirven de mecanismo de restricción de acceso y son de gran ayuda para dar forma al mundo virtual. Las puertas refuerzan la distribución espacial de las habitaciones, acortando las distancias entre las habitaciones comunicadas.

Estructura



Requestor: El demandante es el objeto que quiere atravesar la puerta. Este invoca el mensaje cross() sobre la puerta, y esta realiza el resto del trabajo. El que realiza el pedido debe implementar Locomotion. Este puede conocer muchas puertas en un determinado instante de tiempo. El diagrama representa estas relaciones como una referencia a un objeto (puertas), sin embargo esto está usualmente dado por una percepción del contexto – un objeto virtual conoce a todos los objetos dentro de su alcance -.

Gate: La puerta conoce a las habitaciones que conecta. Cuando un objeto invoca cross(), la puerta averigua la ubicación del demandante, decide si el objeto puede pasar o no por ella, y en caso afirmativo lo mueve hacia la otra habitación.

Area: Implementa Area. Las puertas se pueden ver desde ambas áreas.

Consecuencias

1. Mediante el uso de puertas se refuerza una senda de navegación. Los objetos pueden entrar a una habitación sólo si pueden acceder a la puerta que las conecta.
2. Los objetos que implementan Locomotion son los únicos que pueden usar las puertas.

Implementación

1. *Cerraduras y llaves.* Las puertas pueden ser implementadas con llaves y cerraduras. Sólo un objeto que posee la correspondiente llave puede cruzar la puerta. Las cerraduras pueden ser implementadas de muchas formas: simulando cerraduras reales donde se necesita una llave; verificando identidad, etc. Una puerta semáforo es otra forma de implementar cerraduras y llaves. Esto sólo permite pasar a un número determinado de objetos, luego se cierra hasta que alguien salga. Hay también puertas donde se necesita pagar para cruzarlas.
2. *Puertas de un solo sentido.* Las puertas pueden ser implementadas de modo que sólo puedan ser atravesadas en una solo dirección. La decisión de que la puerta sea visible desde ambos lados queda a cargo del desarrollador.
3. *Entradas, salidas*: Las puertas son comúnmente llamadas entradas o salidas; sin embargo estos términos tienen diferente semántica. Supongamos que estamos dentro de una habitación y queremos cruzar la puerta que nos lleva al corredor. Podríamos decir que cuando salimos de la habitación, entramos al corredor. Desde adentro de la habitación la puerta es una salida, pero si vemos a la puerta desde el corredor, la puerta es una entrada a la habitación. Las entradas generalmente nos indican un camino hacia el interior de alguna estructura espacial, y nos aleja del lugar donde estamos ubicados actualmente. En ciertos casos, darle la semántica de entrada o salida a una puerta ayuda a la navegación.
4. *Caras:* las puertas deben poder verse desde las dos habitaciones que esta conecta. La mayoría de los modelos de Realidad Virtual definen la percepción de un cierto sujeto en términos de los objetos ubicados en la misma habitación o área. De acuerdo a esto, una puerta debe estar contenida en las dos áreas que conecta lo cual va en contra de las restricciones de nuestro modelo. En ese caso, las caras de la puerta pueden ser modeladas como objetos.

Ejemplo:

En LambdaMOO, las puertas ya están implementadas. Se implementan de manera tal que existe un objeto para cada dirección en la cual se quiera permitir navegar entre dos habitaciones, es decir, que se va a estar interactuando con dos objetos diferentes según el room en el que nos encontremos. Este mecanismo facilita la representación de puertas que pueden usarse en una sola dirección. Estos objetos conocen a los dos rooms que conectan. Además, hay que tener en cuenta que se debe mantener consistente la posición de las dos conexiones (si se quiere navegar en cualquier dirección entre los dos rooms que estamos intentando conectar), es decir, que si una puerta de un room1 dirige al norte a un room2, si queremos que se pueda volver al room1, entonces la puerta que pongamos en el room2 para volver al room1 debe estar al sur.

En LambdaMOO, el protocolo para agregar puertas a los ROOMS es:

Agrega una salida del room actual hacia el exit-object-number

@add-exit <exit-object-number>

Agrega una salida del room actual con nombre *up* o *u* hacia un room ya existente

@dig up,u to #7164 (el room existente tiene el número 7164)

Agrega dos salidas: una desde el room actual hacia "The Department of Auto-Musicology" (con nombres west o w) y otra desde "The Department of Auto-Musicology" hasta acá (con nombres east, e u out)

@dig west,w|east,e,out to "The Department of Auto-Musicology"

Para borrar puertas:

Borra del room actual la puerta con el nombre o numero especificado

@remove-exit <exit>

Usos conocidos:

Puerta.

Patrones relacionados:

Area: Las puertas comunican áreas. Debe existir un mecanismo para hacer que las puertas sean visibles en las áreas.

Locomotion: las puertas necesitan que el objeto que realiza el pedido implemente Locomotion.

Nombre: Locomotion

También conocido como:

Movable object, Mobility.

Propósito:

Provee un mecanismo seguro para mover objetos dentro del mundo virtual. Locomotion define un protocolo abstracto para los objetos movibles.

Motivación:

La belleza de los mudos virtuales, depende fuertemente de la habilidad de mover objetos. El ejemplo más simple es un personaje recorriendo un museo virtual. La navegación es la actividad que los personajes llevan a cabo cuando se mueven desde una habitación a otra dentro del museo. Esta actividad involucra, al menos, mover un objeto: el personaje en sí. También puede ser el caso de una herramienta cara o insuficiente que necesita ser compartida. Para usarla, se deberá poder moverla al lugar donde se necesita usarla. Estos ejemplos y muchos otros, involucran cambiar la ubicación de un objeto o un grupo de objetos.

El movimiento es una operación sobre los objetos, que llevan como parámetro una habitación (o un contenedor de objetos) , se remueve el objeto de la ubicación actual, luego cambia la ubicación del objeto a la de la nueva habitación, y lo agrega al contenido de esta. Estos tres pasos deberán ser ejecutados automáticamente para asegurar que el objeto no se encuentre en dos lugares al mismo tiempo, y que además está en algún lado siempre. En lo que concierne a los movimientos y ubicaciones, los objetos virtuales en este modelo se comportan como objetos reales: no pueden estar en dos lugares al mismo tiempo.

Solución:

```
Interface Movable
        Area location();
        Boolean move(Area destination);
```

Los objetos que implementan Locomotion deben implementar una interface para que puedan ser movidos. La mayoría tienen una operación de ubicación, que retorna un “área” donde el objeto se encuentra. Ellos deben implementar el mensaje move() que recibe el área como argumento. Esta operación mueve el objeto desde la posición actual a la posición indicada por el argumento. Esta retorna si la operación fue exitosa o no.

Consecuencias:

1. Locomotion provee un mecanismo seguro para mover objetos dentro del mundo virtual.

Implementación:

1. *Consultar al destino para la aceptación*. La operación de movimiento puede consultar a la habitación destino (y posiblemente a la de origen) para ver si aceptan o no al objeto que se quiere mover. Esta operación es la que se lleva a cabo por defecto en LambdaMOO[7].

2. *Bloquear objetos*. En algunos casos, los objetos necesitan ser bloqueados en su posición para que ninguno pueda moverlos. La operación de movimiento puede consultar al objeto para su aceptación. La aceptación también puede depender de factores que son externos a los objetos, tales como la identidad del objeto que realiza el pedido.
3. *Algunos objetos recuerdan la ubicación de su "casa"*. Cuando permanecen largo tiempo en algún otro lugar que no sea su "casa", regresan automáticamente a la misma. Esta funcionalidad es muy usada, por ejemplo, para el manejo de libros en una biblioteca virtual.

Nota: Las facilidades de Locomotion están usualmente implementadas por el ambiente de VR subyacente.

Usos conocidos:

Personajes, Autos, Animales

Patrones relacionados:

Area: Las áreas definen un mecanismo para construir objetos capaces de contener otros objetos. Locomotion hace posible el movimiento de objetos entre áreas, asegurando que los objetos se encuentren en un único lugar al mismo tiempo.

Gate: Las puertas dependen de que los que realizan el pedido implementen Locomotion.

Transport: Los transportes dependen de Locomotion para poder llevar a cabo su tarea. El Transport en sí puede implementar Locomotion.

Nombre: Transport

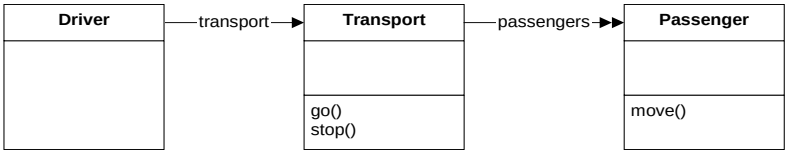
Propósito:

Encapsula un algoritmo para el movimiento de uno o varios objetos. Los transportes ocultan los detalles de la distribución espacial subyacente, y por tanto, simplifica la navegación

Motivación:

Mover objetos utilizando las facilidades de Locomotion es simple. Sin embargo, puede no ser deseable en ciertas circunstancias, como por ejemplo cuando es importante restringir la navegación. Algunos mundos virtuales dependen de las puertas para reforzar las restricciones de acceso y de distribución espacial. Sin embargo, cuando el destino final está ubicado a varias puertas de distancia –tal vez al final de un camino complejo- atravesar puerta por puerta puede resultar un poco engorroso. La solución es construir un objeto que conoce el algoritmo para llegar desde el origen al destino y este objeto conduce a todos los demás objetos. Este se comportará como un ómnibus, transportando todos los objetos que viajan en él, hacia el lugar de destino.

Solución



Driver: Requiere servicios al transporte. Posiblemente está a cargo de la configuración y carga del transporte.

Transport: Conoce cómo llegar al destino, trasladando a todos sus pasajeros mientras éste se mueve. Conoce a todos sus pasajeros.

Passenger: Implementa las operaciones de "Locomotion".

Consecuencias:

1. Los transportes ocultan los detalles de la distribución espacial subyacente, y de allí que simplifica la navegación.
2. Los transportes dejan que el algoritmo de navegación y su estructura subyacente varíen sin afectar a los pasajeros o al conductor.
3. Los pasajeros deben implementar Locomotion.

Implementación:

1. *Conductores*: El conductor es usualmente un tercer objeto que le requiere al transporte que haga su trabajo. Sin embargo, es posible que el transporte se conduzca "solo" (por ejemplo, las escaleras mecánicas) o ser conducidos por sus pasajeros (por ejemplo, elevadores, autos).

2. *Ruta fija*: El ejemplo más común de los objetos transporte, tienen un origen y un destino. Estos mueven otros objetos desde y hacia esos lugares. Los caminos más complejos pueden ser implementados definiendo estaciones terminales y paradas intermedias. Si la meta es esconder el camino que se recorre, entonces se recomienda el uso de una ruta fija con paradas predefinidas. Los pasajeros dependerán de la existencia de paradas y no del camino recorrido para llegar a esas paradas.

3. *Rutas flexibles*: Los objetos transporte no necesariamente deben tener una ruta fija. De hecho, hay casos donde los pasajeros eligen el camino a recorrer, como por ejemplo: los autos. El uso de rutas flexibles proporciona flexibilidad sobre los destinos y caminos de navegación.

4. *Contención y localización:* Hay dos aspectos finales en la implementación de un Transport: el transporte contiene a sus pasajeros y su ubicación. Ambas están relacionadas. En los casos que el transporte deba viajar con sus pasajeros, es mejor que los contenga. Este es el caso de un ómnibus, un elevador y un auto. Cuando el transporte debe estar en un lugar simple , es mejor que los pasajeros no formen parte del contenido del transporte. Este es el caso de los "teletransportadores".

Ejemplo:

PARA PERSONAJES:

Las descripciones de la mayoría de las habitaciones, dejan conocer las direcciones en las cuales existen salidas. Las direcciones típicas son las de los ocho puntos cardinales ('north', 'south', 'east', 'west', 'northeast', 'southeast', 'northwest', and 'southwest'), 'up', 'down', and 'out'.

Para dirigirse hacia una dirección en particular, simplemente hay que tipear la dirección deseada (ej. 'north', 'up'). También se puede tipear 'go <direction>' para moverse; esto es de suma utilidad si se quiere tipear varios movimientos de comandos de una sola vez (ej. go n n e s. Lo que se está haciendo en este ejemplo es ir hacia el norte, luego nuevamente hacia el norte, después hacia el este y por último al sur).

Existe otra manera de moverse dentro de los MOO: la teletransportación. Esta se lleva a cabo con el comando @go <destination>, donde destino puede ser algún lugar que el personaje conozca o directamente el numero de identificación del lugar. Una forma más específica de este tipo de movimientos es el comando @join <character> que lo que hace es, sin saber dónde está, nos lleva al lugar donde se encuentra un carácter en particular.

Este mecanismo de teletransportación no sólo sirve para hacerlo con uno mismo, sino que también se pueden teletransportar objetos con el comando @move <object> to <destination>.

Además, algunas áreas pueden contener objetos que permitan teletransportación y la mayoría de las áreas permiten el uso del comando 'home' para teletransportar a un personaje a su designada 'home' (lo cual se configura utilizando el comando @sethome).

PARA OTROS OBJETOS:

La interfaz que presentan los objetos para que puedan ser desplazados, ya sea de una habitación a la otra, o bien entregados a otro personaje es la siguiente (tener en cuenta que a veces puede que el dueño del objeto que va a ser movido no lo permita o que el mismo contenedor no se lo permita).

get -- Levantar un objeto e incorporarlo a nuestro inventario.
Sintaxis: take < objeto > | get < objeto > | take < objeto > from <contenedor> | get <objeto> from <contenedor> | remove <objeto> from <contenedor>
drop – Permite remover un objeto del inventario del emisor del mensaje y dejarlo en la habitación en la que se encuentre.
Sintaxis: drop <objeto> | throw <objeto>
put- Pone un objeto del inventario del emisor del mensaje en el contenedor especificado.
Sintaxis: put <objeto> into <contenedor> | insert <objeto> in <contenedor>
give- Permite cederle un determinado objeto (que el emisor posee) al personaje especificado.
Sintaxis: give <objeto> to <personaje> | hand <objeto> to <personaje>

Usos Conocidos:
Tren, Ómnibus, Elevador, Teletransportadores.

Patrones Relacionados:
Areas: Los transportes mueven objetos entre áreas. Los objetos que implementan Transport pueden ser áreas y/o áreas dentro de otras.
Locomotion: Los pasajeros implementan Locomotion. Los transportes también pueden implementar Locomotion.

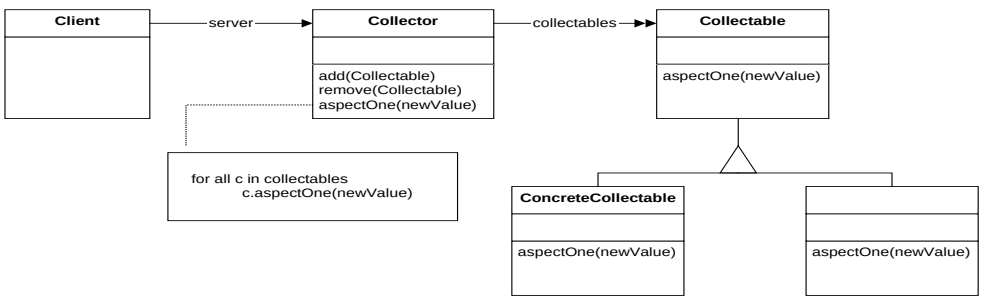
Nombre: Collector

También conocido como:
Propagator

Propósito:
Propagar cambios de las propiedades a un grupo de objetos subrogados. Un objeto colector es usado para ligar un aspecto de ciertos objetos de un grupo, al mismo aspecto de otro objeto distinguido.

Motivación:
Hay casos donde los cambios en aspectos de un objeto deben ser propagados a un grupo de objetos subrogados. Como ejemplo, considerar una bolsa. Los cambios de ubicación son propagados a todos los objetos contenidos dentro de la bolsa. Decimos que una bolsa es un *objeto colector*. Este está interesado en su localización, entonces nosotros lo llamamos *colector de ubicación*.
Un colector está definido por un grupo de objetos subrogados, una propiedad o aspecto, y un mecanismo para propagar cambios en las propiedades de sus objetos subrogados. Los miembros del grupo pueden ser agregados o removidos dinámicamente. La propiedad debe ser significativa para el objeto colector y todos los objetos subrogados. Un colector está construido en base a uno o más aspectos. El colector y los objetos coleccionables pueden tener otros aspectos independientes del comportamiento del colector.

Solución



Client: Efectúa requerimientos al colector.

Collector: Conoce a todos los objetos coleccionables y los aspectos que le son de interés. Propaga cambios.

Collectable: Clase abstracta o interface. El aspecto es significativo para esta. Esta implementa una operación para actualizar el aspecto.

Consecuencias:

1. Los objetos coleccionables pueden ser agregados o removidos dinámicamente.
2. Los objetos coleccionables pueden desenvolverse independientemente.
3. El cliente no hace suposiciones sobre los objetos coleccionables.

Implementación:

1. *Mecanismo de propagación:* El mecanismo de propagación puede ser diferente para cada objeto subrogado

Ejemplo:

Un ejemplo de este patrón puede ser los rooms cuando se los va a remover por completo del MOO. Este aspecto les interesa a los objetos que están dentro del room pues luego de que este sea borrado, tendrían que saber dónde quedarán ubicados. Lo que nos interesa ver es qué propaga el room hacia los objetos que contiene. A continuación, veremos qué políticas se podrían tomar ante esta situación:

- Que deje los objetos en la "nada" (lo cual significa que no tiene ubicación específica).
- Que los deje en algún lugar determinado.
- Que los elimine.

La forma de LambdaMOO para remover por completo un room es la que se describe a continuación:

Si el room está dentro de otro room (rooms anidados), entonces los objetos contenidos dentro del room se dejan en el room exterior al que se quiere borrar.

Si no ocurre lo anterior, lo que se hace es distinguir si los objetos contenidos son personajes o no. Si son personajes, lo que se hace es mandarlos al lugar que especificaron como home; mientras que si son objetos cualquiera, se los devuelve a su dueño.

Una vez que se operó sobre los objetos contenidos, se procede al borrado efectivo del room en cuestión.

Usos conocidos:

Una canasta de compras es un objeto contenedor que lleva productos seleccionados por un usuario en un centro de compras. La ubicación es propagada desde la canasta hacia todos los productos que están dentro de ella.

Patrones relacionados:

Transport: Los transportes triviales, con ningún otro propósito que propagar los cambios de ubicación, pueden ser vistos como colectores de ubicación. Sin embargo, es importante notar que el verdadero poder de los transportes proviene del encapsulamiento de comportamiento navegacional complejo.

4. Conclusiones y trabajo futuro

Los patrones de diseño aparecen como una herramienta extraordinaria para manejar la complejidad del diseño. Favorecen el reuso, la comunicación de diseño y la correctitud. Los patrones ayudan a los desarrolladores de mundos virtuales en muchas áreas para compartir sus experiencias. El lenguaje de patrones presentado en este paper está lejos de ser completo. Hay algunos aspectos del diseño de mundos virtuales que aún deben ser explorados. Los esfuerzos están

actualmente dedicados a los resultados de la aplicación de este lenguaje de patrones en mundos virtuales de diferentes características. Los mundos virtuales existentes, serán explorados en búsqueda de nuevos patrones. Se considerará también la existencia de instancias de patrones de diseño orientados a objetos [6]. En lo que se refiere a la arquitectura de los mundos virtuales, se prestará especial atención en patrones arquitectónicos de Alexander [5].

5. Referencias

[1] High Wired: On the Design, Use, and Theory of Educational MOOs, Cynthia Haynes and Jan Rune Holmevik, Editors the University of Michigan Press.

[2] J. C. Heudin, 1999, *Virtual Worlds. Synthetic Universes, Digital Life and Complexity*. New England Complex Systems Institute, Series on Complexity. ISBN 0-7382-0050-6

[3] Stuart, R. 1996, *Design of Virtual Environment*, ed. McGraw Hill Series on Visual Technology.

[4] Patterns home page. http://hillside.net/patterns/patterns.html

[5] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, S. Angel. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press. 1977.

[6] Erich Gamma, Richard Helm,Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley. October 1994.

[7] LambdaMOO: One of the oldest, most diverse, and largest MOOs in existence. Telnet: telnet://lambda.moo.mud.org:8888

[8] TecfaMOO: Telnet: telnet://tecfamoo.unige.ch:7777

[9] IPL-MOO: Internet Public Library. Telnet: telnet://moo.ipl.org:8888

[10] Diversity University Main Campus: Telnet: telnet://moo.du.org:8888